



Red Hat

Training and Certification

(ROLE)

Red Hat JBoss Enterprise Application Platform 7
AD248

Red Hat JBoss Application Administration I

Edition 8



Red Hat JBoss Application Administration I



Red Hat JBoss Enterprise Application Platform 7 AD248

Red Hat JBoss Application Administration I

Edition 8 20200701

Authors: Douglas Silva, Fernando Lozano, Jim Rigsbee, Ravishankar Srinivasan,
Ricardo Taniguchi, Zachary Guterman
Editor: Steven Bonneville, David Sacco

Copyright © 2019 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are
Copyright © 2019 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but
not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of
Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat,
Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details
contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send
email to training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, Hibernate, Fedora, the Infinity logo, and RHCE are
trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or
other countries.

The OpenStack® word mark and the Square O Design, together or apart, are trademarks or registered trademarks
of OpenStack Foundation in the United States and other countries, and are used with the OpenStack Foundation's
permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the
OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Rob Locke, Bowe Strickland, Scott McBrien, Wander Boessenkool, George Hacker,
Forrest Taylor

Document Conventions	ix
Notes and Warnings	ix
Introduction	xi
JBoss Application Administration I	xi
Orientation to the Classroom Environment	xii
Internationalization	xv
1. Red Hat JBoss Enterprise Application Platform: Architecture and Features	1
Exploring the Architecture of JBoss EAP	2
Quiz: Use cases for standalone server vs. managed domain	12
Installing JBoss EAP	16
Guided Exercise: Installing JBoss EAP	22
Understanding Extensions, Subsystems, and Profiles	29
Quiz: Profile Selection	35
Managing JBoss EAP	39
Guided Exercise: JBoss EAP Administration Console	43
Lab: Red Hat JBoss Enterprise Application Platform: Architecture and Features	49
Summary	54
2. Configuring JBoss EAP as a Standalone Server	55
Running JBoss EAP Standalone Server	56
Guided Exercise: Creating a Standalone Server	62
Configuring JBoss EAP as a Standalone Server	65
Quiz: Configuring Profiles	74
Configuring Interfaces and Socket Binding Groups	76
Quiz: Configuring Interfaces and Socket Binding Groups	79
Lab: Configuring JBoss EAP in Standalone Mode	83
Summary	88
3. Scripting Configuration and Deploying Applications	89
Configuring JBoss EAP with the Command Line Interface	90
Guided Exercise: Exploring the CLI Tool	103
Deploying Applications to a Standalone Server	107
Guided Exercise: Deploying Applications with Management Tools	118
Guided Exercise: Scripting Configuration and Deploying Applications	125
Summary	129
4. Configuring JBoss EAP as a Managed Domain	131
Running JBoss EAP as a Managed Domain	132
Quiz: Managed domains	135
Assigning a Domain Controller	141
Guided Exercise: Assigning a Domain Controller	149
Configuring a Host Controller	153
Guided Exercise: Configuring Host Controllers	159
Configuring a Domain Controller	166
Quiz: Configuring a Domain Controller	171
Lab: Configuring JBoss EAP as a Managed Domain	173
Summary	186
5. Configuring Servers in a Managed Domain	187
Managed Domain Server Architecture	188
Quiz: Hosts and Servers	191
Configuring Server Groups	197
Guided Exercise: Configuring Server Groups	202
Configuring Servers	209
Guided Exercise: Configuring Servers	214
Deploying Applications on a Managed Domain	221

Guided Exercise: Deployment on a Managed Domain	227
Lab: Configuring Servers in a Managed Domain	233
Summary	250
6. Configuring Data Sources	251
Exploring the Data Source Subsystem	252
Quiz: Data Sources	253
Configuring JDBC Drivers	255
Guided Exercise: Configuring JDBC Drivers	258
Configuring Datasources	262
Guided Exercise: Configuring a Data Source	270
Configuring an XA Data Source	276
Guided Exercise: Configuring an XA Datasource	278
Lab: Configuring Data Sources	281
Summary	289
7. Configuring the Logging Subsystem	291
Configuring Logging Handlers	292
Guided Exercise: Configuring Logging Handlers	309
Configuring Loggers	313
Guided Exercise: Controlling Logging Levels	319
Lab: Configuring the Logging Subsystem	323
Summary	333
8. Configuring the Messaging Subsystem	335
Exploring the Messaging Subsystem	336
Quiz: The Messaging Subsystem	341
Configuring Messaging Resources	345
Guided Exercise: Configuring Messaging Resources	353
Configure Journals and Other Settings	363
Guided Exercise: Configuring the Journals and Other Settings	372
Lab: Configuring the Messaging Subsystem	382
Summary	395
9. Securing JBoss EAP	397
Configuring a Database Security Domain	398
Guided Exercise: Securing an Application	403
Configuring an LDAP Security Domain	409
Guided Exercise: Configuring the LDAP Login Module	411
Securing a JMS Destination	415
Quiz: Securing a JMS Destination	418
Configuring the Password Vault	420
Guided Exercise: Encrypting a Password	423
Guided Exercise: Securing JBoss EAP	427
Summary	439
10. Configuring the Java Virtual Machine	441
Configuring the JVM in a Standalone Server	442
Quiz: Configuring the JVM in a Standalone Server	445
Configuring the JVM in a Managed Domain	449
Guided Exercise: Configuring Java Virtual Machines	453
Lab: Configuring the Java Virtual Machine	460
Summary	469
11. Configuring the Web Subsystem	471
Exploring the Features of the Web Subsystem	472
Quiz: Features of Web Subsystem	475
Configuring the Web Subsystem	478

Guided Exercise: Securing HTTP Connections	489
Lab: Configuring the Web Subsystem	495
Summary	506
12. Deploying Clustered Applications	507
Exploring Clustered Applications	508
Quiz: Exploring Clustered Applications	512
Configuring Subsystems that Support Clustered Applications	516
Guided Exercise: Configuring JGroups for a Clustered Web Application	529
Configuring Load Balancer	537
Guided Exercise: Configuring Load Balancing	546
Deploying HA Singleton Applications	551
Quiz: Deploying an HA Singleton	554
Lab: Deploying Clustered Applications	556
Summary	568
13. Configuring the Batch Subsystem	571
Exploring Batch Jobs	572
Quiz: Exploring Batch Jobs	574
Configuring the Batch Subsystem	576
Guided Exercise: Configuring the Batch Subsystem	582
Lab: Configuring the Batch Subsystem	586
Summary	598
14. Discovering what is new in EAP 7	599
Exploring the New Features in JBoss EAP 7	600
Quiz: What's New in JBoss EAP 7	605
Migrating JBoss EAP 6 Applications to JBoss EAP 7	607
Quiz: Migrating Applications to JBoss EAP 7	611
Summary	613
15. Comprehensive Review of Red Hat JBoss Application Administration I	615
Red Hat JBoss Application Administration I Comprehensive Review	616
Lab: Comprehensive Review of Red Hat JBoss Application Administration I	618
Summary	680

Document Conventions



Note

"Notes" are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



References

"References" describe where to find external documentation relevant to a subject.



Important

"Important" boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss, but may cause irritation and frustration.



Warning

"Warnings" should not be ignored. Ignoring warnings will most likely cause data loss.

Introduction

JBoss Application Administration I

JBoss Application Administration I (AD248) focuses on the installation and configuration of the JBoss Enterprise Application Platform 7.0 (EAP7). This hands-on class covers the real-world tasks that a System Administrator needs to know to deploy and manage applications on JBoss EAP.

AD248 revamps the previous course release to support EAP 7 and is meant for System Administrators. The topics in this course cover all of the exam objectives of the RHCJA exam.

Objectives

- Students will learn how to deploy and configure EAP7 in both standalone and a managed domain, as well as use the web management console, the Command Line Interface (CLI), and customize configuration files. Different from the previous version, it will be focused on the CLI management and it was enhanced to be a 5-day course contains over 20 labs and is designed to be 75-80% hands-on training. It includes some new subsystems (batch processing) as well as completely rewritten subsystems (Messaging, Web, and Security).

Audience

- System Administrators who are both new to JBoss and have experience with EAP6.

Prerequisites

Students should meet one or more of the following prerequisites:

- Base experience with system administration on Windows, UNIX, or Linux operating systems
- Understanding of hardware and networking
- No prior knowledge of Java™, scripting, or JBoss Developer Studio is required.

Orientation to the Classroom Environment

In this course, students will begin all hands-on practice exercises and lab work from a virtual machine called **workstation**. From that machine students start SSH sessions to other machines as needed. This is the only VM having a graphical environment. The **workstation** also runs other network services required by some labs such as like an MySQL database and an OpenLDAP server.

All student machines have a standard user account, **student**, with the password **student**. The **root** password on all student systems is **redhat**. Access to the **root** account is available from the **student** account, using the **sudo** command.

Students have two more VMs called **servera** and **serverb** that will be used to create additional EAP server instances. The **classroom** utility server is password-protected from the students and shared by all them.

Classroom Machines

Machine name	IP addresses	Role
content.example.com , materials.example.com	172.25.254.254, 172.25.252.254	Classroom utility server
workstation.lab.example.com , workstationX.example.com	172.25.250.254, 172.25.252.X	Student graphical workstation
servera.lab.example.com	172.25.250.10	Student server
serverb.lab.example.com	172.25.250.11	Student second server

The environment runs a central utility server, **classroom.example.com**, which acts as a NAT router for the classroom network to the outside world. It provides DNS, DHCP, HTTP, and other content services to the students. It uses two alternate names, **content.example.com** and **materials.example.com**, to provide course content used in the practice and lab exercises.

The **workstation.lab.example.com** student virtual machine acts as a NAT router between the student network (**172.25.250.0/24**) and the classroom physical network (**172.25.252.0/24**). **workstation.lab.example.com** is also known as **workstationX.example.com**, where **X** in the host name will be a number that will vary from student to student.

Most activities use the **lab** command, executed on **workstation**, to prepare and evaluate the exercise. **lab** takes two arguments: the activity's name and a verb of **setup**, **grade**, **reset**, **cleanup**, or **solve**.

- The **setup** verb is used at the beginning of an exercise. It will verify that the systems are ready for the activity, possibly making some configuration changes to them.
- The **grade** verb is executed at the end of an exercise. It provides external confirmation that the activity's requested steps were performed correctly.
- The **reset** verb can be used to turn back the clock and start the activity over again, usually followed by **setup**.

Introduction

- The optional **cleanup** verb can be used to selectively undo elements of the activity before moving on to later activities.
- The optional **solve** verb can be used to bring the systems to the desired end result state. This can be used for further investigation of the completed activity or to catch up to a later activity that depends on this one.

In a Red Hat Online Learning classroom, students will be assigned remote computers which will be accessed through a web application hosted at rol.redhat.com [<http://rol.redhat.com>]. Students should log into this machine using the user credentials they provided when registering for the class.

Controlling the stations

The top of the console describes the state of the machine.

Machine States

State	Description
none	The machine has not yet been started. When started, the machine will boot into a newly initialized state (the disk will have been reset).
STARTING	The machine is in the process of booting.
STARTED	The machine is running and available (or, when booting, soon will be.)
STOPPING	The machine is in the process of shutting down.
STOPPED	The machine is completely shut down. Upon starting, the machine will boot into the same state as when it was shut down (the disk will have been preserved).
LOCKED	The virtual machine is locked and waiting for the next stage.
TRANSITIONING	The application is moving from one state to another (e.g., from stopped to running).

Depending on the state of the machine, a selection of the following actions will be available to the student.

Machine Actions

Action (button)	Description
Create Application	Create the application. This creates all of the virtual machines needed for the classroom and starts them. This will take several minutes to complete.
Delete Application	Delete the application. This will destroy all virtual machines in the classroom. Caution: Any work generated on the disk will be lost.
Start Application	Start all machines in the classroom.
Stop Application	Stop all machines in the classroom.

Action (button)	Description
open console	Open a new tab in the browser and connect to the console of the virtual machine. Log in directly to the machine and run commands. In most cases, log in to the workstation machine and use ssh to connect to the other virtual machines.
Start	Start ("power on") the machine. Each machine will have a separate Start button.
Shutdown	Gracefully shutdown the machine, preserving the contents of its disk. Each machine will have a separate button.
Power Off	Forcefully shutdown the machine, preserving the contents of its disk. This is equivalent to removing the power to a physical machine. Each machine will have a separate button.
Restart	Gracefully shutdown the machine, then start it again (reboot). Each machine will have a separate button.
Redeploy	Forcefully shutdown the machine and reset the disk to its initial state. Caution: Any work generated on the disk will be lost. Each machine will have a separate button.
autostop	The timer operates as a "dead man's switch," which decrements as the machine is running. If the timer is winding down to 0, increase the timer if needed.

At the start of a lab exercise, if an instruction to reset **workstation** appears, that means press the **reset** button in the **workstation** console.

At the start of a lab exercise, if an instruction to reset all virtual machines appears, that means on the console of each machine, press the **reset** button.

The station timer

Your Red Hat Online Learning enrollment entitles you to a certain amount of computer time. In order to help you conserve your time, the machines have an associated timer, which is initialized to two hours when your machine is started.

To adjust the timer, click on the **autostop: ...** link. A **Set Autostop Time** window will open. Set the autostop time in hours and minutes (note: there is a four hour maximum time). Press the **Adjust** button to adjust the time accordingly.

Internationalization

Language support

Red Hat Enterprise Linux 7 officially supports 22 languages: English, Assamese, Bengali, Chinese (Simplified), Chinese (Traditional), French, German, Gujarati, Hindi, Italian, Japanese, Kannada, Korean, Malayalam, Marathi, Odia, Portuguese (Brazilian), Punjabi, Russian, Spanish, Tamil, and Telugu.

Per-user language selection

Users may prefer to use a different language for their desktop environment than the system-wide default. They may also want to set their account to use a different keyboard layout or input method.

Language settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the Region & Language application.

Run the command **gnome-control-center region**, or from the top bar, select **(User) → Settings**. In the window that opens, select Region & Language. The user can click the **Language** box and select their preferred language from the list that appears. This will also update the **Formats** setting to the default for that language. The next time the user logs in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications, including **gnome-terminal**, started inside it. However, they do not apply to that account if accessed through an **ssh** login from a remote system or a local text console (such as **tty2**).



Note

A user can make their shell environment use the same **LANG** setting as their graphical environment, even when they log in through a text console or over **ssh**. One way to do this is to place code similar to the following in the user's **~/.bashrc** file. This example code will set the language used on a text login to match the one currently set for the user's GNOME desktop environment:

```
i=$(grep 'Language=' /var/lib/AccountService/users/${USER} \
| sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanese, Korean, Chinese, or other languages with a non-Latin character set may not display properly on local text consoles.

Individual commands can be made to use another language by setting the **LANG** variable on the command line:

```
[user@host ~]$ LANG=fr_FR.utf8 date
jeu. avril 24 17:55:01 CDT 2014
```

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to check the current value of **LANG** and other related environment variables.

Input method settings

GNOME 3 in Red Hat Enterprise Linux 7 automatically uses the IBus input method selection system, which makes it easy to change keyboard layouts and input methods quickly.

The Region & Language application can also be used to enable alternative input methods. In the Region & Language application's window, the **Input Sources** box shows what input methods are currently available. By default, **English (US)** may be the only available method. Highlight **English (US)** and click the **Keyboard** icon to see the current keyboard layout.

To add another input method, click the **+** button at the bottom left of the **Input Sources** window. An **Add an Input Source** window will open. Select your language, and then your preferred input method or keyboard layout.

Once more than one input method is configured, the user can switch between them quickly by typing **Super+Space** (sometimes called **Windows+Space**). A *status indicator* will also appear in the GNOME top bar, which has two functions: It indicates which input method is active, and acts as a menu that can be used to switch between input methods or select advanced features of more complex input methods.

Some of the methods are marked with gears, which indicate that those methods have advanced configuration options and capabilities. For example, the Japanese **Japanese (Kana Kanji)** input method allows the user to pre-edit text in Latin and use **Down Arrow** and **Up Arrow** keys to select the correct characters to use.

US English speakers may find also this useful. For example, under **English (United States)** is the keyboard layout **English (international AltGr dead keys)**, which treats **AltGr** (or the right **Alt**) on a PC 104/105-key keyboard as a "secondary-shift" modifier key and dead key activation key for typing additional characters. There are also Dvorak and other alternative layouts available.



Note

Any Unicode character can be entered in the GNOME desktop environment if the user knows the character's Unicode code point, by typing **Ctrl+Shift+U**, followed by the code point. After **Ctrl+Shift+U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.

For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl+Shift+U**, then **03bb**, then **Enter**.

System-wide default language settings

The system's default language is set to US English, using the UTF-8 encoding of Unicode as its character set (**en_US.utf8**), but this can be changed during or after installation.

From the command line, **root** can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it will display the current system-wide locale settings.

Introduction

To set the system-wide language, run the command `localectl set-locale LANG=locale`, where `locale` is the appropriate `$LANG` from the "Language Codes Reference" table in this chapter. The change will take effect for users on their next login, and is stored in `/etc/locale.conf`.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME, an administrative user can change this setting from Region & Language and clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the login screen will also adjust the system-wide default language setting stored in the `/etc/locale.conf` configuration file.



Important

Local text consoles such as `tty2` are more limited in the fonts that they can display than `gnome-terminal` and `ssh` sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a local text console. For this reason, it may make sense to use English or another language with a Latin character set for the system's text console.

Likewise, local text consoles are more limited in the input methods they support, and this is managed separately from the graphical desktop environment. The available global input settings can be configured through `localectl` for both local text virtual consoles and the X11 graphical environment. See the `localectl(1)`, `kbd(4)`, and `vconsole.conf(5)` man pages for more information.

Language packs

When using non-English languages, you may want to install additional "language packs" to provide additional translations, dictionaries, and so forth. To view the list of available langpacks, run `yum langavailable`. To view the list of langpacks currently installed on the system, run `yum langlist`. To add an additional langpack to the system, run `yum langinstall code`, where `code` is the code in square brackets after the language name in the output of `yum langavailable`.



References

`locale(7)`, `localectl(1)`, `kbd(4)`, `locale.conf(5)`, `vconsole.conf(5)`, `unicode(7)`, `utf-8(7)`, and `yum-langpacks(8)` man pages

Conversions between the names of the graphical desktop environment's X11 layouts and their names in `localectl` can be found in the file `/usr/share/X11/xkb/rules/base.lst`.

Language Codes Reference

Language Codes

Language	\$LANG value
English (US)	en_US.utf8
Assamese	as_IN.utf8

Language	\$LANG value
Bengali	bn_IN.utf8
Chinese (Simplified)	zh_CN.utf8
Chinese (Traditional)	zh_TW.utf8
French	fr_FR.utf8
German	de_DE.utf8
Gujarati	gu_IN.utf8
Hindi	hi_IN.utf8
Italian	it_IT.utf8
Japanese	ja_JP.utf8
Kannada	kn_IN.utf8
Korean	ko_KR.utf8
Malayalam	ml_IN.utf8
Marathi	mr_IN.utf8
Odia	or_IN.utf8
Portuguese (Brazilian)	pt_BR.utf8
Punjabi	pa_IN.utf8
Russian	ru_RU.utf8
Spanish	es_ES.utf8
Tamil	ta_IN.utf8
Telugu	te_IN.utf8

Chapter 1

Red Hat JBoss Enterprise Application Platform: Architecture and Features

Overview

Goal Describe the architecture and features of the Red Hat JBoss Enterprise Application Platform.

- Objectives**
- Describe the architecture and features of JBoss EAP.
 - Describe the available methods for installation and how to install JBoss EAP.
 - Describe the architecture of extensions, subsystems, modules, and profiles.
 - Describe the management options available for JBoss EAP.

- Sections**
- Exploring the Architecture of JBoss EAP (and Quiz)
 - Installing JBoss EAP (and Guided Exercise)
 - Understanding Extensions, Subsystems, and Profiles (and Quiz)
 - Managing JBoss EAP (and Guided Exercise)

- Lab**
- Red Hat JBoss EAP Architecture and Features

Exploring the Architecture of JBoss EAP

Objectives

After completing this section, you should be able to:

- Describe the architecture and features of JBoss EAP
- Describe the relationship between JBoss EAP 7 and JEE 7
- Describe the relationship between JBoss EAP 7 and Wildfly
- Describe what changed in JBoss EAP 7 compared to JBoss EAP 6

What is JBoss EAP?

Red Hat JBoss Enterprise Application Platform 7, JBoss EAP 7, or simply **EAP 7**, is an application server that works as a *middleware platform*, and provides the necessary environment and infrastructure to host and manage Java EE applications.

EAP 7 is built on open standards, based on the **Wildfly** open source software, and provides the following features:

- It provides a reliable, performant, light-weight and supported infrastructure for deploying applications.
- It provides a modular structure that allows service enabling, only when required. This improves performance, has security benefits, and reduces start-up and restart times.
- Web-based management console and management command line interface (CLI) make editing XML configuration files unnecessary, and provide the ability to script and automate tasks.
- It is Java Enterprise Edition 7 full, and web profile, certified.
- It provides centralized management of multiple server instances and physical hosts, while a standalone server allows for a single server instance.
- Pre-configured options for features such as high-availability clustering, messaging, and distributed caching are also provided.

JBoss EAP 7 is a **cloud-ready** application server. It has a very small footprint, fast start-up time, and is automation-friendly. Those features are essential enablers for having an efficient deployment of an application server to a cloud infrastructure. Beyond that, Red Hat provides another offerings for improving EAP 7 cloud deployments:

- **xPaaS** container images for many JBoss middleware products, including EAP 7, as part of the **OpenShift Enterprise** product
- Container images for use with Red Hat Enterprise Linux Atomic Host and other Docker-based container hosts, provided by the Red Hat container image registry
- Ready to use, supported VM images for Amazon AWS, and other cloud platforms such as **Red Hat Enterprise Linux Open Stack Platform** and Microsoft Azure deployments to follow

- Discovery of managed domain server instances deployed as cloud instances
- Discovery of clustered server instances deployed as cloud instances or as containers

JBoss EAP 7 is part of a growing family of Red Hat JBoss Middleware products, such as:

- **JBoss A-MQ**: Multiprotocol, high-performance messaging platform that delivers information reliably, enabling real-time integration
- **JBoss BPM Suite**: Business Process Management
- **JBoss BRMS**: Business rule management, business resource optimization, and complex event processing (CEP)
- **JBoss Data Grid**: In-memory, distributed, NoSQL datastore solution
- **JBoss Data Virtualization**: Data supply and integration solution that sits in front of multiple data sources and allows them to be treated as single source
- **JBoss Developer Studio**: Integrated development environment (IDE) for developing, testing, and deploying rich web apps, mobile web apps, transactional enterprise apps, and service-oriented architecture (SOA)-based integration apps and services
- **JBoss Fuse**: Lightweight, flexible integration platform that enables rapid integration across the extended enterprise

To better understand the role EAP 7 plays inside a IT infrastructure, two application scenarios are presented. EAP 7 subsystems, referred to by the figures, will be explained later in this student guide. There is no need to understand what the Undertow or Infinispan subsystems are to understand those example scenarios.

The following figure shows an example of how EAP 7 fits into a simple environment:

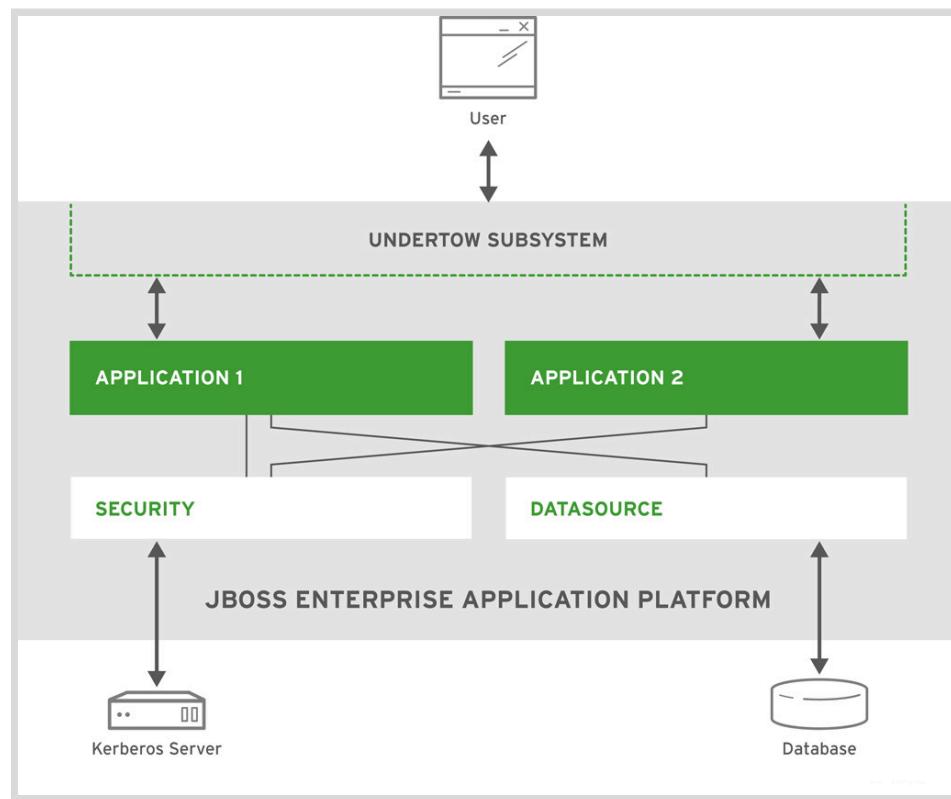


Figure 1.1: A single EAP 7 server instance

In this first example, the EAP 7 server instance has two applications deployed to it. It is also configured to connect to a database and a Kerberos server. The JBoss EAP instance handles network requests and directs those requests to the appropriate application. The applications use the APIs exposed by JBoss EAP to connect to the database and Kerberos server, and perform their implemented business logic.

A more complex environment is presented in the next figure:

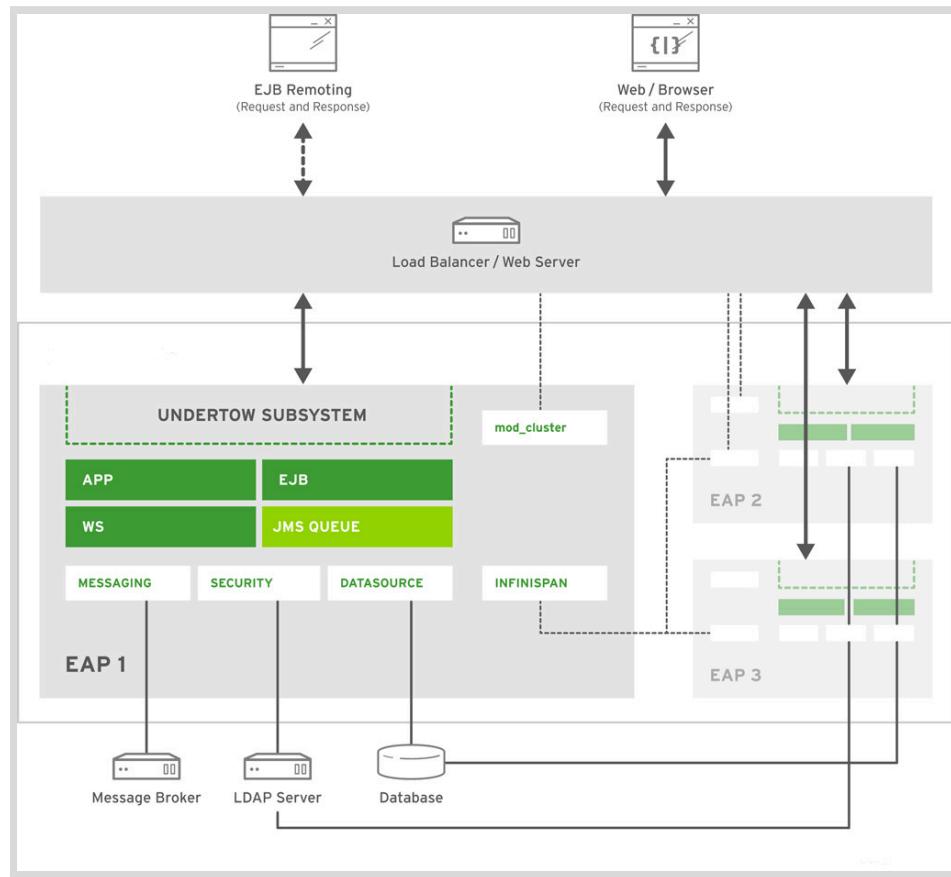


Figure 1.2: A cluster of three EAP 7 server instances

In this second example, three JBoss EAP instances are configured as part of a cluster, and fronted by a load balancer or a web server. State is replicated between the server instances so that any one of them can take over user sessions being served by a failed instance . All three EAP instances have a web application, a web service, and Enterprise JavaBeans (EJB) deployed, besides having connections to a database and an Lightweight Directory Access Protocol (LDAP) server. One EAP instance has also a Java Messaging Service (JMS) queue configured to connect to an external message broker.

In both example scenarios, EAP 7 makes developing applications easier because it provides JEE APIs for accessing databases, authentication, and messaging. Common application functionality is also supported by JEE APIs and frameworks, provided by EAP, for developing web user interfaces, exposing web services, implementing cryptography, and other features. JBoss EAP also makes management easier by providing runtime metrics, clustering services, and automation.

Java EE and the JCP

EAP implements the Java EE 7 specification also known as JEE 7, and is compliant with the various profiles that are defined in Java EE 7. Profiles are a new concept introduced in Java EE 6. A

profile is a collection of Java technologies that target a specific audience. There are currently two profiles defined in Java EE 7:

- **Full Profile:** Contains all of the Java EE technologies.
- **Web Profile:** Contains a full stack of Java EE APIs for developing modern dynamic web applications.

There are over 30 different technologies that comprise the Full Profile of Java EE. Each of these technologies have their own specification and version number. Combined, they provide an impressive list of capabilities that allow Java EE applications to connect to databases, publish and consume Web Services, serve up Web applications, perform transactions, implement security policies, and connect to a multitude of outside resources for tasks like messaging, naming, sending emails, and communicating with non-Java applications.

The following figure illustrates the main set of APIs required by JEE7, showing which APIs are new, which ones had major updates, and which ones had only minor updates.

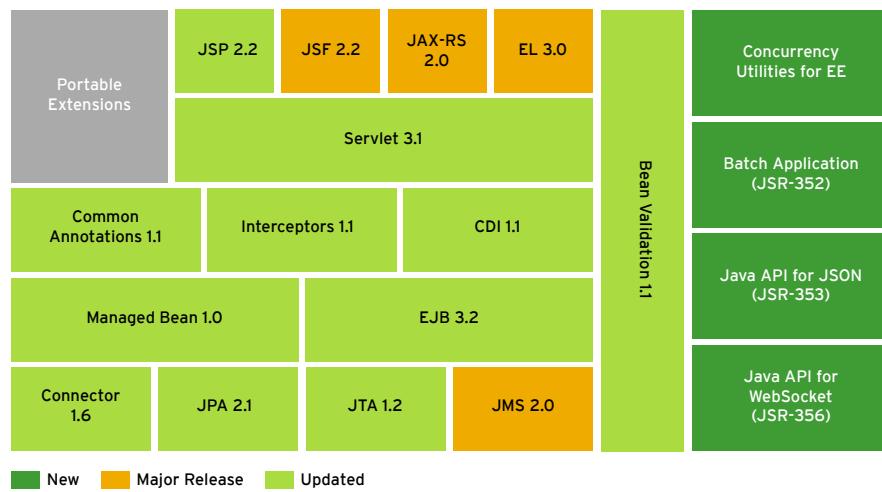


Figure 1.3: Java EE 7 new and updated APIs

The Web Profile contains the web-based technologies of Java EE that are commonly used by web developers, like Servlets, Java Server Pages, Java Server Faces, CDI, JPA, JAX-RS, WebSockets, and a slimmed-down version of Enterprise Java Beans (EJBs) known as EJB Lite.

Note

EJB Lite is a subset of the EJB 3 specification and is designed to allow web developers to gain some of the advantage of using EJBs without requiring all the capabilities of the Full Profile. EJB Lite does not provide remote interfaces, message-driven beans, and SOAP Web Services endpoints, but it does provide support for Stateful and Stateless Session Beans and Entity Beans. The Java EE 7 specification contains more detail.

The JEE standards are defined by the **Java Community Process (JCP)**, a formalized mechanism that allows interested parties to develop standard technical specifications for Java technology. JCP membership for organizations and commercial entities requires annual fees, but is free for individuals.

Each JCP standard is a **Java Specification Request (JSR)**, identified by a number. A JSR starts with a formal specification document that defines how the technology is supposed to work

from the perspective of application developers, system administrators, and middleware vendors. Formal public reviews of JSRs take place before a JSR becomes final, and is approved by the JCP Executive Committee.

A final JSR provides both a **Reference Implementation (RI)**, that shows how the technology is supposed to work, and a **Test Compatibility Kit (TCK)**, that competing implementations are required to have for a 100% pass score, in order to be able to claim they are compatible with the standard described by the JSR.

For example, JEE 7 is defined by **JSR 342**, which itself refers to a number of other JSRs to specify individual technologies and profiles. One of them is the **CDI API**, is specified by **JSR 365**. Its Reference Implementation (RI) is a JBoss community projects named **Weld**.

The JBoss product family and open source projects

The JBoss community develops a number of open source projects, ranging from development tools, to middleware platforms and utility libraries. The most famous of them is the **JBoss Application Server**, or simply **JBoss AS**, later renamed to **Wildfly** application server, which is the basis of the Red Hat JBoss EAP product.

One of the stated goals of the Wildfly project is to be very modular, so individual features can be developed and experiments run with without impact the development schedule of other features. Users do not have runtime overhead created by unused features. Because of this, release 9 of the Wildfly application server was split into two main projects:

- **Wildfly Core** provides the application server management kernel.
- **Wildfly Full** provides a certified JEE application server, built over Wildfly Core.

When Red Hat productizes an open source project, it assumes contractual responsibilities of providing hardened, stable, and dependable software to its customers. This includes support SLAs, timely bug fixes, and keeping backwards compatibility, among others. Sometimes those responsibilities are at odds with the community goals, which are provide fast innovations and allow wide participation in the software development.

Because the community and the product goals are different, there is no one-to-one correspondence between a community project release and a supported product release. New features are usually developed in the community. When they become sufficiently stable and there is enough market demand, they are brought into the product, without bringing together other community features that are not considered yet ready for production environments.

What is new in JBoss EAP 7

JBoss EAP 6 was the most ambitious release ever of the JBoss Enterprise Application Platform to date. It was a complete rewrite, implementing a new modular architecture, to solve the main pain points of previous EAP releases and of JEE application servers in general. This includes faster startup, faster deployments, easier management, lower footprint, and cloud readiness.

EAP 7 builds on the EAP 6 architecture, adding new components and replacing old components with new, innovative open source projects. Here are the main new features of EAP 7, from an application developer perspective:

- **Requires Java 8:** Takes full advantage of Java memory management improvements (no more Permanent Generation!) and new APIs
- **Java EE 7 full and web profile certification:** Among the new and updated APIs in JEE 7 are:
 - *Batch processes:* Batch 1.0, and Concurrency 1.0

- *Web applications and services:* WebSocket 1.1, Servlet 3.1, JSF 2.2, JSP 2.3, EL 3.0, JAX-RS 2.0, JAX-WS 2.2, and JSON-P 1.0
- *Enterprise components:* EJB 3.2, JPA 2.1, JMS 2.0, JCA 1.7, and JTA 1.2
- *IoC and DI framework:* CDI 1.1, CDI Extensions, Interceptors 1.2, Common Annotations 1.1, Managed Beans 1.0, and Bean Validation 1.1
- **JavaScript support:** Based on the Nashorn engine, with transparent access to JEE APIs from JavaScript code
- **Interoperability with EAP5 and 6:** Remote calls from EAP 7 to EAP 5 and 6 are supported using RMI, IIOP, and SOAP
- **Clustered Singleton:** Create components and deployments that are activated in a single server instance, in a clustered group, and failover to a surviving instance if needed
- **Embedable:** Run EAP 7 components inside an application to create even lighter environments

Here are the new features from a system administrator perspective:

- **Port reduction:** Based on HTTP websockets, now a single port (8080) serves all client protocols. A second port (9990) serves both the Management Console, the Management CLI, and the Management API
- **Streamlined and richer Management Console:** Is capable of dealing with large managed domains and perform operations on resources that previously required using the Management CLI
- **Off-line CLI:** Run an embedded domain controller or standalone server inside the CLI client to update configurations, without risking client requests
- **Graceful shutdown:** allow current request processing to be finished before shutting down servers
- **Suspended mode:** allow a server instance, server group, or managed domain to be unavailable for client requests while performing administrative operations
- **Cloning profiles:** profiles can be cloned, so customizations do not affect the standard factory profiles
- **Hierarchical profiles:** have multiple profiles that inherit configurations from a common parent profile, to share customizations
- **Native load-balancer:** No more need of an external component (for example, Apache Httpd + mod_jk) to load balance HTTP requests to clustered applications
- **HTTP/2:** Is the improved new release of the HTTP protocol, including Servlet API 4.0 features from Java EE 8 for application developers
- **OAuth 2.0/SAML:** Provides native support for common Web SSO standards
- **Remote log access:** Now server instance logs can be viewed and downloaded using the management CLI and API
- **Managing EAP 6 instances:** an EAP 7 domain controller can manage hosts and server instances running EAP 6, for supporting deprecated technologies or staged upgrades.

Management objects and operations supported from those servers are available to the Management Console and CLI, even when they are not supported by EAP 7

A few features and components from EAP 6 were dropped by EAP 7 to avoid carrying dead weight. They were either deprecated and not required by Java EE 7 anymore, or replaced by newer EAP 7 components:

- **CMP:** Rewrite applications persistence layer to use EJB 3 and JPA.
- **JAXR:** The SOAP-based web services registry never got market traction.
- **JAX-RPC:** Rewrite SOAP web services to use JAX-WS.
- **JSR-88:** The deployment API never got market traction.
- **Threads subsystem:** now each subsystem manages its own thread pools, using the new JEE 7 Concurrency API. Some subsystems now use non-blocking I/O and do not need thread pools anymore.
- **OSGi:** The Enterprise OSGi specifications were never finished and had not gotten much market traction. Red Hat JBoss Fuse provides a supported OSGi runtime, based on Apache Karaf.

Replaced components are of special relevance to system administrators, because they mean using different configuration objects, commands, and syntax compared to previous releases. Most of the time they should not directly affect application developers, if they stick to the standard JEE APIs. Only system administrators are affected as follows:

- **Messaging (HornetQ):** Is replaced by the **ActiveMQ-Artemis** project, which is a merge of the original HornetQ code into ActiveMQ.
- **JBoss Web (Tomcat):** Is replaced by the **Undertow** web server.
- **Jacorb:** Is replaced by the **OpenJDK IIOP broker** which provides better interoperability with both older EAP releases, other vendor application servers, and CORBA brokers.
- **Picketbox and Picketlink:** are being replaced by **Elytron** as the security subsystem and **Keycloak** as the SSO provider.
- **Apache CXF Spring configuration:** Is replaced by **CDI beans** and **CDI Extensions**, avoiding the need to expose the Spring framework annotations, configuration files, and classes to applications that would not need them otherwise.

The management CLI provides migration operations that perform a best effort conversion of configuration, using the replacement components for the retired components.

Standalone server vs. managed domain

A JBoss EAP 7 installation can be run in two operating modes: **standalone server** and **managed domain**, as illustrated by the following figure:

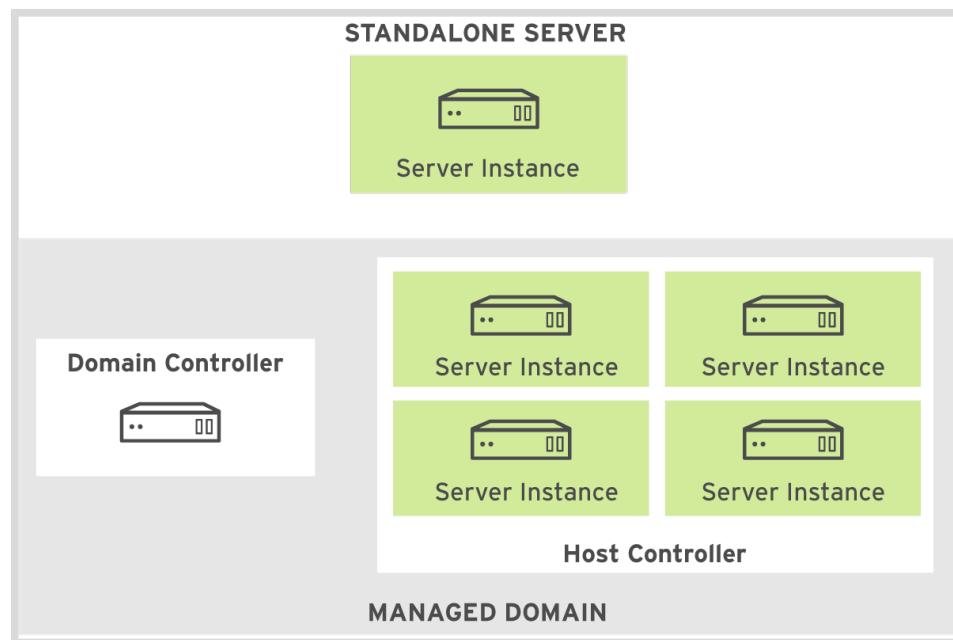


Figure 1.4: JBoss EAP 7 standalone server vs managed domain operating modes

Standalone Server

JBoss EAP versions up to version 5 had a single operating mode which is similar to what EAP 7 calls the **standalone server** operating mode. EAP 6 referred to this operating mode as the **standalone mode**. If centralized management capabilities are not needed, EAP 7 can be provisioned as a standalone server.

Standalone server mode represents a single server instance with a single configuration file named **standalone.xml**.

Managed Domain

To have the ability of managing multiple server instances, possibly deployed into multiple hosts, from a centralized location, EAP can be provisioned in a **managed domain**. This is the same operating mode that EAP 6 referred to as the **domain mode**.

A **domain controller** process acts as the central management control point, communicating with the various **host controllers** in the managed domain. All of the hosts share a common management policy and the domain controller ensures that each server instance is configured according to that policy.

Each host in a domain has a host controller. Exactly one of the host controllers in a domain is designated as the domain controller. That is, the domain controller **is also** a host controller.

The domain controller is also referred to as the **master**, and the other hosts controllers in the domain are referred to as **slaves**. Nothing prevents a domain controller having its own server instances, but having it as a dedicated administration host controller, as suggested by the previous figure, is a common practice.

A managed domain can have server instances using different configurations. This is possible because the managed domain configuration defines a number of different **profiles**, and groups of server instances are assigned to a single profile.

**Note**

The term *profile*, when referring to an EAP 7 managed domain configuration is unrelated to JEE specification profiles.

The managed domain is configured in a file named **domain.xml**, which exists only in the domain controller. Each host controller, including the domain controller, also has a specific configuration, which is in a file named **host.xml**. When a host starts, its settings from **host.xml** are combined with the settings from **domain.xml** on the domain controller.

The only difference between standalone server and managed domain operating modes is the centralized management capability. All EAP features, like security, transactions, messaging, clustering, high availability, and other Java EE technologies, are available to server instances running in either standalone server or managed domain operating modes.

Host vs. server

A **host** in EAP 7 represents a single machine, from an OS point of view, that runs application server instances. EAP does not care whether it is a virtual machine or a physical machine.

When deploying EAP as a managed domain, each host is managed by a single host controller and contains zero or more server instances. The host controller acts as a management agent, relaying commands sent by the domain controller to the affected server instances.

Server instances and host controllers run as their own OS processes, each one inside a dedicated Java Virtual Machine (JVM). Each host in a managed domain also runs a **process controller** process in another dedicated JVM. The process controller is the OS process that starts new processes for host controllers and server instances.

The domain controller JVM runs the Management Console and Management API, so all administrative requests are sent to the domain controller. It relays administrative commands to the specific host controllers that manage the affected server instances.

Comparing server instances in a managed domain with standalone servers, the former runs only application code, not management services. Standalone servers run the Management Console and the Management API under the same JVM that runs applications. This is the same as with older JBoss EAP versions where a single, running instance of EAP represented a single server.



References

For details on profiles, and more information on Java EE 7, please visit:

The JEE 7 Whitepaper

<http://www.oracle.com/technetwork/java/javaee/javaee7-whitepaper-1956203.pdf>

JEE 7 Specification

<https://jcp.org/en/jsr/detail?id=342>

For details on JBoss community and Wildfly, please visit:

The JBoss Community

<http://www.jboss.org>

Wildfly community

<http://wildfly.org>

For details on the JBoss EAP 7 product and the JBoss Middleware products, please visit:

JBoss Middleware Products

<http://www.redhat.com/en/technologies/jboss-middleware>

JBoss EAP Product Page

<https://www.redhat.com/en/technologies/jboss-middleware/application-platform>

JBoss EAP Product Documentation

<https://access.redhat.com/documentation/en/jboss-enterprise-application-platform/>

To find definitions for terms used in this section, like middleware and application server, please refer to the Wikipedia:

Definition of middleware

<https://en.wikipedia.org/wiki/Middleware>

Definition of application server

https://en.wikipedia.org/wiki/Application_server

► Quiz

Use cases for standalone server vs. managed domain

Choose the correct answer to the following questions:

- ▶ 1. A developer is working on his desktop, developing Java EE applications using JBoss Developer Studio and testing his code on the same desktop machine. Which would be the most appropriate operating mode for EAP 7? (Choose only one)
 - a. Standalone server
 - b. Managed domain
 - c. Embedded server
 - d. EAP 7 can not be integrated into current JBDS releases

- ▶ 2. You need to deploy a two-node cluster for fail-over purposes. Which operating modes can be implemented with this requirement? (Choose two.)
 - a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. Standalone server plus an external clustering middleware for example Red Hat RHEL7 High Availability Add-On

- ▶ 3. Your server topology requires EAP running in a cluster of four machines, each machine with a single EAP server instance and each server having identical configurations. Which EAP 7 operating mode would provide the easiest management? (Choose only one.)
 - a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. Both standalone server and managed domain require manual duplication of configuration between server instances

- **4. Three Java EE applications are deployed on one EAP Server instance. Which EAP 7 operating mode would provide the simplest and lightest setup? (Choose only one.)**
- a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. No difference between standalone server and managed domain in this scenario
- **5. A single Java EE application is deployed on one EAP server instance. Which EAP 7 operating mode can be evolved into a cluster with two server instances? (Choose two.)**
- a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
- **6. You have 12 EAP Servers that all need to be configured identically. Which EAP 7 operating mode would allow the quickest setup time for this scenario? (Choose only one.)**
- a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. Both standalone server and managed domain would be the same

► Solution

Use cases for standalone server vs. managed domain

Choose the correct answer to the following questions:

- ▶ 1. A developer is working on his desktop, developing Java EE applications using JBoss Developer Studio and testing his code on the same desktop machine. Which would be the most appropriate operating mode for EAP 7? (Choose only one)
 - a. Standalone server
 - b. Managed domain
 - c. Embedded server
 - d. EAP 7 can not be integrated into current JBDS releases

- ▶ 2. You need to deploy a two-node cluster for fail-over purposes. Which operating modes can be implemented with this requirement? (Choose two.)
 - a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. Standalone server plus an external clustering middleware for example Red Hat RHEL7 High Availability Add-On

- ▶ 3. Your server topology requires EAP running in a cluster of four machines, each machine with a single EAP server instance and each server having identical configurations. Which EAP 7 operating mode would provide the easiest management? (Choose only one.)
 - a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. Both standalone server and managed domain require manual duplication of configuration between server instances

- **4. Three Java EE applications are deployed on one EAP Server instance. Which EAP 7 operating mode would provide the simplest and lightest setup? (Choose only one.)**
- a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. No difference between standalone server and managed domain in this scenario
- **5. A single Java EE application is deployed on one EAP server instance. Which EAP 7 operating mode can be evolved into a cluster with two server instances? (Choose two.)**
- a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
- **6. You have 12 EAP Servers that all need to be configured identically. Which EAP 7 operating mode would allow the quickest setup time for this scenario? (Choose only one.)**
- a. Standalone server
 - b. Managed domain
 - c. Clustered domain
 - d. Embedded server
 - e. Both standalone server and managed domain would be the same

Installing JBoss EAP

Objectives

After completing this section, you should be able to:

- Describe the available methods for JBoss EAP 7 installation
- Install JBoss EAP 7

EAP 7 installation methods

There are four main methods to install EAP 7:

1. **Zip File:** EAP 7 is distributed as a zip file that can be extracted and executed on an operating system with Java 8 installed. It contains all the libraries, configuration files, and scripts needed to start EAP, without further customization. Normally, it is the choice for most developers interested in using EAP7 out-of-the-box.
2. **RPM:** EAP 7 is available as an RPM package for users with a valid subscription to the JBoss Enterprise Application Platform sub-channel, in the JBoss Enterprise Application Platform group. The RPMs will be downloaded, by the **yum** command, to the host OS. All the files needed to start and manage EAP will be installed.



Note

The RPM installation method is only relevant for RHEL OS.

3. **The GUI Installer:** It is a Java based application that is responsible for providing a step-by-step way to customize the environment setup. Due to its portable nature, it can be used on multiple platforms and it requires Java 8 installed on the host OS. It can be run using either a graphical interface or a text based console. Due to its flexible nature, it can be installed using a response file, with all the options passed as an XML file, to allow installation of EAP 7 on multiple hosts in an automated fashion.



References

EAP 7 can be downloaded from: <https://access.redhat.com/jbosssnetwork/restricted/listSoftware.html?product=appplatform&downloadType=distributions>

4. **Containerized EAP:** Alternatively, EAP is also provided as a container image based on Docker and it can be pulled from the Red Hat Docker Registry. It was developed to support Red Hat Enterprise Linux Atomic Host and OpenShift Java EE deployment and it is part of the **xPaaS** initiative from Red Hat for widespread EAP deployment. Using this image, developers can easily build, scale, and test applications deployed across hybrid environments.



References

For more details, refer to the documentation on OpenShift:

https://docs.openshift.com/enterprise/3.1/using_images/xpaas_images/eap.html



Note

This installation method is out of scope for this course. It is addressed in the RH270 and DO276 courses.



Note

Refer to comparisons on the different methods available for EAP installation:

https://access.redhat.com/sites/default/files/attachments/considerations_installation_zip_vs_rpm_vs_wizard_installer.pdf

Configuration Management Tools

In large deployment scenarios, where EAP 7 has to be installed on multiple servers with uniform settings, configuration management tools like Ansible and Puppet can be used. Multiple Puppet and Ansible modules were developed to simplify the deployment and upgrade process for JBoss EAP.



References

Ansible JBoss modules

https://docs.ansible.com/ansible/jboss_module.html

Puppet

<https://forge.puppetlabs.com/coi/jboss>

EAP 7 installation

To install EAP, use one of the following commands:

- The simplest way to install EAP is by extracting the ZIP file. This is the usual package choice for developers since no configuration is needed. To unzip the EAP 7 zip file, run the following command in a terminal:

```
$ unzip jboss-eap-7.0.0.zip
```

- To install EAP 7 on a RHEL machine, using the RPM method, run the following command in a terminal either as root or using **sudo**:

```
# yum groupinstall jboss-eap7
```

- To start the EAP Installer, enter the command:

```
$ java -jar jboss-eap-7.0.0-installer.jar
```

The following screen shot shows the EAP installer in GUI mode:



Figure 1.5: The EAP 7 GUI Installer

If no GUI access is available on a machine and it can only be accessed via a terminal, then the EAP Installer can be executed from a console. Both the graphical installer and the text console based installer follow the same set of steps. To launch the installer in console mode, run :

```
# java -jar jboss-eap-7.0.0-installer.jar -console
Select language :
0: eng
1: chn
2: deu
3: fra
4: jpn
5: por
6: spa
Please choose [0] :
END USER LICENSE AGREEMENT
JBoss(r) ENTERPRISE MIDDLEWARE(tm)
...
press 1 to continue, 2 to quit, 3 to redisplay
1
Select the installation path: [/opt/jboss-eap-7.0]
```

Whether using the GUI or the console, the EAP installer creates an XML document that saves all settings. This XML document can be saved to repeat an installation without any human interaction:

```
$ java -jar jboss-eap-7.0.0-installer.jar filename.xml
```

You would replace **myinstall.xml** with the appropriate name of your auto-install XML file. This command does not require any human input, allowing you to install a custom EAP 7 instance using scripting.

Uninstalling EAP 7

The process to uninstall EAP 7 depends in the installation method used:

- If EAP 7 was installed by using the zip file, then simply delete the top level folder where it was unzipped.

```
# rm -rf jboss-eap-7.0
```

- To uninstall EAP 7 on a RHEL machine that was installed using the RPM method, run the following command in a terminal either as root or using **sudo**:

```
# yum groupremove jboss-eap7
```

- If EAP 7 was installed using the EAP 7 GUI Installer or console, navigate to **\$JBOSS_HOME/uninstaller** and run the uninstaller:

```
# java -jar uninstaller.jar
```

Starting and Stopping EAP

Unix shell scripts and Windows batch scripts are provided to start EAP 7 as either a standalone server or in domain mode. They are **standalone.sh** and **domain.sh** and are found under the **bin** folder in the EAP installation directory.

The EAP installation directory will be referred throughout this book as **JBOSS_HOME**. This is also the name of an OS environment variable used by the EAP startup scripts and other utility scripts to find its installation directory and also find configuration files, log files, and other needed files. If this variable is not defined, it is assumed to point to the parent folder of the script.

To start EAP with out-of-the-box configurations, run the startup script specific for the desired operating mode:

- For a standalone server:

```
$ ${JBOSS_HOME}/bin/standalone.sh
```

- For a managed domain host controller (which will be the master):

```
$ ${JBOSS_HOME}/bin/domain.sh
```

Most of the time, the environment variable can be referred to by shell commands as simply **\$JBOSS_HOME** instead of **\${JBOSS_HOME}**, as in the previous command examples.

To stop EAP, there are three choices:

- Interrupt the server instance (or host controller) process using **Ctrl+C**
- Kill the server instance (or host controller) process using the **kill** command (Unix and OS X) or a GUI task manager.

- Use the management CLI (shown later in this student guide.)

Test if EAP is working

After EAP starts, it listens for HTTP connections on port 8080. Using a web browser to visit the <http://localhost:8080> URL should show a welcome page.

EAP installation folders

A fresh EAP installation has the following folders under **JBoss_HOME**:

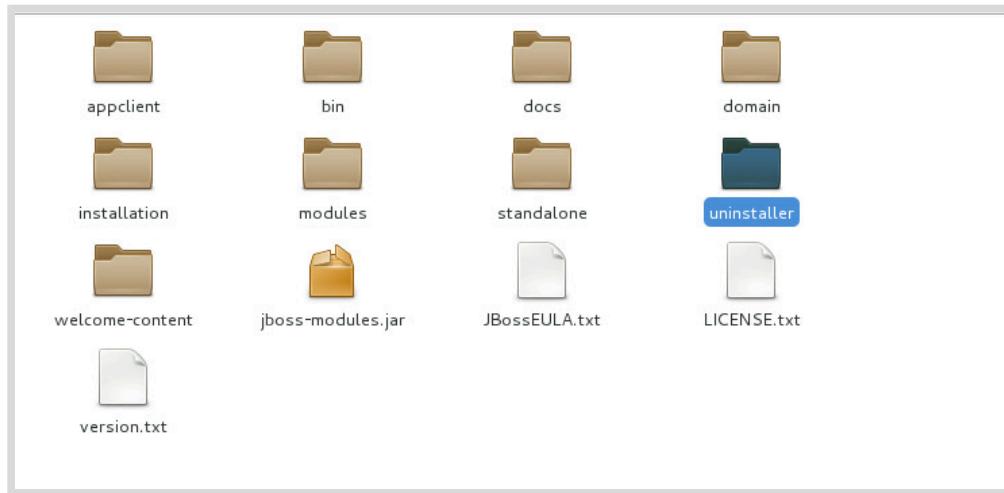


Figure 1.6: EAP 7 installation folders

The main folders are described by the following bullets:

- **bin**: Contains scripts used for starting EAP in managed domain and standalone server, start the management CLI and other utility functions for developers.
- **domain**: Contains configuration and data files for running EAP in a managed domain
- **standalone**: Contains the configuration and data files for running EAP standalone server.
- **modules**: Contains most of the code that implement JEE services in EAP.

The contents of these and other EAP installation folders will be examined throughout the remainder of this book. For now, notice that **domain** and **standalone** have similar structures, including a **configuration** folder that contains the XML configuration files for each operating mode, and a **logs** folder that may be useful in troubleshooting startup failures.

Production environment recommended practices

A developer may install EAP 7 into his personal home folder and run a standalone server instance or host controller under his own OS user account. But for production environments, there are a series of recommended practices to follow for any server-based software.

Please check with the senior system administrator for the specific set of recommended practices for your organization. They will probably include at least the following ones for Linux OSes:

- Install to a folder reserved for third-party applications such as **/opt**.
- The owner of the EAP installation folder and its files should be a **system account**, which is an user who is NOT allowed to login to either the local server machine console or to remote sessions.

Starting EAP manually requires using either **sudo** or **su** to run the server startup script, using the system account created exclusively for EAP.

- EAP standalone server instances and host controllers should be started as system services, in the background, not tied to a user session. For RHEL 7, this means running either as a **systemctl** service unit or a **System V** init script. Follow those steps:
 - The EAP installation folder **JBoss_HOME/bin/init.d** contains a sample System V init script named **jboss-eap-rhel.sh** that was written for RHEL7, but can be customized for other Unix OSes.
 - The same folder also contains a configuration file, named **jboss-eap.conf**, that defines parameters specific for a particular installation. For example, the correct value for **JBoss_HOME** is defined.
 - Copy the init script to **/etc/init.d**, and the script configuration file to **/etc/default**.
 - Reload the **systemctl** in-memory configuration:

```
# systemctl reload-daemon
```

- Use the **systemctl** command to start EAP. For example:

```
# systemctl start jboss-eap-rhel.sh
```

- The **systemctl** command can also be used to stop EAP. For example:

```
# systemctl stop jboss-eap-rhel.sh
```

- At last, **systemctl** can be used to automatically start EAP during boot.

```
# systemctl enable jboss-eap-rhel.sh
```

Using **systemctl** and init scripts is a common Linux system administration task. Please check the RHEL 7 product documentation for more details, or ask for help from an experienced Unix system administrator.

The most common recommended practices just described are already implemented by the RPM installation method, but have to be implemented manually for the ZIP file method and the JAR installer.

EAP 7 also includes batch scripts and executable files for installation and running as a Windows service, but this will not be covered by this student guide.



References

Systemd documentation

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Managing_Services_with_systemd.html

► Guided Exercise

Installing JBoss EAP

In this lab, you will install EAP using the EAP installer file(JAR file).

Resources	
Files	/home/student/JB248/install
Application URL	http://localhost:8080

Outcome

You should be able to install an instance of EAP 7.

Before You Begin

Use the following command to download the relevant lab files, and to verify that no prior JBoss EAP installation exists:

```
[student@workstation ~]$ lab install-eap setup
```

- 1. EAP can be installed in multiple ways (**yum install**, JAR installer, zip file). However, for the purposes of this lab, the JAR-based GUI installer is the simplest. The procedure is the same on different Operating Systems (Windows, Linux, Mac OS X, etc).

Start the EAP GUI Installer:

- 1.I. Open a terminal window and change to the **/home/student/JB248/install** folder.
Open a terminal window from the workstation (**Applications → Utilities → Terminal**) and run the following command:

```
[student@workstation ~]$ cd /home/student/JB248/install
```

- 1.2. The EAP installer is a GUI based application that will uncompress the EAP server files on your workstation and help you set up an admin user for managing EAP. The installer should be run with root privileges, since you will be installing EAP into the **/opt** folder on your workstation.

Enter the following command:

```
[student@workstation installs]$ sudo java -jar \
jboss-eap-7.0.0-installer.jar
```

- 1.3. The EAP installer supports a number of languages. For this course, select the language as English (default) and click **OK**. Also select the **I accept the terms of the license agreement** check box.

- 1.4. Briefly read through and accept the End User License Agreement (EULA) and click **Next**.
- 1.5. For the installation path, use:

```
/opt/jboss-eap-7.0
```

Installation Path

Select the installation path:

- 2. Complete the GUI Installer:

- 2.1. The Component Selection screen shows which packs (server components) are going to be installed. The **AppClient** and **Docs** packs are optional and can be unselected. The **AppClient** is a set of files used to start EAP programmatically. The **Docs** pack contains some configuration file templates for usage. Click the **Next** button to continue without changing the default values.

Component Selection

Note: Disabled packs are required.

<input type="checkbox"/>	<input checked="" type="checkbox"/> Red Hat JBoss Enterprise Application Platform	184 MB
	<input checked="" type="checkbox"/> AppClient	38.94 KB
	<input checked="" type="checkbox"/> Docs	8.4 MB
	<input checked="" type="checkbox"/> Modules	157.19 MB
	<input checked="" type="checkbox"/> Welcome Content	2.11 MB

- 2.2. You need to create an administrative user to access the EAP Management Console and manage the EAP server after installation.

Use **jbossadm** as the Admin username and **JBoss@RedHat123** for the user password.

Create an administrative user

This user will be added to the host container's management realm for administrative purposes. It can be used to access the management console, the management CLI or other applications secured in this realm.

The password must be at least eight characters long, with one alphabetic character, one digit, and one non-alphanumeric character not including &.

Create an administrative user.

Admin username:

Admin password:

Confirm admin password:

- 2.3. Review the information entered so far and click the **Next** button to continue.

Installation Overview

Red Hat JBoss Enterprise Application Platform is now ready to install. The important data points are listed below. Click "Next" to start the installation.

Installation path

/opt/jboss-eap-7.0

Chosen installation packs

Red Hat JBoss Enterprise Application Platform
AppClient
Docs
Modules
Welcome Content

Administrative User

Username: admin

- 2.4. The EAP installer will begin installing selected components. After the installation is complete, you should see the following screen:

Component Installation

Currently installing: welcome-content/noredirect.html

Red Hat JBoss Enterprise Application Platform
AppClient
Docs
Modules
Welcome Content



Click the **Next** button to continue.

- 2.5. You can customize the EAP installation and select the subsystems you want to customize in this screen. For the purposes of this lab you should choose the default option. Select **Perform default configuration** when asked to configure runtime environment and click the **Next** button.

Configure runtime environment

There are several additional options for configuring Red Hat JBoss Enterprise Application Platform now that the server has been installed. Each option can be individually chosen, and will be configured in the order displayed upon pressing next. What would you like to do now?

- Perform default configuration
 Perform advanced configuration

- 2.6. Some post-install tasks will now execute. When those are finished, review the information and click the **Next** button.

Processing finished

IzPack variable state written to /opt/jboss-eap-7.0/installation/InstallationLog.txt

- 2.7. On the final step of the wizard, click the button labeled **Generate installation script and properties file**. Name the file **myinstall.xml** and save it in your **/opt/jboss-eap-7.0** folder (which it defaults to).

Installation has completed successfully.

An uninstaller program has been created in:

/opt/jboss-eap-7.0/uninstaller

[Generate installation script and properties file.](#)

This **myinstall.xml** file can be used by an administrator to automatically perform an EAP installation using the selected options without running the installer again.

- 2.8. Click the **Done** button to close the installer.
- 2.9. Verify that you now have a folder named **jboss-eap-7.0** under your **/opt** folder. This folder will now be referred to as **JBOSS_HOME**.
- 2.10. You need to set an environment variable called **JBOSS_HOME** pointing to the EAP installation directory. It will be used by various scripts to locate where EAP is installed. Similarly, change the **PATH** variable to run EAP scripts in the **JBOSS_HOME/bin** folder from any directory. In order to set these variables, open **/home/student/.bashrc** with your preferred text editor and add the following lines at the end of the file:

```
JBOSS_HOME=/opt/jboss-eap-7.0  
PATH=$PATH:$JBOSS_HOME/bin  
export JBOSS_HOME PATH
```

Logout and login as student to make these changes visible for the student user.

- 2.11. You need to create a user called **jboss** to start and run EAP. The **-r** flag will be used to create a system user. Run the following command:

```
[student@workstation ~]$ sudo useradd -r jboss
```

To check if the user is a system account, run the following command:

```
[student@workstation installs]$ id jboss
```

The UID from the **jboss** user should be a number below 1000.

- 2.12. The **JBOSS_HOME** folder needs to be owned by this user.

```
[student@workstation installs]$ sudo chown -R jboss:jboss /opt/jboss-eap-7.0
```

- 2.13. Open the file **JBOSS_HOME/myinstall.xml** as root using **sudo**. This is the file created when you clicked the **Generate installation script and properties file** button at the last step of the Installer wizard. You can use this file to install another EAP instance with the same settings.

Open the **/opt/jboss-eap-7.0/myinstall.xml** file and check the value of the *installPath* attribute.

Observe the 2 lines:

```
<entry key="adminUser" value="jbossadm">
<entry autoPrompt="true" key="adminPassword">
```

Instead of hard coding the admin password in the file directly, it is set in the **myinstall.xml.variables** file.

- 2.14. Open the file **/opt/jboss-eap-7.0/myinstall.xml.variables** as root using **sudo** and add *JBoss@RedHat123* as the value for the *adminPassword* property.

- 2.15. You should see a folder in **JBOSS_HOME** named **uninstaller**, which contains a single executable JAR file named **uninstaller.jar**.

► 3. Start EAP standalone server:

- 3.1. On the workstation VM, enter the following command to start EAP standalone server as the **jboss** user.

On RHEL:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh
```

The following output is expected:

```
10:01:17,873 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3262ms - Started 261 of 509 services (332 services are lazy, passive or on-demand)
```

- 3.2. EAP 7 starts up very quickly. It should be up and running within 5-10 seconds!

► 4. Verify EAP is running:

- 4.1. Point your web browser to <http://localhost:8080>. You should see the following welcome page:

- 5. Before moving on to the next lab, shutdown the EAP 7 server by pressing **Ctrl+C** in the terminal window where you started EAP 7.

► 6. **(Optional)** Configure EAP to start as a service.

In a production environment, starting up EAP from the command line is not usual because it requires manual intervention. To avoid the need of running a command from a bash script,

EAP can be installed as a service to initialize EAP during the boot time from an OS. In the following steps, a set of commands will be executed to achieve this goal.

6.1. Inspect the script responsible for starting and stopping EAP.

The file is provided by EAP in **JBOSS_HOME/bin/init.d** directory and it will be responsible for checking if EAP is running, starting EAP, and loading a configuration file to customize the EAP runtime environment. Open the file **/opt/jboss-eap-7.0/bin/init.d/jboss-eap-rhel.sh** with your favorite text editor as the **jboss** user using **sudo -u jboss <textEditor>**.

The script will start EAP as a background process, creating a process ID (PID) file, starting it as a specific user, generating a log file to a defined place, and running EAP standalone server or in a managed domain. In order to allow some customization, a configuration file (line 21) with environment variables is loaded to identify the directories and configuration files used by an EAP instance.

6.2. Inspect the init script configuration file available at **/opt/jboss-eap-7.0/bin/init.d/jboss-eap.conf** with your favorite text editor as the **jboss** user using **sudo -u jboss <textEditor>**, to evaluate its contents.

There are environment variables such as **JBOSS_HOME** (pointing to the directory where EAP was installed), **JAVA_HOME** (the directory where the Java is installed), **JBOSS_USER** (the login responsible for running the EAP process), and **JBOSS_MODE** (the approach used to start EAP based on a standalone server or in a managed domain). Fortunately, the environment variables are clearly documented by comments in the file.

Please remove the # in front of each variable and update the following environment variable:

- **JAVA_HOME**: **/etc/alternatives/java_sdk** (The directory where Java is installed)
- **JBOSS_HOME**: **/opt/jboss-eap-7.0** (The directory where EAP is installed)
- **JBOSS_USER**: **jboss** (The owner of the EAP process)
- **JBOSS_MODE**:**standalone** (The mode to start EAP (standalone or domain))
- **JBOSS_CONFIG**:**standalone.xml** (The configuration file that should be used by the process)
- **JBOSS_CONSOLE_LOG**: **/var/log/jboss-eap/console.log** (The file where all the logs will be stored)

6.3. Copy the file **jboss-eap.conf** file to the **/etc/default** directory by running the following command:

```
[student@workstation ~]$ sudo cp \
/opt/jboss-eap-7.0/bin/init.d/jboss-eap.conf \
/etc/default/jboss-eap.conf
```

6.4. To make the init script visible for **systemctl**, the script must be stored at **/etc/init.d** with execution permission. Copy the file to the **/etc/init.d** directory and change its permission to be executable by running the following commands:

```
[student@workstation ~]$ sudo cp \
/opt/jboss-eap-7.0/bin/init.d/jboss-eap-rhel.sh \
/etc/init.d/jboss-eap
[student@workstation ~]$ sudo chmod 755 /etc/init.d/jboss-eap
```

- 6.5. To make the script on the host part of the current **systemctl** configuration, run the following command:

```
[student@workstation ~]$ sudo systemctl daemon-reload
```

- 6.6. To trigger the EAP startup during the boot process, run the following command:

```
[student@workstation ~]$ sudo systemctl enable jboss-eap
```

- 6.7. To verify if the setup was successful, run:

```
[student@workstation ~]$ sudo systemctl start jboss-eap
```

or reboot the system.

- 6.8. Access the EAP instance by opening a web browser and accessing `http://localhost:8080`.
- 6.9. To make sure this service does not conflict with the next laboratories, disable and stop the service by running:

```
[student@workstation ~]$ sudo systemctl disable jboss-eap
[student@workstation ~]$ sudo systemctl stop jboss-eap
```

This concludes this guided exercise.

Understanding Extensions, Subsystems, and Profiles

Objectives

After completing this section, you should be able to:

- Describe the architecture of extensions, subsystems, modules, and profiles.
- Describe the relationship between JBoss EAP 7 modules, extensions, and subsystems.
- Describe how JBoss Modules affect application classloading.

Modules in EAP

EAP 7 is based on a core infrastructure provided by the **WildFly Core** project that, among other tasks, manages the loading and activation of **modules**, following the architecture provided by the **JBoss Modules** project. A module basically provides code (Java Classes) to be used by EAP services and/or by applications.

The following figure shows the main components inside EAP 7. It shows, on the upper half, components that are part of the WildFly Core project (the EAP core), most of them modules themselves. The lower half shows modules provided by other JBoss community projects, that are integrated into the WildFly Full project and then productized and supported by Red Hat as JBoss EAP 7.

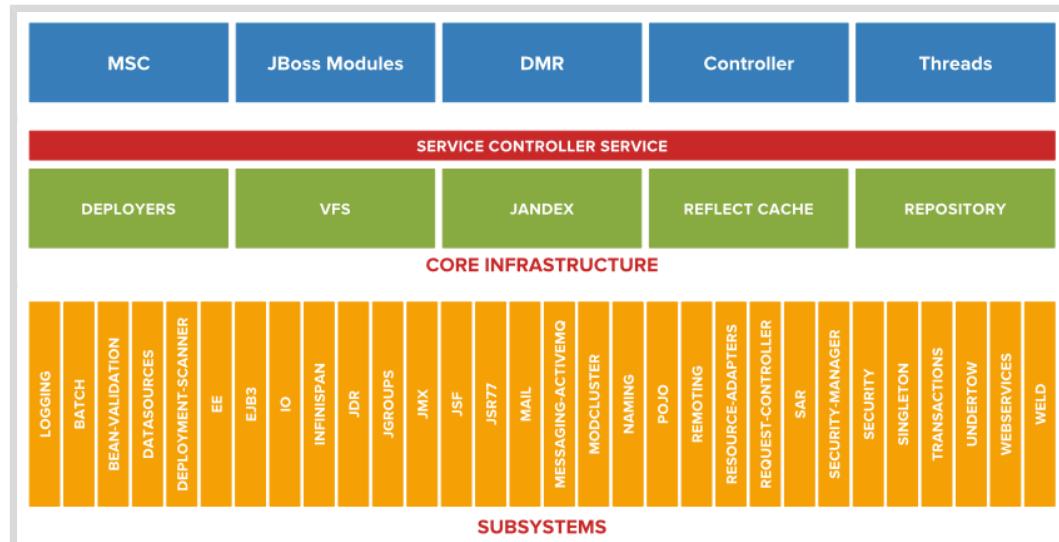


Figure 1.16: EAP 7 high-level architecture

Modules are loaded into an isolated **Classloader**, and can only see classes from other modules when explicitly requested. This means a module is free to be implemented without any concerns about potential conflicts with the implementation of other modules. All code running in EAP (including the code provided by the core) is run inside modules. Application code is no exception, so applications are thus isolated between each other and from EAP services as well.

The JBoss Modules architecture allow very fine-grained control of code visibility. An application can see a module that exposes a particular version of an API, while another application may be see a second module that exposes a different version of the same API.

An application developer may control this visibility manually and it can be very useful in a few scenarios. But for most common cases, EAP 7 automatically decides with modules to expose to an application, based on its use of JEE APIs.

All modules available in an EAP 7 installation are folders inside **JBOSS_HOME/modules**:

- The **JBOSS_HOME/modules/system/layers/base** folder contains the modules provided by the EAP 7 product.
- Third-party products that add modules to EAP 7 are expected to create their own folders inside **JBOSS_HOME/modules/system/layers**.
- The system administrator is expected to add local modules as folders, directly below **JBOSS_HOME/modules**.

Inside one of the module folders, a module name is used to create a folder tree much like compiled Java classes. For example, the extension named **org.wildfly.extension.undertow** is under **JBOSS_HOME/modules/system/layers** as **org/wildfly/extension/undertow**.



Note

Mode details about module file structure and how to add modules to an EAP 7 installation will be presented later in this course.

Modules and Extensions

A module that provides features and capabilities to the application server is called an **extension**. An extension does not simply provide code, but also provides a **management model**, consisting of one or mode **subsystems**, that allows the extension to be configured by the core.

The EAP core only loads and activates a module (and consequently an extension) when it is required, so features that are not in use do not waste memory or CPU. For example, a non-clustered EAP server instance does not have any clustering overhead.

Most of the features of EAP that relate to the deployment and configuration of servers and Java EE applications are implemented through extensions. A subset of all the extensions provided by EAP is sufficient to implement the JEE Web Profile. Add a few more extensions, and EAP becomes compliant to the JEE Full Profile. But if the JEE web profile is too much for an application, it can use a smaller subset of the available extensions.

EAP 7 even provides a few extensions that add capabilities beyond the JEE 7 standards. An example of this is clustered singletons.

Extensions are added to an EAP instance using the **<extension>** element, which appears at the beginning of the EAP main configuration file (**standalone.xml** or **domain.xml**). The following XML listing shows some of the extensions declared in the out-of-the-box **standalone.xml** configuration file:

```
<?xml version="1.0" ?>
<server xmlns="urn:jboss:domain:4.1">
    <extensions>
        <extension module="org.jboss.as.clustering.infinispan"/>
```

```
<extension module="org.jboss.as.clustering.jgroups"/>
<extension module="org.jboss.as.connector"/>
<extension module="org.jboss.as.ee"/>
<extension module="org.jboss.as.ejb3"/>
...
</server>
```

Adding an extension to the configuration files does not make that extension module fully loaded and active. It only makes the extension manageable, meaning that the extension management model is loaded into memory and can be changed using the EAP 7 management tools.

For example, clustering configurations can be done if the related extensions are available, but the related modules will only be loaded and activated when an application that requires clustering services is deployed. If no clustered application is deployed, none of the extensions related to clustering will be activated, even if the clustering extension management model was changed by the administrator.



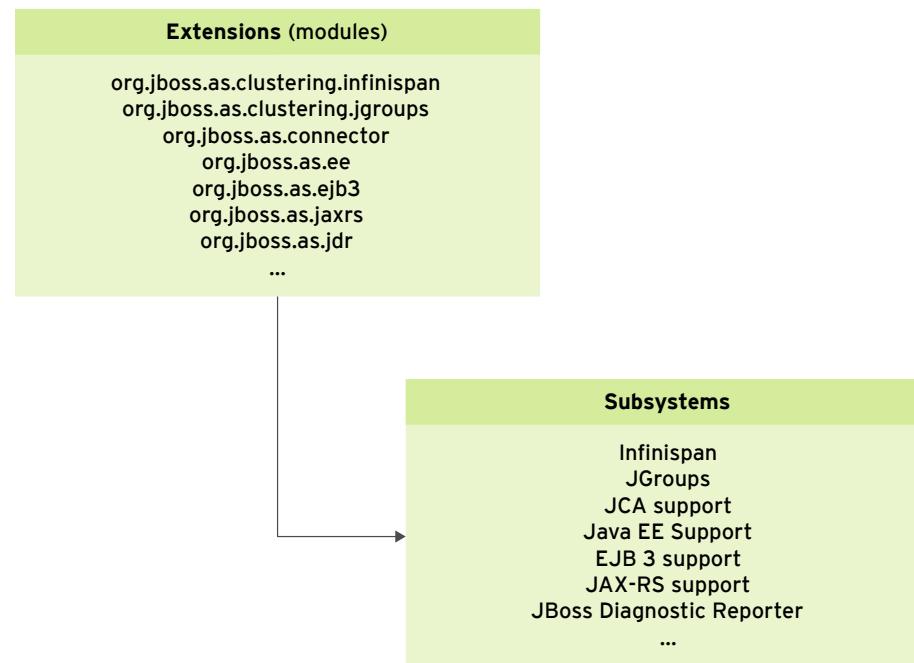
Note

If an organization applications do not need a particular extension, the system administrator can disable the extension so EAP will not load its management model anymore. Just remove the **<extension>** entry and its corresponding **<subsystem>** section (discussed next) from the configuration file. This can be accomplished either by editing the XML directly, or using the Management CLI.

Extensions and Subsystems

An extension management model provides one or more **subsystems**, which can be configured and managed using the Management API provided by the core, by using the Management Console and the Management CLI. When an extension is added to an EAP instance, the capabilities and attributes of the subsystems provided by the extension management model are configured within the **<subsystem>** element in the EAP configuration file.

Most of the time, an extension module provides a single subsystem, as illustrated by the following figure:

**Figure 1.17: EAP 7 Extensions and Subsystems**

For example, the **org.jboss.as.jpa** extension provides the **jpa** subsystem. This subsystem configuration, in the out-of-the-box **standalone.xml** configuration file, looks like:

```
<subsystem xmlns="urn:jboss:domain:jpa:1.1">
  <jpa default-datasource="" default-extended-persistence-inheritance="DEEP"/>
</subsystem>
```

The most commonly used subsystems can be configured using easy forms and assistants provided by the Management Console. All subsystems can be configured using the CLI, so usually there is no need to edit the XML directly to configure a subsystem.

The elements that appear within a subsystem are entirely unique to that particular subsystem. In fact, each subsystem has its own XML schema to define what is allowed within its **<subsystem>** element. All EAP 7 subsystem schema definitions can be found in the **JBoss_HOME/docs/schema** folder.



Note

If a subsystem does not require any specific settings, an empty **<subsystem>** entry is still required in the configuration file. For example, the **jaxrs** subsystem is configured out-of-the-box, without any specific settings:

```
<subsystem xmlns="urn:jboss:domain:jaxrs:1.0"/>
```

Subsystems and Profiles

A subsystem configuration does not stand by itself in the EAP 7 configuration files. A set of subsystem configurations is grouped inside a **<profile>** element.

The standalone server instance configuration file **standalone.xml** contains a single, anonymous, profile definition. The managed domain configuration file **domain.xml** contains out-of-the-box, four pre-defined profiles that should satisfy most use cases for applications deployed on EAP:

1. **default**: Is the most commonly used subsystems, including **logging**, **security**, **datasources**, **infinispan**, **weld**, **webservices**, and **ejb3**. The **default** implements not only the JEE Web Profile, but also most of the JEE Full Profile.
2. **ha**: Contains the exact same subsystems as the **default** profile, with the addition of clustering capabilities, provided mainly by the **jgroups** subsystem.
3. **full**: Is similar to the **default** profile, but notably adds the messaging (**messaging-activemq**) and a few other less used subsystems.
4. **full-ha**: Is same as the **full** profile, but with the addition of clustering capabilities.

The following figure illustrates the four out-of-the box profiles from the out-of-the box **domain.xml** configuration file:

default profile:	full profile:
Includes, among others, the following subsystems: logging batch-jberet datasources ejb3 infinispan jaxrs jca jpa jsf mail remoting webservices security transactions weld undertow	All subsystems from default profile plus: iiop-openjdk jsr77 messaging-activemq
ha profile:	full-ha profile:
All subsystems from default profile plus: jgroups modcluster singletont	All subsystems from full profile plus: jgroups modcluster singletont

Figure 1.18: EAP 7 profiles

The profiles provided out-of-the-box can be in use by different server instances in the same managed domain, but profiles are not associated directly to server instances. Server instances are grouped. The profile is associated to the group.

EAP profiles are supposed to be customized to a particular application and environment needs, so two real-world standalone server instances will probably have different profiles, containing different subsystem configurations.



Note

Look in the **JBOSS_HOME/standalone/configuration** folder. Notice there are four standalone configuration files:

- **standalone.xml**: Which compares to the **default** profile in **domain.xml**.
- **standalone-ha.xml**: Which compares to the **ha** profile in **domain.xml**.
- **standalone-full.xml**: Which compares to the **full** profile in **domain.xml**.
- **standalone-full-ha.xml**: Which compares to the **full-ha** profile in **domain.xml**.

They are provided so a standalone server instance can easily be started with more or less subsystems available.

In a managed domain, the administrator can create new profiles, either from scratch, or cloned from the ones provided out-of-the-box, and then customize the new profiles before associating them to their respective groups. A profile can also be a child of another profile, thus inheriting the subsystem configuration from its parent profile, so common configurations can be shared and maintained in a single place.

In a standalone server instance, the single anonymous profile has to be customized. But later, this course will show how to start a standalone server instance using a configuration file named other than **standalone.xml**, so the original file can be preserved as a reference.



Note

Be careful not to confuse an EAP profile with a Java EE profile. An EAP profile is a collection of subsystem configurations that are used to define the capabilities and services of an EAP server instance. A Java EE profile is a collection of Java EE standards, that is, JSRs.

► Quiz

Profile Selection

Using the information you just learned on the previous pages about profiles, choose the correct answer to the following questions:

- ▶ **1. Which profiles use the messaging-activemq subsystem? (Choose two.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha

- ▶ **2. Which profile(s) use the logging subsystem? (Choose only one.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. All of them.

- ▶ **3. Which profile would you use if you had a Java EE application that used MessageDriven EJBs, that is, the messaging-activemq subsystem, and you were NOT going to setup a cluster of server instances? (Choose only one.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.

- ▶ **4. Which profile would you use for a WAR file deployed onto a clustered group of four server instances? Assume this WAR contains a simple web application that requires only database access. (Choose only one.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.

- 5. Which profile would you use if you needed to deploy a clustered application that requires IIOP support for remote EJBs, that is, the `openjdk-iiop` subsystem? (Choose only one.)
- a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.
- 6. Which profile would you use for a WAR file deployed onto a single server? Assume this WAR contains a simple web application that requires only database access. (Choose only one.)
- a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.

► Solution

Profile Selection

Using the information you just learned on the previous pages about profiles, choose the correct answer to the following questions:

- ▶ **1. Which profiles use the messaging-activemq subsystem? (Choose two.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha

- ▶ **2. Which profile(s) use the logging subsystem? (Choose only one.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. All of them.

- ▶ **3. Which profile would you use if you had a Java EE application that used MessageDriven EJBs, that is, the messaging-activemq subsystem, and you were NOT going to setup a cluster of server instances? (Choose only one.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.

- ▶ **4. Which profile would you use for a WAR file deployed onto a clustered group of four server instances? Assume this WAR contains a simple web application that requires only database access. (Choose only one.)**
 - a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.

- 5. Which profile would you use if you needed to deploy a clustered application that requires IIOP support for remote EJBs, that is, the `openjdk-iiop` subsystem? (Choose only one.)
- a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.
- 6. Which profile would you use for a WAR file deployed onto a single server? Assume this WAR contains a simple web application that requires only database access. (Choose only one.)
- a. default
 - b. ha
 - c. full
 - d. full-ha
 - e. None of them. You would need to create a custom profile.

Managing JBoss EAP

Objectives

After completing this section, a System Administrator should be able to:

- Describe the management options available for JBoss EAP

EAP 7 management options

EAP 7 is designed in a modular fashion, with multiple subsystems that can be customized to support Java EE application requirements, such as database connectivity, batch processing, and integration tools, such as message queues. The subsystem configuration is maintained in the file **standalone.xml** (for the standalone mode) and **domain.xml** (for a managed domain).

These configuration files can be customized using three different approaches:

1. **Management Console:** Using a browser, this web-based application allows a System Administrator to manage most of the capabilities of your standalone or managed domain deployments.
2. **Management CLI (Command Line Interface):** Using a terminal window, the CLI offers a management model for viewing and modifying attributes and for performing operations, including batch operations, on a standalone server or a managed domain. The CLI includes user-friendly features like contextual auto-complete, built-in documentation of server configuration attributes, and command execution history.
3. **Edit XML Configuration files manually:** The settings of a standalone server or a managed domain are maintained in XML-based configuration files which can be modified directly.



Note

No matter which technique is used to modify a configuration setting, all changes are synchronized to the XML configuration files. For example, if a setting is modified using the management console, the underlying XML file is immediately changed, and the CLI will immediately become aware of the change. However, the management console may not get the latest updates made by the CLI because of the web browser caching. The management console displays data that was cached in the browser and it may display obsolete configuration data. A page refresh is needed to solve the problem.



Warning

Do not edit the XML configuration files manually while EAP is running. Any changes made to the XML file will most likely be lost. Either stop all EAP controllers and servers first, or use the management console or CLI.

To access the EAP 7 management console, launch the web browser and navigate to <http://localhost:9990>

Chapter 1 | Red Hat JBoss Enterprise Application Platform: Architecture and Features

By default, the EAP 7 management console is secured and you need to provide the username and password that you entered during the installation process (**jbossadm/JBoss@RedHat123**).

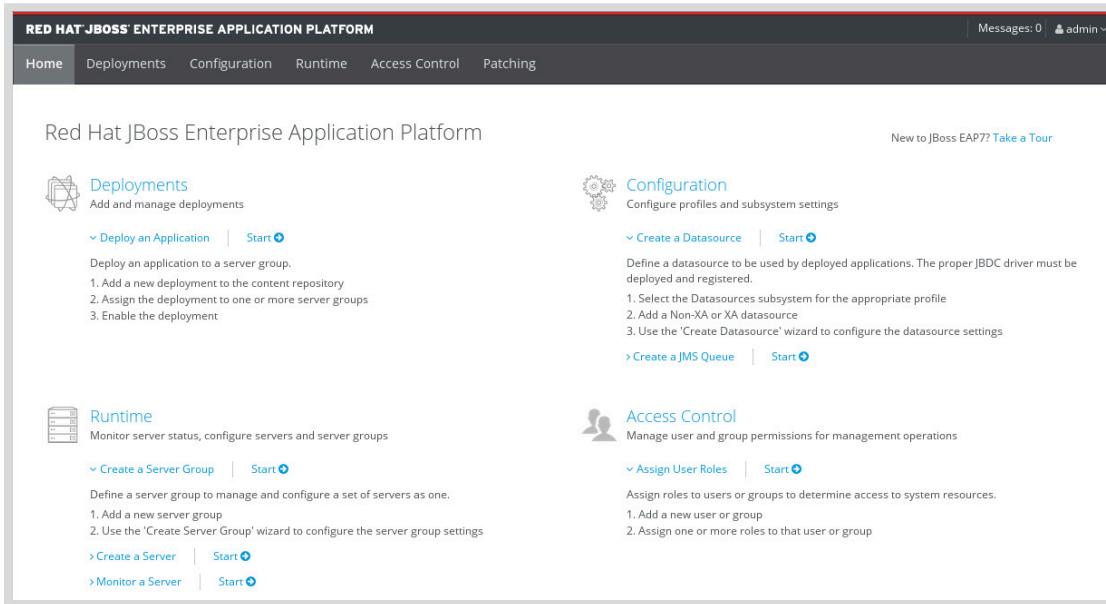


Figure 1.19: The EAP 7 Management Console

To access the JBoss EAP CLI, launch a terminal and navigate to the **JBoss_HOME/bin** folder and run the **jboss-cli.sh** script. For example:

```
$ ./jboss-cli.sh --connect
[standalone@localhost:9990 /] :product-info
{
    "outcome" => "success",
    "result" => [{"summary" => {
        "host-name" => "workstation.lab.example.com",
        "instance-identifier" => "6b407cda-46ea-44e9-b9d5-23236000f2a8",
        "product-name" => "JBoss EAP",
        "product-version" => "7.0.0.GA",
        "product-community-identifier" => "Product",
        "product-home" => "/opt/jboss-eap-7.0",
        "standalone-or-domain-identifier" => "STANDALONE_SERVER",
        "host-operating-system" => "Red Hat Enterprise Linux Server 7.2 (Maipo)",
        ...
    }}]
}
```

Enter '**quit**' or '**q**' to exit the CLI.

If you invoke the CLI without any arguments, it will launch in 'disconnected' mode. To connect to the server, use the **--connect** command option.

Another option to connect to a EAP 7 standalone server instance, or to a domain controller, is to use the CLI command **connect <IPAddress>**.

For an EAP instance running locally, no credentials will be requested, by default, by the JBoss CLI tool, but for EAP instances running on a different host, the admin user credentials will be

requested during login. Furthermore, it is possible to disable automatic authentication for local access for security reasons. This topic will be discussed later.

Ultimately, the XML configuration file can be edited using a text editor. For the standalone server, the file is available at **JBOSS_HOME/standalone/configuration/standalone.xml**.

```
<server xmlns="urn:jboss:domain:4.1">
    <extensions>
        <extension module="org.jboss.as.clustering.infinispan"/>
        ...
    </extensions>
    <management>
        ...
    </management>
    <profile>
        <subsystem xmlns="urn:jboss:domain:logging:3.0">
            ...
        </subsystem>
    </profile>
</server>
```

The file is composed of multiple extension tags that represents all the modules needed by EAP to start a subsystem. Additionally, multiple subsystem tags are declared, nested to the profile tag, which represents each customization needed by a subsystem.

Similarly, the managed domain also has the same set of configurations defined at **JBOSS_HOME/domain/configuration/domain.xml**. The structure and differences will be discussed later in this course.

Creating administrative users

The EAP installer creates an administrative user during the installation process.

If additional administrative users are required, or if the installation was NOT done using the EAP installer, new users can be created by using the **add-user.sh** script that is present in the **JBOSS_HOME/bin** folder.

This script can also be used to manage accounts for testing applications using Java EE security features (Java Authentication and Authorization System - JAAS), if the applications were not configured to use external identity stores such as an database or LDAP server.

There are two different types of users that can be added:

- The **Management** user type is responsible for accessing the administrative tools from EAP. The credentials will be stored at **JBOSS_HOME/standalone/configuration/mgmt-users.properties** file, with a encrypted password.
- The **Application** user type will be used by Java EE apps using JAAS and similarly to the management user option, the contents will be stored at **JBOSS_HOME/standalone/configuration/app-users.properties**

There is no need to restart an EAP server instance (or managed domain) when changing the user files. EAP will automatically detect changes to these files. They are plain text files, and the passwords stored in them are encrypted (technically the correct term is *hashed*). Lines beginning with a pound sign (#) are comments.

Later in this book it will also be shown that host controllers in a managed domain may need special users to connect to the domain controller. Those users can also be created using the **add-user.sh** script.



Note

By default, all management operations are available for all Management users, but EAP also supports the concept of Role Based Authentication Control (RBAC) that requires some customization in order to be activated. RBAC allows configuring users who are not given full administrative privileges, but only a subset of them, and only for a subset of the server instances in a managed domain.

► Guided Exercise

JBoss EAP Administration Console

In this lab, you will configure and manage the installed EAP 7 server.

Resources	
Files	NA
Application URL	http://localhost:9990

Outcome

You should be able to login as an admin user and explore the various features of the administration console.

Before You Begin

Run the following command to check if EAP was installed at **/opt/jboss-eap-7.0** and if it is not running:

```
[student@workstation ~]$ lab manage-eap setup
```

- 1. The EAP 7 server you installed in the previous lab must be started and running. If not, start the EAP 7 server as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh
```

- 2. Add a new admin user:

- 2.1. In a new terminal window, run the **add-user.sh** script as the **jboss** user.

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/add-user.sh
```

- 2.2. Add a new user:

- **User Type:** Management User (choose option **a**)
- **Username:** **admin**
- **Password:** **JBoss@RedHat123**
- **A List of groups that the user belongs to:** none (leave blank)
- **Is this new user going to be used for one AS process to connect to another AS process?:** no

**Note**

Right after the username prompt, the following output is expected.

```
User 'admin' already exists and is disabled, would you like to...
```

- a) Update the existing user password and roles
- b) Enable the existing user
- c) Type a new username

Select the option **a**.

- 2.3. Inspect the **mgmt-users.properties** file which is for defining users who need access to the EAP management console, either through the web interface, or using the CLI.

Using a text editor, open the file **JBOSS_HOME/standalone/configuration/mgmt-users.properties**.

- 2.4. Verify that you have a user named **admin** in this file, and the user has a hashed password.

- 2.5. The same file is also available on the **domain** folder of the EAP 7 installation. It will be used by the managed domain to secure the administrative console too.

Using a text editor, open the file **JBOSS_HOME/domain/configuration/mgmt-users.properties**.

Verify that the **admin** user also appears in this file. Notice that the same credentials were defined for both standalone and the managed domain.

- 2.6. Close the two **mgmt-users.properties** files.

**Note**

You should also see the **jbossadm** user in these files. This was added during the Install EAP guided exercise.

► 3. Login to the management console:

- 3.1. Point your web browser to **http://localhost:9990/**, which is the location of the management console. You should be prompted to login. Login as the **admin** user you created in the previous step, with password **JBoss@RedHat123**.

You should see the home page of the EAP 7 management console:

The screenshot shows the Red Hat JBoss Enterprise Application Platform management console. At the top, there is a navigation bar with tabs: Home, Deployments, Configuration, Runtime, Access Control, and Patching. The Home tab is selected. Below the navigation bar, the title "Red Hat Jboss Enterprise Application Platform" is displayed, along with a "New to EAP7? Take a Tour" link. The main content area is divided into two main sections: "Deployments" and "Configuration".

Deployments: This section includes a "Deploy an Application" button with a "Start" link, and a list of steps: 1. Use the 'Add Deployment' wizard to deploy the application and 2. Enable the deployment.

Configuration: This section includes a "Create a Datasource" button with a "Start" link, and a list of steps: 1. Select the Datasources subsystem, 2. Add a Non-XA or XA datasource, and 3. Use the 'Create Datasource' wizard to configure the datasource settings.

At the bottom of the screen, there is a footer bar with icons for "Tools" and "Settings".

- 4. The JVM (Java Virtual Machine) is responsible for managing the amount of memory used by the application server and deployed applications, as well as to manage classloading. In this step, you will read the metrics obtained from the JVM, using the EAP management console.

View the JVM Status page of the EAP management console. Navigate to (**Runtime** → **Standalone Server** → **JVM**) and click the **View** button to view the JVM details.

The screenshot shows the JVM Status page of the EAP management console. It displays various metrics and charts related to the Java Virtual Machine.

VIRTUAL MACHINE STATUS:

- OpenJDK 64-Bit Server VM
- Operating System: Linux 4.4.6-201.fc22.x86_64
- Processors: 4
- JVM Uptime: 18 min, 11 s

Heap Usage:

Max	1250
Used	218
Committed	1250

Used Heap in MB: 218 (Bar chart)

Committed Heap in MB: 1250 (Bar chart)

Thread Usage:

Live	52
Daemon	9

Daemon number of Threads: 9 (Bar chart)

► 5. View the EAP server logs.

The EAP server logs are stored in the local filesystem, but sometimes due to filesystem access restrictions, the administrator may need to access it via a web console.

Click the blue **<<Back** link on the top left corner of the JVM Status page to go back and navigate to (**Runtime** → **Standalone Server** → **Log Files**) and click the **View** button to view the log viewer.

Select the **server.log** entry in the table displayed and click the **View** button to view the server logs.

The screenshot shows the 'Monitor: Log Files' interface. At the top, there are tabs for 'LOG FILES' and 'SERVER.LOG'. Below the tabs, the title 'server.log' is displayed. A search bar with 'Find' and navigation buttons ('<', '>') is present. The main area contains log entries from file 90 to 97. The log entries are as follows:

```
90 user.timezone = America/Sao_Paulo
91 2016-04-15 13:30:52,369 DEBUG [org.jboss.as.config] (MSC service thread 1-5) VM Arguments: -D[Standalone] -verbose:gc -Xloggc:/hom
92 2016-04-15 13:30:53,585 INFO  [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using s
93 2016-04-15 13:30:53,607 INFO  [org.xnio] (MSC service thread 1-3) XNIO version 3.3.2.Final-redhat-1
94 2016-04-15 13:30:53,618 INFO  [org.xnio.nio] (MSC service thread 1-3) XNIO NIO Implementation Version 3.3.2.Final-redhat-1
95 2016-04-15 13:30:53,658 INFO  [org.wildfly.extension.io] (ServerService Thread Pool -- 37) WFLYIO0001: Worker 'default' has auto-cc
96 2016-04-15 13:30:53,666 INFO  [org.jboss.as.jsf] (ServerService Thread Pool -- 44) WFLYJSF0007: Activated the following JSF Implem
97 2016-04-15 13:30:53,667 INFO  [org.jboss.as.naming] (ServerService Thread Pool -- 46) WFLYNAM0001: Activating Naming Subsystem
```

At the bottom of the log viewer, it says '2.8.15.Final-redhat-1'. On the right side, there are links for 'Tools' and 'Settings'.

► 6. Change the deployment scanner interval.

The EAP deployment scanner is a subsystem that periodically scans and detects new application deployments (WAR, EAR, JAR files, etc.) to the application server. In this step, the time interval which EAP will scan for new files in the **deployments** directory will be updated.

- 6.1. Click the blue **<<Back** link on the top left corner of the **View Logs** page to go back and navigate to (**Configuration** → **Subsystems** → **Deployment Scanners**) and click the **View** button to view the **Deployment Scanners** configuration page.

Click the blue **Edit** link in the **Attributes** table and change the **Scan Interval** value from the default value of **5000** (5 seconds) to **8000** (8 seconds).

The screenshot shows the 'Attributes' configuration page for a deployment scanner subsystem. It includes fields for 'Auto deploy exploded*', 'Auto deploy xml*', 'Auto deploy zipped*', 'Deployment timeout' (set to 600), 'Path*' (set to 'deployments'), 'Relative to' (set to 'jboss.server.base.dir'), 'Runtime failure causes rollback*' (set to '\${jboss.deployment.scanner.rollback.on.failure:false}'), 'Scan enabled*' (checked), and 'Scan interval' (set to 8000). A 'Need Help?' link is in the top right corner.

- 6.2. Click the blue **Save** button when done.

► 7. Verify Configuration Changes:

- 7.1. Open the file **JBOSS_HOME/standalone/configuration/standalone.xml** in a text editor.
- 7.2. Verify that the **deployment-scanner** subsystem reflects the changes you made via the administration console.

```
....  
<subsystem xmlns="urn:jboss:domain:deployment-scanner:2.0">  
    <deployment-scanner path="deployments" relative-to="jboss.server.base.dir"  
        scan-interval="8000"  
        runtime-failure-causes-  
rollback="${jboss.deployment.scanner.rollback.on.failure:false}"/>  
</subsystem>  
....
```

- 7.3. Launch the JBoss EAP CLI tool and verify that the configuration changes are visible.

The CLI tool will be presented later, but it uses an approach similar to a bash script to customize EAP configuration files. For a while, to get information about the deployment scanner subsystem, open a terminal window, run the **jboss-cli.sh** script, and connect to the running instance of EAP using the following commands:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \  
--connect  
[standalone@localhost:9990 /] /subsystem=deployment-scanner/scanner=default:read-  
resource  
{  
    "outcome" => "success",  
    "result" => {
```

```
"auto-deploy-exploded" => false,  
"auto-deploy-xml" => true,  
"auto-deploy-zipped" => true,  
"deployment-timeout" => 600,  
"path" => "deployments",  
"relative-to" => "jboss.server.base.dir",  
"scan-enabled" => true,  
"scan-interval" => 8000  
}  
}
```

- 7.4. Exit the CLI by running the exit command:

```
[standalone@localhost:9990 /] exit
```

► **8.** Shut down EAP.

- 8.1. Before moving on to the next lab, if EAP is running, shutdown the EAP 7 server by pressing **Ctrl+C** in the terminal window where you started EAP 7.

This concludes the guided exercise.

► Lab

Red Hat JBoss Enterprise Application Platform: Architecture and Features

In this lab, you will uninstall the existing EAP instance, reinstall it using the automated installation method (`myinstall.xml` file), and configure the server.

Resources	
Files	<code>/home/student/JB248/install/jboss-eap-7.0.0-installer.jar</code> <code>/home/student/JB248/labs/features-eap/myinstall.xml</code> <code>/home/student/JB248/labs/features-eap/myinstall.xml.variables</code>
EAP 7 Administration Console URL	<code>http://localhost:9990</code>

Outcome

You should be able to install an instance of EAP 7.

Before You Begin

Use the following command to download the relevant lab files and ensure that EAP is not running:

```
[student@workstation ~]$ lab features-eap setup
```

1. In this step, we will uninstall EAP 7 that was installed from a previous lab using the uninstaller. Later, EAP 7 will be reinstalled using an automated approach, via an answer file.



Note

The grading script can be used as a guideline to test the progress right after running the step 1.1, due to the lab environment limitation.

- 1.1. Uninstall the existing EAP 7 instance in `/opt/jboss-eap-7.0` using the EAP 7 uninstaller as **root**. (Hint: **sudo**)
Mark the **Force the deletion of /opt/jboss-eap-7.0** checkbox and click the **Uninstall** button to remove the EAP installation.
- 1.2. Verify that the `/opt/jboss-eap-7.0` folder no longer exists.

2. An administrator installed EAP on an existing host and you would like to replicate the same installation process on the workstation. In this step, the answer file generated during the installation process will be provided to install EAP in a repeatable manner.

Install EAP 7 using the automated installer with the following characteristics:

- **Admin username:** `jbossadm`
- **Admin password:** `JBoss@RedHat123`
- **Install Path (JBoss_HOME):** `/opt/jboss-eap-7.0`

The variable file (`myinstall.xml.variables`) and the answer file (`myinstall.xml`) are available at `/home/student/JB248/labs/features-eap/`. The EAP 7 installer is located at `/home/student/JB248/install`s. Please check if the configuration from these files follows the characteristics expected:

3. Verify that the folder `/opt/jboss-eap-7.0` is created and check that the startup and other scripts (`standalone.sh`, `domain.sh`, `add-user.sh` etc...) are available in the folder `JBoss_HOME/bin`.
4. The recommended approach is to run EAP as a non-root user to avoid security breaches that allow malicious users to access the host with administrative permissions. A user called '`jboss`' has already been created for you.

Change ownership of the `/opt/jboss-eap-7.0` folder and files all the files within the folder to user '`jboss`' and group '`jboss`' via `chown -R` command.

Verify the ownership change by running the following command:

```
[student@workstation ~]$ ls -la /opt/jboss-eap-7.0
```

5. Start an EAP standalone server with the `jboss` user. In a new terminal window, login as the `jboss` user to start the EAP 7 instance. (Hint: Use `sudo -u jboss` command.)
6. In order to check if the admin credentials, used to install EAP, are working, login to the EAP 7 administration console using the credentials for the `jbossadm` user (password is `JBoss@RedHat123`).
7. Shutdown the EAP 7 server by pressing `Ctrl+C` in the terminal window where you started EAP 7.
8. Open a new terminal window to verify the completion of this lab by running the following command:

```
[student@workstation ~]$ lab features-eap grade
```

9. This concludes the lab.

► Solution

Red Hat JBoss Enterprise Application Platform: Architecture and Features

In this lab, you will uninstall the existing EAP instance, reinstall it using the automated installation method (`myinstall.xml` file), and configure the server.

Resources	
Files	<code>/home/student/JB248/installs/jboss-eap-7.0.0-installer.jar</code> <code>/home/student/JB248/labs/features-eap/myinstall.xml</code> <code>/home/student/JB248/labs/features-eap/myinstall.xml.variables</code>
EAP 7 Administration Console URL	<code>http://localhost:9990</code>

Outcome

You should be able to install an instance of EAP 7.

Before You Begin

Use the following command to download the relevant lab files and ensure that EAP is not running:

```
[student@workstation ~]$ lab features-eap setup
```

1. In this step, we will uninstall EAP 7 that was installed from a previous lab using the uninstaller. Later, EAP 7 will be reinstalled using an automated approach, via an answer file.



Note

The grading script can be used as a guideline to test the progress right after running the step 1.1, due to the lab environment limitation.

- 1.1. Uninstall the existing EAP 7 instance in `/opt/jboss-eap-7.0` using the EAP 7 uninstaller as **root**. (Hint: `sudo`)

Open a terminal window from the workstation VM (**Applications → Utilities → Terminal**) and run the following commands:

```
[student@workstation ~]$ sudo \
java -jar /opt/jboss-eap-7.0/uninstaller/uninstaller.jar
```

Mark the **Force the deletion of /opt/jboss-eap-7.0** checkbox and click the **Uninstall** button to remove the EAP installation.

- 1.2. Verify that the **/opt/jboss-eap-7.0** folder no longer exists.

```
[student@workstation ~]$ ls -la /opt
```

2. An administrator installed EAP on an existing host and you would like to replicate the same installation process on the workstation. In this step, the answer file generated during the installation process will be provided to install EAP in a repeatable manner.

Install EAP 7 using the automated installer with the following characteristics:

- **Admin username:** jbossadm
- **Admin password:** JBoss@RedHat123
- **Install Path (JBoss_HOME):** /opt/jboss-eap-7.0

The variable file (**myinstall.xml.variables**) and the answer file (**myinstall.xml**) are available at **/home/student/JB248/labs/features-eap/**. The EAP 7 installer is located at **/home/student/JB248/install**s. Please check if the configuration from these files follows the characteristics expected:

```
[student@workstation ~]$ cd /home/student/JB248/install
[student@workstation ~]$ sudo java -jar jboss-eap-7.0.0-installer.jar \
.../labs/features-eap/myinstall.xml
```

3. Verify that the folder **/opt/jboss-eap-7.0** is created and check that the startup and other scripts (**standalone.sh**, **domain.sh**, **add-user.sh** etc...) are available in the folder **JBoss_HOME/bin**.

```
[student@workstation ~]$ ls -la /opt/jboss-eap-7.0
[student@workstation ~]$ ls -la /opt/jboss-eap-7.0/bin
```

4. The recommended approach is to run EAP as a non-root user to avoid security breaches that allow malicious users to access the host with administrative permissions. A user called '**jboss**' has already been created for you.

Change ownership of the **/opt/jboss-eap-7.0** folder and files all the files within the folder to user '**jboss**' and group '**jboss**' via **chown -R** command.

```
[student@workstation ~]$ sudo chown -R jboss:jboss /opt/jboss-eap-7.0
```

Verify the ownership change by running the following command:

```
[student@workstation ~]$ ls -la /opt/jboss-eap-7.0
```

5. Start an EAP standalone server with the **jboss** user. In a new terminal window, login as the **jboss** user to start the EAP 7 instance. (Hint: Use **sudo -u jboss** command.)

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh
```

6. In order to check if the admin credentials, used to install EAP, are working, login to the EAP 7 administration console using the credentials for the **jbossadm** user (password is **JBoss@RedHat123**).
Open a browser and navigate to `http://localhost:9990`
7. Shutdown the EAP 7 server by pressing **Ctrl+C** in the terminal window where you started EAP 7.
8. Open a new terminal window to verify the completion of this lab by running the following command:

```
[student@workstation ~]$ lab features-eap grade
```

9. This concludes the lab.

Summary

In this chapter, you learned:

- EAP 7 is an application server, certified to the **Java EE 7** Web and Full Profiles defined by the JCP.
- EAP7 is based on the **Wildfly** open source projects, by the JBoss community, but productized and supported by Red Hat.
- Among EAP 7 features and benefits, it is fast, lightweight, flexible, modular, and cloud-ready.
- EAP 7 can be deployed into two distinct operation modes:
 - **Standalone server**, where a single server instance runs inside a single OS process and VM, and runs its own management.
 - **Managed domain**, where multiple server instances, in the same and/or different hosts, are managed by a dedicated server process called the domain controller.
- There are several different ways to install EAP 7 - zip file, GUI Installer, Automated install and RPM.
- Most EAP features are provided as extension modules, which are loaded and activated only when required by an application. The modules architecture makes EAP a fast, lightweight, and flexible application server.
- An extension provides one or more subsystem, and a profile contains configuration for different subsystems.
- EAP 7 provides, out-of-the-box, four different profiles, named: **default**, **ha**, **full** and **full-ha**.
 - Only the **ha** and **full-ha** provide clustering services.
- There are several different options to configure and manage EAP 7, manually by editing XML files, the management console, and the JBoss EAP CLI.
- Configuration changes performed via the management console or the JBoss EAP CLI are automatically synchronized to the XML files. The JBoss EAP CLI will immediately become aware of the changes.
- On RHEL 7, EAP 7 can be configured to start at boot and managed as an Operating System service via the **systemctl** facility of RHEL.

Chapter 2

Configuring JBoss EAP as a Standalone Server

Overview

Goal Configure a standalone server instance of JBoss EAP.

- Objectives**
- Configure an instance of JBoss EAP to run a standalone server.
 - Describe the configuration files for a standalone server and make configuration changes.
 - Configure JBoss EAP network interfaces and socket binding groups.

- Sections**
- Running JBoss EAP Standalone Server (and Guided Exercise)
 - Configuring JBoss EAP Standalone Server (and Quiz)
 - Configuring Interfaces and Socket Binding Groups (and Quiz)

- Lab**
- Configuring JBoss EAP as a Standalone Server

Running JBoss EAP Standalone Server

Objectives

After completing this section, students should be able to:

- Configure an instance of Enterprise Application Platform (EAP) to run a standalone server.
- List the characteristics of an EAP standalone server.

Overview of Standalone Server

Standalone servers are ideal for running a single instance of EAP as a single server. In a standalone server, all settings appear in a single configuration file: **JBoss_HOME/standalone/configuration/standalone.xml**. Within this file, users can configure different subsystems that comprise the features of this server, such as logging, messaging, and managing datasources.



Note

It is not generally recommended to directly edit this file. Instead use the EAP command line interface (CLI) or the Management Console which provides a safer outlet for making configurations. Changes made with the CLI or Management Console happen immediately whereas changes to the XML do not take place immediately and can be overwritten by someone making CLI/Management Console changes.

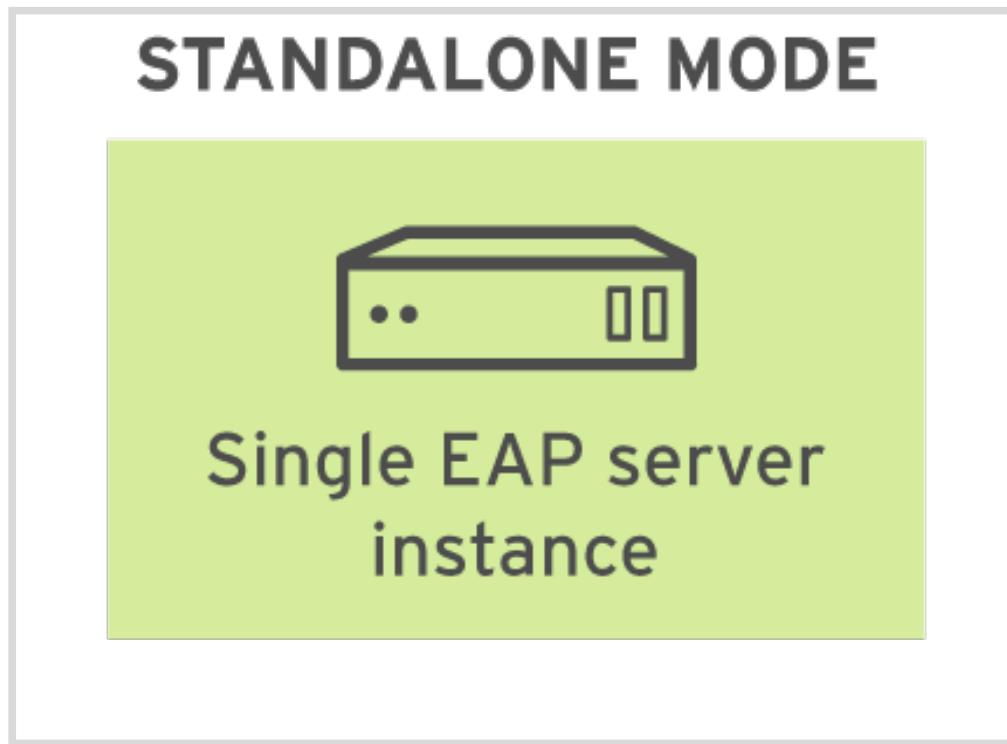


Figure 2.1: Standalone server installed on a single server

A standalone server only has one **profile** defined by default, whereas a managed domain has four defined by default. A profile refers to a set of subsystems and their configurations used by the server.

Running multiple instances of EAP standalone servers can create several complexities, such as requiring administrators to manage port conflicts, having multiple directories with repeated files, and keeping configuration in sync for each server. To resolve these issues, EAP can be run in a managed domain that will be discussed more in depth, later in this course.

EAP standalone server provides the capabilities for clustering and high availability. These features can be configured using several of the other configuration files provided by EAP, in addition to **standalone.xml**.

The Standalone Directory Structure

After installing an EAP distribution, the **standalone** folder initially contains the following subfolders:

```
configuration/  
deployments/  
lib/
```

Here is a description of each of these folders:

- **configuration:** contains the configuration files for a standalone server, along with a **standalone_xml_history** subfolder that maintains a version history of the configuration files. This folder also contains the properties file **logging.properties** for configuring loggers, a users file **mgmt-users.properties**, for defining login credentials for securing the management interfaces and a groups file **mgmt-groups.properties**, for defining login roles.
- **deployments:** contains application deployment files used by Java EE, such as EAR, WAR, and JAR files. Marker files (discussed later in this section) also appear in this folder.
- **lib:** is used for deploying common JAR files.



Note

it is a recommended practice to install Java libraries (JAR files) as modules, so this folder will often be empty.

These directories are essential to starting EAP. Without any of them, EAP will fail to start.

After running EAP the first time, the following subfolders of **standalone** are created:

```
data/  
log/  
tmp/
```

Here is a description of each of these folders:

- **data:** a location available to subsystems that store content in the file system, such as message queue or an in-memory database.
- **log:** the default location for the server log files, including **gc.log.0.current** and **server.log**, which contains startup logs from EAP.

- **tmp**: for temporary files, such as the shared-key mechanism used by the command line interface (CLI), to authenticate local users to the server.

The EAP Base Directory

The configuration and data folders of an EAP instance can be separated from the folder where EAP is installed. This allows users to maintain configurations separate from the installation, making it easier to upgrade to a newer version of EAP. It also allows users to install EAP once and run it multiple times on the same machine.

There are two command line properties that allow users to create this separation between the install folder and the folders used by the running instance:

1. **jboss.home.dir**: This property represents the root directory where EAP is installed. If this property is not defined, it defaults to **JBOSS_HOME**.
2. **jboss.server.base.dir**: This property represents the base directory for the server's configuration files. If this property is not defined, it defaults to the **JBOSS_HOME/standalone** folder.

For example, these properties can be set when running the **standalone.sh** script to start the EAP server:

```
$ ./standalone.sh -Djboss.server.base.dir=/path/to/base/directory \
-Djboss.home.dir=/path/to/home/directory
```



Note

The **port-offset** attribute can be changed without editing **standalone.xml** by using the **jboss.socket.binding.port-offset** property on the command line. For example:

```
$ ./standalone.sh -Djboss.socket.binding.port-offset=10000
```

Using a customized configuration file

By default, EAP will use the **BASE_DIR/configuration/standalone.xml** to start up a standalone server. However, an alternative configuration file may be used by passing the file name as a parameter of the **standalone.sh** script. This approach is useful to customize the configuration file without touching the default configuration files. To use a different configuration file, the **--server-config** parameter may be used as follows:

```
$ ./standalone.sh --server-config standalone-full.xml
```

An alternative parameter (**-c**) can be used as well:

```
$ ./standalone.sh -c standalone-full.xml
```

Demonstration: Running JBoss EAP from a Custom Location

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command to create the lab directory and verify that EAP is installed and not currently running:

```
[student@workstation ~]$ demo standalone-custom-location setup
```

2. Often users want to be able to run multiple standalone instances without needing to install EAP multiple times on the same host. This demonstration will create an alternative directory in order to customize the EAP instance and leave the original installation untouched.

Copy the following folders from **/opt/jboss-eap-7.0/standalone** into the new EAP location **/home/student/JB248/labs/custom-eap**:

- **configuration**
- **deployments**
- **lib**

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/standalone  
[student@workstation standalone]$ cp -r configuration deployments lib \  
~/JB248/labs/custom-eap
```



Note

Do not copy the **data**, **log**, and **tmp** folders. These get created automatically when EAP initially starts up.

3. Open the file **/home/student/JB248/labs/custom-eap/configuration/standalone.xml** with a text editor. The subsequent steps will customize the server's configuration by editing the **standalone.xml** file, to verify that EAP is using the configuration files in the new folder.
4. In the **standard-sockets** socket binding group, change the default value of the **port-offset** attribute from **0** to **10000**. This will add 10,000 to every port number used in the **standard-sockets** binding group.

```
...  
<socket-binding-group name="standard-sockets" default-interface="public" port-  
offset="${jboss.socket.binding.port-offset:10000}">  
...
```

5. Save your changes to **standalone.xml**.
6. The variable **jboss.server.base.dir** allows users to pass in a directory path to use it for an alternate base directory for the server content. Run the following command to start the EAP server using the **standalone.sh** script in the original EAP installation while using the new EAP configuration files:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/custom-eap/
```

The server should start up successfully with the following message:

```
17:03:30,497 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP  
7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3313ms -Started 261 of  
509 services (332 services are lazy, passive or on-demand)
```

7. On the workstation, point the browser to `http://localhost:18080` to see the EAP welcome page with the new port offset.
8. On the workstation, point the browser to `http://localhost:19990` to see the EAP management console that is running with the same port offset.



Note

The admin username is **jbossadm** and the password is **JBoss@RedHat123**.

9. Explore the contents of the folder `/home/student/JB248/labs/custom-eap` in a new terminal window.

```
[student@workstation ~]$ ls /home/student/JB248/labs/custom-eap/  
configuration data deployments lib log tmp
```

Notice the three new folders: **data**, **log**, **tmp**. The folders automatically get created when the EAP server starts.

10. In the new terminal window, start a second EAP server without specifying a `jboss.server.base.dir` with the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh
```



Note

The `/opt/jboss-eap-7.0` directory is owned by the **jboss** user. Other users only have read and execution privileges. Because starting the EAP instance within that directory creates folders and writes to files, you must be the **jboss** user to start the server with that base directory.

11. Notice that the server starts up without any port conflicts by verifying that there are no **ERROR** messages in the server log. For this instance of EAP, the original configuration files still have a port-offset of **0**, which means that this instance will not conflict with the other instance that has a port-offset of **10000**.
12. Open a browser on the workstation and go to `http://localhost:8080`. You should see the welcome page on both ports **8080** and **18080**. EAP is now running twice on the same machine, using the same installation folders, but each server has its own configuration and deployments.

13. Using a browser on the workstation, verify the management console on port **9990** is accessible by visiting <http://localhost:9990>. Use the admin credentials to log in. Click on **Runtime**, then **Standalone Server**, then **Environment** and click **View**.

Filter the environment variables by searching for **jboss.server.base.dir** to see the base directory for this instance. Repeat these steps in the other instance of EAP running by visiting the management console at <http://localhost:19990>.

Each server is managed separately in Standalone mode -- they do not share configurations. If you want multiple servers to be configured identically, then you need to run EAP in a managed domain.

14. Stop both instances of EAP by pressing **Ctrl+C** on each terminal window that is running EAP.

This concludes the demonstration.



Note

Visit the official documentation for a full list of command line parameters, when starting EAP, at the following link: https://docs.jboss.org/author/display/WFLY10/Command+line+parameters?_sscc=t

► Guided Exercise

Creating a Standalone Server

In this lab, you will create a standalone EAP server.

Resources	
Files:	/home/student/JB248/labs/standalone-instance
Application URL:	http://localhost:18080 http://localhost:19990

Outcome(s)

You should be able to run a standalone EAP server using a custom location for the server base directory.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to download the files for the exercise:

```
[student@workstation ~]$ lab standalone-instance setup
```

► 1. Copy the Standalone Directories to a New Location

In this guided exercise, you will create an alternative directory in order to customize the EAP server and leave the original installation untouched. Copy the **configuration**, **deployments**, and **lib** directories to the new location.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/standalone
[student@workstation standalone]$ cp -r configuration deployments lib \
~/JB248/labs/standalone-instance
```

► 2. Modify the Port Offset for the Standalone Server

To verify that the EAP server is utilizing the new, copied configuration files, you will update the port offset to adjust the ports this instance of EAP uses, allowing for multiple instances of EAP to run at the same time, without port conflicts.

- 2.1. Update the new configuration file by editing the **standalone.xml** file in order to verify that EAP is using the configuration files in the new folder. Open the file **/home/student/JB248/labs/standalone-instance/configuration/standalone.xml** with a text editor.
- 2.2. In the **standard-sockets** socket binding group, change the default value of the **port-offset** attribute from **0** to **10000**. This will add 10,000 to every port number used in the **standard-sockets** binding group.

```
...  
<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:10000}">  
...
```

- 2.3. Save your changes to the **standalone.xml** and close the text editor.

► 3. Run and Test the EAP Server

- 3.1. Run the following command to start the EAP server using the **standalone.sh** script in the original EAP installation while using the new EAP configuration files:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone-instance/
```

The server should start up successfully with an output similar to the following:

```
17:03:30,497 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP  
7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3313ms -Started 261 of  
509 services (332 services are lazy, passive or on-demand)
```



Note

Students may receive an error when starting the server for the first time similar to the following:

```
java.lang.IllegalArgumentException: Failed to instantiate class  
"org.jboss.logmanager.handlers.PeriodicRotatingFileHandler" for handler  
"FILE"  
...  
Caused by: java.io.FileNotFoundException: /opt/jboss-eap-7.0/standalone/log/  
server.log (Permission Denied)
```

This error can be safely ignored as it is a bug in EAP. Subsequent start ups of the server should not render the same error message.

- 3.2. On the workstation, point the browser to `http://localhost:18080` to see the EAP welcome page with the new port offset.
- 3.3. On the workstation, point the browser to `http://localhost:19990` to see the EAP Management Console that is running with the same port offset.



Note

The admin username is **jbossadm** and the password is **JBoss@RedHat123**.

- 3.4. Explore the contents of the folder `/home/student/JB248/labs/standalone-instance` in a new terminal window:

```
[student@workstation ~]$ ls /home/student/JB248/labs/standalone-instance/
configuration  data  deployments  lib  log  tmp
```

Notice the three new folders: **data**, **log**, **tmp**. The folders automatically get created when the EAP server starts.

► 4. Clean Up

- 4.1. Stop the running instance of EAP that was started in the previous step. Press **Ctrl+C** in the terminal window in which the server is running.

This concludes the guided exercise.

Configuring JBoss EAP as a Standalone Server

Objectives

After completing this section, students should be able to:

- Describe the `standalone.xml` configuration file.
- Make configuration changes and updates to EAP as a standalone server.

The `standalone.xml` Configuration File

Similar to EAP 6, configuring EAP 7 occurs in a single XML file. For a standalone server, that file is `JBoss_HOME/standalone/configuration/standalone.xml`. The general structure of `standalone.xml` looks like:

```
<server xmlns="urn:jboss:domain:4.1">
  <extensions>
    ...list of extensions here
  </extensions>

  <system-properties>
    ...system properties defined here
  </system-properties>

  <management>
    ...management interfaces defined here
  </management>

  <profile>
    ...list of subsystems and their configurations
  </profile>

  <interfaces>
    ...interface definitions
  </interfaces>

  <socket-binding-group>
    ...socket binding definitions
  </socket-binding-group>

  <deployments>
    ...deployed applications go here
  </deployments>
</server>
```

Extensions

Extensions are modules that extend the core capabilities of the server. A **module** is a bundle composed of a library developed in Java and an XML configuration file that provides Java EE

compliant functionalities. An extension defines one or more subsystems based on a module. Within the `<extensions>` element is a list of `<extension>` elements that make a subsystem available to that server. For example, the following excerpt describes a list of extensions that are made available to a server, including `ejb3`, which is responsible for providing transactional features as required by JEE 7 specs:

```
<extensions>
    <!-- list all extensions that you want made available to this server -->
    <extension module="org.jboss.as.clustering.infinispan"/>
    <extension module="org.jboss.as.deployment-scanner"/>
    <extension module="org.jboss.as.ejb3"/>
    <extension module="org.jboss.as.jpa"/>
</extensions>
```

Each extension must be declared in the `standalone.xml` file and each module should be stored at `JBOSS_HOME/modules/system/layers/base/`. In order to manage the extension, EAP requires a `<subsystem>` tag where all customization needed for that subsystem will be declared. This will be discussed later during the course.



Note

To find the module, EAP uses module name, such as `org.jboss.as.jpa`, and finds the folder in the `modules` directory `org/jboss/as/jpa/` to locate the module.

To demonstrate the `<extensions>` section of a configuration file, suppose an EAP instance that includes all extensions for Java EE 7 Full Profile, as well as clustering and high availability, is needed. The `<extensions>` element in this case should contain the following:

```
<extensions>
    <extension module="org.jboss.as.clustering.infinispan"/>
    <extension module="org.jboss.as.connector"/>
    <extension module="org.jboss.as.deployment-scanner"/>
    <extension module="org.jboss.as.ee"/>
    <extension module="org.jboss.as.ejb3"/>
    <extension module="org.jboss.as.jaxrs"/>
    <extension module="org.jboss.as.jdr"/>
    <extension module="org.jboss.as.jmx"/>
    <extension module="org.jboss.as.jpa"/>
    <extension module="org.jboss.as.jsf"/>
    <extension module="org.jboss.as.logging"/>
    <extension module="org.jboss.as.mail"/>
    <extension module="org.jboss.as.naming"/>
    <extension module="org.jboss.as.pojo"/>
    <extension module="org.jboss.as.remoting"/>
    <extension module="org.jboss.as.sar"/>
    <extension module="org.jboss.as.security"/>
    <extension module="org.jboss.as.transactions"/>
    <extension module="org.jboss.as.webservices"/>
    <extension module="org.jboss.as.weld"/>
    <extension module="org.wildfly.extension.batch.jberet"/>
    <extension module="org.wildfly.extension.bean-validation"/>
    <extension module="org.wildfly.extension.io"/>
```

```
<extension module="org.wildfly.extension.request-controller"/>
<extension module="org.wildfly.extension.security.manager"/>
<extension module="org.wildfly.extension.undertow"/>
</extensions>
```

Management Interfaces

After the `<extensions>` section in `standalone.xml` is the `<management>` section, which is used for defining the *management interfaces*. The management interfaces allow remote clients to connect to the EAP instance for the purpose of managing the instance. EAP exposes two management interfaces:

- **HTTP Interface**: provides access to the management console.
- **Native Interface**: the native interface allows for management operations to be executed over a proprietary binary protocol, which is what the Command Line Interface (CLI) tool uses.

The `<management>` element contains the settings for making these interfaces available. The configuration looks like:

```
<management>
  ...
  <management-interfaces>
    <native-interface security-realm="ManagementRealm">
      <socket-binding native="management-native"/>
    </native-interface>
    <http-interface security-realm="ManagementRealm">
      <socket-binding http="management-http"/>
    </http-interface>
  </management-interfaces>
  ...
</management>
```

The `management-native` and `management-http` socket bindings are defined later in the configuration file in the `<socket-binding-group>` section. This is where the actual host and port are defined that these interfaces listen on.

**Note**

A user can restrict access through the HTTP interface, simply by removing it from the list of **management-interfaces**:

```
<management>
  ...
  <management-interfaces>
    <native-interface security-realm="ManagementRealm">
      <socket-binding native="management-native"/>
    </native-interface>
  </management-interfaces>
  ...
</management>
```

This is the easiest way to disable the management console, because without an interface there will not be any way to access it.

In the following XML code, there is a **<security-realm>** tag named **ManagementRealm** and it is used by both management interfaces from the previous XML code. It refers to a text file named **mgmt-users.properties** where user credentials are stored.

**Note**

In a production environment, this realm will be replaced with a database or LDAP realm. Security realms are discussed in detail later on in this course.

The **security-realm** tag named **ApplicationRealm** is used by client applications that need to access EAP remotely.

```
<management>
  ...
  <security-realms>
    <security-realm name="ManagementRealm">
      <authentication>
        <local default-user="$local" skip-group-loading="true"/>
        <properties path="mgmt-users.properties" relative-
to="jboss.server.config.dir"/>
      </authentication>
      <authorization map-groups-to-roles="false">
        <properties path="mgmt-groups.properties" relative-
to="jboss.server.config.dir"/>
      </authorization>
    </security-realm>
    <security-realm name="ApplicationRealm">
      <authentication>
        <local default-user="$local" allowed-users="*" skip-group-
loading="true"/>
        <properties path="application-users.properties" relative-
to="jboss.server.config.dir"/>
      </authentication>
      <authorization>
```

```
<properties path="application-roles.properties" relative-
to="jboss.server.config.dir"/>
</authorization>
</security-realm>
</security-realms>
...
</management>
```

Profiles and Subsystems

A **profile** is a collection of subsystems. A **subsystem** is where the extensions of the EAP standalone instance can be configured. Adding a subsystem to a profile has two purposes:

- If a subsystem appears within a profile, then that subsystem will be made available to the server using that profile.
- The subsystem allows for configuring the extension to suit the user's specific needs.

When starting an EAP standalone server, there is only one profile available.



Note

There is a direct relationship between a subsystem and its extension. Only a subsystem can be added to a profile if the corresponding **<extension>** element appears in the **<extensions>** section.

The **<profile>** element has a collection of **<subsystem>** child elements, and each **<subsystem>** entry consists of the unique configuration settings of that particular extension. Some subsystem definitions do not require any settings, like the **jsf** and **jaxrs** subsystems:

```
<profile>
    <subsystem xmlns="urn:jboss:domain:jsf:1.0"/>
    <subsystem xmlns="urn:jboss:domain:jaxrs:1.0"/>

    ...other subsystem definitions
</profile>
```



Note

jaxrs and **jsf** are both configured in locations outside of the subsystems. For example, **jaxrs** uses annotations and the configurations are made within the source code. **jsf** uses its own configuration file, **faces-config.xml**.

Some subsystems have lots of configuration settings, like the **datasources** subsystem:

```
<profile>
    <subsystem xmlns="urn:jboss:domain:datasources:4.0">
        <datasources>
            <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-
name="ExampleDS" enabled="true" use-java-context="true">
                <connection-
url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
```

```
<driver>h2</driver>
<security>
    <user-name>sa</user-name>
    <password>sa</password>
</security>
</datasource>
<drivers>
    <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
    </driver>
</drivers>
</datasources>
</subsystem>
</profile>
```

**Note**

The child elements of **<subsystem>** are unique for each individual subsystem. View the corresponding schema to learn about and determine how to configure a particular subsystem. The schema documents are found in the **JBoss_HOME/docs/schema** folder.

For example, to view the settings for configuring the **datasources** subsystem, view the contents of its schema at **JBoss_HOME/docs/schema/wildfly-datasources_4_0.xsd**.

Demonstration: Configuring Profiles

We have seen the XML that represents the profiles and subsystems of a standalone server. It can be edited directly in **standalone.xml**. However, most of the common administrative tasks can and should be accomplished using the management console. For example, the following screenshot shows how to edit log handlers in the management console:

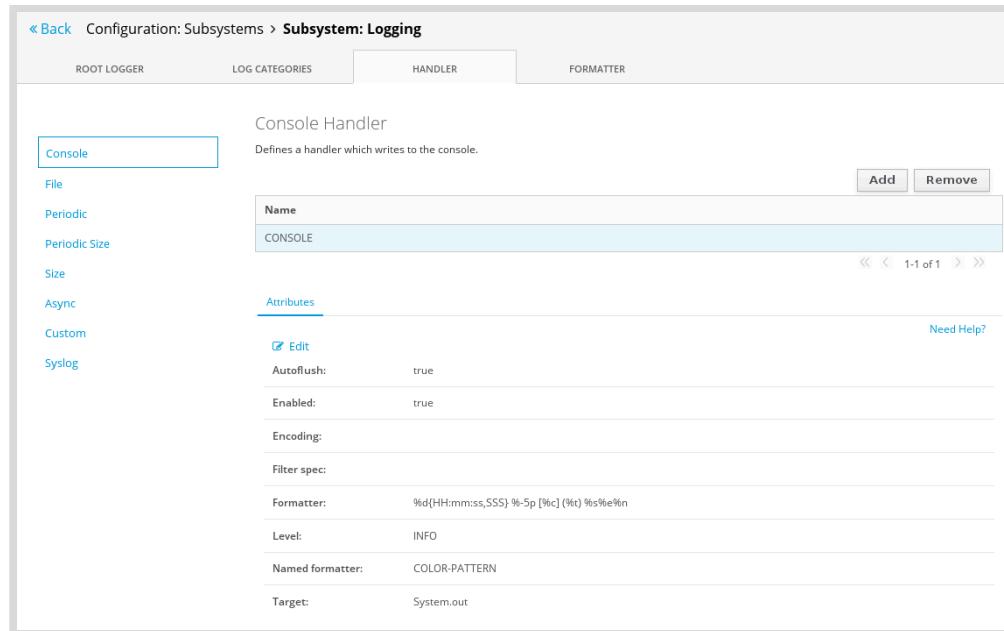


Figure 2.2: Log handler editing in the Management Console

Please review the video to follow along with the steps. Replay it as often as needed.

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command to verify that EAP is installed and not currently running and to create a new base directory for EAP:

```
[student@workstation ~]$ demo configuring-profiles setup
```

2. This demonstration will run an EAP standalone instance using a custom server folder located at **/home/student/JB248/labs/configuring-profiles**. Open the **standalone.xml** file and view the three **<logger>** entries in the **logging** subsystem. The file is located at **/home/student/JB248/labs/configuring-profiles/configuration/standalone.xml**.

```
...
<profile>
    <subsystem xmlns="urn:jboss:domain:logging:3.0">
        ...
        <logger category="com.arjuna">
            <level name="WARN"/>
        </logger>
        <logger category="org.jboss.as.config">
            <level name="DEBUG"/>
        </logger>
        <logger category="sun.rmi">
            <level name="WARN"/>
        </logger>
        ...
    </subsystem>
...
<profile>
...
```

**Note**

The Logging Subsystem will be covered in much more detail later on in this course. For now, this demonstration will illustrate how the management console updates the **standalone.xml** file.

Ignore the following logger that appears outside of the profile as it is outside of the scope of this demonstration:

```
<logger log-boot="true" log-read-only="false" enabled="false">
    <handlers>
        <handler name="file"/>
    </handlers>
</logger>
```

3. Start the EAP server using the **standalone.sh** script at **/opt/jboss-eap-7.0/bin** and point to the lab directory at **/home/student/JB248/labs/configuring-profiles/configuration/standalone.xml** as the server base directory. In order to access the EAP management console to create a new logger, the standalone server must be running:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/configuring-profiles/
```

Wait for the server to fully start up.

4. Access the management console by visiting <http://localhost:9990>

**Note**

The admin username is **jbossadm** and the password is **JBoss@RedHat123**

5. Click on **Configuration** at the top of the management console. Click on **Subsystems** in the first column and then click on **Logging** in the second column. Then click on **View** to access the configuration page for the Logging subsystem.
6. Using the management console, add a new log category to the configuration. Select **LOG CATEGORIES** at the top of the page and then click **Add** above the list of categories.
7. Take note that the categories currently populated are the same as the loggers seen in the configuration file previously. Set the **Name** of the category as **org.jboss.as.clustering** and set the level to **DEBUG**. Select **Save**. You should see a notification indicating that the category was successfully saved.
8. Open the **/home/student/JB248/labs/configuring-profiles/configuration/standalone.xml** file again and view the **logging** subsystem to see the changes reflected in the configuration file. You should see a fourth entry that looks like the following:

```
...
<logger category="org.jboss.as.clustering" use-parent-handlers="false">
    <level name="DEBUG"/>
</logger>
...
```

9. Return to the management console and delete the **org.jboss.as.clustering** category, by clicking on **org.jboss.as.clustering** and then clicking **Remove**. Confirm the removal in the pop up to return the server to its original state.
10. Reopen the **/home/student/JB248/labs/configuring-profiles/configuration/standalone.xml** file and observe that the number of loggers has returned to three in the **logging** subsystem.
11. Stop the instance of EAP by pressing **Ctrl+C** on the terminal window that is running EAP.

This concludes the demonstration.

► Quiz

Configuring Profiles

Choose the correct answer to the following questions:

- ▶ 1. Which sections in the `standalone.xml` are required to add and configure a module in the EAP server? (Select two.)
 - a. Profile
 - b. Extensions
 - c. Management Interface
 - d. Socket Binding Group

- ▶ 2. What section in `standalone.xml` is responsible for describing which users can access the Management Console? (Select one.)
 - a. Profile
 - b. Extensions
 - c. Management Interface
 - d. System Properties

- ▶ 3. Which management interface is responsible for providing access to the Management Console? (Select one.)
 - a. Native Interface
 - b. HTTP Interface
 - c. HTTPS Interface
 - d. Insecure Interface

- ▶ 4. How many profiles can be defined in `standalone.xml`? (Select one.)
 - a. One
 - b. Four
 - c. As many as required.
 - d. Zero

- ▶ 5. Which methods of configuring EAP are recommended? (Select two.)
 - a. Directly editing `standalone.xml` with a text editor.
 - b. The Management Console
 - c. The EAP Command Line Interface
 - d. Making minor changes to the `standalone.xml` and major changes with the Management Console.

► Solution

Configuring Profiles

Choose the correct answer to the following questions:

- ▶ 1. Which sections in the `standalone.xml` are required to add and configure a module in the EAP server? (Select two.)
 - a. Profile
 - b. Extensions
 - c. Management Interface
 - d. Socket Binding Group

- ▶ 2. What section in `standalone.xml` is responsible for describing which users can access the Management Console? (Select one.)
 - a. Profile
 - b. Extensions
 - c. Management Interface
 - d. System Properties

- ▶ 3. Which management interface is responsible for providing access to the Management Console? (Select one.)
 - a. Native Interface
 - b. HTTP Interface
 - c. HTTPS Interface
 - d. Insecure Interface

- ▶ 4. How many profiles can be defined in `standalone.xml`? (Select one.)
 - a. One
 - b. Four
 - c. As many as required.
 - d. Zero

- ▶ 5. Which methods of configuring EAP are recommended? (Select two.)
 - a. Directly editing `standalone.xml` with a text editor.
 - b. The Management Console
 - c. The EAP Command Line Interface
 - d. Making minor changes to the `standalone.xml` and major changes with the Management Console.

Configuring Interfaces and Socket Binding Groups

Objectives

After completing this section, students should be able to:

- Configure JBoss EAP network interfaces and socket binding groups

Interfaces

An *interface* is a logical name for a network interface, IP address, or host name to which a socket can be bound. The `<interfaces>` element is used to define an interface using the `<interface>` child element. For example, the `<interfaces>` section defined in the default `standalone.xml` configuration file looks like the following:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}"/>
    </interface>
</interfaces>
```



Note

The properties in the \$ and curly braces are system runtime properties. The value after the colon is the default value if the runtime property is not defined.

There are three pre-defined interfaces, but users can add as many interfaces as necessary based on the user's particular network and hardware. The default value of each interface is 127.0.0.1, which represents the loopback network interface (`localhost`). By binding to `localhost`, an out-of-the-box installation of EAP will not be exposed to the outside world as a security feature. The interface named "management" is kept separate from the interface named "public" to make it easier to secure access to the management features of EAP.

Users can configure the IP address for accessing the Management Console, when starting EAP, using the command line variable `-bmanagement`. For example:

```
$ ./standalone.sh -bmanagement 127.0.0.1
```

The purpose of the `<interfaces>` element is to provide a single location to define physical addresses used in an environment, then refer to those physical addresses as many times as needed throughout the configuration file. Do not specify ports or protocols here. Those details get defined later in the socket binding groups.

The name of the interface can be any value and there are various options for defining interfaces. If a host has multiple Network Interface Cards (NICs) with multiple IP addresses and only a certain

NIC should be exposed, then use the first of the following interface configurations. If all the network interfaces need to be exposed, use the second interface configuration.

```
<interfaces>
  <interface name="my_ip_address">
    <inet-address value="128.164.0.15"/>
  </interface>
</interfaces>
```

The following interface uses the wildcard **any-address** to bind to any and all IP addresses available on the server:

```
<interfaces>
  <interface name="global">
    <any-address/>
  </interface>
</interfaces>
```

The interface above demonstrates the use of an *interface criteria*, which is used at runtime to determine what IP address to use for an interface. The following interface also uses a criteria that will only bind to an IPv4 address:

```
<interfaces>
  <interface name="ipv4-global">
    <any-ipv4-address/>
  </interface>
</interfaces>
```

Criteria can be combined to be very specific about an interface definition. For example, the following interface defines a specific subnet that supports multicast. The address must be up, and cannot be point-to-point:

```
<interfaces>
  <interface name="my_specific_interface">
    <subnet-match value="192.168.0.0/16"/>
    <up/>
    <multicast/>
    <not>
      <point-to-point/>
    </not>
  </interface>
</interfaces>
```

On a Linux™ environment, users can define an interface for a specific network card:

```
<interface name="internal">
  <nic name="eth1"/>
</interface>
```

Socket Binding Groups

A *socket binding group* is a named collection of socket bindings, which allows users to define all of the ports needed for an EAP instance. A socket binding group is defined using the **<socket-binding-group>** element, which consists of a collection of **<socket-binding>** child elements. For example, here is the default socket binding group defined in **standalone.xml**:

```
<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:0}">
    <socket-binding name="management-http" interface="management" port="${jboss.management.http.port:9990}"/>
    <socket-binding name="management-https" interface="management" port="${jboss.management.https.port:9993}"/>
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
</socket-binding-group>
```

The **name** of this socket binding group is **standard-sockets**, but any name can be used when defining a socket binding group. The **default-interface** attribute must be set to a named interface defined in the **<interfaces>** section. Each **<socket-binding>** entry has a protocol name and a port number. Notice the **<socket-binding>** can also define an interface attribute that points to an **<interface>** definition, which overrides the **default-interface** value.

The **port-offset** allows users to easily modify all the port numbers by specifying a positive offset value. This is especially convenient in situations where multiple instances of EAP are running on the same host and an offset can be used to avoid port conflicts that prevent an instance from starting up. Notice the **port-offset** can be defined using the **jboss.socket.binding.port-offset** runtime property. This allows for users to provide the port-offset for the server as a property when starting the server.

A new feature in EAP 7 is the port reduction facility. EAP port reduction tunnels protocols under HTTP using **HTTP UPGRADE** method. It allows different protocols to use the same port such as those ones provided by EJBs, messaging, or any other subsystem that can be externally accessed.

► Quiz

Configuring Interfaces and Socket Binding Groups

Based on the provided configuration, select the correct answers to the following questions:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:192.168.0.1}" />
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}" />
    </interface>
    <interface name="global">
        <any-address />
    </interface>
</interfaces>
<socket-binding-group name="standard-sockets" default-interface="public"
port-offset="${jboss.socket.binding.port-offset:10000}">
    <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}" />
    <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}" />
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}" />
    <socket-binding name="http" port="${jboss.http.port:8080}" />
    <socket-binding name="https" port="${jboss.https.port:8443}" />
    <socket-binding name="txn-recovery-environment" port="4712" />
    <socket-binding name="txn-status-manager" port="4713" />
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25" />
    </outbound-socket-binding>
</socket-binding-group>
```

► **1. Which of the following commands do NOT start an EAP standalone server that exposes the public IP address to 127.0.0.1, based on the configuration? (Select one.)**

- a. Run from a terminal window:

```
$ standalone.sh -Djboss.bind.address.management=0.0.0.0 \
-Djboss.bind.address=205.25.238.172
```

- b. Run from a terminal window:

```
$ standalone.sh -Djboss.bind.address=127.0.0.1
```

- c. Run from a terminal window:

```
$ standalone.sh
```

- d. Run from a terminal window:

```
$ standalone.sh -Djboss.bind.address=0.0.0.0
```

► **2. Which port is used by the EAP management console, based on the previous configuration with all default values used? (Select one.)**

- a. 9990
- b. 8080
- c. 19990
- d. 18080

► **3. Which interfaces bind to 127.0.0.1 in the previous configuration? (Select one.)**

- a. Global
- b. Management
- c. Public
- d. None of the above.

► **4. Which ports can be used with the HTTPS binding, based on the previous configuration? (Select one.)**

- a. 8443
- b. 8443, 9993
- c. 18443
- d. 18443, 19993

► Solution

Configuring Interfaces and Socket Binding Groups

Based on the provided configuration, select the correct answers to the following questions:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:192.168.0.1}" />
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}" />
    </interface>
    <interface name="global">
        <any-address />
    </interface>
</interfaces>
<socket-binding-group name="standard-sockets" default-interface="public"
port-offset="${jboss.socket.binding.port-offset:10000}">
    <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}" />
    <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}" />
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}" />
    <socket-binding name="http" port="${jboss.http.port:8080}" />
    <socket-binding name="https" port="${jboss.https.port:8443}" />
    <socket-binding name="txn-recovery-environment" port="4712" />
    <socket-binding name="txn-status-manager" port="4713" />
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25" />
    </outbound-socket-binding>
</socket-binding-group>
```

► **1. Which of the following commands do NOT start an EAP standalone server that exposes the public IP address to 127.0.0.1, based on the configuration? (Select one.)**

- a. Run from a terminal window:

```
$ standalone.sh -Djboss.bind.address.management=0.0.0.0 \
-Djboss.bind.address=205.25.238.172
```

- b. Run from a terminal window:

```
$ standalone.sh -Djboss.bind.address=127.0.0.1
```

- c. Run from a terminal window:

```
$ standalone.sh
```

- d. Run from a terminal window:

```
$ standalone.sh -Djboss.bind.address=0.0.0.0
```

► **2. Which port is used by the EAP management console, based on the previous configuration with all default values used? (Select one.)**

- a. 9990
- b. 8080
- c. 19990
- d. 18080

► **3. Which interfaces bind to 127.0.0.1 in the previous configuration? (Select one.)**

- a. Global
- b. Management
- c. Public
- d. None of the above.

► **4. Which ports can be used with the HTTPS binding, based on the previous configuration? (Select one.)**

- a. 8443
- b. 8443, 9993
- c. 18443
- d. 18443, 19993

▶ Lab

Configuring JBoss EAP in Standalone Mode

In this lab, you will create two new directories outside of the EAP installation folder as a base directory, run two standalone servers with a port offset, and use the Management Console to increase the log level for a category.

Resources	
Files	/opt/jboss/standalone /opt/jboss/standalone2
Application URL	http://localhost:19990 http://localhost:9990
Resources	N/A

Outcome(s)

You should be able to run two instances of EAP with a port-offset and a custom base directory as well as use the Management Console to customize a standalone server.

Before You Begin

Use the following command to download the relevant lab directory and to verify no instance of EAP is running:

```
[student@workstation ~]$ lab configuring-lab setup
```

1. In order to avoid interfering with the base install of EAP, and to maintain a backup of the server's configuration, create a new base directory for EAP. Using the current EAP installation owned by user **jboss** located at **/opt/jboss-eap-7.0**, copy the required directories as the **jboss** user into the lab directory located at **/opt/jboss/standalone**.
2. Change to the **jboss** user and start an EAP standalone server using the **/opt/jboss/standalone** directory as the EAP base directory. Wait for the server to finish starting up before proceeding.
3. Your organization wants to be able to run two standalone servers of EAP concurrently in order to configure the server differently for separate applications.
Open a new terminal window and create a new base directory for another server based on the original installation of EAP located at **/opt/jboss-eap-7.0**. Use the directory **/opt/jboss/standalone2** as the base directory for the new instance of EAP.
4. Leave the other instance of EAP running and open a new terminal and start a new instance using the command line parameter to set a port offset of **10000** in order to avoid port conflicts. The new server management console should be listening on port 19990 with the offset.
In addition, be sure to use a command line argument when starting EAP as user **jboss**, in order to use **/opt/jboss/standalone2** as the EAP base directory.

5. Wait for the second server to finish starting. Then access the management console. Update the log level of the **CONSOLE** handler and set it to **DEBUG**. This can be configured in the logging subsystem's Handler section. Restart the server to view the debug information displayed the console.
6. Access the management console for the first server and observe that the **Console** handler log level is still set to **INFO**. The first server did not change because the two servers have different configuration files in each of their respective base directories.
7. Stop the instances of EAP by pressing **Ctrl+C** in each terminal window that is running EAP. Use the following command to run the grading script:

```
[student@workstation bin]$ lab configuring-lab grade
```

This concludes the lab.

► Solution

Configuring JBoss EAP in Standalone Mode

In this lab, you will create two new directories outside of the EAP installation folder as a base directory, run two standalone servers with a port offset, and use the Management Console to increase the log level for a category.

Resources	
Files	/opt/jboss/standalone /opt/jboss/standalone2
Application URL	http://localhost:19990 http://localhost:9990
Resources	N/A

Outcome(s)

You should be able to run two instances of EAP with a port-offset and a custom base directory as well as use the Management Console to customize a standalone server.

Before You Begin

Use the following command to download the relevant lab directory and to verify no instance of EAP is running:

```
[student@workstation ~]$ lab configuring-lab setup
```

1. In order to avoid interfering with the base install of EAP, and to maintain a backup of the server's configuration, create a new base directory for EAP. Using the current EAP installation owned by user **jboss** located at **/opt/jboss-eap-7.0**, copy the required directories as the **jboss** user into the lab directory located at **/opt/jboss/standalone**.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/standalone
[student@workstation standalone]$ sudo -u jboss \
cp -r configuration deployments lib /opt/jboss/standalone
```

2. Change to the **jboss** user and start an EAP standalone server using the **/opt/jboss/standalone** directory as the EAP base directory. Wait for the server to finish starting up before proceeding.

 - 2.1.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ sudo -u jboss ./standalone.sh \
-Djboss.server.base.dir=/opt/jboss/standalone/
```

- 2.2. Verify the server is running, by accessing the management console at `http://localhost:9990` and by checking the terminal output for a message similar to the following:

```
17:03:30,497 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP  
7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3313ms -Started 261 of  
509 services (332 services are lazy, passive or on-demand)
```

3. Your organization wants to be able to run two standalone servers of EAP concurrently in order to configure the server differently for separate applications.

Open a new terminal window and create a new base directory for another server based on the original installation of EAP located at `/opt/jboss-eap-7.0`. Use the directory `/opt/jboss/standalone2` as the base directory for the new instance of EAP.

```
[student@workstation bin]$ cd /opt/jboss-eap-7.0/standalone  
[student@workstation standalone]$ sudo -u jboss \  
cp -r configuration deployments lib /opt/jboss/standalone2
```

4. Leave the other instance of EAP running and open a new terminal and start a new instance using the command line parameter to set a port offset of **10000** in order to avoid port conflicts. The new server management console should be listening on port 19990 with the offset.

In addition, be sure to use a command line argument when starting EAP as user **jboss**, in order to use `/opt/jboss/standalone2` as the EAP base directory.

```
[student@workstation standalone]$ sudo -u jboss \  
/opt/jboss-eap-7.0/bin/standalone.sh \  
-Djboss.server.base.dir=/opt/jboss/standalone2/ \  
-Djboss.socket.binding.port-offset=10000
```

5. Wait for the second server to finish starting. Then access the management console. Update the log level of the **CONSOLE** handler and set it to **DEBUG**. This can be configured in the logging subsystem's Handler section. Restart the server to view the debug information displayed the console.

- 5.1. Access the Management Console at `http://localhost:19990`.
- 5.2. Click **Configuration** at the top of the management console. In the first column, click **Subsystems** and then in the second column click **Logging** and then **View**.
- 5.3. Click **HANDLER** at the top of the page and then click **Console** to access the configuration page for the **Console** handler.
- 5.4. Under the **Attributes** header, click the **Edit** button.
- 5.5. Change the **Level** from **INFO** to **DEBUG**. Then click **Save**.
- 5.6. Stop the instance of EAP by pressing **Ctrl+C** on the terminal window that is running EAP and use the following command, the same that was used previously, to start the server again as user **jboss**:

```
[student@workstation standalone]$ sudo -u jboss \
/opt/jboss-eap-7.0/bin/standalone.sh \
-Djboss.server.base.dir=/opt/jboss/standalone2/ \
-Djboss.socket.binding.port-offset=10000
```

Verify the server is running by accessing the management console at <http://localhost:19990>.

- 5.7. Observe the additional debug information provided by the server in the console that reflects the log level change made in the management console:

```
...
10:05:20,503 DEBUG [org.jboss.as.config] (MSC service thread 1-3) Configured
system properties:
[Standalone] =
awt.toolkit = sun.awt.X11.XToolkit
file.encoding = UTF-8
file.encoding.pkg = sun.io
file.separator = /
java.awt.graphicsenv = sun.awt.X11GraphicsEnvironment
java.awt.headless = true
java.awt.printerjob = sun.print.PSPrinterJob
java.class.path = /opt/jboss-eap-7.0/jboss-modules.jar
java.class.version = 52.0
java.endorsed.dirs = /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.71-2.b15.el7_2.x86_64/
jre/lib/endorsed
java.ext.dirs = /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.71-2.b15.el7_2.x86_64/jre/
lib/ext:/usr/java/packages/lib/ext
java.home = /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.71-2.b15.el7_2.x86_64/jre
java.io.tmpdir = /tmp
...
```

6. Access the management console for the first server and observe that the **Console** handler log level is still set to **INFO**. The first server did not change because the two servers have different configuration files in each of their respective base directories.
 - 6.1. Access the Management Console at <http://localhost:9990>.
 - 6.2. Click **Configuration** at the top of the management console. In the first column, click **Subsystems** and then in the second column click **Logging** and then **View**.
 - 6.3. Click **HANDLER** at the top of the page and then click **Console** to access the configuration page for the **Console** handler and observe the log level.
7. Stop the instances of EAP by pressing **Ctrl+C** in each terminal window that is running EAP. Use the following command to run the grading script:

```
[student@workstation bin]$ lab configuring-lab grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- Running EAP as a standalone server provides users a way to run a single instance of EAP with a single server.
- The default configuration for a standalone server is contained within the **standalone.xml** configuration file.
- Port-offsets can be a simple solution for managing port conflicts when multiple instances of EAP are running on the same host.
- The three folders required to use as an EAP base directory are: **deployments**, **lib**, and **configuration**.
- Extensions are modules that extend the core capabilities of the server.
- The **profile** section of the **standalone.xml** has a collection of **subsystem** child elements, each with its own custom configuration scheme.
- It is not recommended for the **standalone.xml** file to be directly edited. Instead, users should use the EAP command line interface (CLI) tool or the Management Console.
- Runtime properties can be overridden at the command line by using the "-D" option when starting the server.
- The multiplexed approach in EAP 7 allows multiple connections to be opened on the same two ports (9990 and 8080, by default).

Chapter 3

Scripting Configuration and Deploying Applications

Overview

Goal Configure JBoss EAP using the Command Line Interface tool and deploy Java EE applications.

- Objectives**
- Connect to an instance of JBoss EAP with the CLI and execute general commands.
 - Deploy a Java EE application to a server instance running as a standalone server.

- Sections**
- Configuring JBoss EAP with the Command Line Interface (and Guided Exercise)
 - Deploying Applications in a Standalone Server (and Guided Exercise)

- Lab**
- Scripting Configuration and Deploying Applications

Configuring JBoss EAP with the Command Line Interface

Objectives

After completing this section, a system administrator should be able to:

- Connect to an instance of JBoss EAP standalone with the CLI and execute general commands.

Overview of the CLI tool

In JBoss EAP 7 most configurations reside in XML files. There are three different options for configuring for EAP servers:

1. **Management Console:** a web application that modifies the underlying XML configuration files.
2. **CLI:** The Command Line Interface, or CLI for short, to manage and configure EAP instances remotely from a command line, including writing scripts for repetitive tasks.
3. **XML:** Edit the XML configuration files directly is possible, although this is not considered a recommended practice.

Some of the benefits of using the CLI include:

- manage servers without the need of a GUI.
- Commands can be written in a separate file and executed as a batch, allowing you to write scripts for common tasks.
- The Management Console has limitations, but the CLI configures almost every aspect of the XML configuration files.

Admin-only mode of CLI

JBoss EAP 7 introduced a new feature to the CLI. Now it is possible to embed an EAP server instance inside the CLI process. Running an embedded EAP 7 server instance inside the CLI process means you have access to all the same CLI remote administration commands without actually connecting to a remote server..

The main goal of using this approach is to enable the local administration of EAP without requiring a socket-based connection. This can be very useful when for security reasons the management port (9990) is not available.

Another example of how this is useful is that you can perform some configuration offline and apply in production only when the update can be made.

To use this feature, it is necessary to start the CLI without providing the connection parameter:

```
# $JBoss_HOME/bin/jboss-cli.sh
```

After this, it is possible to start the embedded server:

```
[disconnected /] embed-server --server-config=standalone-ha.xml
```

In the previous example, every change made by the CLI will be persisted on the **standalone-ha.xml** file that is available in the **JBOSS_HOME/standalone/configuration** folder. If the **--server-config** parameter is not specified, by default, the **standalone.xml** will contain the changes.

It is possible to start with an empty configuration using the parameter **--empty-config**

```
[disconnected /] embed-server --server-config=production-ha.xml --empty-config
```

The **production-ha.xml** is created under the **JBOSS_HOME/standalone/configuration** folder. This file only contains the XML tags related to configurations performed in the CLI. If the file already exists, the command fails to avoid accidental deletion of a configuration file.



Note

The file specified in the **--server-config** parameter must exist on the **JBOSS_HOME/standalone/configuration** folder if the **--empty-config** is not specified.

Introducing the DMR syntax

EAP 6 introduced a new, de-typed syntax for representing the underlying Java objects that are used by both the Management Console and the CLI (and any other management tools that may come along).

This syntax is referred to as *Dynamic Model Representation (DMR)*, and EAP7 still uses this syntax.

The DMR syntax is a way of looking at EAP settings. DMR syntax understanding is required to work with the CLI. However the DMR syntax is intuitive and less verbose than XML.

To compare DMR and XML syntax from EAP, the following excerpt is part of the **standalone.xml** file, configuring the logging subsystem:

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
    <console-handler name="CONSOLE">
        <level name="INFO"/>
        <formatter>
            <named-formatter name="COLOR-PATTERN"/>
        </formatter>
    </console-handler>

    ...remainder of the logging subsystem
</subsystem>
```

The same settings in DMR syntax:

```
"logging" => {

    "console-handler" => {"CONSOLE" => {
        "autoflush" => true,
```

```

    "enabled" => true,
    "encoding" => undefined,
    "filter" => undefined,
    "filter-spec" => undefined,
    "formatter" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n",
    "level" => "INFO",
    "name" => "CONSOLE",
    "named-formatter" => "COLOR-PATTERN",
    "target" => "System.out"
  },
  ...remainder of the logging subsystem
}

```

One of the advantages of seeing the DMR syntax is that it is possible to see all the attributes including the attributes that have a default value. For example, in the XML syntax it is not clear that the **autoflush** attribute has the **true** value while this is clear in the DMR syntax.

Notice that the DMR is "flatter" than the XML. In DMR, most settings appear at the same level, as opposed to XML where settings can be attributes or nested within elements. For example, the **formatter** pattern in XML is several levels deep, while in the DMR syntax it is simply a property of the **CONSOLE** handler.

As the progress through this course increase, the configurations and examples discussed will gradually shift from XML to the DMR syntax. The CLI is an important tool designed for both the casual user of EAP and the "power user", and the DMR syntax is something required to be able to read and be comfortable with using.

Executing commands

The CLI is started by running the **jboss-cli.sh** script in the **bin** folder of the EAP installation. This script does not require any arguments and if arguments are not provided, it will start in disconnected mode:

```
# $JBoss_HOME/bin/jboss-cli.sh
[disconnected /]
```

The **connect** command is responsible for connecting to the server. If no argument is provided, it will try to connect to the default host (localhost) using the default port (9990).

```
[disconnected /] connect localhost:9990
```

To automatically connect using the **jboss-cli.sh** script, use the **--connect** argument:

```
# $JBoss_HOME/bin/jboss-cli.sh --connect
```

In the example above it will try to connect to the default localhost using the default port 9990.

**Note**

The OS user running CLI should be the same as the one that started EAP. The authentication is based on creating a file to pass a one-time token. EAP instance creates the file; if CLI can read the file, it sends the token to EAP and CLI is authenticated.

If EAP and CLI were started by different OS users, then the CLI will not be able to read the file and the local authentication fails. Thus, authentication falls back, and the CLI requests for a user name and password.

It is possible to use a shortcut to connect to a server:

```
# $JBOSS_HOME/bin/jboss-cli.sh -c
```

To connect to a different host machine, use the **--controller** argument:

```
# $JBOSS_HOME/bin/jboss-cli.sh --connect --controller=ip:port
```

**Note**

By default, the CLI will not request credentials if it is running in the same operating system that started the server. Credentials can be passed with **--user** and **--password** arguments.

Operations

Operations are a low level way to manage the EAP server. If such management cannot be done with an operation, it means that it cannot be done.

An operation in the CLI has the following format:

```
[node] : operation_name [ parameters ] [ headers ]
```

The **node** represents the target resource address or the node where the operation should be invoked. It as a key/value pair consisted of the node type and the node name. The **node / subsystem=datasources** is a data source.

After specifying the desired node, the operation name should be defined with an optional list of parameters. The colon is required before the operation name because it serves as a separator between the node and the operation. A colon is required even if the node is empty. In this case the operation will be executed in the current node level.

The following operations are very common:

- **:read-resource**: Reads a model resource's attribute values and either basic or complete information about any child resources.
- **:read-operation-names**: Gets the names of all the operations for the given resource.
- **:read-operation-description**: Gets the description of given operation.

- **:reload**: Reloads the server by shutting down all its services and starting them again. The JVM itself is not restarted.
- **:read-attribute**: Gets the value of an attribute for the selected resource.
- **:write-attribute**: Sets the value of an attribute for the selected resource.
- **:remove**: Removes the node.

Commands

Commands contains a user friendly syntax and most of them translate into operation requests.

The following commands are the most basic supported commands for the CLI:

- **cn** or **cd**: Change the current node path to the argument.
- **connect**: Connect to the server or domain controller.
- **data-source**: Used to manage resources of type /subsystem=datasources/data-source.
- **deploy**: Deploy an application.
- **help**: Display the help page.
- **history**: Print or disable,enable, or clear the history expansion;
- **ls**: list the contents of the node path.
- **pwn** or **pwd**: Prints the current working node.
- **exit** or **quit**: Quit the command line interface.
- **undeploy**: Undeploy an application.
- **version**: Prints the version and environment information.

Tab completion

Tab completion shows all possible commands available at any point in a current command. For example, enter / then press **Tab** to view all possible values that is possible to enter next after the /.

```
[standalone@localhost:9990 /] /  
core-service deployment-overlay interface socket-binding-  
group system-property  
deployment extension path subsystem
```

Start typing **interface** after the /, and press **Tab**, CLI not only completes the **interface** sub-level, but it adds an equals sign, because an equals sign is the only possible value after / **interface**.

Press **Tab** again and all of the interfaces will display:

```
[standalone@localhost:9990 /] /interface=  
management public
```

Running a CLI script file

EAP 7 supports using a text file to the CLI as a script. Using this approach, it is possible to create scripts for repetitive tasks. For example, it is possible to create a script file that will configure a datasource and test a connection from the pool:

```
/subsystem=datasources/data-source=appDs:add\  
  (jndi-name=java:jboss/datasources/appDS,driver-name=h2,user-name=jb248, \  
  password=jb248,connection-  
  url="jdbc:h2:mem:app;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE")  
/subsystem=datasources/data-source=appDs:test-connection-in-pool
```

If the previous content is available in the `/home/student/create-datasource.cli` file, it can be executed with:

```
# $JBOSS_HOME/jboss-cli.sh --connect --controller=localhost:9990 \  
--file=/home/student/create-datasource.cli
```

The CLI has a **batch** command that supports multiple commands to be executed as one atomic unit. If at least one of the commands or operations fails, all the other successfully executed commands and operations in the batch are rolled back.

In the previous example, it is a good idea to create the data source using the **batch** command. If testing a connection from the pool fails, the data source creation will be rolled back. To execute a batch, use the **run-batch** command:

```
batch  
/subsystem=datasources/data-source=appDs:add\  
  (jndi-name=java:jboss/datasources/appDS,driver-name=h2,user-name=jb248, \  
  password=jb248,connection-  
  url="jdbc:h2:mem:app;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE")  
/subsystem=datasources/data-source=appDs:test-connection-in-pool  
run-batch
```

Running a command from an external script

The CLI provides a feature that enables integration with external scripts. It is possible to pass a command or an operation to the CLI using the **--command** attribute:

```
# $JBOSS_HOME/jboss-cli.sh -c --controller=localhost:9990 \  
--command="/subsystem=datasources:read-resource"
```

The result of the operation will be returned using the **DMR** syntax to the script that invoked the CLI. It is possible to specify a set of commands using the **commands** attribute. This attribute specifies a comma-separated list (the list must not contain whitespace) of commands and operations that should be executed in the CLI session.

```
# ./jboss-cli.sh -c --controller=localhost:9990 --command="cd /  
subsystem=datasources,ls"
```

Demonstration: Executing CLI commands

1. Open a terminal window from the workstation VM (**Applications** → **Favorites** → **Terminal**) and run the following command to verify that EAP is installed and not currently running and to create a new base directory for EAP:

```
[student@workstation ~]$ demo executing-cli setup
```

The base directory is copied from **/opt/jboss-eap-7.0/standalone** directory. This will be used to start a new standalone server to be connected using CLI.

2. Start the new standalone server using the **/home/student/JB248/labs/executing-cli** directory as the EAP base directory. Wait for the server to finish starting before proceeding:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/executing-cli
```

3. Open a new terminal window and start the CLI by running the **jboss-cli.sh** script in the **bin** directory of EAP. Use the **--controller** property to specify the host and port of the EAP instance you want to connect to.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect \
--controller=localhost:9990
```

4. A common and useful task with the CLI is reading resource properties to discover or verify your configurations.

The CLI input is a hierarchical structure that starts at standalone server level. For example, to view the settings at the top level, enter the following command:

```
[standalone@localhost:9990 /] /:read-resource
```

5. The forward slash “/” is used to separate levels. For example, the following command reads the resources of the public interface:

```
[standalone@localhost:9990 /] /interface=public:read-resource
```

A level is similar to a folder in the Linux operating system. In the previous example, it is possible to make the analogy that the **public interface** resource is a file inside the **/** folder. It means that **/** is a top level and the **public** interface is a child level.

6. It is possible to keep drilling down further levels in the hierarchy using the forward slash. For example:

```
[standalone@localhost:9990 /] /subsystem=ejb3/thread-pool=default:read-resource
```

**Note**

A number of **bash** CLI shortcuts are also available within the JBoss CLI. For example, **Ctrl+U** clears from the cursor to the beginning of the line, **Ctrl+K** clears from the cursor to the end of the line, **Up Arrow** and **Down Arrow** step forward and backward through the CLI history. Feel free to explore which other key combinations maybe applicable.

7. It is possible to use an asterisk as a wildcard. For example, to view all the configurations of the **undertow** subsystem:

```
[standalone@localhost:9990 /] /subsystem=undertow/configuration=*:read-resource
```

The **undertow** subsystem is a web server responsible for processing Java EE based web pages.

8. Add the **recursive** flag at the end of **:read-resource** to view a resource and all of its sublevels. For example, compare the output of the following operations:

```
[standalone@localhost:9990 /] /subsystem=undertow:read-resource
[standalone@localhost:9990 /] /subsystem=undertow:read-resource(recursive=true)
```

Notice with the **recursive** flag set to **true** that all child elements of the **undertow** resource are displayed, and their child elements are displayed, and so on.

9. It is possible to view all resources on the server by simply starting at the top level and turn on recursion:

```
[standalone@localhost:9990 /] ::read-resource(recursive=true)
```

The output shows the settings of the entire server configuration, which is about 2000 lines of output. Redirect the output to a file for better viewing:

```
[standalone@localhost:9990 /] ::read-resource(recursive=true) \
> /tmp/output.txt
```

Notice no output appears in the CLI, but a new file named **/tmp/output.txt** should be available with the entire settings of your standalone server in DMR syntax.

10. The colon is used to invoke an operation, as for example with **read-resource**. Enter a colon then press **Tab**, all the available operations will be displayed for whatever level the operation is executed. For example, at the root level enter a colon and press **Tab** to view the operations at the root level:

```
[standalone@localhost:9990 /] :
```

The following output is expected:

add-namespace	map-put	read-operation-names
take-snapshot		
add-schema-location	map-remove	read-resource
undefine-attribute		
clean-obsolete-content	product-info	read-resource-description
upload-deployment-bytes		
delete-snapshot	query	reload
upload-deployment-stream		
full-replace-deployment	read-attribute	remove-namespace
upload-deployment-url		
list-add	read-attribute-group	remove-schema-location
validate-address		
list-clear	read-attribute-group-names	replace-deployment
validate-operation		
list-get	read-children-names	resolve-expression
whoami		
list-remove	read-children-resources	resolve-internet-address
write-attribute		
list-snapshots	read-children-types	resume
map-clear	read-config-as-xml	shutdown
map-get	read-operation-description	suspend

11. When an operation is invoked, the CLI displays the output in DMR syntax. Try invoking the **product-info** operation:

```
[standalone@localhost:9990 /] :product-info
```

The following output is expected:

```
{
    "outcome" => "success",
    "result" => [{"summary" => {
        "host-name" => "workstation.lab.example.com",
        "instance-identifier" => "b22faa6f-b32b-4e22-8ee6-c6e138e67447",
        "product-name" => "JBoss EAP",
        "product-version" => "7.0.0.GA",
        "product-community-identifier" => "Product",
        "product-home" => "/opt/jboss-eap-7.0",
        "standalone-or-domain-identifier" => "STANDALONE_SERVER",
        "host-operating-system" => "Red Hat Enterprise Linux Server 7.2 (Maipo)",
        "host-cpu" => {
            "host-cpu-arch" => "amd64",
            "host-core-count" => 2
        },
        "jvm" => {
            "name" => "OpenJDK 64-Bit Server VM",
            "java-version" => "1.8",
            "jvm-version" => "1.8.0_77",
            "jvm-vendor" => "Oracle Corporation",
            "java-home" => "/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.77-0.b03.el7_2.x86_64/jre"
        }
    }}]
```

```
        }
    }}}
```

12. Browsing resources in the CLI is similar to navigating a folder system from a Linux command prompt. For example, if you need to enter multiple commands on the subsystem level, it is possible to **cd** to that level. For example, if it is required to invoke multiple operations on the **datasource** subsystem, then **cd /subsystem=datasources** to that level. Enter the following commands:

```
[standalone@localhost:9990 /] cd /subsystem=datasources
[standalone@localhost:9990 subsystem=datasources] ./
data-source jdbc-driver xa-data-source
```

Notice the use of the **.** to display the sub-levels relative to the current level.

13. Use the **ls** command to view a more detailed list of the resources at the current level:

```
[standalone@localhost:9990 subsystem=datasources] ls
```

14. Use the **pwd** command to print the current working node:

```
[standalone@localhost:9990 subsystem=datasources] pwd
```

The following output is expected:

```
/subsystem=datasources
```

15. Until now, the CLI was used only to read information. However, the **write-attribute** operation can be used to modify a resource's attributes using the CLI. To demonstrate this operation, change the **min-pool-size** attribute of the **ExampleDS** data source resource. Data source configuration will be discussed later, but for now just note that this attribute defines the minimum number of connections held open for a data source.

```
[standalone@localhost:9990 /] cd /subsystem=datasources/data-source=ExampleDS
[standalone@localhost:9990 data-source=ExampleDS] :write-attribute\
(name=min-pool-size,value=5)
{"outcome" => "success"}
```



Note

Not all resource attributes are writeable. Use the **:read-resource-description** command to check if the **access-type** of the attribute you want is **read-write**. Other valid types include **read-only** and **metric**.

16. Use the **:read-attribute** command and verify the change occurred:

```
[standalone@localhost:9990 data-source=ExampleDS] :read-attribute\
(name=min-pool-size)
```

The following output is expected:

```
{  
    "outcome" => "success",  
    "result" => 5  
}
```

17. View the **standalone.xml** file of your instance (in the folder **/home/student/JB248/labs/executing-cli/configuration/standalone.xml**). Notice in the **ExampleDS** definition in the **datasource** subsystem that the **<min-pool-size>** element is **5**, and you have now seen that the CLI and the underlying XML files are synchronized.
18. The CLI **add** operation is used to add new resources to a configuration. To add a new configuration, start typing in the name of the new resource at the level where the resource is to appear. For example, the **path** element is at the root level. The following statement adds a new path named **files**:

```
[standalone@localhost:9990 data-source=ExampleDS] /path=files:add\  
(path=/files,relative-to=user.home)
```

Attributes are defined in an **add** operation as a comma-separated list of **name=value** pairs.

A path is a logical name available for an instance referring to a file system location.

19. The CLI **remove** operation is used to remove resources. To remove a configuration, navigate to the resource and run the **remove** operation.

```
[standalone@localhost:9990 data-source=ExampleDS] cd /path=files  
[standalone@localhost:9990 path=files] :remove
```

20. JBoss EAP 7 has a new management mode for the CLI. Now it is possible to start an embedded instance from the CLI. This is useful when you need to prepare some configuration without changing the production environment. Exit the CLI and stop the running instance.
21. Execute the CLI without the **-c** parameter:

```
[student@workstation bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh
```



Note

Because the files in the **/opt/jboss-eap-7.0** are owned by the **jboss** user, use the **sudo -u jboss** command to start the CLI to ensure you have suitable permissions for writing configurations.

22. The **embed-server** command is responsible for starting an embedded server to perform offline configurations. By default, the changes will be applied in the **standalone.xml** configuration file available in the **standalone** folder from the JBoss installation path. Start the embedded server:

```
[disconnected /] embed-server
```

**Note**

It is possible to define the desired file with the **--server-config** parameter. The file should already exist on the **standalone** folder.

**Note**

It is also possible to start an embedded process to configure the managed domain mode with the **embed-host-controller** command.

23. Create a new system property named **course** whose value is **JB248** and exit the CLI:

```
[standalone@embedded /] /system-property=course:add(value=JB248)
[standalone@embedded /] exit
```

24. Verify in the **/opt/jboss-eap-7.0/standalone/configuration/standalone.xml** file that the new system property is available.

```
...
</extensions>
<system-properties>
  <property name="course" value="JB248"/>
</system-properties>
<management>
  ...

```

25. The offline mode starts with an empty configuration file using the parameter **--empty-config**:

```
[student@workstation bin]$ sudo ./jboss-cli.sh
[disconnected /] embed-server --server-config=my-config.xml --empty-config
```

26. Create a new system property named **country** whose value is **US** and exit the CLI:

```
[standalone@embedded /] /system-property=country:add(value=US)
[standalone@embedded /] exit
```

27. Verify that a new configuration file was created with only the new system property defined:

```
[student@workstation bin]$ cat \
/opt/jboss-eap-7.0/standalone/configuration/my-config.xml
```

The following output is expected:

```
<server xmlns="urn:jboss:domain:4.1">

    <system-properties>
        <property name="country" value="US"/>
    </system-properties>

</server>
```

This concludes the demonstration.

► Guided Exercise

Exploring the CLI Tool

In this lab, you will explore the CLI using the offline mode.

Resources	
Files:	/opt/jboss-eap-7.0/standalone/configuration/exploring-cli.xml
Application URL:	N/A

Outcomes

You should be able to configure a stand-alone server of EAP using the CLI.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to copy the file for the exercise:

```
[student@workstation ~]$ lab exploring-cli setup
```

► 1. Start the CLI

- 1.1. Open a terminal window from the workstation VM (**Applications** → **Favorites** → **Terminal**) and run the following commands to start the CLI:

```
[student@workstation ~]$  
sudo -u jboss  
/opt/jboss-eap-7.0/bin/jboss-cli.sh
```

- 1.2. During the setup of this laboratory, a file named **exploring-cli.xml** was copied to the **/opt/jboss-eap-7.0/standalone/configuration** folder. Open this file and compare it to the original **standalone.xml** file to see that they have the same content.
- 1.3. During this laboratory, you should use the offline mode to explore the CLI tool. Start a new embedded server using the **exploring-cli.xml** as the server configuration file:

```
[disconnected /] embed-server --server-config=exploring-cli.xml
```

The previous command started an embedded instance from the CLI.

► 2. Getting information about configuration elements

- 2.1. Verify the settings at the top level using the following command:

```
[standalone@embedded /] /:read-resource
```

Remember that the CLI input is a hierarchical structure that starts at /. The previous commands execute the **read-resource** operation at the top level. The / level refers to the XML structure from **exploring-cli.xml**. When you run :**read-resource**, you will request information about the first level of the **exploring-cli.xml** file. You are getting information from the server.

- 2.2. The **ejb3** subsystem is responsible for configuring the Enterprise Java Beans specification in the EAP server. This subsystem has a resource named thread pool. A thread pool contains a number of EJBs already instanced available for the application when it is required. Using this approach, the latency will decrease during the requisitions.

Using the absolute path, verify the configuration of the **default** thread pool in the **ejb3** subsystem:

```
[standalone@embedded /] /subsystem=ejb3/thread-pool=default:read-resource
```

The previous command should display that the default thread pool can create up to ten threads.

- 2.3. Navigate to the **logging** subsystem and verify the configuration recursively:

```
[standalone@embedded /] cd /subsystem=logging  
[standalone@embedded subsystem=logging] :read-resource(recursive=true)
```

Unlike the previous step, this step uses the **cd** command to navigate to the subsystem tree. This helps to minimize the commands required to fetch and change information.

Observe that the **CONSOLE** handler has the **INFO** level defined. Logging handlers and logging levels will be discussed in a later chapter.



Note

Remember to use **Tab** to auto complete the command to avoid typos.

- 2.4. A data source is a resource in the server that create connections responsible for accessing the database from a Java application. The data source specifies the number of connections that should be created to minimize the amount of time spent waiting for the connection to be available.

Navigate to a data source named **ExampleDS** in the **datasources** subsystem and view the description of the **test-connection-in-pool** operation:

```
[standalone@embedded subsystem=logging] cd \  
/subsystem=datasources/data-source=ExampleDS  
[standalone@embedded data-source=ExampleDS] :read-operation-description\  
(name=test-connection-in-pool)
```

This operation is very useful to test if a data source is working. According to the description, you can pass a user name and a password to test the connection.

- 2.5. View the description of all attributes from the **ExampleDS** data source:

```
[standalone@embedded data-source=ExampleDS] :read-resource-description
```

Observe that the command displays if an attribute can be edited or if it is read only. Verify what the **min-pool-size** attribute is responsible for.

The **read-resource-description** is an alternative to the product documentation.

► 3. Modifying resources

- 3.1. Modify the data source to be at least 5 connections and up to ten connections within the pool:

```
[standalone@embedded data-source=ExampleDS] :write-attribute\  
  (name=min-pool-size,value=5)  
[standalone@embedded data-source=ExampleDS] :write-attribute\  
  (name=max-pool-size,value=10)
```

- 3.2. Check that the values were changed:

```
[standalone@embedded data-source=ExampleDS] :read-attribute(name=min-pool-size)  
[standalone@embedded data-source=ExampleDS] :read-attribute(name=max-pool-size)
```

- 3.3. Open the **/opt/jboss-eap-7.0/standalone/configuration/exploring-clixml** and verify that the **ExampleDS** has a pool defined with the specified values in the previous step. You should see the following content:

```
<pool>  
  <min-pool-size>5</min-pool-size>  
  <max-pool-size>10</max-pool-size>  
</pool>
```

- 3.4. Create a new system property named **env** whose value is **production**:

```
[standalone@embedded data-source=ExampleDS] /system-property=\  
  env:add(value=production)
```

Remember that when you want to create a new resource, you need to specify a name after the **=** and call the **:add** operation, passing the required attributes. You can discover the available attributes using the **Tab** key for auto completion.

- 3.5. Try to remove the path named **user.home**. It is available in the **/path** structure:

```
[standalone@embedded data-source=ExampleDS] /path=user.home:remove
```

It was not possible to remove the **user.home** path because it is read-only.

- 3.6. It is possible to remove a subsystem if necessary. For example, if your company does not have projects that use JSF, you can remove this subsystem. Remove the **jsf** subsystem:

```
[standalone@embedded data-source=ExampleDS] /subsystem=jsf:remove
```

► 4. Clean up

4.1. Exit the CLI tool:

```
[standalone@embedded data-source=ExampleDS] exit
```

4.2. Remove the **exploring-cli.xml** file:

```
[student@workstation ~]$ sudo \
rm /opt/jboss-eap-7.0/standalone/configuration/exploring-cli.xml
```

This concludes the guided exercise.

Deploying Applications to a Standalone Server

Objectives

After completing this section, a system administrator should be able to:

- Deploy a Java EE application to a server instance running as a standalone server.

Deployments section of standalone.xml

The `<deployments>` section of `standalone.xml` lists the deployed applications on the standalone server. The `<deployments>` element contains a single child element named `<deployment>`.



Note

Remember that applications are deployed using the EAP management tools, so do not add these entries manually to the XML.

Here is an example of a `<deployments>` section:

```
<deployments>
  <deployment name="bookstore.war" runtime-name="bookstore.war">
    <content sha1="e1e57cb8b89371794d6c7e80baeb8bf0e3da4fcf"/>
  </deployment>
  <deployment name="example.war" runtime-name="example.war" enabled="false">
    <content sha1="0a07b224819ce516b231b1afba0eadc45b272298"/>
  </deployment>
  <deployment name="version.war" runtime-name="version">
    <fs-exploaded path="deploying-filesystem/version.war" relative-to="labs"/>
  </deployment>
</deployments>
```

In the example above, three applications are deployed: **bookstore.war**, **example.war** and **version.war**. The first two applications are managed by EAP and the third was deployed using the unmanaged method.

The `<content>` element represents a secure hash algorithm (SHA) value of the deployment file, and is used internally as a unique identifier for the deployment.

The managed deployments are stored in the **JBOSS_HOME/data/content** folder which is the deployment cache folder. EAP creates a new folder in which the name is defined by the first two characters of the SHA1 code. Inside this folder, another folder is created which the name is defined by the other characters of the SHA1 code.



Note

If the server is using a custom base directory, the managed deployments are stored in the **BASE_DIR/data/content** folder.

For the deployments listed above, the following directory should be available:

```
[student@workstation labs]$ tree -d $JBoss_HOME/data/content/
standalone-instance/data/content/
└── 0a
    └── 07b224819ce516b231b1afba0eadc45b272298
        └── e1
            └── e57cb8b89371794d6c7e80baeb8bf0e3da4fcf
```

Because the version.war application is not managed, it will not have the application file available in the **content** folder.



Note

Adding a **<deployment>** element to the **<deployments>** section of an EAP configuration is not intended to be done manually. However, it is possible to delete entries manually and restart the server to successfully undeploy an application.

Deploying applications with management tools

The deploying process is responsible for installing an application in JBoss EAP 7. This includes the copy of the package and also the configurations that should be executed in the server.

There are three ways to deploy an application using a standalone server:

- The management console
- The CLI
- The file system deployer

Deploying using the management console

The management console has a section specific to managing the deployments available in the standalone server. It can be accessed in the management console clicking on the **Deployments** menu at the top of the page.

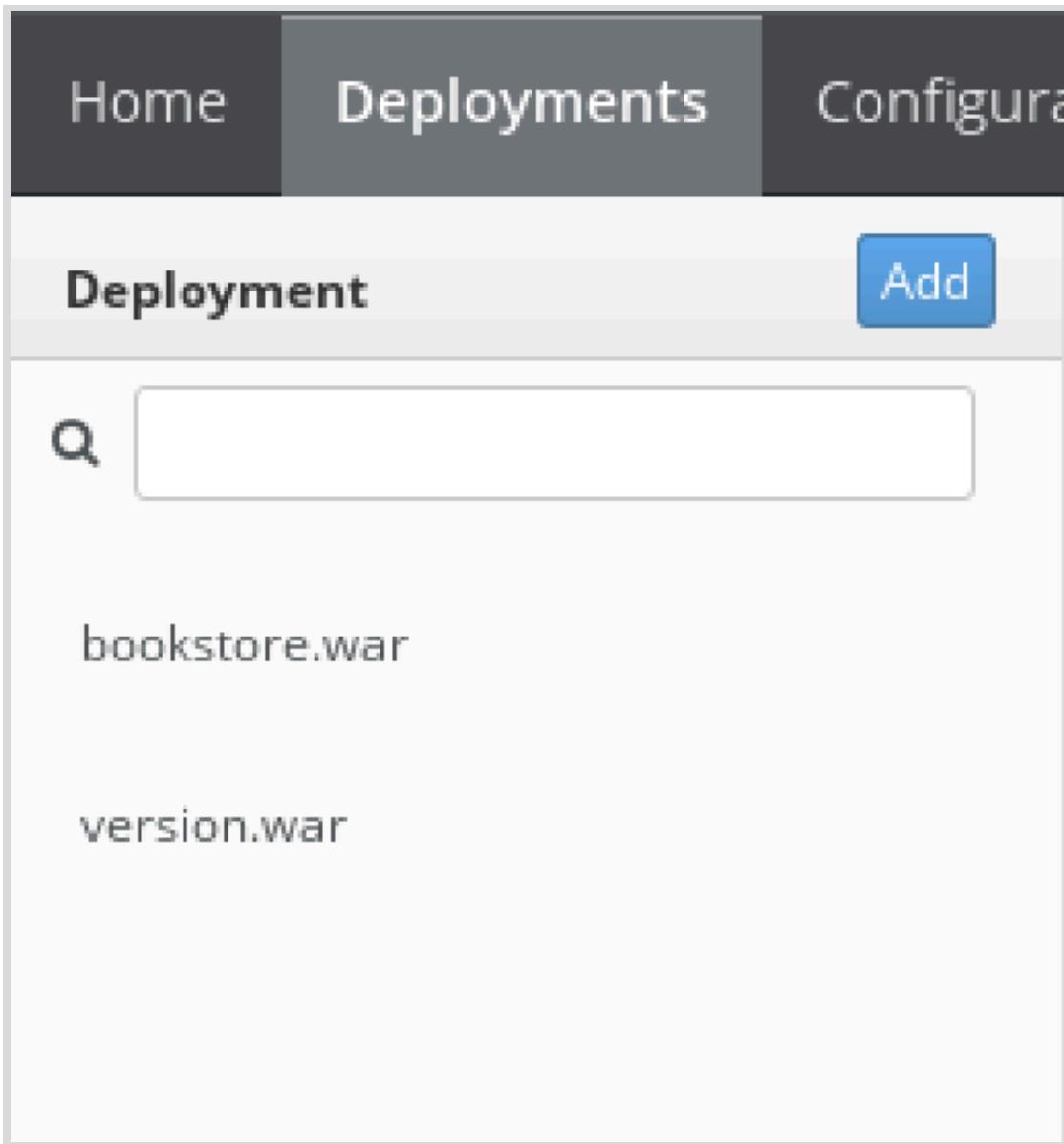


Figure 3.1: Deployments view

A new deployment can be created clicking **Add**. A wizard will start to deploy a new application. The first step of the wizard asks for the type of the deployment. Two options are available:

- **Upload a new deployment:** Using this option, an application should be uploaded and will be available in the cache. A reference to the application will be created in the **standalone.xml**.
- **Create an unmanaged deployment:** Using this option, a path must be specified to where the application archive is available. It is called unmanaged because EAP does not persist the application archive to the deployment cache and any user who has permission to the file can remove it and the application is undeployed and unavailable to the server.

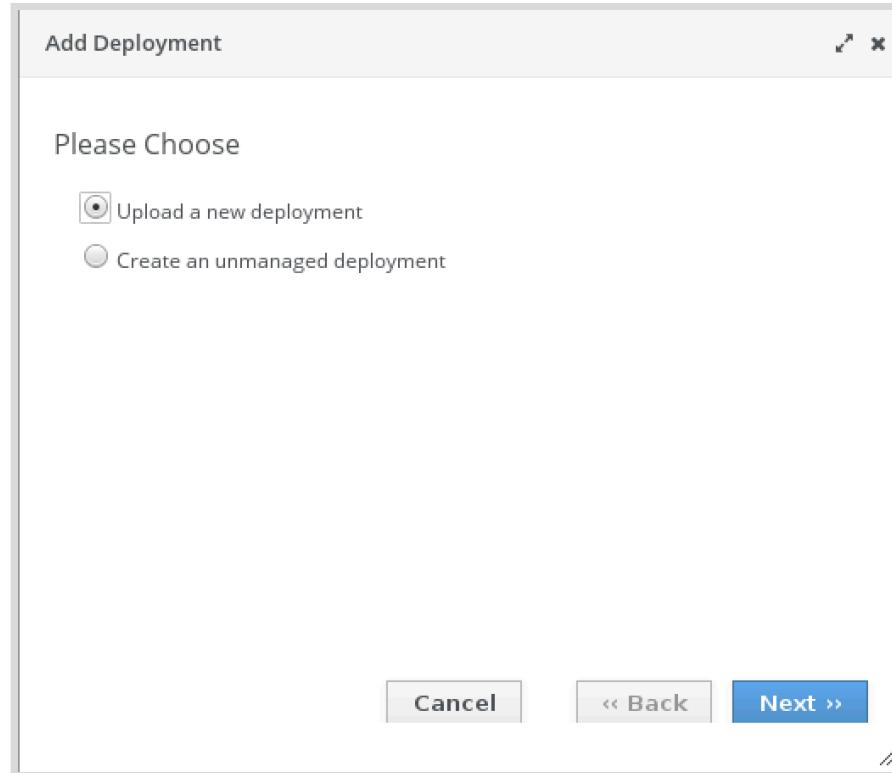


Figure 3.2: Available options for deployment

Using the **Upload a new deployment** option, the second step will ask for the file that should be uploaded. It is possible to select the desired file by clicking **Browse**.

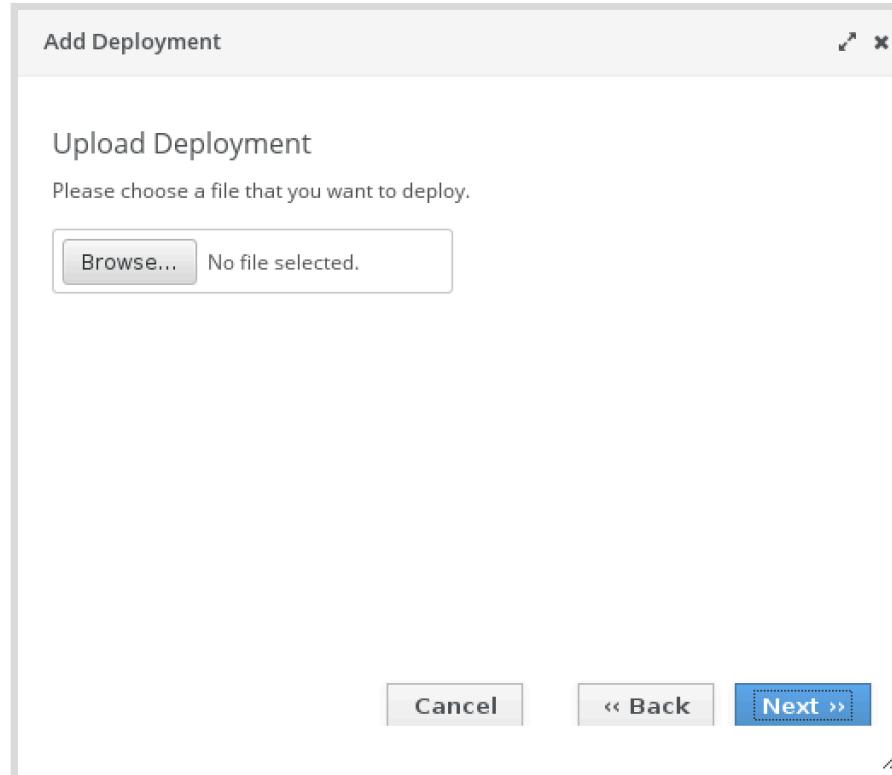


Figure 3.3: Selecting deployment file

The next step on the wizard has three options to be defined:

- **Name:** Identifies the deployment and must be a unique value across all deployments.
- **Runtime Name:** Defines the context of the application. The context is the name of the application in the runtime environment. If a deployment has a runtime name defined as **myapp**, it will be available on <http://server:port/myapp>.
- **Enabled:** Defines if the deployment should start immediately. If it is not checked, it is possible to enable it later.

Enabling and disabling a deployment using the management console

An application can be configured to start or not during the startup of EAP 7. Configure the deployment as enabled if it should start during EAP 7 startup.

The management console enable and disable a deployment using the **Deployments** menu. To disable an application, click on the application. A combo box should appear. Clicking the down arrow, the **Disable** option will be displayed:

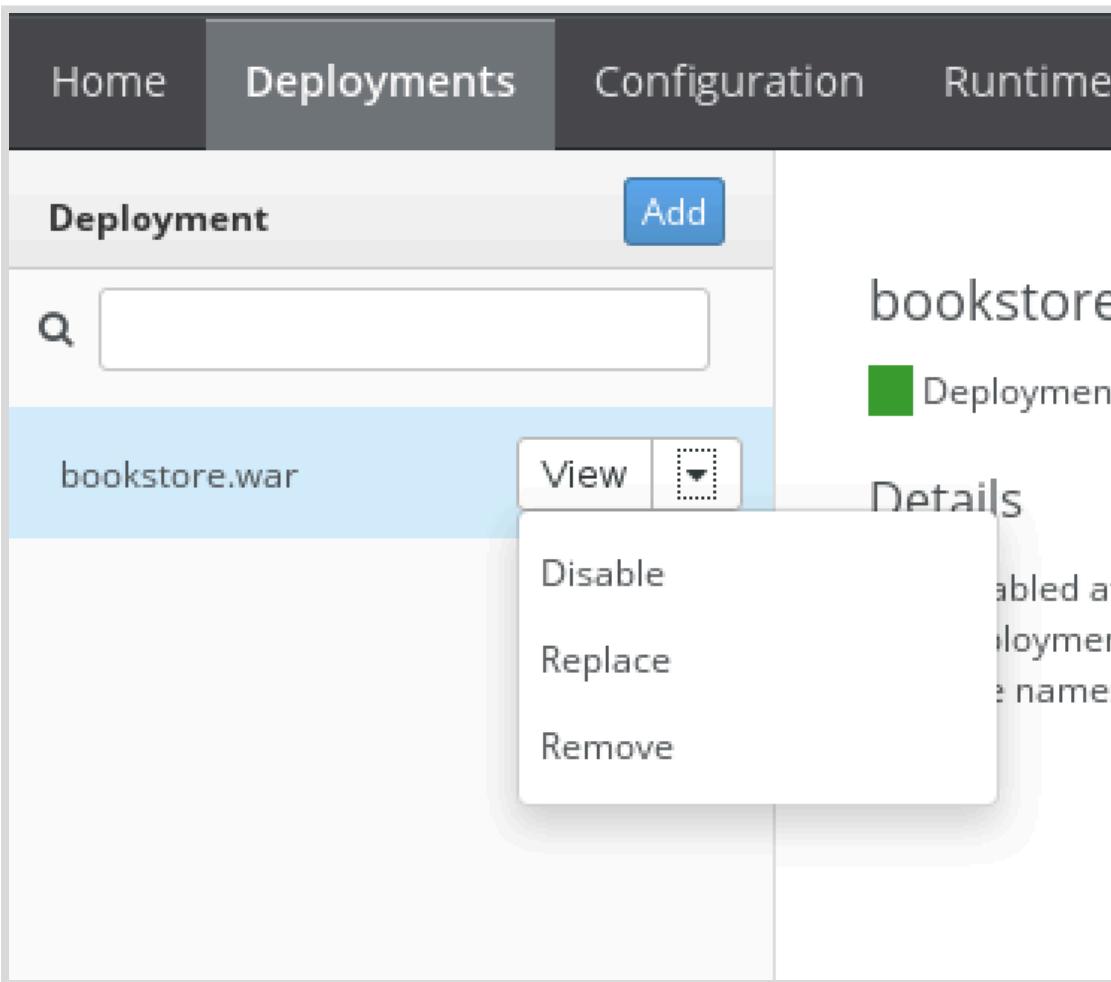


Figure 3.4: Disabling a deployed application

A confirmation screen should be displayed. Click **Confirm** and the deployment will be disabled.

To enable a deployment, click on the application and again the combo box will be displayed. Click on the down arrow to see the **Enable** option.

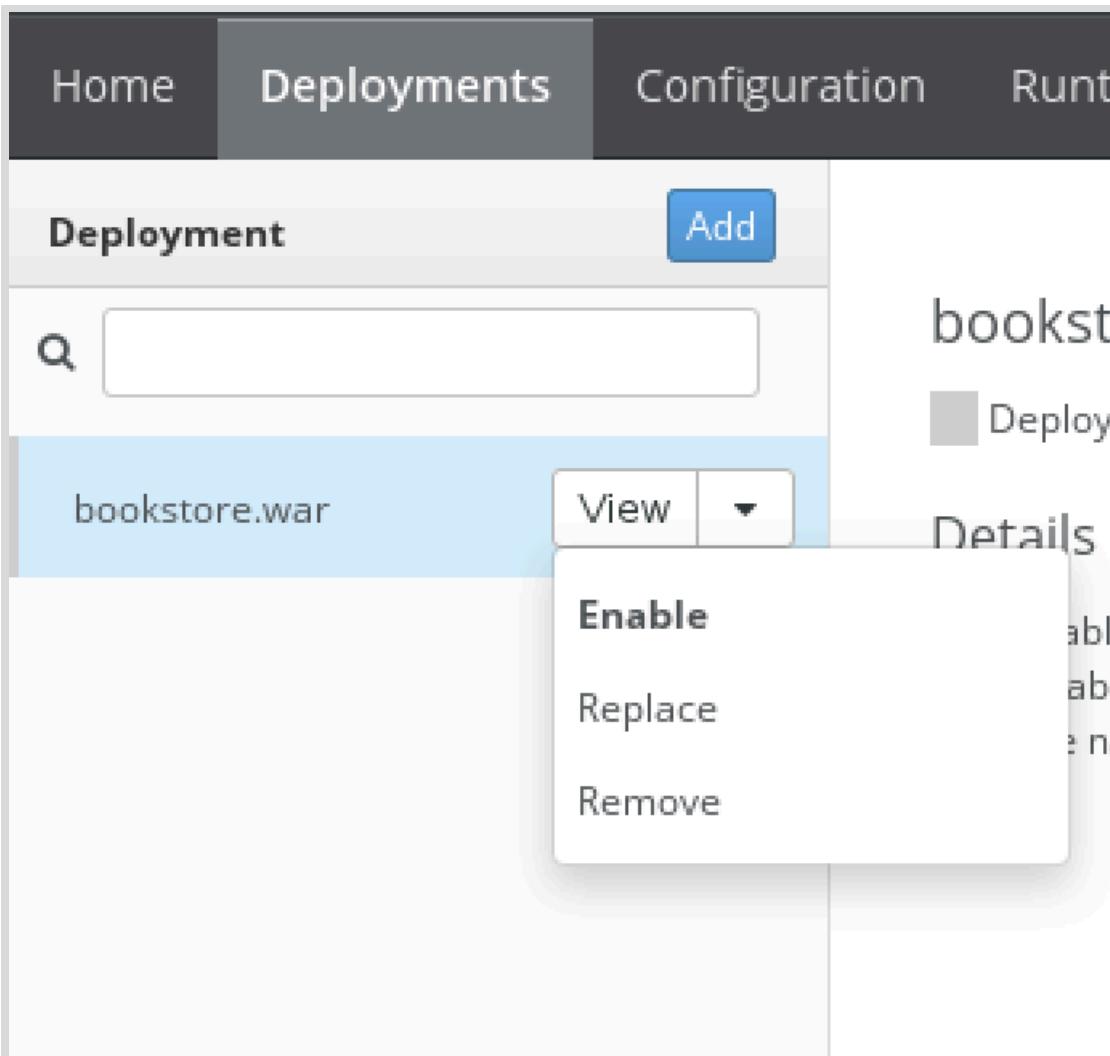


Figure 3.5: Enabling an application

Undeploying an application using the management console

The management console undeploys an application using the **Deployments** menu. To undeploy an application, click the application. A combo box should appear. Clicking the down arrow, the **Remove** option will be displayed:

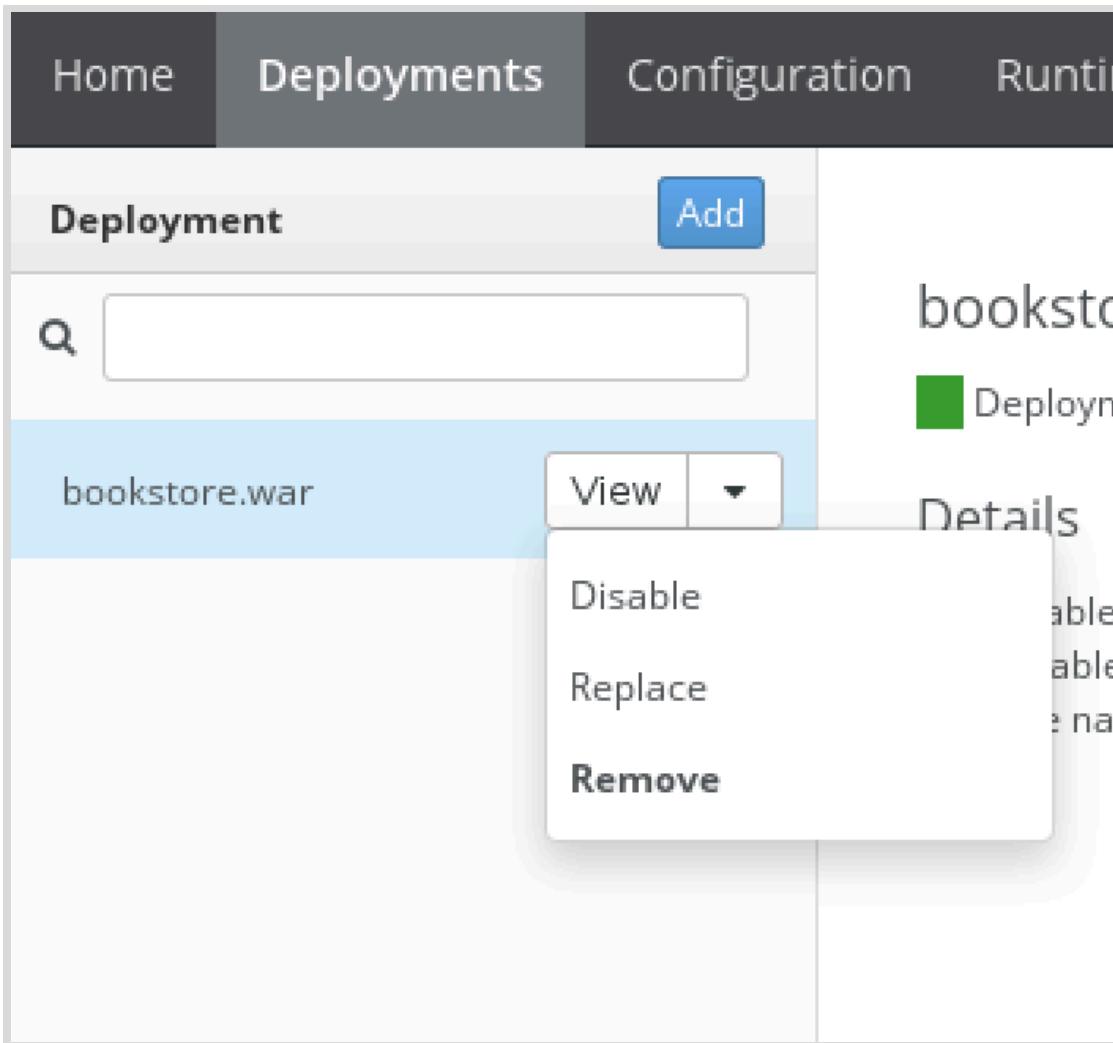


Figure 3.6: Undeploying a deployed application

A confirmation screen is displayed. Click **Confirm** and the application will be undeployed.



Note

Undeploying is different from disabling a deployment. Undeploying will uninstall the application from EAP removing the application from the server while disabling will not start it but the deployment is available and can be started without a new upload.

Deploying using the CLI

The CLI provides the **deploy** command to start a new deployment. This command has arguments and the principal arguments are listed:

- **file_path**: Path to the application to deploy.
- **--url**: URL at which the deployment content is available for upload to the deployment content repository.
- **--name**: The unique name of the deployment. If no name is provided then the file name is used.
- **--runtime-name**: Optional, defines the runtime name for the deployment.

- **--force**: If the deployment with the specified name already exists, by default, the deployment is aborted and the corresponding message is printed. The **--force (-f)** forces the replacement of the existing deployment with the one specified in the command arguments.
- **--disabled**: Indicates that the deployment has to be added to the repository in a disabled state.

To deploy an application that is located at **/home/student/myapp.war** use the following command:

```
[standalone@localhost:19990 /] deploy /home/student/myapp.war --name=myapp.war
```

Undeploying using the CLI tool

The CLI provides the **undeploy** command to remove a deployment. This command has several arguments and the most common are:

- **name**: The name of the application to undeploy.
- **--keep-content**: Disable the deployment but do not remove its content from the repository.

```
[standalone@localhost:19990 /] undeploy myapp.war
```

Deploying applications using the file system deployer

The file system deployer is a subsystem from EAP that will scan for JEE compliant app at a certain directory. Now it is possible to execute operations in this type of deployment using the management tools while in EAP 6 it was possible only to manage directly from the file system. For example, a deploy using the file system deployer can be disabled using the Management console.

The standalone server supports deployment of a new application manually. To start a manually deployment, it is only required to copy the application to the **JBOSS_HOME/deployments** folder.

The deployment process is managed using marker files. A marker file is an empty file that uses the same name as the application, and adds the objective of the marker file to the end of the file name. The marker file should be created in the **deployments** folder. The following marker files extensions can be used:

- **.dodeploy**: It is created by the user. It triggers a new deployment of the application.
- **.deployed**: It is created by the deployment scanner. It indicates that the application has been deployed.
- **.isdeploying**: It is created by the deployment scanner. It indicates that the application is deploying.
- **.failed**: It is created by the deployment scanner. It indicates that the application has failed.
- **.isundeploying**: It is created by the deployment scanner. It indicates that the application is undeploying.
- **.undeployed**: It is created by the deployment scanner. It indicates that the application has been undeployed.
- **.skipdeploy**: It is created by the user. It indicates that the application should not be deployed.

- **.pending**: It is created by the deployment scanner. It indicates that it has noticed the need to deploy content but has not yet instructed the server to deploy it.

To deploy an application named myapp.war, create the **myapp.war.dodeploy** marker file.

Demonstration: Deploying applications with the file system deployer

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command:

```
[student@workstation ~]$ demo  
deploying-filesystem setup
```

The previous command will verify that:

- EAP is installed.
 - Guided exercise **Creating a Standalone Server** was executed.
 - EAP is not running.
 - Download the required files for this demonstration.
2. Run the following commands to start the EAP server using **/home/student/JB248/labs/standalone-instance** as the base directory:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone-instance/
```

Remember that this base directory has been configured the **port-offset** to 10000. It means that the administration port is **19990** and the server port is **18080**.

3. Locate the directory **/home/student/JB248/labs/deploying-filesystem/version.war**, which represents a simple web application that displays the version of JBoss that it is deployed on. This is an example of an exploded application; the application is not in a single, compressed WAR file.

It important to understand that the app is not a static HTML page and the exploded file is not a Java EE standard web app. It is a convenience provide by EAP 7.

4. Open a new terminal window and copy the directory **version.war** into your **/home/student/JB248/labs/standalone-instance/deployments** directory.

```
[student@workstation bin]$ cp -r \  
/home/student/JB248/labs/deploying-filesystem/version.war \  
/home/student/JB248/labs/standalone-instance/deployments
```

5. Look in the terminal window of your EAP instance and you should see the following output:

```
12:54:37,969 INFO [org.jboss.as.server.deployment.scanner] (DeploymentScanner-  
threads - 1) WFLYDS0004: Found version.war in deployment directory. To trigger  
deployment create a file called version.war.dodeploy
```

A marker file is an empty file read by the **deployment-scanner** subsystem to deploy new apps. It should have the same name as the deployed app, with a predefined extension. In the next step you will define this marker file.

6. To cause EAP to deploy the exploded **version.war** application, you need to create the marker file. Create a new, empty text file in the **/home/student/JB248/labs/standalone-instance/deployments** folder named **version.war.dodeploy**.

```
[student@workstation bin]$ cd \
/home/student/JB248/labs/standalone-instance/deployments
[student@workstation deployments]$ touch version.war.dodeploy
```

7. To verify your marker file was successfully processed by the deployment-scanner subsystem, check the terminal window of EAP. You should see something similar to:

```
12:55:48,100 INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
WFLYSRV0027: Starting deployment of "version.war" (runtime-name: "version.war")
...OUTPUT OMMITED...
12:55:51,237 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2)
WFLYSRV0010: Deployed "version.war" (runtime-name : "version.war")
```

Check the files from the **/home/student/JB248/labs/standalone-instance/deployments/** and verify that the file was renamed. During the deploy, the file is named **version.war.deploying**. When the deploy is finished, the file will be renamed to **version.war.deployed**.

8. Open the **/home/student/JB248/labs/standalone-instance/configuration/standalone.xml** file and check that there are no entries made in the file related to the deployment. You should see that the **deployment** tag is not available. Also, manual deployment does not store files in the managed cache. It is possible to check this verifying that **/home/student/JB248/labs/standalone-instance/data/content** directory is empty.
9. Navigate to <http://localhost:18080/version>. You should see the example application.
EAP 7 can deploy multiple apps and each one will be available at <http://localhost:8080/<appname-without-the-war-extension>> by default. Because EAP was started with a port offset of 10000 then it will be available at <http://localhost:18080/<appname-without-the-war-extension>>.
10. Using a text editor, open the file **/home/student/JB248/labs/standalone-instance/deployments/version.war/index.xhtml**. You will modify the main page of the version application.
11. On line 25, change **1.0** to **2.0** and save the file. Notice in the terminal window of EAP that the **version.war** application was not automatically redeployed. However, the page will be automatically updated when it is accessed by a web browser.
12. Go to your web browser and refresh the page <http://localhost:18080/version>. You should see version 2.0 of the application.
13. To redeploy the entire **version.war** application, you need to change the time stamp on the file **/home/student/JB248/labs/standalone-instance/standalone/deployments/ version.war.deployed**:

```
[student@workstation deployments]$ touch version.war.deployed
```

14. In your EAP terminal window, you should see that the **version.war** app is redeployed:

```
15:10:38,010 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 62) WFLYUT0022: Unregistered web context: /version  
... OUTPUT OMMITED...  
15:10:39,288 INFO [org.jboss.as.server] (DeploymentScanner-threads - 1)  
WFLYSRV0013: Redeployed "version.war"
```

15. To undeploy the version application, delete the file **version.war.deployed**:

```
[student@workstation deployments]$ rm version.war.deployed
```

16. Within a few seconds, the deployment scanner will undeploy the **version.war** app and create a new marker file named **version.war.undeployed** in the **deployments** folder.
17. In your EAP terminal window, you should see that the **version.war** application was undeployed:

```
15:11:54,462 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 67) WFLYUT0022: Unregistered web context: /version  
... OUTPUT OMMITED...  
15:11:54,556 INFO [org.jboss.as.server] (DeploymentScanner-threads - 1)  
WFLYSRV0009: Undeployed "version.war" (runtime-name: "version.war")
```

This concludes the demonstration.

► Guided Exercise

Deploying Applications with Management Tools

In this lab, you will deploy two applications using the management tools provided by EAP 7.

Resources	
Files:	/home/student/JB248/labs/deploy-tools
Application URL:	http://localhost:18080/example http://localhost:18080/version

Outcomes

You should be able to deploy an application using the Management Console and using the CLI.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to copy the files for the exercise:

```
[student@workstation ~]$ lab deploy-tools setup
```

► 1. Deploy using the management console

1. Open a terminal window from the workstation VM (**Applications** → **Favorites** → **Terminal**) and run the following commands to start the EAP server using the **/home/student/JB248/labs/standalone-instance** as the base directory:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/standalone-instance/
```



Note

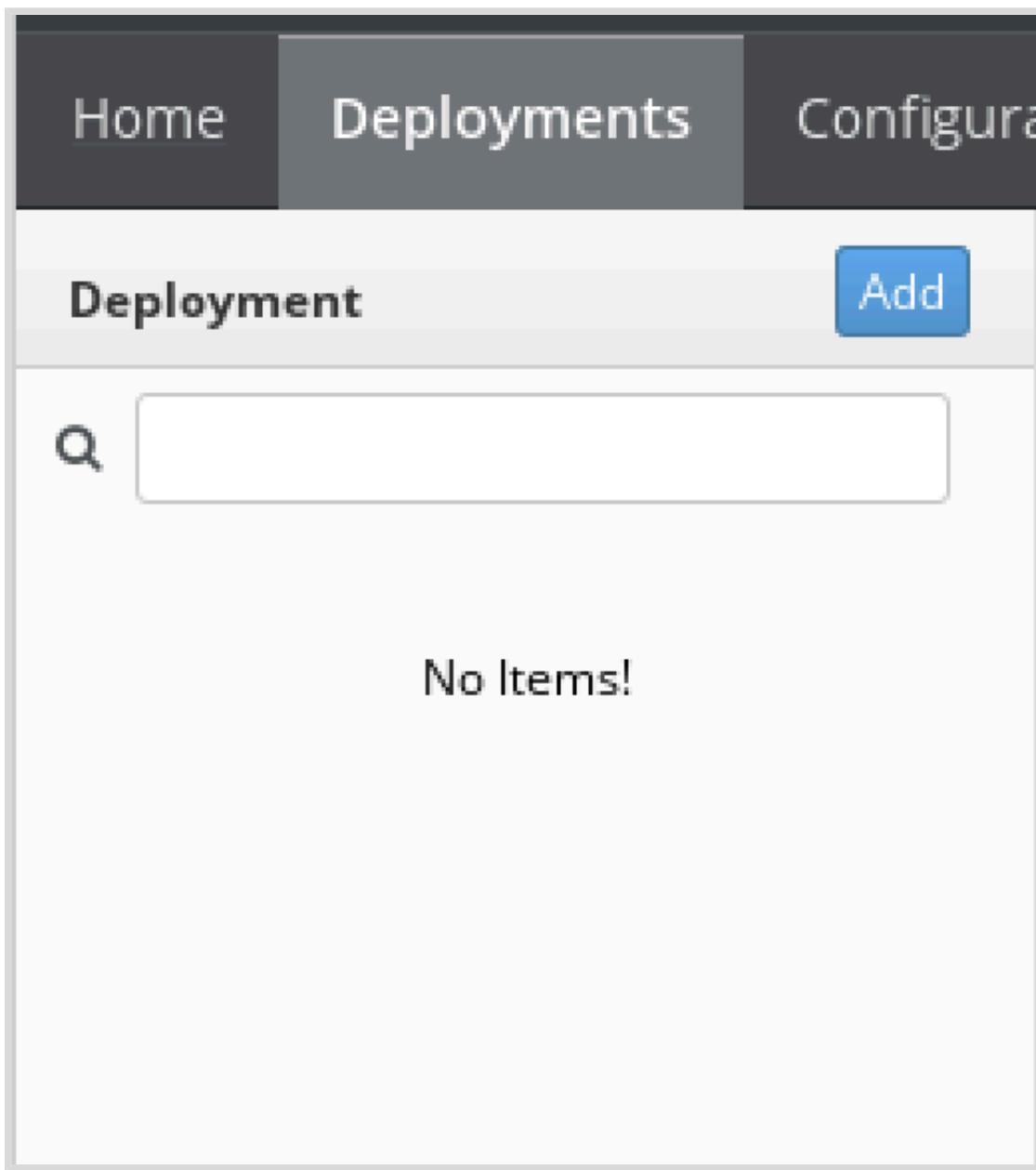
Remember that the **standalone.xml** file from the base directory was configured with the **port-offset** set to **10000**. This means that the management console will be available at port **19990** and the deployed applications will be available at port **18080**.

2. Navigate to **http://localhost:19990** to access the management console page. Use the following credentials to access it:

- **username: jbossadm**

- **password: JBoss@RedHat123**

13. Click **Deployments** in the navigation menu bar. You do not have anything deployed yet, so the list of deployments will be empty.



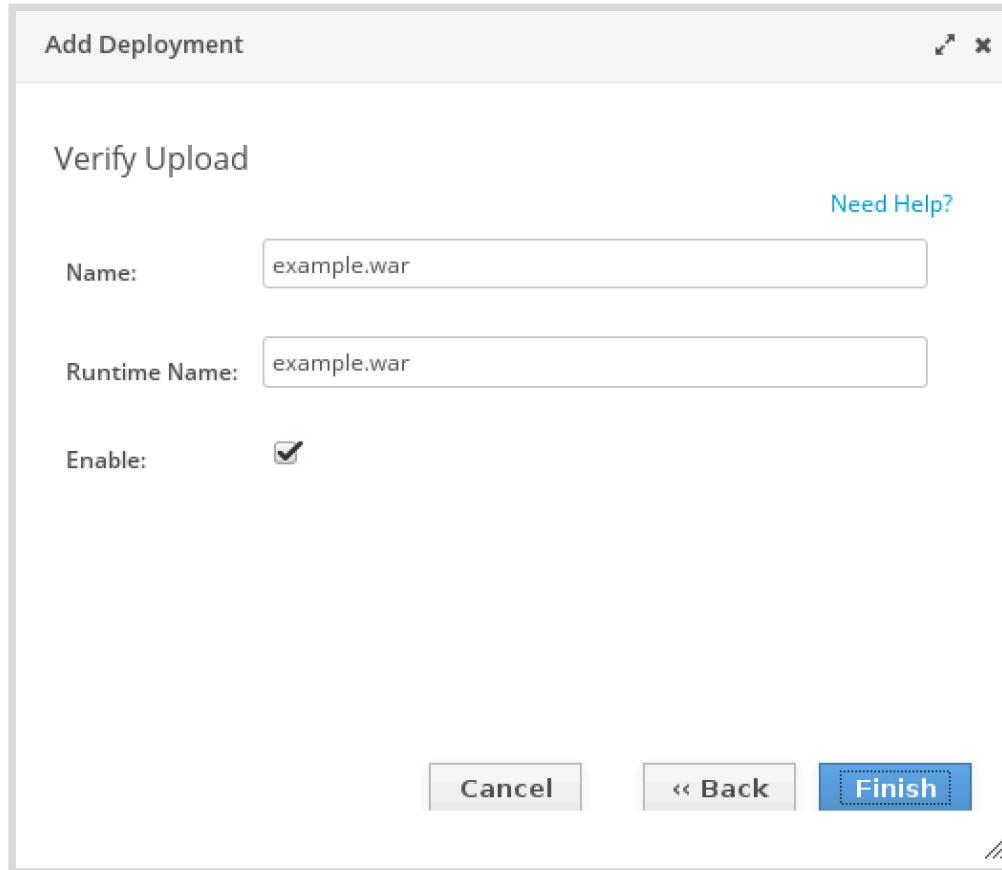
14. Click **Add** to add a new deployment.
15. Select the **Upload a new deployment** option and click **Next**.



Note

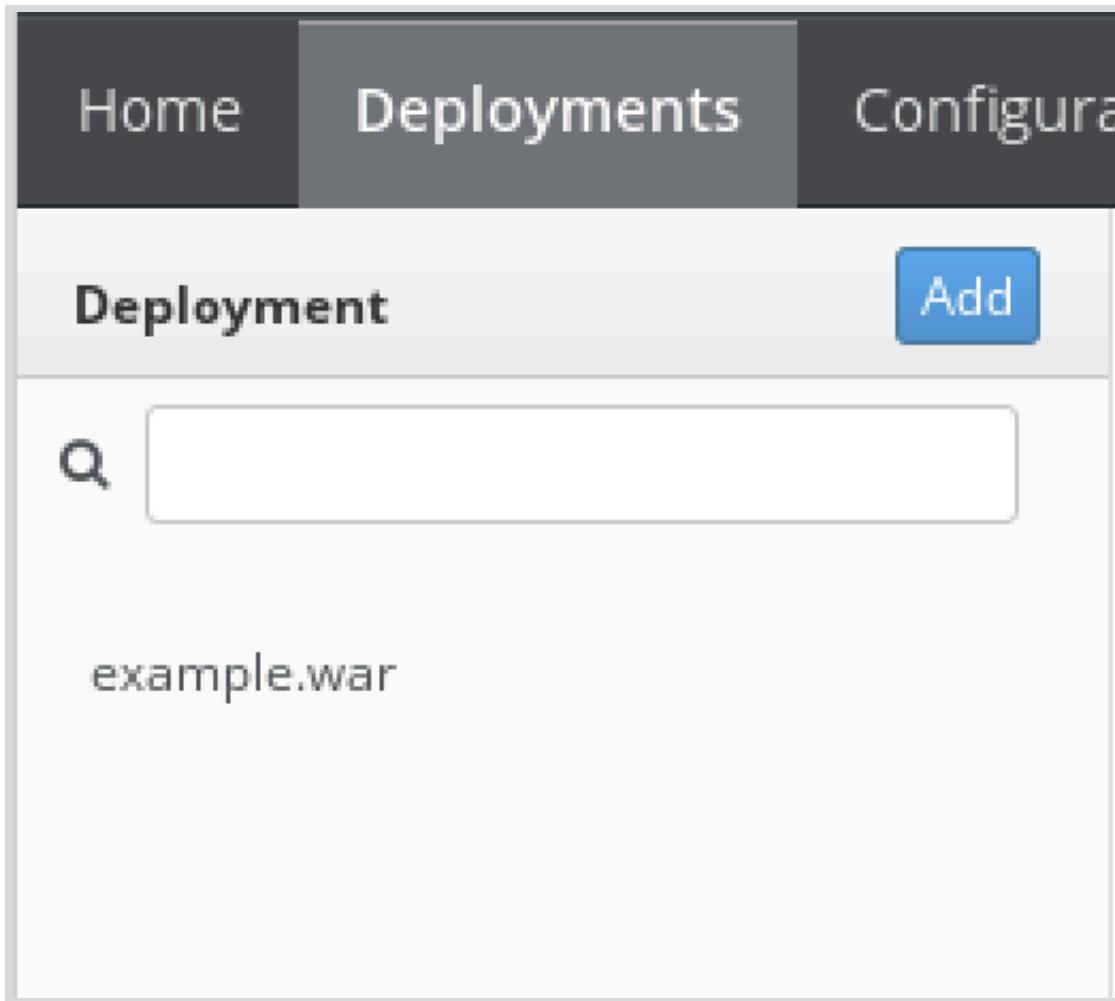
Another option is available in the management console to deploy an application. The **Create an unmanaged deployment** option deploys an application using a configured path pointing to a folder.

16. Click the **Browse** button and select the file **example.war** in your **/home/student/JB248/labs/deploy-tools** folder. Click **Next** to go to step 3 of the wizard.
17. In step 3, you can use the default values or you can change the name and runtime name of the deployment. The name identifies the deployment, and the runtime name specifies the context of the application. A runtime name of myapp means the application is available at <http://server-address:port/myapp>.

**Note**

The uploaded deployment is enabled by default, and it will automatically deploy the application. Clearing it will not deploy the application, and a manual deployment request should be executed.

18. Click **Finish** to complete the wizard. You should now see **example.war** in the list of deployments.



- 1.9. Look in the terminal window of your running instance of EAP. You should see an output similar to the following:

```
INFO [org.jboss.weld.deployer] (MSC service thread 1-2) WFLYWELD0009: Starting weld service for deployment example.war  
...OUTPUT OMMITED...  
INFO [org.jboss.as.server] (XNIO-1 task-6) WFLYSRV0010: Deployed "example.war" (runtime-name : "example.war")
```

- 1.10. Navigate to <http://localhost:18080/example>. You should see the example application, which displays a simple web page.



Welcome to EAP 7

This is a simple web app deployed as compressed WAR file named example.war

- 1.11. Open the **/home/student/JB248/labs/standalone-instance/configuration/standalone.xml** file to view its contents. You should see a **<deployments>** section at the end of this file that contains your **example.war** deployment:

```
<deployments>
  <deployment name="example.war" runtime-name="example.war">
    <content sha1="0a07b224819ce516b231b1afba0eadc45b272298"/>
  </deployment>
</deployments>
```

- 1.12. Navigate to the **/home/student/JB248/labs/standalone-instance/content/data** folder. This folder contains child folders that are responsible for persisting all the applications that were deployed using the management tools provided by EAP 7. You can identify each deployment using the **sha1** code provided in the **standalone.xml** file. The first level of the directory contains a folder named using the first and the second character from the sha1. This folder contains another folder named using the rest of the sha1 code and finally a file named content will be the application.

► 2. Disable the deployment using the Management Console

- 2.1. It is possible to disable a deployment to undeploy the application without removing it from the server. This can be useful if you want to undeploy the application with a possibility to redeploy it again later without uploading the application again. Go back to the **Deployments** page of the management console.
- 2.2. Click **example.war** application. A combo box should be displayed. Click the down arrow and select the **Disable** option to undeploy the application from your standalone server. Click **Confirm** when the confirmation window appears.
- 2.3. Click **example.war** application and notice that the application is no longer enabled.
- 2.4. Look in the terminal window of your running instance of EAP. You should see an output similar to the following:

```
INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 63)
WFLYUT0022: Unregistered web context: /example
...OUTPUT OMITTED...
INFO [org.jboss.as.server] (XNIO-1 task-8) WFLYSRV0009: Undeployed
"example.war" (runtime-name: "example.war")
```

- 2.5. Try reloading the URL `http://localhost:18080/example` in your browser. You should get a 404 error.

► 3. Deploying applications using the CLI

- 3.1. The CLI can deploy an application. Open a new terminal window and start the CLI by running the `jboss-cli.sh` script in the `bin` folder of EAP.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:19990
```

- 3.2. Applications can be deployed using the `deploy` command, passing the location of the file that should be deployed:

```
[standalone@localhost:19990 /] deploy \
/home/student/JB248/labs/deploy-tools/version.war
```

- 3.3. Look in the terminal window of your running instance of EAP. You should see output similar to the following:

```
INFO [org.jboss.weld.deployer] (MSC service thread 1-2) WFLYWELD0009: Starting
weld service for deployment version.war
...OUTPUT OMITTED...
INFO [org.jboss.as.server] (XNIO-1 task-6) WFLYSRV0010: Deployed
"version.war" (runtime-name : "version.war")
```

- 3.4. Navigate to `http://localhost:18080/version`. You should see the version application, which displays a simple web page.

► 4. Disable the deployment using the CLI

- 4.1. It is also possible to disable an application using the CLI tool. List the available applications:

```
[standalone@localhost:19990 /] cd /deployment
[standalone@localhost:19990 deployment] ls
```

You should see the following output:

```
example.war  version.war
```

- 4.2. Disable the deployment with the `undeploy` operation:

```
[standalone@localhost:19990 deployment] ./version.war:undeploy
```

- 4.3. Look in the terminal window of your running instance of EAP. You should see an output similar to the following:

```
INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 63)
WFLYUT0022: Unregistered web context: /version
...OUTPUT OMITTED...
INFO [org.jboss.as.server] (XNIO-1 task-8) WFLYSRV0009: Undeployed
"version.war" (runtime-name: "version.war")
```

- 4.4. Try reloading the URL `http://localhost:18080/version` in your browser. You should get a 404 error.

- 4.5. Use the **redeploy** operation to enable the deployment again:

```
[standalone@localhost:19990 deployment] ./version.war:redeploy
```

- 4.6. Try reloading the URL `http://localhost:18080/version` in your browser. You should see the version application.

▶ 5. Clean up

- 5.1. Remove the example.war application:

```
[standalone@localhost:19990 /] /deployment=example.war:remove
```

- 5.2. Remove the version.war application:

```
[standalone@localhost:19990 /] /deployment=version.war:remove
```

- 5.3. Exit the CLI tool:

```
[standalone@localhost:19990 /] exit
```

- 5.4. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the guided exercise.

► Guided Exercise

Scripting Configuration and Deploying Applications

In this lab, you will configure the standalone server and deploy the bookstore application using the CLI tool.

Resources	
Files:	/home/student/JB248/labs/cli-lab
Application URL:	http://localhost:8081 http://localhost:8081/bookstore

Outcomes

You should be able to configure the standalone server and deploy an application using the CLI tool.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to copy the files for the exercise:

```
[student@workstation ~]$ lab cli-lab setup
```

The **/opt/jboss/standalone2** server was deployed as a second server from laboratory **Configuring JBoss EAP in Standalone Mode**. For this laboratory, the first server (configured at **/opt/jboss/standalone** dir) will be stopped for maintenance purposes, and an application will be deployed on the second server (configured at **/opt/jboss/standalone2**).

A new service will be installed in the operating system by the system administrator that requires the **8080** port. (The service is not related to Java.) To avoid port conflicts and due to network port limitations, apps from the second server should be available at port **8081**. Also the server logging level was too verbose and the sysadmin requested that you to lower the verbosity level to minimize the logging overhead.

- 1. Start an EAP standalone server using the **/opt/jboss/standalone2** directory as the EAP base directory. This server should be started using the **jboss** user. Wait for the server to finish starting before proceeding.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ sudo -u jboss ./standalone.sh \
-Djboss.server.base.dir=/opt/jboss/standalone2/
```

Verify the server is running by accessing the management console at <http://localhost:9990>.

Open a web browser and navigate to the management console to check if the management console was started.

- ▶ 2. By default, applications deployed on EAP 7 is available on the **8080** port. However, the sysadmin uses port **8080** for another service and it conflicts with the EAP port responsible for providing access to web applications. In this step, change only the web application port to be served at **8081**. This port is defined in a **socket-binding** named **http**. This socket-binding is available in a **socket-binding-group** named **standard-sockets**. Using the CLI tool, discover the attribute responsible for defining the default port and change its value to **8081**.

- 2.1. Open a new terminal window and connect to the CLI tool using **localhost** on the **9990** port.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ sudo -u jboss ./jboss-cli.sh -c --  
controller=localhost:9990
```

- 2.2. Navigate to the **socket-binding-group** named **standard-sockets**:

```
[standalone@localhost:9990 /] cd /socket-binding-group=standard-sockets
```

- 2.3. Navigate to the **socket-binding** named **http**:

```
[standalone@localhost:9990 socket-binding-group=standard-sockets] cd \  
socket-binding=http
```

- 2.4. Change the **port** attribute to **8081**:

```
[standalone@localhost:9990 socket-binding=http] :write-attribute\  
(name=port,value=8081)
```

The following output is expected:

```
{  
    "outcome" => "success",  
    "response-headers" => {  
        "operation-requires-reload" => true,  
        "process-state" => "reload-required"  
    }  
}
```

- 2.5. Read the resource properties of the **http** socket binding:

```
[standalone@localhost:9990 socket-binding=http] :read-resource
```

The following output is expected:

```
{  
    "outcome" => "success",  
    "result" => {  
        "client-mappings" => undefined,
```

```
"fixed-port" => false,  
"interface" => undefined,  
"multicast-address" => undefined,  
"multicast-port" => undefined,  
"name" => "http",  
"port" => expression "${jboss.http.port:8081}"  
}  
}
```

- 2.6. The standalone server must be reloaded to start listening on the **8081** port. Reload the server:

```
[standalone@localhost:9990 socket-binding=http] cd /  
[standalone@localhost:9990 /] :reload
```

- 2.7. Open a web browser and point it to `http://localhost:8081` to test the new port.

▶ 3. Change the log level

The running standalone server is configured to display debug messages in the **CONSOLE**. The logging output was huge and the sysadmin has requested that you decrease the amount of logging. Change the level of this console handler to **INFO** using the CLI tool:

- 3.1. Navigate to the **console-handler** named **CONSOLE** in the **logging subsystem**:

```
[standalone@localhost:9990 /] cd /subsystem=logging/console-handler=CONSOLE
```

- 3.2. Change the level to **INFO**:

```
[standalone@localhost:9990 console-handler=CONSOLE] :write-attribute\  
(name=level,value=INFO)
```

- 3.3. Read the resource properties of the **CONSOLE** console handler:

```
[standalone@localhost:9990 console-handler=CONSOLE] :read-resource
```

The following output is expected:

```
{  
    "outcome" => "success",  
    "result" => {  
        "autoflush" => true,  
        "enabled" => true,  
        "encoding" => undefined,  
        "filter" => undefined,  
        "filter-spec" => undefined,  
        "formatter" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n",  
        "level" => "INFO",  
        "name" => "CONSOLE",  
        "named-formatter" => "COLOR-PATTERN",  
        "target" => "System.out"  
    }  
}
```

► 4. Deploy the bookstore application

The bookstore application is an e-commerce application that sells books. Deploy the application using the CLI tool. The application is available in the **/tmp** folder.

- 4.1. Deploy the application using the **deploy** command:

```
[standalone@localhost:9990 console-handler=CONSOLE] cd /
[standalone@localhost:9990 /] deploy \
/tmp/bookstore.war
```

- 4.2. Look in the terminal window of your running instance of EAP. You should see output similar to the following:

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3) WFLYSRV0027:
Starting deployment of "bookstore.war" (runtime-name: "bookstore.war")
...OUTPUT OMMITED...
INFO [org.jboss.as.server] (XNIO-2 task-6) WFLYSRV0010: Deployed
"bookstore.war" (runtime-name : "bookstore.war")
```

- 4.3. Open a web browser and navigate to <http://localhost:8081/bookstore>. You should see the bookstore application.

► 5. Grade

- 5.1. Exit from the CLI tool:

```
[standalone@localhost:9990 deployment] exit
```

- 5.2. Run the following command to grade your work on this laboratory:

```
[student@workstation bin]$ lab cli-lab grade
```

- 5.3. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the lab.

Summary

In this chapter, you learned:

- There are three ways to configure EAP servers:
 - Using the management console
 - Using the CLI
 - Using XML files
- The CLI provides a new feature to embed an EAP server inside the CLI process.
- EAP 7 still uses the DMR syntax.
- The CLI is started by running the **jboss-cli.sh** script in the **bin** folder of the EAP installation.
- Operations are a low-level way to manage the EAP server.
- Commands use a simple syntax and most commands translate into operation requests.
- There are three ways to deploy an application:
 - The management console
 - The CLI
 - The file system deployer
- The deployment process should specify a name, a runtime name and if it is enabled or disabled.
- Using the file system deployer, the deployment process is managed using marker files.

Chapter 4

Configuring JBoss EAP as a Managed Domain

Overview

Goal Configure JBoss EAP as a Managed Domain

Objectives

- Describe managed domain architecture.
- Assign the domain controller and start the managed domain.
- Describe the configuration options for a host controller and make configuration changes to a host controller.
- Describe the configuration options for a domain controller and make configuration changes to the domain controller.

Sections

- Running JBoss EAP as a Managed Domain (and Quiz)
- Assigning a Domain Controller (and Guided Exercise)
- Configuring a Host Controller (and Guided Exercise)
- Configuring a Domain Controller (and Quiz)

Lab

- Configuring JBoss EAP as a Managed Domain

Running JBoss EAP as a Managed Domain

Objectives

After completing this section, students should be able to:

- Describe managed domain architecture.

Managed domain architecture

To understand EAP managed domains, you need to understand the following terms and how they relate to each other:

Domain

A collection of EAP server instances.

Domain controller

A single process that acts as the central management control point for a domain. The domain controller is also referred to as the **master** host controller.

Server

An EAP server instance running in its own JVM process. It runs application code; that is, it acts as an application server.

Host controller

A process running on a host machine that relays configuration information, runtime status, and management commands to EAP server instances on that particular machine. The host controller is also referred to as a **slave**.

Process controller

A process running on a host machine that starts host controllers and server instances on that particular machine.

Host

The set of processes started by the same process controller; a host controller and zero or more application server instances.

Server group

A collection of servers that are managed and configured as one.

Profile

A named set of EAP subsystem configurations.

The following figure helps to visualize the relationships between the terms described above:

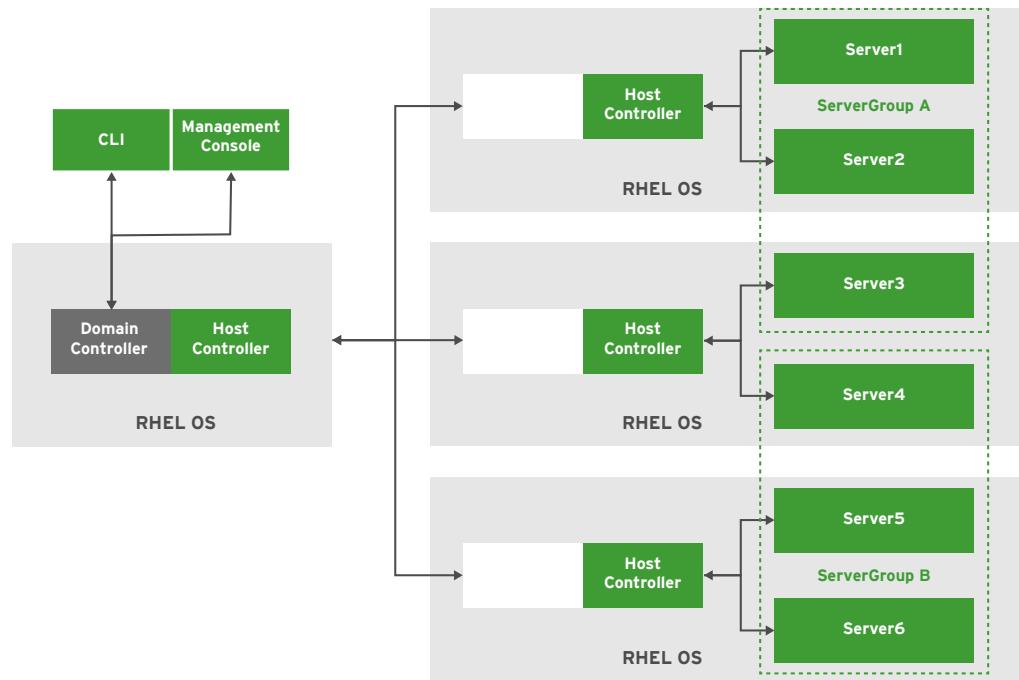


Figure 4.1: EAP managed domain

In the previous figure, the light gray boxes represent machines. They could be either physical or virtual machines, and could run OSes other than Red Hat Enterprise Linux (RHEL). Each machine corresponds to a **host** and runs a **process controller** (not shown in the figure) and a **host controller**. Each machine also runs two **server** instances, but there could be more or less of them.

In a managed domain, one of the host controller instances is configured to act as the central management point, that is, to act as the **domain controller**. In the previous figure, the host managed by the domain controller does not have any server instances. This is a recommended but NOT required approach because a domain controller is also a host controller it could directly manage its own server instances.

Each **host controller** interacts with the **domain controller** to ensure each server instance is configured according to the policies of the domain. All management interfaces, such as the graphical Management Console and the CLI communicate to the domain controller only. The Management Console runs as part of the domain controller OS process and JVM.

Any host controller could be configured as a replacement domain controller in case the original one is unavailable, but this promotion is not automated.

A host controller does not perform application server tasks such as serving up Java EE applications and handling client requests. This is done exclusively by **server** instances. The host controller's sole responsibilities are to interact with the domain controller to help manage the host servers.

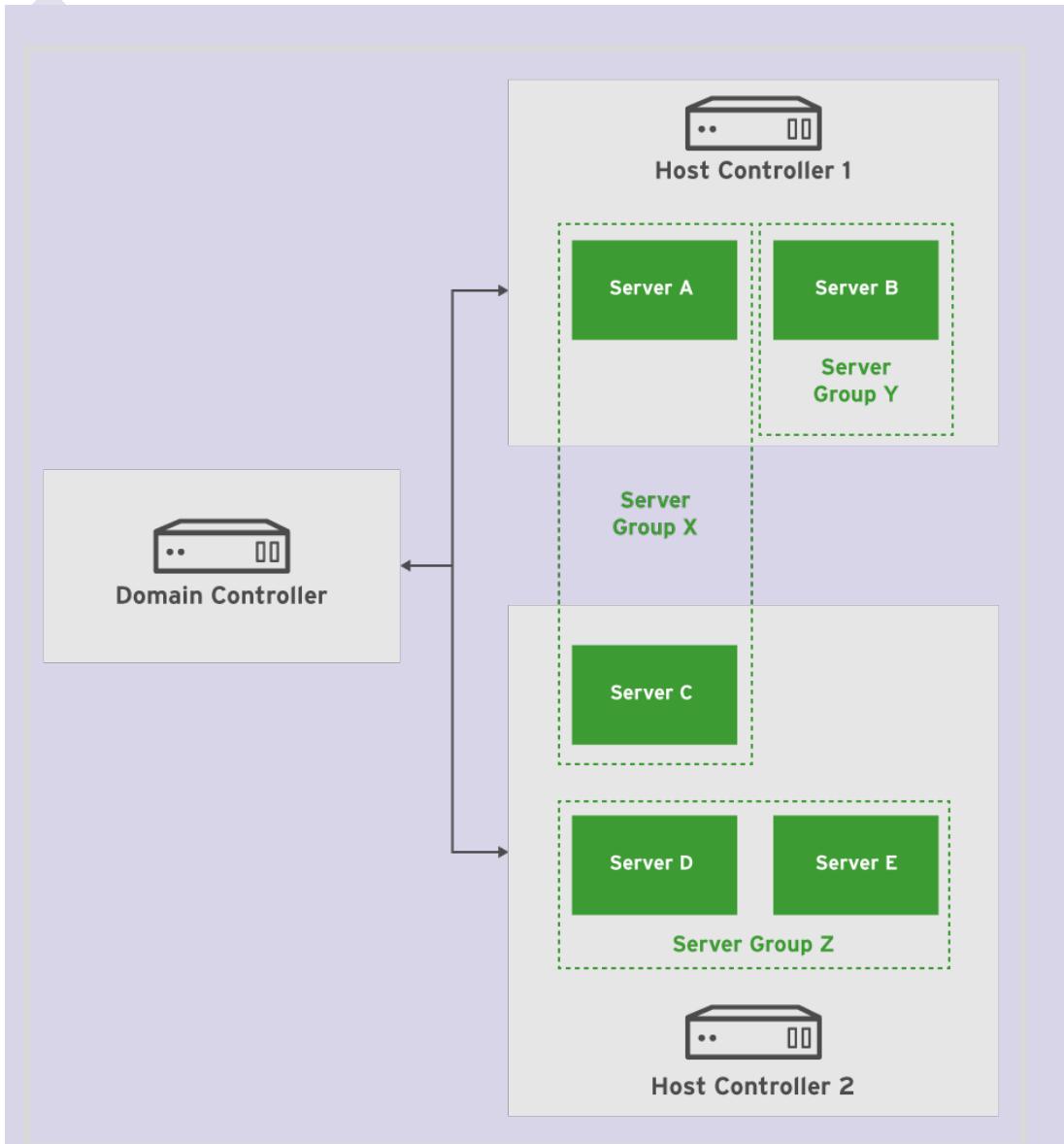
Hosts provide a physical grouping that impacts performance metrics such as available processor cores and RAM (heap). Server groups provide a logical grouping that impacts configuration settings; EAP profiles are associated with server groups, not with individual server instances. Applications are also deployed and activated only to server groups and never to individual server instances in a managed domain.

The EAP managed domain start up script **domain.sh** starts a host process controller, and the process controller starts the host controller. Then the host controller requests that the process

controller start server instances according to the host configuration and later when requested to do so by the domain controller.

► Quiz

Managed domains



Based on the previous diagram, choose the correct answer to the following questions:

► 1. How many host controllers are in the domain? (Choose one)

- a. 5
- b. 3
- c. 2
- d. 1

► **2. How many servers are in the domain? (Choose one.)**

- a. 5
- b. 3
- c. 2
- d. 1

► **3. Which servers are managed by Host Controller 1? (Choose two.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **4. Which servers are managed by Host Controller 2? (Choose three.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **5. Which servers are managed by the domain controller? (Choose one.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E
- f. None of them

► **6. Which servers belong to Server Group Z? (Choose two.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **7. Which servers belong to Server Group X? (Choose two.)**

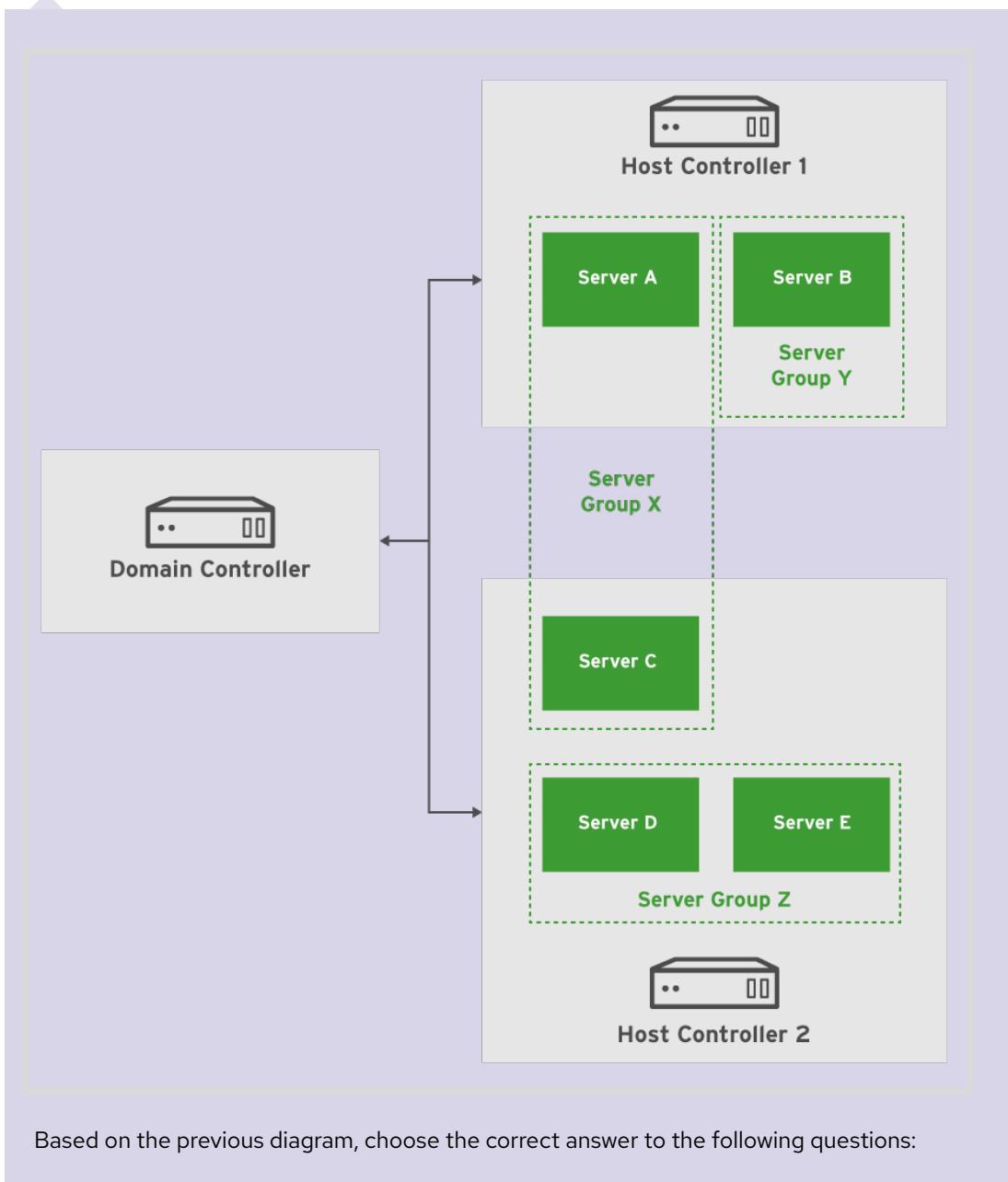
- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **8. Is Server Group Y necessary, or can Server B run outside of a group because there are no other members in the group? (Choose one.)**

- a. Yes, because the server group determines the host where the server instance will be started.
- b. No, standalone server instances can NOT be members of a managed domain server group.
- c. Yes, a server instance can NOT be part of a managed domain without being member of a server group.
- d. No, because the only function of server groups is to ensure that server instances are associated to the same profile, and if Server B is the only one associated to the same profile, a server group is not required.

► Solution

Managed domains



Based on the previous diagram, choose the correct answer to the following questions:

► 1. How many host controllers are in the domain? (Choose one)

- a. 5
- b. 3
- c. 2
- d. 1

► **2. How many servers are in the domain? (Choose one.)**

- a. 5
- b. 3
- c. 2
- d. 1

► **3. Which servers are managed by Host Controller 1? (Choose two.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **4. Which servers are managed by Host Controller 2? (Choose three.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **5. Which servers are managed by the domain controller? (Choose one.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E
- f. None of them

► **6. Which servers belong to Server Group Z? (Choose two.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **7. Which servers belong to Server Group X? (Choose two.)**

- a. Server A
- b. Server B
- c. Server C
- d. Server D
- e. Server E

► **8. Is Server Group Y necessary, or can Server B run outside of a group because there are no other members in the group? (Choose one.)**

- a. Yes, because the server group determines the host where the server instance will be started.
- b. No, standalone server instances can NOT be members of a managed domain server group.
- c. Yes, a server instance can NOT be part of a managed domain without being member of a server group.
- d. No, because the only function of server groups is to ensure that server instances are associated to the same profile, and if Server B is the only one associated to the same profile, a server group is not required.

Assigning a Domain Controller

Objectives

After completing this section, students should be able to:

- Assign the domain controller and start the managed domain.
- Configure slave host controllers to an existing managed domain.

Configuring a host controller as the master

The settings and configuration of a managed domain are split into two files:

- **host.xml**: the configuration file for a host controller. This file needs to be configured for either how to find the domain controller, or it needs to configure itself as the domain controller.

There are other settings in this file, for example server configurations, but they will be presented later in this book. In short, the settings in this file relate to the host hardware and OS specifics.

- **domain.xml**: the configuration file of the domain controller, defining the profiles available and other settings that are not directly influenced by a managed domain host's hardware and OS, such as server groups and socket bindings.

This chapter and the next focus primarily on the **host.xml** configuration file. The **domain.xml** configuration file is covered in later chapters.

A host controller instance is named after its machine host name, but this can be overridden by using the **name** attribute in the **<host>** top-level element, at the beginning of the **host.xml** configuration file. For example:

```
<?xml version="1.0" ?>
<host xmlns="urn:jboss:domain:4.1" name="mydomainmaster">
  ...
```

To denote that a host controller is a domain controller, that is, the master host controller, add the following to its **host.xml** configuration file between the **<management>** and **<interfaces>** elements:

```
<?xml version="1.0" ?>
<host xmlns="urn:jboss:domain:4.1" name="mydomainmaster">
  ...
  </management>

  <domain-controller> ①
    <local/> ②
  </domain-controller>

  <interfaces>
  ...
  </host>
```

- ① The **<domain-controller>** element informs a host controller where to find the domain controller.
- ② If a host controller is supposed to be the **master** for its managed domain, use the **<local/>** element. In other words, the host controller configuration means "the domain controller is the local host controller, that is, the one reading this configuration file".

Besides declaring itself as a master, a domain controller **host.xml** configuration file has to declare a **native management** interface using the **<native-interface>** element. For example:

```
...
</extensions>

<management>
  ...
  <management-interfaces>
    <native-interface security-realm="ManagementRealm">
      <socket interface="management"
        port="${jboss.management.native.port:9999}" />
    </native-interface>
    <http-interface security-realm="ManagementRealm"
      http-upgrade-enabled="true">
      <socket interface="management"
        port="${jboss.management.http.port:9990}" />
    </http-interface>
  </management-interfaces>
</management>

<domain-controller>
  ...

```

The native management interface accepts Management API requests using the EAP native binary protocol. Unlike previous EAP releases, this management interface is NOT required for administrative access, but host controller communication still requires it.

A domain controller is not required to, but it usually defines an **HTTP management** interface as shown by the previous listing. This enables the web Management Console for the domain. If an administrator wishes to, he could remove the HTTP management interface and use only the management CLI to administer the domain.

The native and HTTP management interfaces reference the **management** (network) interface, and so it also has to be declared in the **host.xml** configuration file. For example:

```
...
</domain-controller>

<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}" />
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}" />
  </interface>
</interfaces>
```

```
<jvm>
```

Notice that **host.xml** also defines the **public** network interface. This is used by regular user access to applications. Domain controllers are NOT required to define this interface, unless they have server instances of their own.



Note

Do not confuse **management-interfaces** with (network) **interfaces**. The first refers to remote administrative access. The second refers to segregating different kinds of network traffic to different IP address so they can be subjected to distinct firewall rules. Those configuration elements have the same meaning in managed domain and in standalone server operating modes.

The native management interface also references the **ManagementRealm** security realm. Security in managed domains are presented later in this book.

Configuring host controllers as slaves

To denote that a host controller is not the domain controller, that is, it is supposed to act as a **slave**, specify instead in **host.xml** where the domain controller can be found:

```
...
<domain-controller>
    <remote security-realm="ManagementRealm"> ①
        <discovery-options> ②
            <static-discovery name="primary" ③
                protocol="${jboss.domain.master.protocol:remote}"
                host="${jboss.domain.master.address}"
                port="${jboss.domain.master.port:9999}"/>
        </discovery-options>
    </remote>
</domain-controller>
...
```

- ① The **<remote>** element informs this host controller is a slave, and child elements specify how to find the master.
- ② EAP 7 supports multiple domain controller discovery mechanisms, configured as children of the **<discovery-options>** element. Different combinations of mechanisms allow for implementing fail-over policies when the domain controller is not available and finding the domain controller in highly dynamic cloud environments.
- ③ The **<static-discovery>** mechanism points to the domain controller **native management interface** IP address and TCP port.

In this course only the **<static-discovery>** mechanism is used. For information about other mechanisms, consult the EAP product documentation.

In the configuration above, the **jboss.domain.master.address** system property must be defined when starting the host controller. This is demonstrated later in this section.

An alternative is to modify the **host.xml** configuration file to assign it to the machine name or IP address that the domain controller is running on. For example:

```

<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary"
        protocol="${jboss.domain.master.protocol:remote}"
        host="${jboss.domain.master.address:172.25.14.9}" ①
        port="${jboss.domain.master.port:9999}"/>
    </discovery-options>
  </remote>
</domain-controller>

```

- ① Adding a default value for the `jboss.domain.master.address` system property reference.

The previous example illustrates a common practice with EAP 7 configuration files: to provide attribute values as default values for system properties. This allows overriding the attribute value by defining a system property on the command line.

The following figure shows an example managed domain consisting of three hosts, and shows the critical parts of each host controller `host.xml` configuration file:

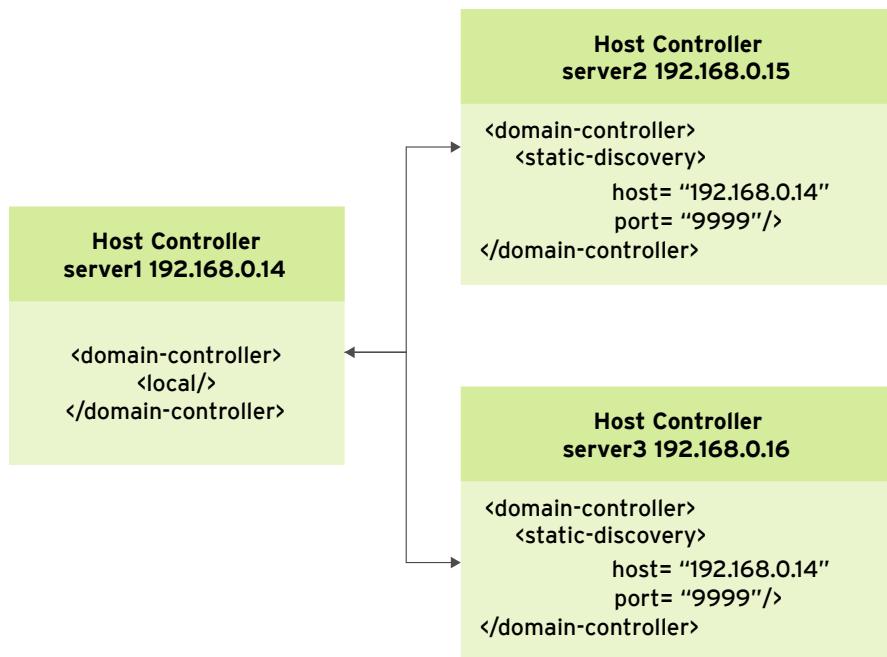


Figure 4.3: Sample managed domain

In the previous figure, the **server1** machine runs the domain controller. The IP address assigned to all domain controller (network) interfaces is **192.168.0.14** and its native management interface is using the default port **9999**. The **server2** and **server3** machines run slave host controllers configured to point to the domain controller running in the **server1** machine.

Starting a managed domain

To start an EAP host controller, use the domain script specific to your platform. The script is found in the `/bin` folder of your EAP installation. On Linux and UNIX, the command is:

```
$ ./domain.sh
```

When starting EAP in managed domain mode, EAP needs to determine if the instance being started is the domain controller or not. Here is the sequence of events that occurs when the domain start-up script runs:

1. The process controller is started in a single JVM process.
2. The process controller starts a host controller in another JVM process.
3. The host configuration file (**domain/configuration/host.xml** if using default settings) is processed first, and the **<domain-controller>** element is checked to determine if this host controller is to act as the domain controller or not.
4. If this particular host controller is configured as a domain controller, then the settings from the domain configuration file (**domain/configuration/domain.xml** if using default settings) are combined with the settings in **host.xml**. A domain controller must expose an addressable management interface binding for the other host controllers to communicate with it.
5. If this particular host controller is not the domain controller, then it must attempt to connect to the domain controller based on the settings in the **<domain-controller>** element in **host.xml**.
6. Whether or not this host controller is the domain controller, it starts up its servers, each in their own JVM process, if the **host.xml** configuration file has any servers defined.

EAP managed domains are secure by default because they only allow accesses from the local machine. To allow for remote access to an EAP domain controller for administrative actions, and also for slave host controllers to join the domain, the **management** interface has to be assigned a different IP address.

Following the example from the previous figure, a domain controller would be started to listen for slave connections on a private IP address by defining the **jboss.bind.address.management** system property. For example:

```
$ ./domain.sh -Djboss.bind.address.management=192.168.0.14
```

The previous example shows a domain controller whose management network interface is assigned the **192.168.0.14** IP address.

Continuing with the same example, a slave host controller would be started to connect to the master in that address, by defining the **jboss.domain.master.address** system property, and also to listen for management commands (from the master) in another private IP address, by defining the **jboss.bind.address.management** system property.

The following example would be used to start the host controller in the **server2** machine:

```
$ ./domain.sh -Djboss.bind.address.management=192.168.0.15 \
-Djboss.domain.master.address=192.168.0.14
```

In the previous example a slave host controller starts and tries to connect to a domain controller running on **192.168.0.14** on port **9999**. Note that the master IP address was specified in the command line using a system property, while the master native management interface TCP port was left as the default value provided by the **host.xml** configuration file.

The previous example also shows the slave accepts management requests from the master on the **192.168.0.15** IP address, specified on the command line using a system property. The slave native management interface TCP port was also left as the default value provided by the **host.xml** configuration file.

For the sake of completeness the following example would be used to start the host controller in the **server3** machine:

```
$ ./domain.sh -Djboss.bind.address.management=192.168.0.16 \
-Djboss.domain.master.address=192.168.0.14
```

Host configuration file names and folders

Most EAP installations need custom configuration files and want to keep the default configuration files unchanged for future reference. This can be done using a few startup host script command-line options:

- **--domain-config** provides an alternative name for the **domain.xml** file.
- **--host-config** provides an alternative name for the **host.xml** file.

For example:

```
$ ./domain.sh --domain-config=mydomain.xml --host-config=myhost.xml
```

The files specified by **--host-config** and **--domain-config** are supposed to exist in the default location which is the **JBOSS_HOME/domain/configuration** folder. This path can be changed by assigning a new value to the **jboss.domain.config.dir** system property.

Creating a custom configuration folder for an EAP host controller normally requires creating custom folders for subsystem data files, logs and other working files. Because of that need EAP also recognizes the **jboss.domain.base.dir** system property. It provides an alternative location for the **JBOSS_HOME/domain/** folder which is expected to have **configuration**, **data**, and **log** folders as children.

For example, to start an EAP host controller using domain configuration files from **/usr/local/eap/configuration**, use the following command:

```
$ ./domain.sh -Djboss.domain.base.dir=/usr/local/eap/configuration
```

You can combine the use of system properties and file name options in a single command:

```
$ ./domain.sh -Djboss.domain.base.dir=/usr/local/eap/configuration \
--domain-config=mydomain.xml --host-config=myhost.xml
```

Recommended practice is to copy the **/domain** folder from a clean installation of EAP 7 before starting EAP in a managed domain. It separates a specific host controller instance configurations, deployments, log files and temporary folders from the default installation folders. There are two key benefits gained from this practice:

1. Multiple instances of EAP in a managed domain, that is multiple host controllers and their associated server instances, can be run on the same machine using the same installation files.
2. It is possible to upgrade EAP without affecting or overwriting already custom configurations.

The **jboss.domain.base.dir** runtime property makes this recommended practice possible.

The default **host.xml** configures the host controller as a master. It also configures sample server groups and server instances so test applications can be deployed right away. It can NOT be used unchanged to start additional (slave) host controllers.

To start additional slave host controllers in the domain, the default **host.xml** configuration file has to be changed to replace the **<local/>** element with a **<remote>** element as shown before.

For convenience, the EAP installation also includes two additional sample host controller configuration files:

- **host-master.xml**: this file configures a minimal domain controller with no server instances. As it defines only the **management** network interface, it can NOT be used as-is when the master host controller is supposed to also contain server instances. Having a master host controller without server instances is a recommended EAP 7 configuration.
- **host-slave.xml**: this file configures a slave host controller with the same sample server groups and server instances as the default **host.xml** configuration file. This configuration file uses the **jboss.domain.master.address** and **jboss.domain.master.port** system properties to provide the master IP address and TCP port.

Recovering from a failed domain controller

An EAP managed domain is a dynamic entity, created by the existence of one or more host controllers running the same domain configuration. Usually the master host controller (that is, the domain controller) is started first, and then slave domain controllers are started. Additional host controllers may join the managed domain at any time.

If the domain controller is not available for any reason, no new host controller may join the managed domain, but existing ones continue to run. Running applications are not affected by domain controller failures; only the administration tasks are affected.

At the time this book was written the domain controller did not have an automatic fail-over mechanism. If a domain controller fails, the host controllers simply keep trying to reconnect until the domain controller comes back up. Note that the server instances are NOT affected if a domain controller fails - they keep running and handling requests! In this manner, a domain controller can fail without affecting the uptime of user applications.

A host controller can keep a cache of the managed domain configuration file so that the host can start if the master is unavailable. Start the slave using the **--backup** option to cause the slave host controller to create and maintain a local copy of the domain configuration file. For example:

```
$ ./domain.sh --backup -Djboss.domain.master.address=192.168.0.14
```

If the slave host controller is unable to contact the master domain controller to get its configuration when it boots, the **--cached-dc** option informs the slave to use a local copy previously created using **--backup**. For example:

```
$ ./domain.sh --cached-dc -Djboss.domain.master.address=192.168.0.14
```

The **--backup** and **--cached-dc** can NOT be used at the same time, so its use as a fail-over mechanism requires manual intervention from the system administrator. Also notice that the **--cached-dc** option will NOT make the slave host controller act as the domain controller: the cached configuration is considered read-only.

The Management Console in a managed domain

The EAP 7 management console is a little different compared to when running from a standalone server instance. Additional tabs and navigation steps are required to deal with the extra elements provided by the managed domain: named profiles, hosts, server groups, and server instances.

The basic navigation is the same, and specific subsystem configuration is also the same. The most visible changes are:

- **Deployments** tab: deployments are organized into server groups, instead of having all applications running in the single server instance.
- **Configuration** tab: a profile must be selected before configuring a subsystem.
- **Runtime** tab: Adds operation to view and configure hosts, server groups, and server instances.

The following figure shows the **Runtime** tab as it appears after starting the EAP 7 as managed domain using the default configuration files:

RED HAT JBOSS® ENTERPRISE APPLICATION PLATFORM

Messages: 0 | admin ▾

Home Deployments Configuration **Runtime** Access Control Patching

Browse Domain By Host (1) Refresh

Hosts master

Hosts

All hosts that are members of this domain.

In a managed domain, multiple server instances are managed from a single control point. A domain can span multiple physical or virtual machines, with all server instances under the control of a host controller. One host controller is configured to act as the central domain controller. All other host controllers interact with the domain controller to control the configuration and lifecycle of the application server instances.

7.0.0.GA ▲ Tools ⚙ Settings

Figure 4.4: Runtime tab from the EAP Management Console in managed domain mode

Similar changes also happen in the management CLI as there are objects representing the additional managed domain configuration elements.

► Guided Exercise

Assigning a Domain Controller

In this lab, you will assign the domain controller and start it from the workstation VM.

Resources	
Files:	/home/student/JB248/labs/domain
Application URL:	http://172.25.254.250:9990

Outcomes

You should be able to start a domain controller from the workstation VM.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to download the files for the exercise:

```
[student@workstation ~]$ lab assign-controller setup
```

► 1. Create a New Domain Base Directory in the Workstation VM

Copy the folder **/opt/jboss-eap-7.0/domain** into a new folder **machine1** located in the lab directory **/home/student/JB248/labs/domain/**. This will create a folder **machine1/domain**, which should contain three subfolders: **configuration**, **data**, and **tmp**:

```
[student@workstation ~]$ cd /home/student/JB248/labs/domain
[student@workstation domain]$ mkdir machine1
[student@workstation domain]$ cp -r /opt/jboss-eap-7.0/domain machine1/domain
```



Note

Eventually you are going to have a Domain Controller and two Host Controllers all running on your student workstation. In reality, you probably would run these three controllers on separate machines, so we are going to simulate separate machines by using subfolders named **machine1**, **machine2**, and **machine3**.

In this lab, you are going to configure **machine1** to run as the master controller. In the next lab, you will create and configure **machine2** and **machine3** as slaves connecting to **machine1**.

► 2. Configure Files in machine1 for a Domain Controller

- 2.1. Using the editor of your choice, open the **host-master.xml** file in the **/home/student/JB248/labs/domain/machine1/domain/configuration** folder.

This host configuration file configures a domain controller that does not manage any local servers.

- 2.2. Look at the following line at the beginning of the **host-master.xml** file:

```
<host xmlns="urn:jboss:domain:4.1" name="master">
```

Observe that this line configures the name of this host to be "**master**".

- 2.3. Notice there is only one interface defined in **host-master.xml**, named **management**.

```
...
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}" />
    </interface>
</interfaces>
...
```

It is assumed that the host machine running the domain controller is not hosting servers; consequently it does not need to define a **public** network interface for the server instances to accept user requests.

- 2.4. Any slaves must be configured to point to the IP address of the domain controller.
- 2.5. The labs are going to simulate multiple machines, and binding to 127.0.0.1 is not going to make the domain controller on the **machine1** folder visible to outside machines.

We could specify the **jboss.bind.address.management** property for the startup script, but instead you will manually edit the XML file. Modify the **management** interface's **inet-address** to bind to the IP address of your workstation machine (172.25.250.254). The **<interfaces>** section of **host-master.xml** should appear as follows:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:172.25.250.254}" />
    </interface>
</interfaces>
```

- 2.6. Save your changes to **host-master.xml** and close the text editor.
- 2.7. Open the **domain.xml** in **/home/student/JB248/labs/domain/machine1/domain/configuration**.
- 2.8. Inside of the **messaging-activemq** subsystem of the **full-ha** profile, edit the **<cluster>** tag (line 1278):

```
<cluster password="${jboss.messaging.cluster.password:JBoss@RedHat123}" />
```

**Note**

Clustering is described in more detail later on in this book. Right now this setting is required to avoid error messages in the terminal.

- 2.9. Save your changes to **domain.xml**.

► **3. Start the Domain Controller on the Workstation VM**

- 3.1. In your terminal window, change to **/opt/jboss-eap-7.0/bin**:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin
```

- 3.2. To start the **master** host controller using the **host-master.xml** file in your **/home/student/JB248/labs/domain/machine1/domain/** folder, enter the following command:

```
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \
--host-config=host-master.xml
```

Because you are starting a domain controller with no application server instances attached, it should start quickly. Look for the following output to confirm that the domain controller is running:

```
[Host Controller] 01:50:11,359 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0060: Http management interface listening on http://172.25.250.254:9990/
management
[Host Controller] 01:50:11,359 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0051: Admin console listening on http://172.25.250.254:9990
```

**Note**

This host controller instance is NOT using the default **host.xml** configuration file, so the **--host-config** option is necessary. The **--domain-config** option is not needed because it IS using the default **domain.xml** managed domain configuration file.

► **4. Verify the Domain Controller is Running**

- 4.1. Open a web browser in the workstation VM and point the browser to <http://172.25.250.254:9990/> to access the EAP Management Console from the domain controller.
- 4.2. You will be prompted for credentials. The administrator user name is **jbossadm** and the password is **JBoss@RedHat123**. These values were inherited from the EAP installation process in Chapter 1.

► 5. Navigating the Management Console in Domain Mode

- 5.1. The Management Console is a little different in a Managed Domain compared to the Management Console when running a standalone server. Under the **Runtime** section, there is the option to browse the Domain by **Hosts** or **Server Groups**.
- 5.2. Click the **Hosts** category under **Browse Domain By** and click **master** under the **Host** column. Note that because there are no servers running (you have not defined and started any yet), the **Runtime** page does not have any useful information to display.
- 5.3. Click the **Configuration** page. You saw some of the details of profiles when you had a standalone server running. Notice this page looks similar to when EAP is run as a standalone server:

The screenshot shows the Red Hat JBoss Enterprise Application Platform 7.0.0.Beta1 Management Console. The top navigation bar includes 'RED HAT JBOSS' ENTERPRISE APPLICATION PLATFORM 7.0.0.Beta1', 'Messages: 0 Red Hat Access', and a user icon 'jbossadm'. Below the navigation bar, the main menu has tabs: Home, Deployments, Configuration (which is selected and highlighted in blue), Runtime, Access Control, and Patching. The 'Configuration' section on the left has a tree view with 'Profiles' expanded, showing four profiles: default, full, full-ha, and ha. To the right of the tree view, a detailed description of 'Profiles' is provided:

Profiles
A profile is a named set of subsystem configurations. A subsystem is a set of capabilities added to the core server by an extension. Subsystems provide capabilities like servlet handling capabilities, an EJB container, JTA, etc.

A profile is a named list of subsystems, along with the details of each subsystem's configuration. A profile with a large number of subsystems results in a server with a large set of capabilities. A profile with a small, focused set of subsystems will have fewer capabilities but a smaller footprint.

- 5.4. Click on the option **Profile**: to see all of the profiles defined in your **domain.xml** configuration file.



Note

Recall that in a managed domain you can have multiple profiles defined, and each profile has a unique name. Contrast this to running EAP as a standalone server, where there is a single anonymous profile.

► 6. Clean Up

- 6.1. Stop the domain controller by pressing **Ctrl+C** in the terminal window running the domain controller.

This concludes the guided exercise.

Configuring a Host Controller

Objectives

After completing this section, students should be able to:

- Describe the configuration options for a host controller and make configuration changes to a host controller.

General host controller settings

A host controller is configured in **JBoss_HOME/domain/configuration/host.xml** file. Parts of this file were already presented by this book but now it is time to start discussing its structure.

A **host.xml** file has the following general structure, as defined by the XML schema in **JBoss_HOME/docs/schema/wildfly-config_4_1.xsd**:

```
<host name="my_hostname" xmlns="urn:jboss:domain:4.1">
    <extensions>
        ... host controller extensions
    </extensions>
    <system-properties>
        ... for defining system properties
    </system-properties>
    <paths>
        ... for defining filesystem paths
    </paths>
    <vault>
        ... for storing encrypted passwords
    </vault>
    <management>
        ... the management interfaces and their security settings appear here
    </management>
    <domain-controller>
        ... the settings for how to connect to the Domain Controller
    </domain-controller>
    <interfaces>
        ... interfaces are defined here
    </interfaces>
    <jvms>
        ... JVMs are defined here
    </jvms>
    <servers>
        ... servers are defined here
    </servers>
    <profile>
        ... subsystems configurations for host controller extensions
    </profile>
</host>
```

Most of the elements are optional and may not be seen in the out-of-the-box host controller configuration files. The only mandatory elements are: **<management>**, **<domain-controller>** and the top-level element **<host>**.

The following are very high-level explanations about each element. Their use and syntax are explained in more detail later in this book.

<extension>

Declares EAP extensions that are to be loaded by the host controller itself, instead of by server instances.

<system-properties>

Defines system properties for this specific host. A system property defined here overrides the same system property defined in **domain.xml** (if the property is defined in both files) but NOT by a system property defined on the command line.

<paths>

Defines the actual file system paths used by the EAP subsystem configuration. This allows each host controller to map those paths to different physical locations.

<vault>

Defines a security vault, a place for storing encrypted passwords.

<management>

Defines the management interfaces available to administrators, alongside their security and audit settings.

<interfaces>

Defines the actual IP addresses different services and subsystems will bind to.

<domain-controller>

Specifies either where this host controller can locate the domain controller or that it is the domain controller.

<jvms>

Defines multiple Java Virtual Machine configurations that can be referenced by different server groups and also by different individual server instances.

<servers>

Defines the server instances on this host controller. Each server definition refers to a **server group** which is defined by the **domain.xml** configuration file.

<profile>

Specifies configurations for subsystems provided by **<extension>** declared earlier.

Elements found in **standalone.xml** for standalone server operating mode usually have the same meaning and syntax in **host.xml** for managed domain operating mode. Some elements, such as **<system-properties>** and **<jvms>**, can be configured in different places in **host.xml** and **domain.xml**. Sometimes this can occur multiple times in the same file in a way that may seem redundant. This is allowed because it provides for configuring domain-wide defaults that can be overridden at the host, server group, and/or server instance levels.

A host configuration usually defines multiple servers, inside the **<server>** element. In this case, each server instance probably requires a different port offset so there are no network port conflicts.

Exploring a host controller configuration

Server and JVM configurations are the same for master and slave host controllers, even if it is common for a master to have none.

The following is a partial, sample **host.xml** configuration file. The following text explains the server instance and JVMs settings:

```
<host name="myhost" ❶ xmlns="urn:jboss:domain:4.1">
    ...
    <jvms> ❷
        <jvm name="default">
            <heap size="64m" max-size="128m"/>
        </jvm>
        <jvm name="myhost-jvm">
            <heap size="512m" max-size="1024m"/>
        </jvm>
    </jvms>
    <servers> ❸
        <server name="server1" group="group1"> ❹
            <jvm name="default"/> ❺
        </server>
        <server name="server2" group="group2"> ❻
            <jvm name="myhost-jvm"/> ❼
            <socket-bindings port-offset="200"/> ❽
        </server>
    </servers>
</host>
```

- ❶ The name of this host controller is **myhost**. If the **name** attribute were not specified, the machine host name would be used instead.
- ❷ Two JVMs are defined, named **default** and **myhost-jvm**. Those JVM names also have to be defined by **domain.xml** because they could be referred to by server group definitions instead of by individual server instances as in this sample.
- ❸ Two server instances are defined, named **server1** and **server2**.
- ❹ The **server1** server instance is a member of the **group1** server group which is defined by **domain.xml**.
- ❺ The **server1** server instance uses the **default** JVM definition and so it gets a very small heap, with just a 128MB maximum size.
- ❻ The **server2** server instance is a member of the **group2** server group which is defined by **domain.xml**.
- ❼ The **server2** server instance uses the **myhost-jvm** JVM definition and so it gets a bigger heap, with a 1024 MB (1 GB) maximum size.
- ❽ The **server2** server instance is configured with a port offset of 200, so its HTTP port will be 8280. The **server1** server instance has no port offset configured and so will get the default 8080 HTTP port.

The previous sample host controller configuration defines two server instances, and thus the host will run four OS processes, each one with its own JVM: the process controller, the host controller, and the two server instances.

Slave host controller configuration

The essential parts for a slave host controller configuration were already presented by this book, but a complete walk through of a slave configuration has not yet been performed. The following walk through highlights the differences between master and slave host controller settings:

As we already know, a host controller instance is named after its machine host name but this can be overridden by using the **name** attribute in the **<host>** top-level element, at the beginning of the **host.xml** configuration file. For example:

```
<?xml version="1.0" ?>
<host xmlns="urn:jboss:domain:4.1" name="mydomainslave1">
    ...

```

To denote that a host controller is a slave, specify where the master can be found:

```
...
<domain-controller>
    <remote security-realm="ManagementRealm">
        <discovery-options>
            <static-discovery name="primary"
                protocol="${jboss.domain.master.protocol:remote}"
                host="${jboss.domain.master.address}"
                port="${jboss.domain.master.port:9999}"/>
        </discovery-options>
    </remote>
</domain-controller>
...
```

Details about the previous settings were already explained; review the previous section for more information.

A slave host controller has to declare a **native management** interface the same way a master has to. For example:

```
...
</extensions>

<management>
    ...
    <management-interfaces>
        <native-interface security-realm="ManagementRealm">
            <socket interface="management"
                port="${jboss.management.native.port:9999}"/>
        </native-interface>
    </management-interfaces>
</management>

<domain-controller>
    ...

```

Remember that all administrative actions are performed on the domain controller. It therefore serves no purpose to define an HTTP management interface for a slave host controller.

The native management interface for a slave host controller references the **management** (network) interface, again in the same way as a master does:

```
...
</domain-controller>

<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}"/>
    </interface>
</interfaces>

<jvm>
...

```

For a slave host controller, the **public** network interface IS required, as it is supposed to have server instances.

Only the domain controller should contact a slave native management interface. It is recommended to enforce this restriction using firewall rules, and that is the main reason the out-of-the-box EAP slave host controller configuration defines the **management** network interface.

Starting a slave host controller

At least two system properties have to be defined to start a slave host controller:

- **jboss.domain.master.address**: provides the IP address of the domain controller.
- **jboss.bind.address.management**: provides the slave with a non-loopback IP address. This is required because the domain controller also connects back to the slave.

Either those variables are defined in the **domain.sh** script command line or their values are inserted directly in the slave **host.xml** configuration file.

Assuming the master management interface IP address is **192.168.0.14** and the slave management interface IP address is **192.168.1.25**, the following command starts the host controller as a slave:

```
$ ./domain.sh -Djboss.bind.address.management=192.168.1.25 \
-Djboss.domain.master.address=192.168.0.14
```

The previous example assumes the native management interface TCP port was left as the default value **9999** for both the master and the slave.

Configuring multiple masters

New to EAP 7 is the ability to define multiple masters in the same slave host controller configuration. For example:

```
<domain-controller>
    <remote security-realm="ManagementRealm">
        <discovery-options>
```

```

<static-discovery name="primary" ①
    protocol="${jboss.domain.master.protocol:remote}"
    host="172.16.81.100" port="${jboss.domain.master.port:9999}"
<static-discovery name="backup" ②
    protocol="${jboss.domain.master.protocol:remote}"
    host="172.16.81.101" port="${jboss.domain.master.port:9999}"/>
</discovery-options>
</remote>
</domain-controller>
```

- ①** This points to a regular domain controller IP address.
- ②** This is a slave host controller supposed to be started with the **--backup** option.

The slave host controller using the previous example configuration will try to connect to each domain controller in the order they appear, and will register to the first one that replies. The names **primary** and **backup** have no special meaning.

If the **primary** domain controller is unavailable, the **backup** domain controller has to be manually stopped and reconfigured to be a domain controller. If it was last started with the **--backup** option it should already have a complete, current **domain.xml** configuration file and can take over domain controller duties.

The promotion of a slave host controller to the master role is still a manual process for EAP 7.0.0. The sample configuration above just shows a way to NOT require changing each other slave host controller configuration to point to the new master.

Authenticating with a domain controller

A host controller connected to a domain controller has access to all configuration used by all profiles. Nonetheless, an external host controller may connect to a managed domain and get these configurations from the domain controller if no authentication is required. To avoid this scenario, EAP requires that a host controller must authenticate to the domain controller to be part of the managed domain. This authentication requires:

- *A management user in the domain controller:* Created during the installation process or later, running the **JBOSS_HOME/bin/add-user.sh** script in the domain controller.
- *An XML element with a encrypted password in the host controller configuration file:* Provided by the **JBOSS_HOME/bin/add-user.sh** script, it will be updated on each host controller configuration file (**host-slave.xml**). The XML element is called **secret** and the value attribute is an encrypted password obtained from the **add-user.sh** script.
- *A username in the host controller configuration file:* Added using the **JBOSS_HOME/bin/add-user.sh** in the host controller, it must be added to the **remote** element in the host controller configuration file (**host-slave.xml**).

In the **host-slave.xml** file from the host controller, the following XML contains the encrypted password:

```
<secret value="c2xhdmVfdXNlc19wYXNzd29yZA=="/>
```

Additionally, the **host-slave.xml** file must have the user name added to the domain controller by changing the following XML excerpt:

```
<remote username="jbossadm" security-realm="ManagementRealm">
```

► Guided Exercise

Configuring Host Controllers

In this lab, you will create two new host controllers to be slaves to the previously created domain controller.

Resources	
Files:	/home/student/JB248/labs/host
Application URL:	http://172.25.250.254:8230/ http://172.25.250.254:9080/ http://172.25.250.254:8080/

Outcomes

You should be able to start two host controllers that are slaves to a domain controller.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and the previous guided exercise set up a domain controller at **/home/student/JB248/labs/domain**:

```
[student@workstation ~]$ lab configure-host setup
```

► 1. Create Additional Base Directories For Two New Host Controllers

- 1.1. In your **/home/student/JB248/labs/host** folder, create two new folders: **machine2** and **machine3**.

```
[student@workstation ~]$ cd /home/student/JB248/labs/host  
[student@workstation host]$ mkdir machine2  
[student@workstation host]$ mkdir machine3
```

- 1.2. Copy the folder **opt/jboss-eap-7.0/domain** into both the **machine2** folder and the **machine3** folder. These two folders symbolize two separate machines that will connect to your domain controller.

```
[student@workstation host]$ cp -R /opt/jboss-eap-7.0/domain machine2/domain  
[student@workstation host]$ cp -R /opt/jboss-eap-7.0/domain machine3/domain
```

► 2. Configure Distinct Host Names

- 2.1. Using a text editor, open the **host-slave.xml** file in the **/home/student/JB248/labs/host/machine2/domain/configuration** folder.

- 2.2. Notice this host does not have a name. Each host in a domain needs a distinct name, so assign value “**host2**” to the **name** attribute, as follows:

```
<host name="host2" xmlns="urn:jboss:domain:4.1">
```

► 3. Specify the IP Address of the Domain Controller

- 3.1. Notice that **host-slave.xml** is configured to be a slave based on the following **domain-controller** setting that includes the **<remote>** element:

```
<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary"
        protocol="${jboss.domain.master.protocol:remote}"
        host="${jboss.domain.master.address}"
        port="${jboss.domain.master.port:9999}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

The system property **jboss.domain.master.address** passes in the IP address of the domain controller when you start **host2** and **host3** later in this guided exercise.

► 4. Avoid Port Conflicts From Multiple Host Controllers

- 4.1. To avoid port conflicts, different ports must be set for each host controller running on the workstation VM.

Because the domain controller from the previous lab is already bound to port **9999**, you need to change the native interface's port number for **host2**. In the **<native-interface>** section, change the default port to **29999**:

```
<native-interface security-realm="ManagementRealm">
  <socket interface="management"
    port="${jboss.management.native.port:29999}"/>
</native-interface>
```

- 4.2. Notice in all **<interfaces>** elements that the IP address is assigned to **127.0.0.1**, the loopback interface by default.

This will not work for **jboss.bind.address.management** in a multi-machine environment, because the master will not be able to connect to this host. Neither will it not work for **jboss.bind.address** if you want your servers to be accessible to the outside world.

Replace the loopback IP address **127.0.0.1** in both of the default values with the IP address of the workstation VM, **172.25.250.254**:

```
<interface name="management">
  <inet-address value="${jboss.bind.address.management:172.25.250.254}"/>
</interface>
<interface name="public">
  <inet-address value="${jboss.bind.address:172.25.250.254}"/>
</interface>
```

► 5. Observe the Host's Servers

- 5.1. In the **host-slave.xml** file for **host2**, notice that there are already two servers defined, **server-one** and **server-two**:

```
<server name="server-one" group="main-server-group"/>
<server name="server-two" group="other-server-group">
  <socket-bindings port-offset="150"/>
</server>
```

server-one belongs to the server group named **main-server-group** and **server-two** belongs to the server group **other-server-group**.

Also note that **server-two** uses a port offset of **150** to avoid conflicts between the two servers.

Save the changes to **host-slave.xml** and close your text editor.

► 6. Start Domain Controller

- 6.1. Before running the host controller, start the domain controller created in the previous exercise on **machine1** using a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \
--host-config=host-master.xml
```

- 6.2. Start **host2** using the **host-slave.xml** configuration file that has its management interface bind to 172.25.250.254 on port **29999**.

Run the following command from your **/opt/jboss-eap-7.0/bin** folder in the original terminal window:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine2/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254
```



Note

Notice that the prefix of each log entry in the terminal window is either [HostController] or the name of the server that caused the log event, which is either [Server:server-one] or [Server:server-two] in your deployment.

- 6.3. Inspect the terminal window of the host controller of machine2. Carefully review the log output and you should see the host controller connecting to the master, and also messages indicating that server-one and server-two are starting.

```
[Host Controller] 16:42:57,307 INFO [org.jboss.as.host.controller]
(Controller Boot Thread) WFLYHC0148: Connected to master host controller at
remote://172.25.250.254:9999
[Host Controller] 16:42:57,367 INFO [org.jboss.as.host.controller] (Controller
Boot Thread) WFLYHC0023: Starting server server-one
```

- 6.4. Look in the terminal window of the domain controller. You should see a log entry showing the slave connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host
Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
"host2", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

► 7. Verify host2 is in the Domain

- 7.1. In your browser, navigate to `http://172.25.250.254:9990/`, which is the URL for the Domain Controller's management tool. The administrator user name is **jbossadm** and the password is **JBoss@RedHat123**.
- 7.2. Verify that **server-one** and **server-two** are displayed in the list on the **Runtime** page under the **host2** host.

The screenshot shows the 'Browse Domain By' interface. On the left, there are two main categories: 'Hosts' and 'Server Groups'. Under 'Hosts', 'host2' is selected, and its details are shown on the right. Under 'Server Groups', 'master' is selected, and its details are also shown. The 'host2' section includes a 'JVM' dropdown menu. The 'master' section includes a search bar and two entries: 'server-one (main-server-group)' and 'server-two (other-server-group)'.

► 8. Configure the host3 Host Controller

- 8.1. Using a text editor, open the file **host-slave.xml** in your `/home/student/JB248/labs/host/machine3/domain/configuration` folder. Be sure to leave the domain controller and other host controller running.
- 8.2. Add a name attribute to the **<host>** element and assign it the value **host3**:

```
<host name="host3" xmlns="urn:jboss:domain:4.1">
```

- 8.3. Change the native management interface port to **39999**:

```
<native-interface security-realm="ManagementRealm">
    <socket interface="management"
        port="${jboss.management.native.port:39999}" />
</native-interface>
```

- 8.4. In the **<interfaces>** section, replace **127.0.0.1** in the default value of **jboss.bind.address.management** and **jboss.bind.address** with the IP address of the workstation VM, 172.25.250.254:

```
<interface name="management">
  <inet-address value="${jboss.bind.address.management:172.25.250.254}" />
</interface>
<interface name="public">
  <inet-address value="${jboss.bind.address:172.25.250.254}" />
</interface>
```

- 8.5. Delete **server-one** from the **<servers>** section.
- 8.6. Rename **server-two** to **server-three** and change its **port-offset** to be **1000**. Your **<servers>** section should appear as follows:

```
<servers>
  <server name="server-three" group="other-server-group">
    <socket-bindings port-offset="1000" />
  </server>
</servers>
```

- 8.7. Save your changes to **host-slave.xml** and exit your text editor.

▶ 9. Start host3

- 9.1. Open a new terminal window and run the following commands to start **host3** as a slave in your domain:

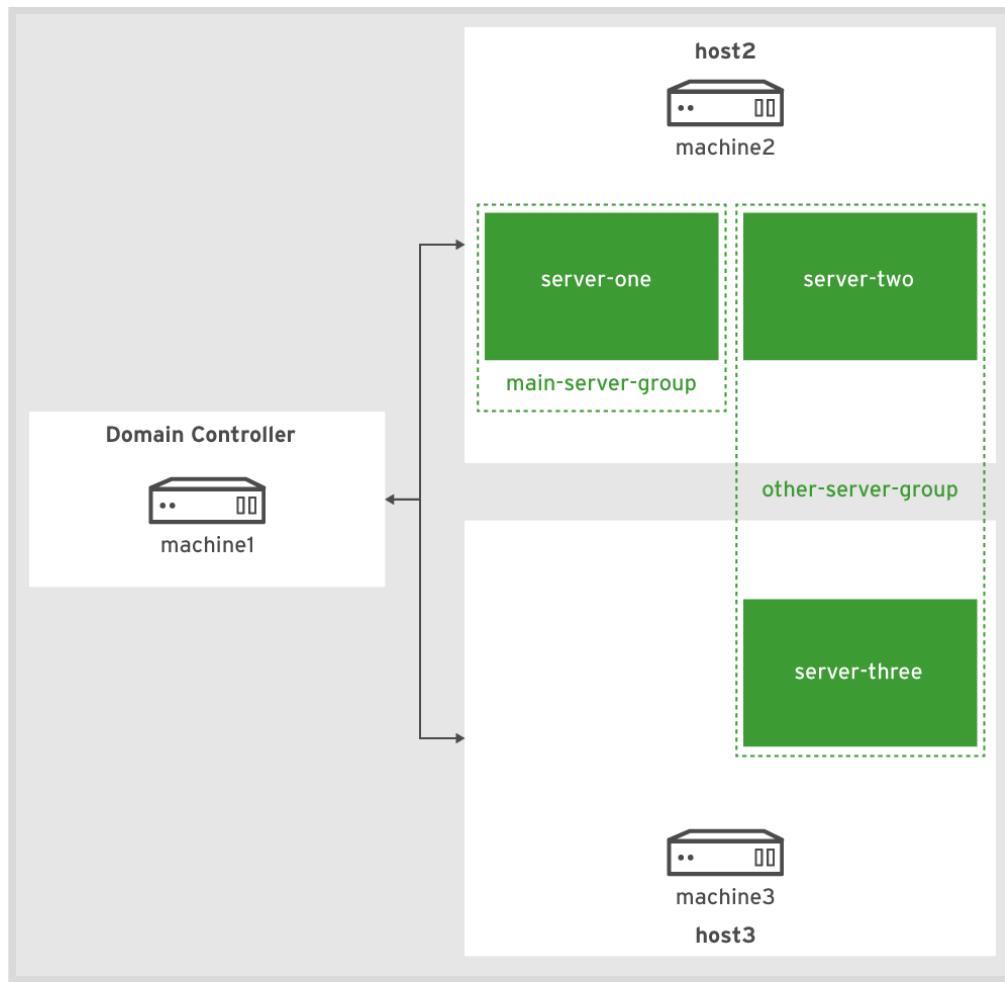
```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine3/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254
```

- 9.2. Look in the terminal window of the domain controller. You should see a log entry showing the slave connecting:

```
[Host Controller] 01:34:29,845 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 35) WFLYHC0019: Registered remote slave host "host3", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

▶ 10. Verify host3 is in the Domain

- 10.1. Go back to your management console and refresh the **Runtime** page.
- 10.2. Verify that the **server-three** is running on **host3**.
- 10.3. The following diagram shows what your domain now looks like:



Domain Servers

Server	Address	Port-Offset
server-one	172.25.250.254:8080	0
server-two	172.25.250.254:8230	150
server-three	172.25.250.254:9080	1000

► 11. Verify the Servers are Running

- 11.1. On the workstation VM, open a web browser and point to <http://172.25.250.254:8080/>. You should see the default EAP Welcome page that is being served by **server-one**.
- 11.2. Point your web browser to <http://172.25.250.254:8230/>. Again, you should see the default EAP Welcome page, but this time the page is being served by **server-two**.
- 11.3. Point your web browser to <http://172.25.250.254:9080/>. Again, you should see the default EAP Welcome page, but this time the page is being served by **server-three**.

► 12. Stop a Host Controller

- 12.1. Press **Ctrl+C** in the terminal window of **host2**, which will start the shutdown of **host2**.
- 12.2. Watch the output in the terminal window and notice that the **server-one** and **server-two** processes are stopped, followed by the host controller process.

```
19:47:21,305 INFO [org.jboss.as.process.Host Controller.status] (reaper for Host Controller) WFLYPC0011: Process 'Host Controller' finished with an exit status of 130
19:47:21,305 INFO [org.jboss.as.process] (Shutdown thread) WFLYPC0016: All processes finished; exiting
```

- 12.3. Look in the terminal window of the domain controller. You should see a log event similar to the following, stating that **host2** has been removed from the managed domain:

```
[Host Controller] 01:42:49,331 INFO [org.jboss.as.domain.controller] (management task-7) WFLYHC0026: Unregistered remote slave host "host2"
```

- 12.4. Start **host2** again. After it is started, you should see a log event in the **machine1** terminal window showing **host2** being registered again with the Domain.

```
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine2/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254
```

► 13. Stop the Domain Controller

- 13.1. Press **Ctrl+C** in the terminal window of **machine1** to shutdown the domain controller.
- 13.2. Refresh the web page at <http://172.25.250.254:8080/>. The page should display properly, even though the **master** host controller is no longer running.
- 13.3. Start the domain controller again:

```
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \
--host-config=host-master.xml
```

- 13.4. Watch the output in the terminal window and wait. You should see **host2** and **host3** reconnect to the master after a few seconds.

► 14. Clean Up

- 14.1. Use **Ctrl+C** to stop the domain controller and the two host controllers.

This concludes the guided exercise.

Configuring a Domain Controller

Objectives

After completing this section, students should be able to:

- Describe the configuration options for a domain controller and make configuration changes to the domain controller.

Domain controller settings

The domain controller settings are found in two locations: **host.xml** and **domain.xml**. The settings from both files are combined to configure and create the domain controller. The **domain.xml** file has almost the same structure as the **standalone.xml** file. The differences are described below:

- **standalone.xml** can only define a single profile. **domain.xml** can define any number of different profiles.
- **domain.xml** contains a **<server-groups>** section for defining a common setup for a group of EAP servers and mapping them to a profile. Because a standalone server is composed of a single server instance, the concept of server groups is nonsensical.

The **domain.xml** file is structured as follows:

```
<domain xmlns="urn:jboss:domain:4.1">
  <extensions>
    ... domain controller extensions
  </extensions>
  <system-properties>
    ... for defining system properties
  </system-properties>
  <paths>
    ... for defining filesystem paths
  </paths>
  <management>
    ... the management interfaces and their security settings appear here
  </management>
  <profiles>
    <profile name="profile_name">
      <subsystem xmlns="subsystem_xmlns">
        ... Configuration from subsystems
      </subsystem>
    </profile>
  </profiles>
  <interfaces>
    ... interfaces are defined here
  </interfaces>
  <socket-binding-groups>
    ... Socket binding groups configuration are defined here
  </socket-binding-groups>
```

```
<deployments>
    ... deployments are defined here
</deployments>
<server-groups>
    ... server groups are defined here
</server-groups>
</domain>
```

Notice that profiles are defined within the `<profiles>` section, and each `<profile>` must have a unique name.

The `<server-groups>` section is for defining and grouping servers, something that is not possible in standalone mode. Servers and server groups will be discussed later in this course.

A Comparison of domain.xml and host.xml

Both `domain.xml` and `host.xml` use a range of XML tags to specify their configuration options. Some of these tags appear in both files. The following table describes which tags appear in which file.-

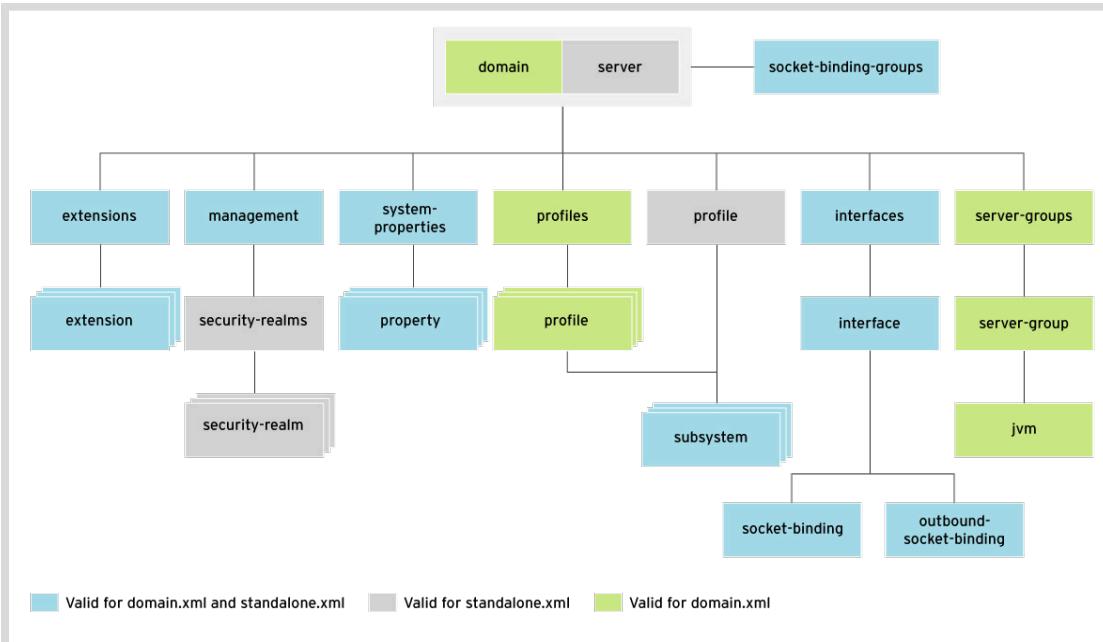
domain.xml or host.xml

Configuration Name	domain.xml	host.xml
<code><extensions></code>	x	x
<code><system-properties></code>	x	x
<code><paths></code>	x	x
<code><management></code>	x	x
<code><profiles></code>	x	x
<code><interfaces></code>	x	x
<code><domain-controller></code>		x
<code><jvms></code>		x
<code><servers></code>		x
<code><socket-binding-groups></code>	x	
<code><server-groups></code>	x	
<code><deployments></code>	x	



Note

The settings in `host.xml` are combined with the settings in `domain.xml` to configure the host controller. If the same setting appears in both files, the value in `host.xml` takes precedence.

**Figure 4.8: XML structure comparison**

Deployed applications of a domain are configured in the domain controller's **domain.xml** file. This does not mean every host and server in the domain has to deploy every application. An application is not deployed onto an entire domain, but instead onto a server group within the domain. Adding an Enterprise Application Archive (EAR file) or a Web Application Archive (WAR file) in the **<deployments>** section simply makes that application available for deployment onto a server group.

A **<path>** defined in **domain.xml** does not actually have a value, only a name. The actual value is defined in **host.xml**. This allows variable to be defined at the domain level, but allows each host have a unique value for that variable. This is extremely useful because hosts might not be running the same operating systems or may have files deployed in entirely different locations.

An **<interface>** definition is similar to a **<path>** in **domain.xml**. When configuring the interfaces of a domain, the domain controller does not need to know the specific addresses of each host, so in **domain.xml** the **<interface>** element acts as a placeholder that is overridden in the **host.xml** file of the host controller.

For example, only the name of the interface is defined on **domain.xml**:

```
<interface name="my_interface"/>
```

Continuing the example, the actual IP address or device name for the interface is specified on **host.xml**:

```
<interface name="my_interface">
  <nic name="eth1"/>
</interface>
```

Each server is managed by the host controller and not by the domain controller and each server is declared as part of the **host.xml** configuration file.

Notice that **<server-groups>** is defined at the domain level, but **<servers>** is defined at the host level. This means that the **domain.xml** and **host.xml** files need to be compatible with each other, because a **<server>** definition in **host.xml** must refer to a **<server-group>** definition in **domain.xml**. If a **<server>** definition in **host.xml** refers to a **<server-group>** that is not defined in **domain.xml**, then the host controller will not start.

Notice also that **<deployments>** is defined at the domain level, not the host level. It is not possible to deploy applications onto servers. They should be deployed onto server groups.

In EAP 7 it is possible to define **<extensions>** in the **host.xml** file to support subsystems running on specific hosts.

Configuring a managed domain with CLI

The way to navigate in domain mode with the CLI is different from standalone mode. New resources are available at the top level:

```
[domain@172.25.250.254:9990 /] cd
```

Using tab completion, the following levels are returned:

-help	extension	profile
--no-validation	host	server-group
core-service	interface	socket-binding-group
deployment	management-client-content	system-property
deployment-overlay	path	

The first difference is the **host** level. This level list all host controllers managed by the domain:

```
[domain@172.25.250.254:9990 /] cd host  
[domain@172.25.250.254:9990 host] ls  
host2 host3 master
```

The **host** level is used to manage and get runtime metrics from the available hosts. For example, it is possible to add a new server to the **host3** host:

```
[domain@172.25.250.254:9990 /] cd /host=host2/server-config  
[domain@172.25.250.254:9990 server-config] ./server-c:add\  
(auto-start=true, group=main-server-group)
```

To remove a server, use the **remove** operation:

```
[domain@172.25.250.254:9990 /] /host=host2/server-config/server-c:\  
remove()
```

Another important item on the host level is the ability to get runtime information. For example, to get memory runtime metrics from a server, use the following commands:

```
[domain@172.25.250.254:9990 /] cd /host=host2/server=server-one/  
[domain@172.25.250.254:9990 server=server-one] cd core-service=platform-mbean/  
[domain@172.25.250.254:9990 core-service=platform-mbean] ./type=memory:\  
read-attribute(name=heap-memory-usage)
```

A similar output is expected:

```
{  
    "outcome" => "success",  
    "result" => {  
        "init" => 67108864L,  
        "used" => 126501384L,  
        "committed" => 206045184L,  
        "max" => 477626368L  
    }  
}
```

The top level also provides the **server-group** level. This level is responsible for creating and managing the server groups. To create a new group, use the following command:

```
[domain@172.25.250.254:9990 core-service=platform-mbean] cd /  
[domain@172.25.250.254:9990 /] /server-group=production:add\  
(profile=ha,socket-binding-group=ha-sockets)
```

To remove a server group called **production**, use the **remove** operation:

```
[domain@172.25.250.254:9990 /] /server-group=production:remove()
```

To remove a server called **server-two** from a host controller called **serverb**, use the following operation:

```
[domain@172.25.250.254:9990 /] /host=serverb/server-config=server-two:remove()
```

To configure or manage a subsystem, it is required to first navigate to the desired named profile:

```
[domain@172.25.250.254:9990 /] cd /profile=ha/subsystem=ejb3
```

The **socket-binding-group** level is responsible for configuring the available sockets for use by the server groups.

The **:take-snapshot** operation is available to backup the **domain.xml** configurations. This operation generates a backup file in the **JBOSS_HOME/domain/configuration/domain_xml_history/snapshot** folder:

```
[domain@172.25.250.254:9990 /] :take-snapshot
```

► Quiz

Configuring a Domain Controller

Choose the correct answer to the following questions:

► 1. Which tags are used in both `domain.xml` and `host.xml` files? (Choose three.)

- a. `extensions`
- b. `interfaces`
- c. `servers`
- d. `paths`
- e. `server-groups`

► 2. Which of the following sentences are correct? (Choose two.)

- a. `domain.xml` configuration definitions take precedence over `host.xml`
- b. `host.xml` configuration definitions take precedence over `domain.xml`.
- c. Deployments are stored in the `host.xml` file.
- d. Network configuration IP addresses are defined in the `host.xml` file.

► 3. Which of the following is can be used to create a snapshot of a domain controller configuration? (Choose one.)

- a. `:take-snapshot`
- b. `:save-snapshot`
- c. `:snapshot`
- d. `:save-snapshot <filename>`

► Solution

Configuring a Domain Controller

Choose the correct answer to the following questions:

► 1. Which tags are used in both `domain.xml` and `host.xml` files? (Choose three.)

- a. `extensions`
- b. `interfaces`
- c. `servers`
- d. `paths`
- e. `server-groups`

► 2. Which of the following sentences are correct? (Choose two.)

- a. `domain.xml` configuration definitions take precedence over `host.xml`
- b. `host.xml` configuration definitions take precedence over `domain.xml`.
- c. Deployments are stored in the `host.xml` file.
- d. Network configuration IP addresses are defined in the `host.xml` file.

► 3. Which of the following is can be used to create a snapshot of a domain controller configuration? (Choose one.)

- a. `:take-snapshot`
- b. `:save-snapshot`
- c. `:snapshot`
- d. `:save-snapshot <filename>`

► Lab

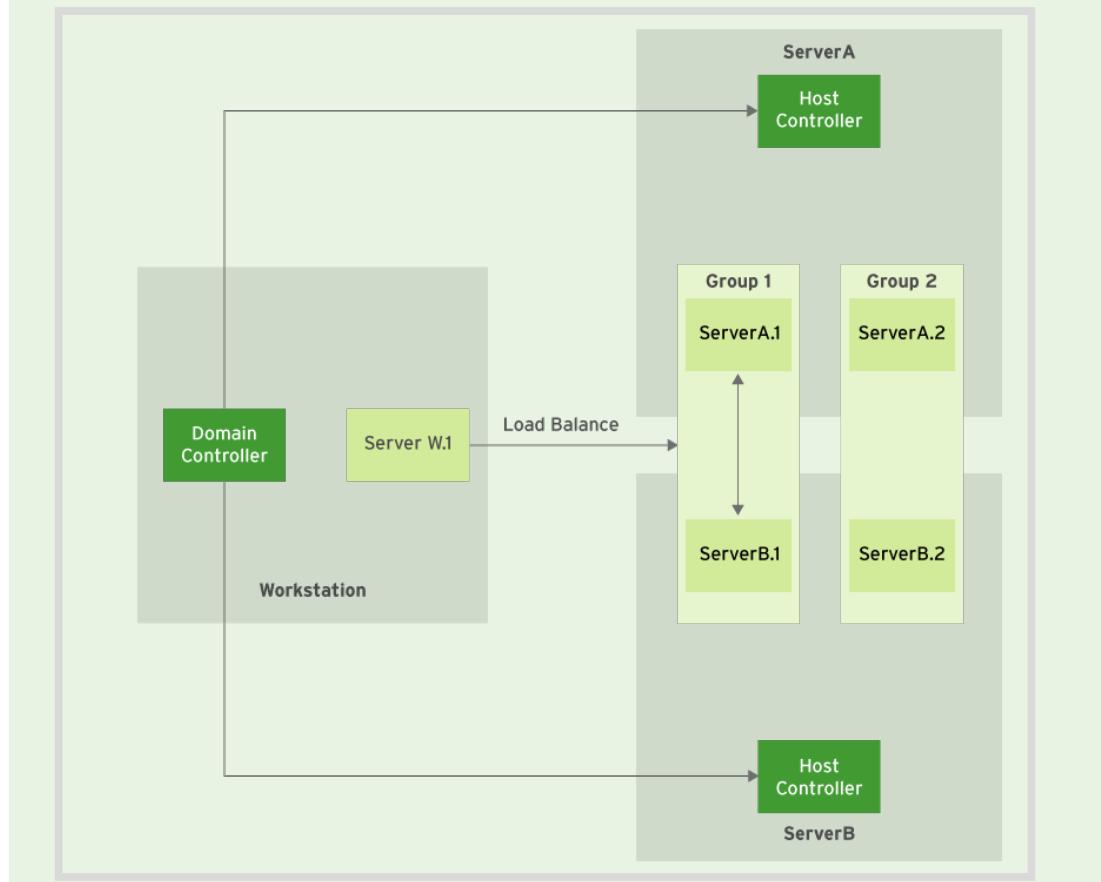
Configuring JBoss EAP as a Managed Domain

In this lab, you will create a managed domain.

Resources	
Files	/opt/domain
Application URL	172.25.250.10:8080 172.25.250.11:8080 172.25.250.254:9990
Resources	N/A

Outcomes

You should be able to start a domain controller on the workstation and run a host controller on both server A and server B. The final solution should be in line with the following architecture with regards to the host controller and the domain controller. The servers will be configured in later labs:



Before You Begin

In this lab, you create a managed domain comprised of a domain controller on the workstation, a host controller on server A and a host controller on server B. This lab sets the groundwork for the managed domain for the final solution for the course.

Check that **servera** can be accessed using SSH by running the following command from a terminal window in the workstation:

```
[student@workstation ~]$ ssh servera hostname
```

The expected output is:

```
servera.lab.example.com
```

If the following output is presented then start **servera** and re-run the command:

```
ssh: connect to host servera.lab.example.com port 22: No route to host
```

Check that **serverb** can be accessed using SSH by running the following command from a terminal window in the workstation:

```
[student@workstation ~]$ ssh serverb hostname
```

The expected output is:

```
serverb.lab.example.com
```

If the following output is presented then start **serverb** and re-run the command:

```
ssh: connect to host serverb.lab.example.com port 22: No route to host
```

Use the following command to download the relevant lab directory, verify the EAP installation on **workstation**, and to install EAP on **servera** and **serverb** with user **jboss**:

```
[student@workstation ~]$ lab managed-domain-lab setup
```

1. To get started creating the domain controller, copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on the workstation. Set the owner of the **/opt/domain** directory to user **jboss**.
2. The EAP instance on **workstation** is be the domain controller, and exposes the management interface to an internal network. The network where all servers are connected to is the **172.25.250.X** network. Other network interfaces should not be exposed to guarantee that external host controllers may not get sensitive information from the domain controller.

Update the address of the management interface for the domain controller to point to the workstation's IP address (**172.25.250.254**) using the **host-master.xml** configuration file.

- Because the domain controller will run an application with high availability features, including the messaging subsystem, some extra requirements are needed. The domain controller is responsible for providing security information to allow server instances to be part of a clustered environment, for all subsystems.

The messaging subsystem requires authentication to exchange data among cluster members, and the credentials are declared at the **domain.xml** file. The details about the subsystem will be explained later, but for the environment to function correctly, the cluster password must be declared at the domain controller configuration file.

Update **domain.xml** on **workstation** to have the cluster password **JBoss@RedHat123**.

- As the **jboss** user, open the **/opt/domain/configuration/domain.xml** file on **workstation**.

```
[student@workstation ~]$ sudo -u jboss vi /opt/domain/configuration/domain.xml
```

- Inside of the **messaging-activemq** subsystem of the full-ha profile, edit the **<cluster>** tag (line 1278):

```
<cluster password="${jboss.messaging.cluster.password:JBoss@RedHat123}" />
```



Note

Clustering will be described in more detail later on in the course.

- Save your changes to **domain.xml**.

- Start the domain controller on the workstation.

- In your terminal window, change directories to **/opt/jboss-eap-7.0/bin**:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin
```

- To start the domain controller using the **host-master.xml** file in your **/opt/domain/** folder, enter the following command:

```
[student@workstation bin]$ sudo -u jboss ./domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-master.xml
```

Look for the following output to confirm that the domain controller is running:

```
[Host Controller] 01:50:11,359 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0060: Http management interface listening on http://172.25.250.254:9990/
management
[Host Controller] 01:50:11,359 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0051: Admin console listening on http://172.25.250.254:9990
```

- Access the **server A** VM using the **ssh** command:

```
[student@workstation bin]$ ssh servera
```

The setup script installed an instance of EAP onto both **server A** and **server B**. Copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on **server A** and set the owner as **jboss**.

6. Set the name of the host to **servera**. Update the address of the management interface for the host controller to point to server A's IP address (**172.25.250.10**) using the **host-slave.xml** configuration file.
7. Start the host controller on **server A** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.
8. Access the **server B** VM using the **ssh** command:

```
[student@workstation bin]$ ssh serverb
```

Copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on **server B** and set the owner as **jboss**.

9. Set the name of the host to **serverb**. Update the address of the management interface for the host controller to point to **server B**'s IP address (**172.25.250.11**) using the **host-slave.xml** configuration file and configure the servers so that there is only one named **server-three** that has no port offset in server group **other-server-group**.
10. Start the host controller on **server B** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.
11. On **workstation**, open a web browser and navigate to **172.25.250.254:9990** to access the domain's management console. Verify that both hosts appear under the **Runtime** section.
 - 11.1. On **workstation**, open a web browser and navigate to **172.25.250.254:9990**.
 - 11.2. At the top of the page click **Runtime** and then select **Hosts** in the first column.
 - 11.3. Verify that both host **servera** and host **serverb** appear.
12. Stop and remove all of the servers (**server-one**, **server-two**, and **server-three**). In the next chapter's final lab, you create new servers to deploy applications on these hosts.
 - 12.1. In a new terminal on the **workstation**, connect to the EAP CLI for the domain controller:

```
[student@workstation bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```



Note

The administrator user name is **jbossadm** and the password is **JBoss@RedHat123**.

- 12.2. Use the following command to stop all of the servers in the managed domain. The servers must be stopped before removing them:

```
[domain@172.25.250.254:9990] :stop-servers
```

12.3. After the servers are all stopped, use the following command to remove **server-one**:

```
[domain@172.25.250.254:9990] /host=servera/server-config=server-one:remove
```

Run the command again, but navigate to **server-two** that is on **servera**:

```
[domain@172.25.250.254:9990] /host=servera/server-config=server-two:remove
```

Run the command to remove **server-three** that is on **serverb**:

```
[domain@172.25.250.254:9990] /host=serverb/server-config=server-three:remove
```

13. Remove the server-groups (**main-server-group**, **other-server-group**) as these will be configured in the next chapter's final lab in order to align with the final course solution. After removing the server groups, exit the EAP CLI.

13.1. Run the following command to remove **main-server-group** on **workstation** with the EAP CLI:

```
[domain@172.25.250.254:9990] /server-group=main-server-group:remove
```

13.2. Run the following command to remove **other-server-group** on **workstation** with the EAP CLI:

```
[domain@172.25.250.254:9990] /server-group=other-server-group:remove
```

13.3. Exit the EAP CLI.

14. Clean Up and Grading

14.1. Press **Ctrl+C** to stop the domain controller on **workstation** and the two host controllers on **servera** and **serverb**.

14.2. Run the following command to grade the exercise:

```
[student@workstation bin]$ lab managed-domain-lab grade
```

This concludes the lab.

► Solution

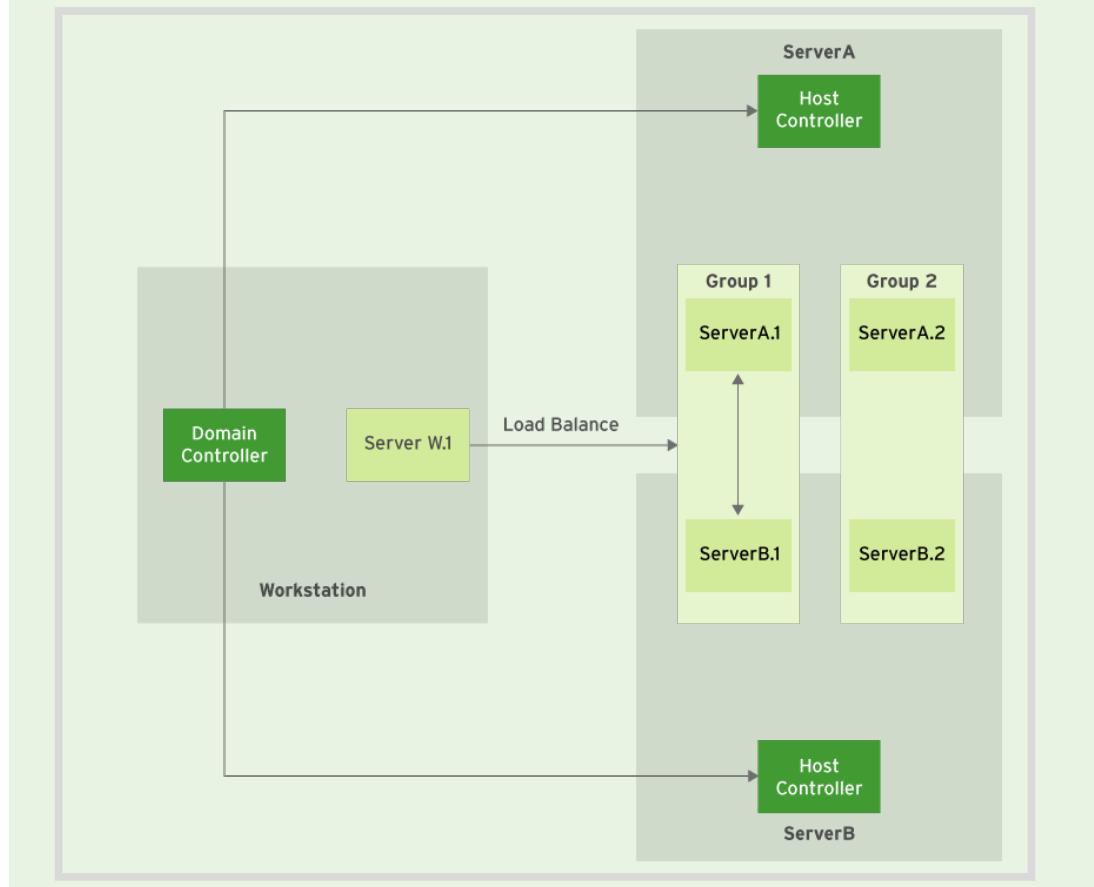
Configuring JBoss EAP as a Managed Domain

In this lab, you will create a managed domain.

Resources	
Files	/opt/domain
Application URL	172.25.250.10:8080 172.25.250.11:8080 172.25.250.254:9990
Resources	N/A

Outcomes

You should be able to start a domain controller on the workstation and run a host controller on both server A and server B. The final solution should be in line with the following architecture with regards to the host controller and the domain controller. The servers will be configured in later labs:



Before You Begin

In this lab, you create a managed domain comprised of a domain controller on the workstation, a host controller on server A and a host controller on server B. This lab sets the groundwork for the managed domain for the final solution for the course.

Check that **servera** can be accessed using SSH by running the following command from a terminal window in the workstation:

```
[student@workstation ~]$ ssh servera hostname
```

The expected output is:

```
servera.lab.example.com
```

If the following output is presented then start **servera** and re-run the command:

```
ssh: connect to host servera.lab.example.com port 22: No route to host
```

Check that **serverb** can be accessed using SSH by running the following command from a terminal window in the workstation:

```
[student@workstation ~]$ ssh serverb hostname
```

The expected output is:

```
serverb.lab.example.com
```

If the following output is presented then start **serverb** and re-run the command:

```
ssh: connect to host serverb.lab.example.com port 22: No route to host
```

Use the following command to download the relevant lab directory, verify the EAP installation on **workstation**, and to install EAP on **servera** and **serverb** with user **jboss**:

```
[student@workstation ~]$ lab managed-domain-lab setup
```

1. To get started creating the domain controller, copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on the workstation. Set the owner of the **/opt/domain** directory to user **jboss**.

- 1.1. Run the following command to copy the EAP domain configuration to the **/opt/domain** directory on the **workstation**.

```
[student@workstation ~]$ sudo cp -r /opt/jboss-eap-7.0/domain /opt/
```

- 1.2. Use the following command to set the directory owner as user **jboss**:

```
[student@workstation ~]$ sudo chown -R jboss:jboss /opt/domain
```

2. The EAP instance on **workstation** is to be the domain controller, and exposes the management interface to an internal network. The network where all servers are connected to is the **172.25.250.X** network. Other network interfaces should not be exposed to guarantee that external host controllers may not get sensitive information from the domain controller.

Update the address of the management interface for the domain controller to point to the workstation's IP address (**172.25.250.254**) using the **host-master.xml** configuration file.

- 2.1. Open the file **/opt/domain/configuration/host-master.xml** with a text editor as user **jboss**.
- 2.2. Modify the interface sections of the configuration file as follows:

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:172.25.250.254}" />
  </interface>
</interfaces>
```

- 2.3. Save your changes and exit the editor.

3. Because the domain controller will run an application with high availability features, including the messaging subsystem, some extra requirements are needed. The domain controller is responsible for providing security information to allow server instances to be part of a clustered environment, for all subsystems.

The messaging subsystem requires authentication to exchange data among cluster members, and the credentials are declared at the **domain.xml** file. The details about the subsystem will be explained later, but for the environment to function correctly, the cluster password must be declared at the domain controller configuration file.

Update **domain.xml** on **workstation** to have the cluster password **JBoss@RedHat123**.

- 3.1. As the **jboss** user, open the **/opt/domain/configuration/domain.xml** file on **workstation**.

```
[student@workstation ~]$ sudo -u jboss vi /opt/domain/configuration/domain.xml
```

- 3.2. Inside of the **messaging-activemq** subsystem of the full-ha profile, edit the **<cluster>** tag (line 1278):

```
<cluster password="${jboss.messaging.cluster.password:JBoss@RedHat123}" />
```



Note

Clustering will be described in more detail later on in the course.

- 3.3. Save your changes to **domain.xml**.

4. Start the domain controller on the workstation.

- 4.1. In your terminal window, change directories to **/opt/jboss-eap-7.0/bin**:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin
```

- 4.2. To start the domain controller using the **host-master.xml** file in your **/opt/domain** folder, enter the following command:

```
[student@workstation bin]$ sudo -u jboss ./domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-master.xml
```

Look for the following output to confirm that the domain controller is running:

```
[Host Controller] 01:50:11,359 INFO  [org.jboss.as] (Controller Boot Thread)
WFLYSRV0060: Http management interface listening on http://172.25.250.254:9990/
management
[Host Controller] 01:50:11,359 INFO  [org.jboss.as] (Controller Boot Thread)
WFLYSRV0051: Admin console listening on http://172.25.250.254:9990
```

5. Access the **server A** VM using the **ssh** command:

```
[student@workstation bin]$ ssh servera
```

The setup script installed an instance of EAP onto both **server A** and **server B**. Copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on **server A** and set the owner as **jboss**.

```
[student@servera ~]$ sudo cp -r /opt/jboss-eap-7.0/domain /opt/
[student@servera ~]$ sudo chown -R jboss:jboss /opt/domain
```

6. Set the name of the host to **servera**. Update the address of the management interface for the host controller to point to server A's IP address (**172.25.250.10**) using the **host-slave.xml** configuration file.

- 6.1. Open the file **/opt/domain/configuration/host-slave.xml** on **server A** with a text editor as user **jboss**.
- 6.2. Update the name of the host to **servera** by adding the **name** property to the **<host>** tag:

```
<host name="servera" xmlns="urn:jboss:domain:4.1">
```

- 6.3. Update the public IP address as follows:

```
<interface name="public">
  <inet-address value="${jboss.bind.address:172.25.250.10}" />
</interface>
```

- 6.4. Update the management IP address as follows:

```
<interface name="management">
    <inet-address value="${jboss.bind.address.management:172.25.250.10}"/>
</interface>
```

6.5. Save the file and exit the editor.

7. Start the host controller on **server A** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.

7.1. Run the following command to start the host controller and connect to the domain controller:

```
[student@servera ~]$ cd /opt/jboss-eap-7.0/bin
[student@servera bin]$ sudo -u jboss ./domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254
```

7.2. Look in the terminal window of the host controller on **servera**. Carefully review the log output and you should see the host controller connecting to the master, and also **server-one** and **server-two** starting up.

```
[Host Controller] 16:42:57,307 INFO [org.jboss.as.host.controller]
(Controller Boot Thread) WFLYHC0148: Connected to master host controller at
remote://172.25.250.254:9999
[Host Controller] 16:42:57,367 INFO [org.jboss.as.host.controller] (Controller
Boot Thread) WFLYHC0023: Starting server server-one
```

7.3. Look in the terminal window of the domain controller running on the workstation. You should see a log entry showing the slave connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host
Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
"servera", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

8. Access the **server B** VM using the **ssh** command:

```
[student@workstation bin]$ ssh serverb
```

Copy files from **JBoss_HOME/domain** into the lab directory at **/opt/domain** on **server B** and set the owner as **jboss**.

```
[student@serverb ~]$ sudo cp -r /opt/jboss-eap-7.0/domain /opt/
[student@serverb ~]$ sudo chown -R jboss:jboss /opt/domain
```

9. Set the name of the host to **serverb**. Update the address of the management interface for the host controller to point to **server B's IP address (172.25.250.11)** using the **host-slave.xml** configuration file and configure the servers so that there is only one named **server-three** that has no port offset in server group **other-server-group**.

- 9.1. Open the file **/opt/domain/configuration/host-slave.xml** on **server B** with a text editor as user **jboss**.
- 9.2. Update the name of the host to **servera** by adding the **name** property to the **<host>** tag:

```
<host name="serverb" xmlns="urn:jboss:domain:4.1">
```

- 9.3. Update the public IP address as follows:

```
<interface name="public">
    <inet-address value="${jboss.bind.address:172.25.250.11}" />
</interface>
```

- 9.4. Update the management IP address as follows:

```
<interface name="management">
    <inet-address value="${jboss.bind.address.management:172.25.250.11}" />
</interface>
```

- 9.5. Update the **servers** tag to match the following:

```
<servers>
    <server name="server-three" group="other-server-group"/>
</servers>
```

- 9.6. Save the file and exit the editor.

10. Start the host controller on **server B** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.

- 10.1. Run the following command to start the host controller and connect to the domain controller:

```
[student@serverb ~]$ cd /opt/jboss-eap-7.0/bin
[student@serverb bin]$ sudo -u jboss ./domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254
```

- 10.2. Look in the terminal window of the host controller of **serverb**. Carefully review the log output and you should see the host controller connecting to the master, and also **server-three** starting up.

```
[Host Controller] 16:42:57,307 INFO [org.jboss.as.host.controller]
(Controller Boot Thread) WFLYHC0148: Connected to master host controller at
remote://172.25.250.254:9999
[Host Controller] 16:42:57,367 INFO [org.jboss.as.host.controller] (Controller
Boot Thread) WFLYHC0023: Starting server server-three
```

- 10.3. Look in the terminal window of the domain controller running on **workstation**. You should see a log entry showing the slave connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host
Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
"serverb", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

11. On **workstation**, open a web browser and navigate to 172.25.250.254:9990 to access the domain's management console. Verify that both hosts appear under the **Runtime** section.
- 11.1. On **workstation**, open a web browser and navigate to 172.25.250.254:9990.
 - 11.2. At the top of the page click **Runtime** and then select **Hosts** in the first column.
 - 11.3. Verify that both host **servera** and host **serverb** appear.
12. Stop and remove all of the servers (**server-one**, **server-two**, and **server-three**). In the next chapter's final lab, you create new servers to deploy applications on these hosts.
- 12.1. In a new terminal on the **workstation**, connect to the EAP CLI for the domain controller:

```
[student@workstation bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```



Note

The administrator user name is **jbossadm** and the password is **JBoss@RedHat123**.

- 12.2. Use the following command to stop all of the servers in the managed domain. The servers must be stopped before removing them:

```
[domain@172.25.250.254:9990] :stop-servers
```

- 12.3. After the servers are all stopped, use the following command to remove **server-one**:

```
[domain@172.25.250.254:9990] /host=servera/server-config=server-one:remove
```

Run the command again, but navigate to **server-two** that is on **servera**:

```
[domain@172.25.250.254:9990] /host=servera/server-config=server-two:remove
```

Run the command to remove **server-three** that is on **serverb**:

```
[domain@172.25.250.254:9990] /host=serverb/server-config=server-three:remove
```

13. Remove the server-groups (**main-server-group**, **other-server-group**) as these will be configured in the next chapter's final lab in order to align with the final course solution. After removing the server groups, exit the EAP CLI.

- 13.1. Run the following command to remove **main-server-group** on **workstation** with the EAP CLI:

```
[domain@172.25.250.254:9990] /server-group=main-server-group:remove
```

- 13.2. Run the following command to remove **other-server-group** on **workstation** with the EAP CLI:

```
[domain@172.25.250.254:9990] /server-group=other-server-group:remove
```

- 13.3. Exit the EAP CLI.

14. Clean Up and Grading

- 14.1. Press **Ctrl+C** to stop the domain controller on **workstation** and the two host controllers on **servera** and **serverb**.

- 14.2. Run the following command to grade the exercise:

```
[student@workstation bin]$ lab managed-domain-lab grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- Each machine that hosts server instances runs a **host controller** process, which acts as a **slave** to the domain controller. The master is also a host controller.
- The host configuration file, **host.xml**, configures a host controller, including defining its role as master or slave, its IP addresses, and its server instances.
- Host controllers require a native management interface for master/slave communication, but only the master should have an active HTTP management interface.
- Host controllers are started by the **domain.sh** script. The script invocation usually defines system properties for the IP addresses to be assigned to the host controller network interfaces, the master IP address if it is a slave, and the configuration files folder.
- Deploying applications to a managed domain involves three steps:
 1. Uploading the application package to the domain controller repository.
 2. Assigning the application to one or more server groups.
 3. Enabling the application in each of its assigned server groups. This is when the application becomes deployed from the user and developer point of view.

Chapter 5

Configuring Servers in a Managed Domain

Overview

- Goal** Configure servers and server groups in a managed domain.
- Objectives**
- Describe the relationship between hosts, server groups, and servers in a managed domain.
 - Configure server groups in a managed domain.
 - Configure servers on a host controller.
 - Deploy
- Sections**
- Managed Domain Server Architecture (and Quiz)
 - Configuring Server Groups (and Guided Exercise)
 - Configuring Servers (and Guided Exercise)
 - Deploying Applications on a Managed Domain (and Guided Exercise)
- Lab**
- Configuring Servers in a Managed Domain

Managed Domain Server Architecture

Objectives

After completing this section, students should be able to:

- Describe the relationship between hosts, server groups, and servers.

Understanding hosts and servers

A managed domain consists of a:

- Domain controller*: Responsible for all configuration management using *profiles*.
- Host controller*: Responsible for managing server instances. In a production environment, a host controller is usually installed on a separate host (physical or VM).
- Server Groups*: A logical grouping of EAP server instances that are configured and managed together as a single unit. Server groups can span any number of servers across multiple host controllers.
- Server*: Responsible for running JavaEE applications (JAR, EAR, WAR files).

To simplify server instance management, EAP defines the concept of a *server group*, which shares the same set of applications and reuses the same profile and all embedded configurations.



Note

A server cannot be part of multiple server groups.

Server groups are configured in the **domain.xml** configuration file on the domain controller, and servers are configured in the **host.xml** configuration file on each host.

The topology of an EAP managed domain can be viewed from two perspectives:

- Host View**: A managed domain consists of various hosts, where a host is an EAP instance running in managed domain mode. An EAP managed domain is a collection of hosts.
- Server View**: An EAP managed domain can be viewed as a collection of server instances, with each server instance belonging to a server group.

To demonstrate, the following diagram shows a domain from the perspective of the hosts within the managed domain:

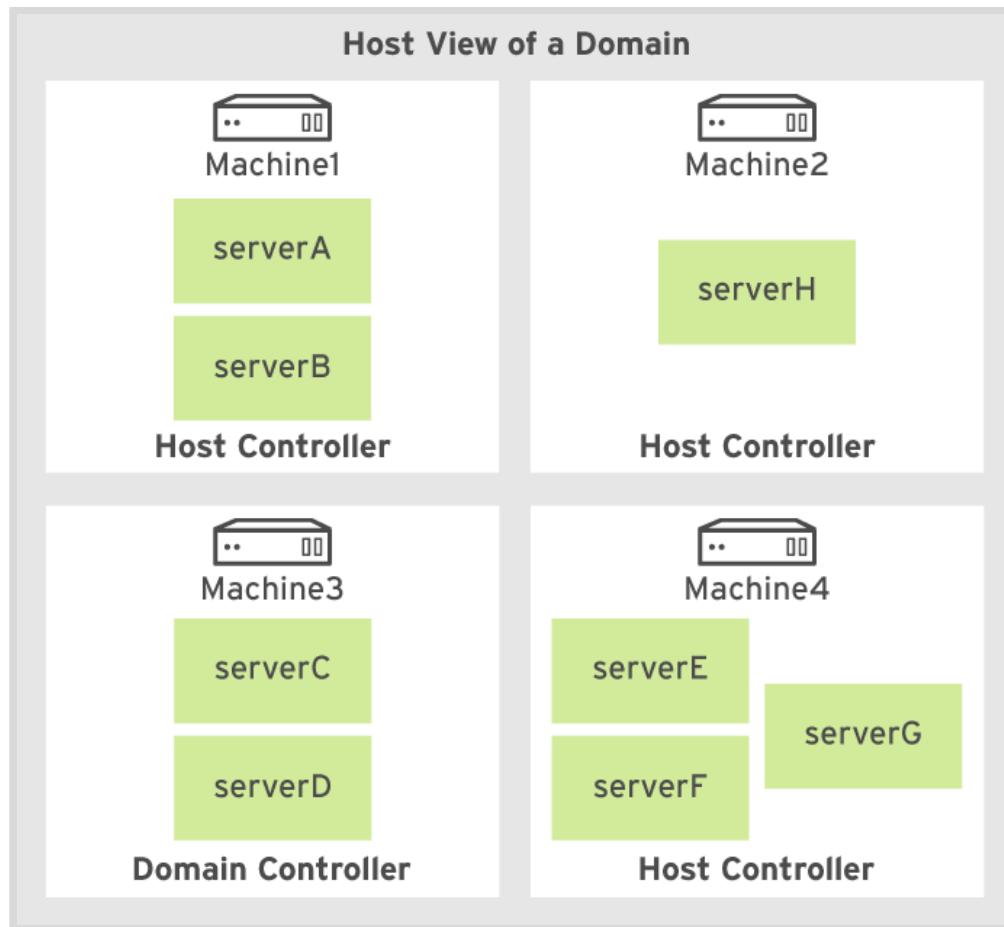
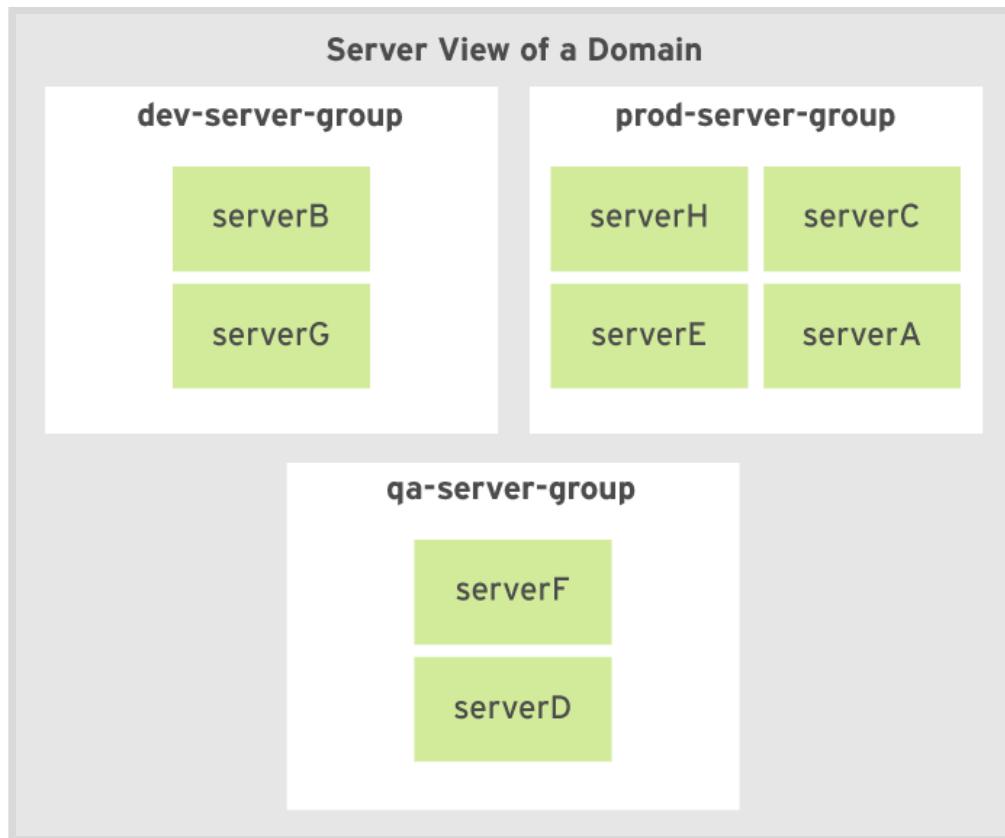


Figure 5.1: Host view of a domain

The diagram above shows which servers are running on a host, without addressing which server groups each one belongs to. It is hard to identify which servers share the same configuration set, but it is useful for an administrator to see how many processes are running on each host (physical or VM). An administrator can plan the amount of memory and CPU capacity needed for each host based on this information.

The following diagram shows the same EAP managed domain, except that this diagram shows the managed domain from the perspective of the servers:

**Figure 5.2: Server view of a domain**

The previous diagram shows which servers and the server group they belong to, providing a logical view of an EAP managed domain. It does not address hardware concerns, but shows which servers share the same configuration set. Notice that in the server view, the important distinction is the server group that the server belongs to, and not the machine that the server is running on. When designing a managed domain, the server view is critical because applications are not deployed onto servers; they are deployed onto server groups.

These views complement each other and it helps an administrator to plan how many EAP host controllers must be deployed and how many servers must be created.

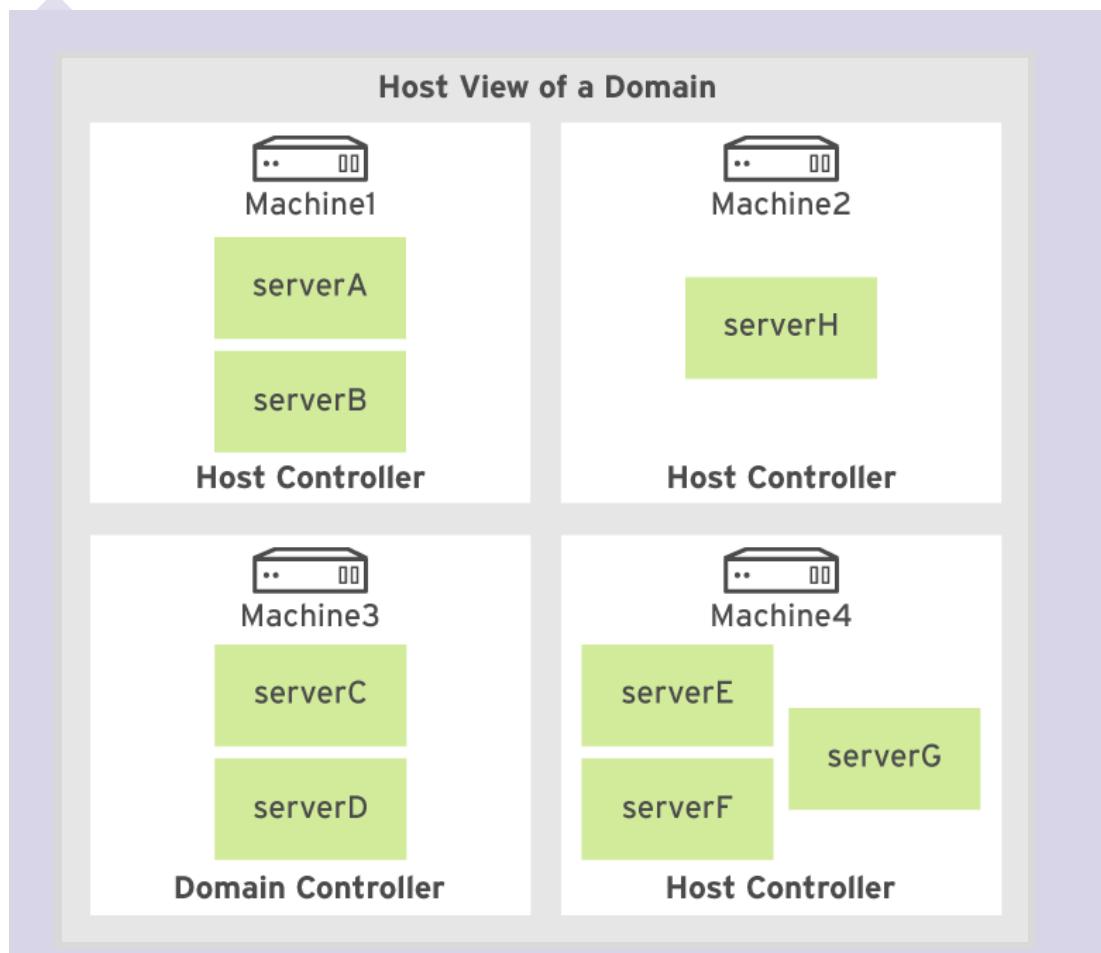


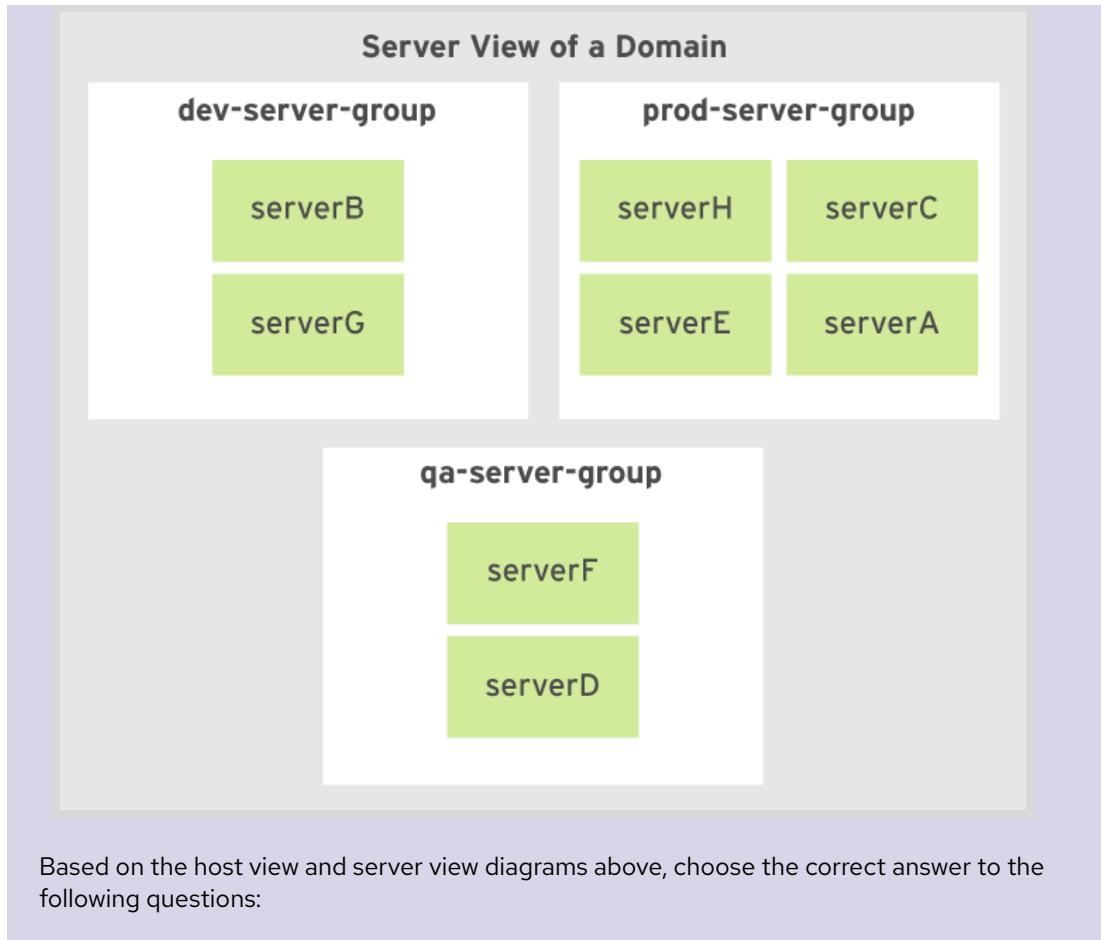
Note

Different server groups can be configured with different *profiles* and *deployments* (*applications*). Different server groups can run the same profile and have the same deployments. A benefit of having identical server groups is to support rolling application upgrade scenarios, where a complete service outage is avoided by first upgrading the application on one server group, and then upgrading the application on the second server group.

► Quiz

Hosts and Servers





Based on the host view and server view diagrams above, choose the correct answer to the following questions:

- ▶ **1. How many server groups are defined in the managed domain? (Choose one.)**
 - a. 1
 - b. 2
 - c. 3
 - d. 4

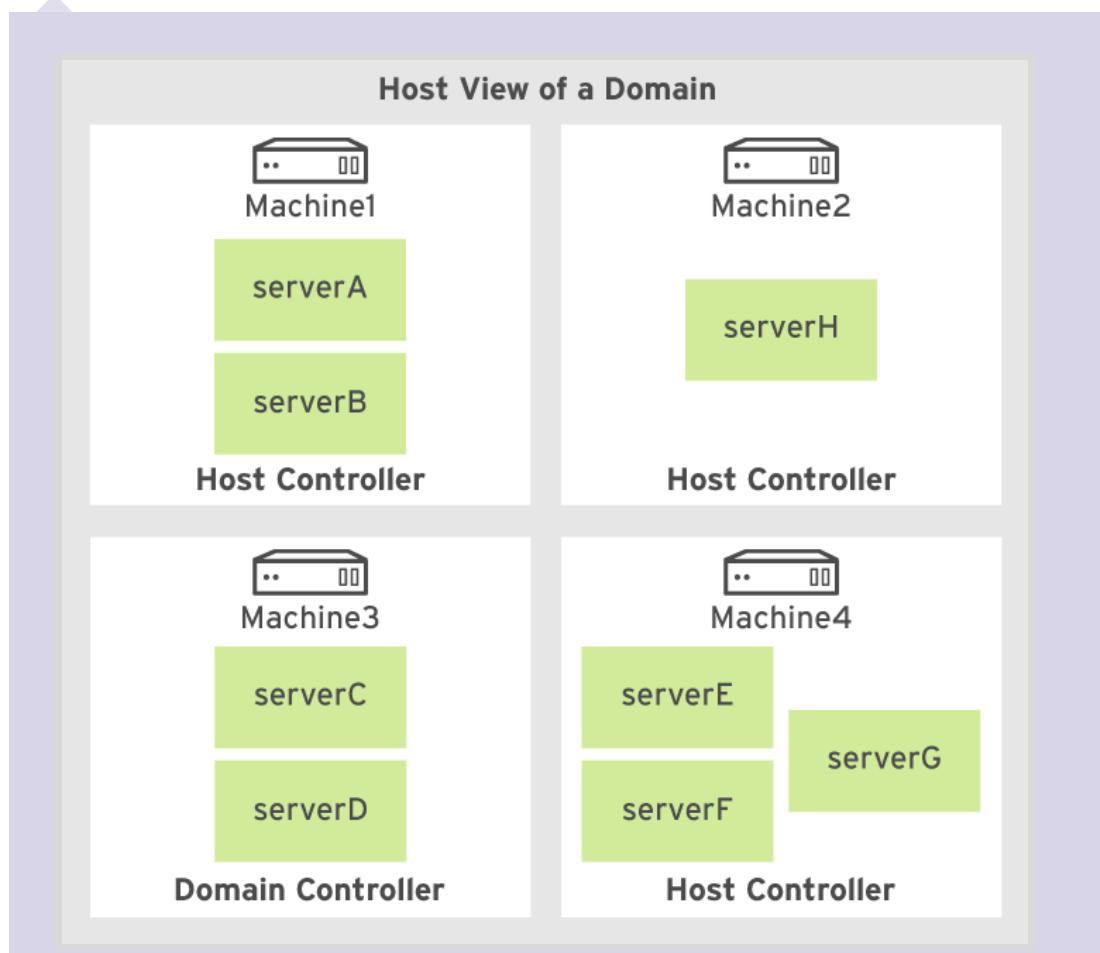
- ▶ **2. How many servers are defined in the managed domain? (Choose one.)**
 - a. 3
 - b. 8
 - c. 6
 - d. 5

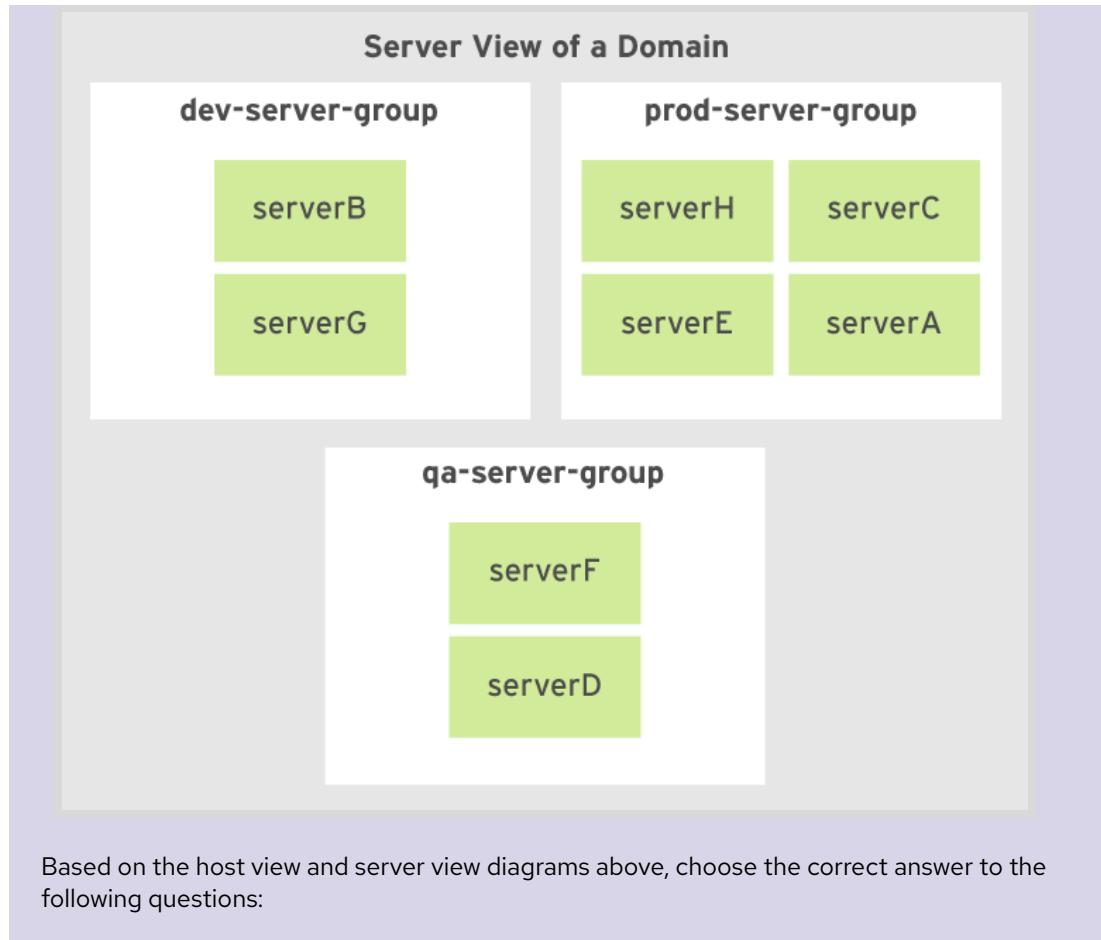
- ▶ **3. How many host controllers are in the managed domain? (Choose one.)**
 - a. 2
 - b. 4
 - c. 6
 - d. 8

- 4. If an application **example.war** is accessible from **serverA**, which of the following statements are true? (Choose two.)
- a. **example.war** is accessible from **serverB**.
 - b. **example.war** is deployed to the **qa-server-group**.
 - c. **example.war** is accessible from **serverH**.
 - d. **example.war** is accessible from **serverG**.
 - e. **example.war** is deployed to the **dev-server-group**.
 - f. **example.war** is deployed to the **prod-server-group**.
- 5. Which of the following statements about the **prod-server-group** are correct? (Choose two.)
- a. An application can be deployed such that it only runs on **serverH** and not on other servers in the **prod-server-group**.
 - b. **serverH** is part of the **prod-server-group**.
 - c. All the servers in the **prod-server-group** must be running on a single host.
 - d. Multiple applications can be deployed on the **prod-server-group**.
- 6. Which of the following statements about the host **Machine4** are correct? (Choose two.)
- a. **serverE**, **serverF** and **serverG** must belong to the same server group.
 - b. There can only be one server group created on host **Machine4**.
 - c. **serverE**, **serverF** and **serverG** can be assigned to any server group.
 - d. Multiple applications can be running simultaneously on the host **Machine4**.
 - e. The configuration for host **Machine4** is stored in a file called **domain.xml**.
- 7. Which of the following statements about the managed domain are correct? (Choose three.)
- a. **prod-server-group** and **dev-server-group** must be configured with the same profile.
 - b. **prod-server-group** and **dev-server-group** can be configured with different profiles.
 - c. All the server groups must be configured with the same profile and deployments (applications).
 - d. Multiple different applications can be deployed on **prod-server-group** and **dev-server-group**.
 - e. A server group can have no applications.
 - f. A host must always have at least one server configured and running.

► Solution

Hosts and Servers





- ▶ **1. How many server groups are defined in the managed domain? (Choose one.)**
 - a. 1
 - b. 2
 - c. 3
 - d. 4

- ▶ **2. How many servers are defined in the managed domain? (Choose one.)**
 - a. 3
 - b. 8
 - c. 6
 - d. 5

- ▶ **3. How many host controllers are in the managed domain? (Choose one.)**
 - a. 2
 - b. 4
 - c. 6
 - d. 8

► 4. If an application `example.war` is accessible from `serverA`, which of the following statements are true? (Choose two.)

- a. `example.war` is accessible from `serverB`.
- b. `example.war` is deployed to the `qa-server-group`.
- c. `example.war` is accessible from `serverH`.
- d. `example.war` is accessible from `serverG`.
- e. `example.war` is deployed to the `dev-server-group`.
- f. `example.war` is deployed to the `prod-server-group`.

► 5. Which of the following statements about the `prod-server-group` are correct? (Choose two.)

- a. An application can be deployed such that it only runs on `serverH` and not on other servers in the `prod-server-group`.
- b. `serverH` is part of the `prod-server-group`.
- c. All the servers in the `prod-server-group` must be running on a single host.
- d. Multiple applications can be deployed on the `prod-server-group`.

► 6. Which of the following statements about the host `Machine4` are correct? (Choose two.)

- a. `serverE`, `serverF` and `serverG` must belong to the same server group.
- b. There can only be one server group created on host `Machine4`.
- c. `serverE`, `serverF` and `serverG` can be assigned to any server group.
- d. Multiple applications can be running simultaneously on the host `Machine4`.
- e. The configuration for host `Machine4` is stored in a file called `domain.xml`.

► 7. Which of the following statements about the managed domain are correct? (Choose three.)

- a. `prod-server-group` and `dev-server-group` must be configured with the same profile.
- b. `prod-server-group` and `dev-server-group` can be configured with different profiles.
- c. All the server groups must be configured with the same profile and deployments (applications).
- d. Multiple different applications can be deployed on `prod-server-group` and `dev-server-group`.
- e. A server group can have no applications.
- f. A host must always have at least one server configured and running.

Configuring Server Groups

Objectives

After completing this section, students should be able to:

- Configure a server group within a managed domain.
- Describe the different options available to create and manage server groups.
- Describe the different attributes of a server group that are configured in the `domain.xml` file of the domain controller.

Server groups

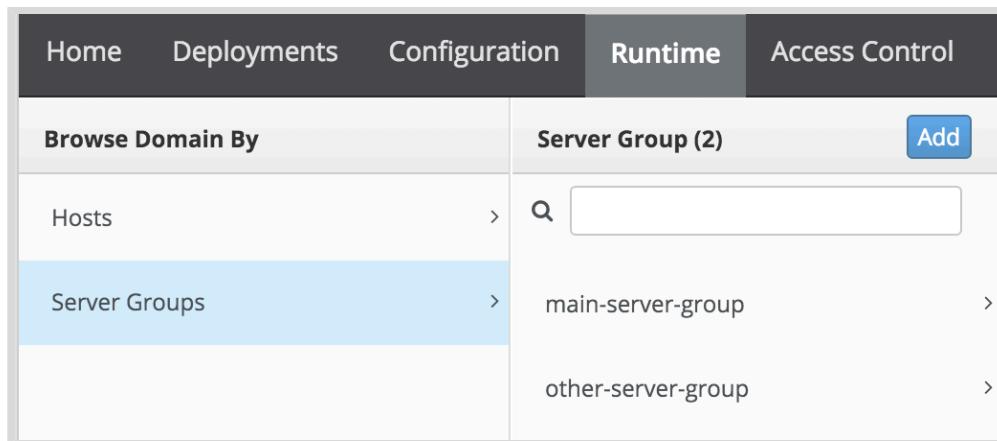
A server group is a logical grouping of servers in an EAP managed domain that are managed as a single unit. The server groups are defined and managed by the domain controller, but each individual server is managed by the respective host controllers which communicate with the domain controller and ensure that all the servers within a managed domain have the same configuration.

Each server group is assigned a unique **profile**. A **profile** consists of the list of EAP **subsystems** and their configuration that will be used by the different subsystems that make up the profile.

Server groups are configured in the `domain.xml` file on the domain controller. The recommended approach to create and manage the server groups in a managed domain is to either use the EAP management console or the JBoss EAP CLI (preferred).

Configuring server groups using the management console

Server groups in a managed domain can be created and managed in the **Runtime** section of the management console.



The screenshot shows the 'Runtime' tab selected in the top navigation bar. On the left, there's a sidebar with 'Browse Domain By' options: 'Hosts' and 'Server Groups'. The 'Server Groups' option is highlighted with a blue background. To the right, a main panel displays a list of 'Server Group (2)'. It shows two entries: 'main-server-group' and 'other-server-group', each preceded by a magnifying glass icon for search. A blue 'Add' button is located at the top right of this list area.

Figure 5.5: Server groups view in the Runtime page

To add a new server group, three key attributes must be provided:

- a unique **Name** for the server group.
- a valid **Profile** from the list of profiles defined in the **domain.xml** configuration file, which control the subsystems that are active in a given profile.
- a **Socket Binding** which defines a set of network ports that will be used by servers running within the server group.

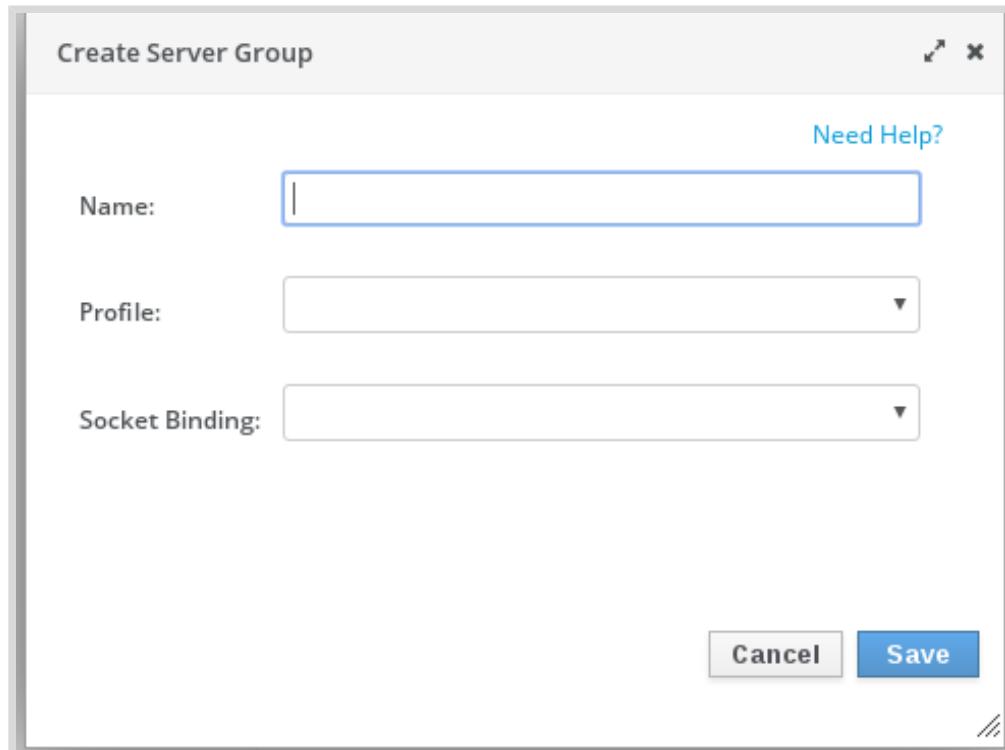


Figure 5.6: Creating a server group

To remove a server group, ensure that there are no servers assigned to the server group. If there are servers assigned, stop them first and then remove the servers from the server group. Finally, remove the server group.

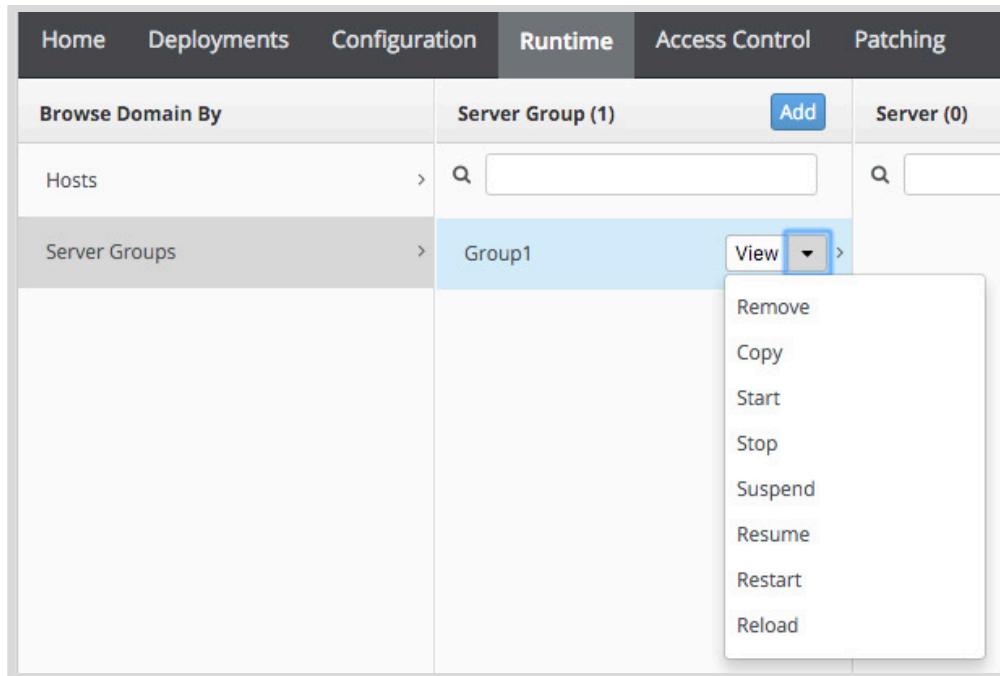


Figure 5.7: Removing a server group

Configuring server groups using the JBoss EAP CLI

Server groups can also be created and managed using the JBoss EAP CLI. The advantage of this approach is that the steps to create and manage the server groups can be scripted as part of an automated work flow and it is faster to create multiple server groups in batched mode. This approach also has the option to atomically commit and/or roll back multiple steps in batch mode, thereby ensuring the integrity of the `domain.xml` configuration file in case of an error during execution of the CLI commands.

The server groups are configured in the `/server-group` namespace of the JBoss EAP CLI. Commands exist to create, view, modify, and remove server groups in this namespace. To add a new server group, run the following command:

```
[domain@workstation:9990 /] /server-group=Group1:add\
(profile=full,socket-binding-group=full-sockets)
{
    "outcome" => "success",
    ...
}
```

To verify that the server groups have been created, use the `read-resource` attribute in the EAP CLI:

```
[domain@workstation:9990 /] /server-group=Group1:read-resource
{
    "outcome" => "success",
    "result" => {
        "management-subsystem-endpoint" => false,
        "profile" => "default",
        "socket-binding-default-interface" => undefined,
        "socket-binding-group" => "standard-sockets",
```

```

        "socket-binding-port-offset" => 0,
        "deployment" => undefined,
        "deployment-overlay" => undefined,
        "jvm" => undefined,
        "system-property" => undefined
    }
}

```

To remove a server group, ensure that there are no servers assigned to the server group. Then run the following command to remove the server group:

```
[domain@workstation:9990 /] /server-group=Group1:remove()
{
    "outcome" => "success",
    ...
}
```



Note

Regardless of which technique is used to modify a configuration setting, all changes are synchronized to the `domain.xml` configuration file. For example, if a setting is modified using the management console, the underlying `domain.xml` file is updated, and the CLI will immediately become aware of the change. In the next section, the server group configuration attributes in the `domain.xml` file will be discussed in more detail.

Server group configuration

Server groups in a managed domain are defined in the `domain.xml` file of the domain controller. Server groups are defined using the `<server-groups>` element. The child element `<server-group>` within the `<server-groups>` parent element can be used to define a server group. A sample `<server-group>` definition follows:

```

<server-groups>
    <server-group name="main-server-group"① profile="default"②>
        <jvm name="default"③>
            ...
        </jvm>
        <socket-binding-group ref="standard-sockets"④/>
        <deployments>⑤
            ...
        </deployments>
    </server-group>
</server-groups>

```

- ① The `name` attribute is required and must be unique in the managed domain. When defining a server in `host.xml`, the server references this `name` attribute.
- ② The `profile` attribute is also required and references the name of a `<profile>` defined in the domain configuration file.
- ③ The `jvm name` attribute references a `<jvm>` memory configuration for all the servers in this server group. In addition, the server group's settings for the JVM can be overridden by the server in `host.xml`. JVM memory allocation in EAP is discussed later in the course.

- ④ The **socket-binding-group** attribute references the name of a socket binding group defined in this domain configuration file. This setting can be overridden by the server in `host.xml`.
- ⑤ The **<deployments>** section lists the applications to be deployed on every server in the group. Do not add **<deployment>** entries manually, but instead deploy the applications using the management console or CLI.



Note

It is the responsibility of the domain controller and the host controllers to ensure that all servers in a server group have a consistent configuration. The servers are all be configured with the same profile assigned to the server group, and they will have the same deployment content deployed.

► Guided Exercise

Configuring Server Groups

In this lab, you will configure server groups in a managed domain using the EAP 7 management console.

Resources	
Files	/home/student/JB248/labs/domain and /home/student/JB248/labs/host
Application URL	http://172.25.250.254:9990

Outcomes

You should be able to configure and start a managed domain with two server groups.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and the previous lab set up a domain controller at **/home/student/JB248/labs/domain** and two hosts at **/home/student/JB248/labs/host**:

```
[student@workstation ~]$ lab server-groups setup
```

- 1. You will be simulating running three host machines on the workstation in this guided exercise. In order to not disturb the existing EAP 7 installation at **/opt/jboss-eap-7.0**, the setup script has created a domain controller in the **machine1** folder under the **/home/student/JB248/labs/domain** directory and two folders under the **/home/student/JB248/labs/host** directory called **machine2** and **machine3** that simulate host controllers.
It has also set up a preconfigured managed domain with the two hosts and the domain controller. There are two server groups and three servers predefined in the managed domain. You will start the managed domain and delete the existing server groups and servers and define two new server groups using the management console.
- 2. Start the domain controller (**machine1**) and the two slave hosts (**machine2** and **machine3**).
2.1. When the slave host controllers are started, they connect to the domain controller and fetch the latest domain configuration. For this reason, the domain controller should be started first, followed by the two slaves.
Start the domain controller using a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \  
--host-config=host-master.xml
```

2.2. Start the host controller on **machine2**.

Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window on your **workstation** to start **host2** using the **host-slave.xml** configuration file:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine2/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

2.3. Look in the terminal window of the domain controller. You should see a log entry showing the slave **host2** connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 36) WFLYHC0019: Registered remote slave host "host2", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

2.4. Start the host controller on **machine3** to join the managed domain.

Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window on your workstation to start **host3** using the **host-slave.xml** configuration file:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine3/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

2.5. Look in the terminal window of the domain controller. You should see a log entry showing the slave **host3** connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 36) WFLYHC0019: Registered remote slave host "host3", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

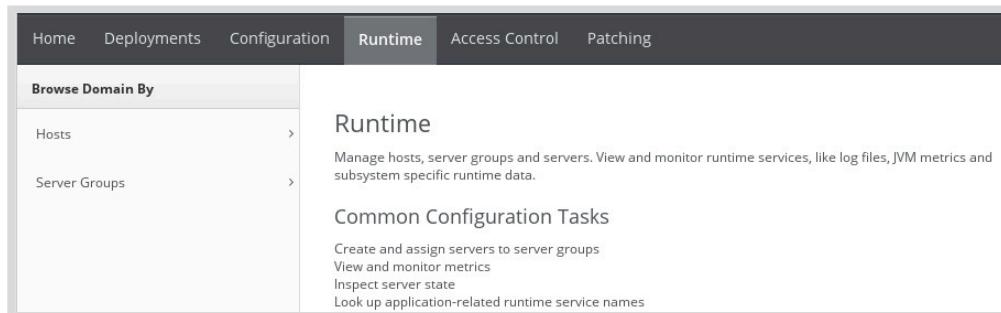
You have now started a managed domain with the domain controller running on **machine1** and two slaves **host2** and **host3** that are managed by the domain controller. In the next step, you will log in to the management console and set up server groups.

- 3. To configure server groups in a managed domain, you need to log in to the management console.

Navigate to <http://172.25.250.254:9990/>, which is the location of the management console. You should be prompted to log in. Login as the **jbossadm** user with password **JBoss@RedHat123**.

- 4. **Server groups** and **servers** can be created and managed in the **Runtime** section of the management console.

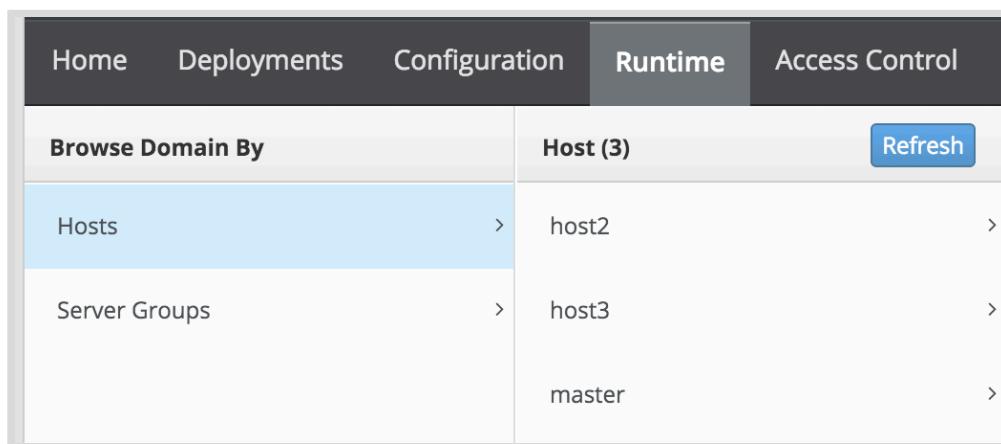
On the management console, click **Runtime** in the top navigation bar.



The screenshot shows the Management Console interface. The top navigation bar has tabs: Home, Deployments, Configuration, **Runtime**, Access Control, and Patching. The Runtime tab is selected. On the left, there's a sidebar titled "Browse Domain By" with options: Hosts and Server Groups. The main content area is titled "Runtime" and describes managing hosts, server groups, and servers. It includes sections for "Common Configuration Tasks" like creating and assigning servers to groups, viewing metrics, inspecting state, and looking up runtime service names.

- 5. A managed domain consists of a number of **hosts** which can run EAP 7 instances called **Servers**.

Click **Hosts** in the left navigation sidebar to view the **hosts** in the managed domain and verify that you can see three hosts called **master**, **host2**, and **host3** respectively.



The screenshot shows the Management Console interface. The top navigation bar has tabs: Home, Deployments, Configuration, **Runtime**, and Access Control. The Runtime tab is selected. On the left, there's a sidebar titled "Browse Domain By" with options: Hosts and Server Groups. The main content area shows a table with one row. The first column contains "Hosts" and "Host Groups". The second column contains "Host (3)" and a "Refresh" button. The third column lists three hosts: "host2", "host3", and "master", each with a right-pointing arrow.

Browse Domain By	Host (3)	Refresh
Hosts	host2	>
Server Groups	host3	>
	master	>

- 6. A **server group** is a logical group of EAP 7 **servers (instances)** that are managed together as a single unit. The server instances may be running on different **host** machines

and it is the job of the **domain controller** to ensure that the configuration across the managed domain is synchronized.

Click **Server Groups** in the left navigation sidebar to view the **server groups** in the managed domain and verify that you can see two server groups named **main-server-group** and **other-server-group**.

Browse Domain By		Server Group (2)	Add
Hosts	>	<input type="text"/>	
Server Groups	>	main-server-group	>
		other-server-group	>

- ▶ 7. You will be creating two new server groups, but before you do that, delete the existing servers and server groups.
- 7.1. Click **main-server-group** in the **Server Group** column to view the **servers** in this group and verify that you can see a single server named **server-one** defined. Before deleting this server, you need to ensure that it is stopped. Click the drop-down button next to **server-one** and click **Stop** to stop this server.

Browse Domain By		Server Group (2)	Add	Server (1)	Add	Monitor
Hosts	>	<input type="text"/>		<input type="text"/>		JVM
Server Groups	>	main-server-group	View ▾	server-one (host2)	View ▾	Environment
		other-server-group	>			Remove Copy Stop Suspend Reload Restart

- 7.2. Click **Confirm** in the confirmation dialog that pops up. After a few seconds, you will see a message in green saying that the server has been successfully stopped.
- 7.3. Click the drop-down button next to **server-one** again and click **Remove** to remove this server. Now, click **Confirm** in the confirmation dialog that pops up. After a few seconds, you will see a message in green saying that the server has been successfully removed.
- 7.4. Repeat this process to stop and remove **server-two** and **server-three** from the **other-server-group**. Ensure that there are no servers defined in a server group, before deleting them.

- 7.5. To delete the **main-server-group**, click the drop-down button next to **main-server-group** and click **Remove** to remove this server group. Click **Confirm** in the confirmation dialog that pops up. After a few seconds, you will see a message in green saying that the server group has been successfully removed.

Repeat the process to remove the **other-server-group** from the managed domain. You should now have no server groups or servers defined in the managed domain.

- 8. After you have deleted the existing server groups, you can create new server groups using the management console.

- 8.1. Click **Add** in **Server Group** column of the management console to add a new server group. This brings up the **Create Server Group** dialog window.

- 8.2. To add a new server group, you need to provide three key attributes as shown in the window. A unique **Name** for the server group, a valid **Profile** from the list of profiles defined in the **domain.xml** configuration file which control the subsystems that are active in a given profile and a **Socket Binding** which defines a set of network ports that will be used by servers running within the server group.

Add a new server group using the details below:

- **Name: Group1**

- **Profile:** default
- **Socket Binding:** standard-sockets

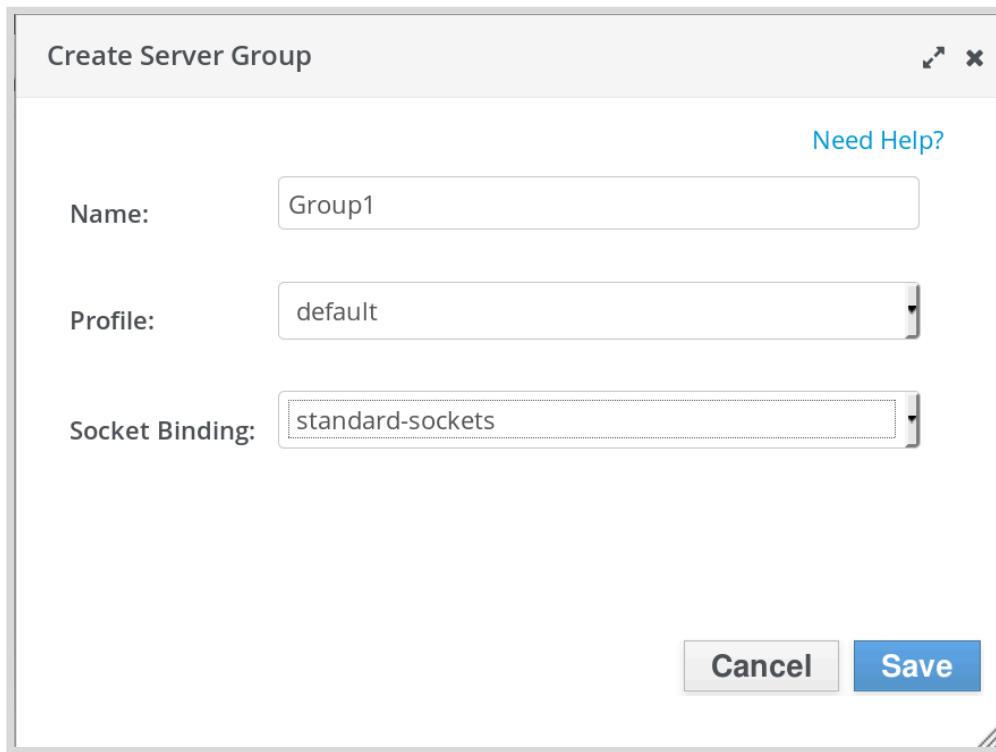
Click **Save** to create the server group **Group1**.

Create Server Group

Need Help?

Name:	Group1
Profile:	default
Socket Binding:	standard-sockets

Cancel **Save**

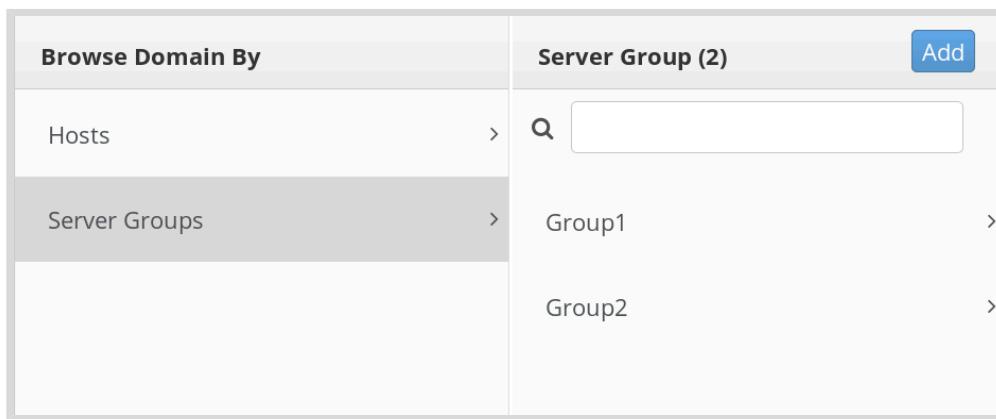


- 8.3. You can create multiple server groups, each with its own profile and socket binding.
Create another server group called **Group2** with the following details:

- **Name:** Group2
- **Profile:** full
- **Socket Binding:** full-sockets

- 9. You should now have two server groups called **Group1** and **Group2** defined in your managed domain.

Browse Domain By	Server Group (2)	Add
Hosts >	<input type="text"/>	
Server Groups >	Group1 >	
	Group2 >	



► **10. Clean Up.**

10.1. Press **Ctrl+C** to stop the domain controller and the two host controllers.

10.2. This concludes the guided exercise.

Configuring Servers

Objectives

After completing this section, students should be able to:

- Configure a server within a host controller.
- Describe the different attributes of a server configuration that are configured in the **host.xml** file of the host controller.
- Describe the different options available to create and manage servers in a managed domain.
- Describe how to shut down servers, server groups and the entire managed domain from the EAP CLI.

Servers definition

A server is a logical runtime instance of EAP running within a managed domain. A server must belong to a server group, even if the group only contains a single server. The individual servers are managed by the respective host controllers which communicate with the domain controller and ensure that all the servers within a managed domain have the same configuration.

Each server belongs to a unique *server group* and runs within a *host*. A host can run multiple servers simultaneously using the EAP provided concept of a **port-offset** to ensure that there are no network port or socket clashes at runtime.

Servers are configured in the **host.xml** file on the individual host controllers, which run on a physical server (or VM).

Server configuration

Servers in a managed domain are defined in the **<servers>** section of **host.xml** on each host controller. The child element **<server>** within the **<servers>** parent element can be used to define a server. A sample **<server>** definition follows:

```

<servers>
    <server name="serverA" ❶ group="main-server-group" ❷>
        <socket-bindings socket-binding-group="standard-sockets" ❸ />
    </server>
    <server name="serverB" group="other-server-group" auto-start="true" ❹>
        <jvm name="default" ❺>
            ...
        </jvm>
        <socket-bindings socket-binding-group="standard-sockets" port-
offset="350" ❻ />
    </server>
</servers>

```

- ❶ The **name** attribute is required and is the name of the server. It must be unique within the host.

- ② The **group** attribute is required and must reference a valid server group name defined previously in **domain.xml**.
- ③ The **socket-binding-group** attribute references a valid **socket-binding-group** value defined in **domain.xml**.
- ④ The **auto-start** attribute defaults to **true** and causes the server to automatically start when the host controller is started or restarted. It can be set to **false** if you want to disable server auto-start on the host, in which case you need to manually start the server if a host controller is started or restarted.
- ⑤ The **<jvm>** element configures the JVM settings for the server. Any values defined here within the **<server>** element will override the **jvm** settings from either the host or server group level. JVM memory allocation in EAP will be discussed later in the course.
- ⑥ The **port-offset** value is used to avoid network port clashes when multiple servers are defined on a host. In this case, **serverB** will run with an offset value of 350 from the base value defined for this profile. For example, on this host, the Undertow web server component will run on port 8080 for serverA because it has no port-offset defined, and for serverB, the web server will be running on port 8430 (for example, 8080 + 350).

**Note**

A host can have any number of servers. To avoid port conflicts, use the **port-offset** attribute of **<socket-bindings>**.

**Note**

Notice the **<server>** element does not contain any information about deployed applications. This is because all deployments for a server are configured at the server group level in the domain configuration file **domain.xml**.

Managing servers using the EAP management console

Servers in a managed domain can be created and managed in the **Hosts** section under the **Runtime** tab of the management console.

Browse Domain By	Host (3)	Server (0)
Hosts	host2 host3	Server (0) No Items!
Server Groups	master	

Figure 5.16: Runtime view

To add a new server, you need to provide three key attributes; a unique **Name** for the server, a valid **server group** from the list of server groups defined in the **domain.xml** configuration file, and a **port offset** value in case there are multiple servers running on this host.

Create New Server Configuration

Create new server

Name:

Host: host2

Server Group: Group1

Port Offset:

Auto Start?:

Need Help?

Cancel Save

Figure 5.17: Creating a new server

To remove a server from a host, ensure that the server is stopped.

Home Deployments Configuration Runtime Access Control Patching

Browse Domain By Host (3) Refresh Server (1) Add ▾

Hosts >	host2	JVM ▾ >	Server (1)	Add ▾
Server Groups >	host3	>	serverA (Group1)	View ▾
	master	>		Remove Copy Start Suspend Reload Restart

Server C
A "Server" r
separate JV
responsible
is

Figure 5.18: Server options - Remove

Individual servers can be started, stopped, restarted, reloaded, and suspended by clicking **View** drop-down menu next to the server in the **Hosts** section under the **Runtime** tab of the management console.

Configuring and managing servers using the JBoss EAP CLI

Servers can also be created and managed using the JBoss EAP CLI. The advantage of this approach is that the steps to create and manage the servers can be scripted as part of an automated workflow and it is faster to create multiple servers in batched mode. This approach also has the option of atomically committing or rolling back multiple steps in batch mode, thereby

Chapter 5 | Configuring Servers in a Managed Domain

ensuring the integrity of the `host.xml` configuration file in case of an error during execution of the CLI commands.

The servers in a managed domain are configured in the `/host` namespace of the JBoss EAP CLI. Commands exist to create, view, start, stop, restart, reload, suspend, and remove servers under this namespace. To add a new server, you can run the following command:

```
[domain@workstation:9990 /] /host=host2/server-config=serverA:add\
(auto-start=true,group=Group1,socket-binding-port-offset=100)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```

To start a single server, run the following command:

```
[domain@workstation:9990 /] /host=host2/server-config=serverA:start
{
    "outcome" => "success",
    "result" => "STARTING"
}
```

To stop a single server, run the following command:

```
[domain@workstation:9990 /] /host=host2/server-config=serverA:stop
{
    "outcome" => "success",
    "result" => "STOPPING"
}
```

All the servers in a server group can be started by running the following command:

```
[domain@workstation:9990 /] /server-group=Group1:start-servers(blocking=true)
{
    "outcome" => "success",
    ...
}
```

Similarly, to stop all the servers in a server group, run the following command:

```
[domain@workstation:9990 /] /server-group=Group1:stop-servers(blocking=true)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

**Note**

The **blocking=true** property ensures that control returns to the CLI prompt only after all servers in the server group have been successfully started or stopped. The default behavior is for control to return immediately to the CLI prompt and the servers are started or stopped asynchronously in the background.

To remove a server, you must first ensure that the server is stopped. To remove the server, run the following command:

```
[domain@workstation:9990 /] /host=host2/server-config=serverA:remove()  
{  
    "outcome" => "success",  
    ...  
}
```

In EAP 7 it is possible to suspend a server using the **suspend** operation. Any server in a suspended mode will not accept new requests from any client until the server is resumed. A suspended server can begin serving requests immediately by running the **resume** operation.

A graceful shut down support was added. It is different from an ordinary shut down because the server is suspended before shutting down. To invoke this behavior a **timeout** parameter must be passed to the **shutdown**, **stop**, or **stop-servers** operations. The server waits until the timeout for all requests to finish before shutting down.

You can also gracefully shut down a host controller from the EAP CLI. To shutdown a host controller, first ensure that all the servers running on this host are stopped and then run the following command:

```
[domain@workstation:9990 /] /host=host2:shutdown(timeout=1000)  
{  
    "outcome" => "success",  
    "result" => undefined  
}
```

In a similar fashion, it is possible to gracefully shutdown the domain controller from the EAP CLI. To shutdown the domain controller (assuming it is named **master**), run the following command:

```
[domain@workstation:9990 /] /host=master:shutdown(timeout=1000)  
{  
    {"outcome" => "success"}  
[domain@localhost:9990 /]  
The controller has closed the connection.  
[disconnected /]  
}
```

► Guided Exercise

Configuring Servers

In this lab, you will configure servers in a managed domain using the JBoss EAP Command Line Interface (CLI) and deploy applications on the managed domain.

Resources	
Files	/home/student/JB248/labs/domain /home/student/JB248/labs/host
Application URL	NA

Outcomes

You should be able to configure and start a managed domain with two server groups and four servers and deploy applications on them.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and the previous lab set up a domain controller at **/home/student/JB248/labs/domain** and two hosts at **/home/student/JB248/labs/host**:

```
[student@workstation ~]$ lab domain-servers setup
```

- ▶ 1. You will be simulating running three host machines on the workstation in this guided exercise. In order to not disturb the existing EAP 7 installation at **/opt/jboss-eap-7.0**, the setup script has created a domain controller in the **machine1** folder under the **/home/student/JB248/labs/domain** directory and two folders under the **/home/student/JB248/labs/host** directory called **machine2** and **machine3** that simulate host controllers. It has also set up a preconfigured managed domain with the two hosts and the domain controller. There are two server groups predefined in the managed domain. You will start the managed domain and define four new servers using the JBoss EAP 7 CLI.
- ▶ 2. Start the domain controller (**machine1**) and the two slave hosts (**machine2** and **machine3**).
 - 2.1. The domain controller should be started first, followed by the slaves, because it will provide all the configuration for the host controllers.

Start the domain controller using a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \  
--host-config=host-master.xml
```

2.2. Start the host controller on **machine2**.

Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window on your workstation to start **machine2** using the **host-slave.xml** configuration file:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine2/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

2.3. Look in the terminal window of the domain controller. You should see a log entry showing the slave **host2** connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 36) WFLYHC0019: Registered remote slave host "host2", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

2.4. Start **machine3** to join the managed domain.

Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window on your workstation to start **machine3** using the **host-slave.xml** configuration file:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine3/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

2.5. Look in the terminal window of the domain controller. You should see a log entry showing the slave **host3** connecting:

```
[Host Controller] 11:42:16,348 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 36) WFLYHC0019: Registered remote slave host "host3", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

You have now started a managed domain with the domain controller running on the host **machine1** and two slaves, **machine2** and **machine3** that are managed by the domain controller. In the next step, you will access the EAP 7 CLI and set up the servers.

► 3. Access the JBoss EAP CLI and configure the servers.

3.1. In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh --connect \  
--controller=172.25.250.254:9990
```

- 3.2. Before creating the servers, you need to verify that the server groups have been created. Verify that server groups **Group1** and **Group2** have been created and are running. Run the following commands in the EAP CLI:

```
[domain@workstation:9990 /] /server-group=Group1:read-resource  
{  
    "outcome" => "success",  
    "result" => {  
        "profile" => "default",  
        "socket-binding-group" => "standard-sockets",  
        ...  
    }  
}
```

```
[domain@workstation:9990 /] /server-group=Group2:read-resource  
{  
    "outcome" => "success",  
    "result" => {  
        "profile" => "full",  
        "socket-binding-group" => "full-sockets",  
        ...  
    }  
}
```

- 3.3. Define a new server called **server-one** on Host **host2** with a port offset value of **100** and assign it to the **Group1** server group.

```
[domain@workstation:9990 /]  
/host=host2/server-config=server-one:add\  
(auto-start=true,group=Group1,socket-binding-port-offset=100)  
{  
    "outcome" => "success",  
    "result" => {  
        ...  
    }  
}
```

- 3.4. Define another server called **server-two** on Host **host2** with a port offset value of **200** and assign it to the **Group2** server group.

```
[domain@workstation:9990 /]  
/host=host2/server-config=server-two:add\  
(auto-start=true,group=Group2,socket-binding-port-offset=200)  
{
```

```
"outcome" => "success",
"result" => {
    ...
}
```

- 3.5. Define a third server called **server-three** on host **host3** with a port offset value of **300** and assign it to the **Group1** server group.

```
[domain@workstation:9990 /] /host=host3/server-config=server-three:add\
(auto-start=true,group=Group1,socket-binding-port-offset=300)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```

- 3.6. Define a fourth server called **server-four** on host **host3** with a port offset value of **400** and assign it to the **Group2** server group.

```
[domain@workstation:9990 /] /host=host3/server-config=server-four:add\
(auto-start=true,group=Group2,socket-binding-port-offset=400)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```



Note

The **auto-start** property is set to true to ensure that the server starts up automatically when the host controller on which it is configured is started or restarted. If you do not want to start the server automatically, then set the property to **false**.

- 4. Start the servers.

- 4.1. Although you have defined and created the servers, they are not yet running. Start the servers in the managed domain.

```
[domain@workstation:9990 /] /host=host2/server-config=server-one:start
{
    "outcome" => "success",
    "result" => "STARTING"
}
```

Monitor the console window of **machine2** and verify that **server-one** is starting up:

```
[Host Controller] 14:02:31,089 INFO [org.jboss.as.host.controller] (Host Controller Service Threads - 44) WFLYHC0023: Starting server server-one
14:02:31,101 INFO [org.jboss.as.process.Server:server-one.status]
(ProcessController-threads - 4) WFLYPC0018: Starting process 'Server:server-one'
...
```

Note that because you defined server-one with a port offset value of 100, the ports are bound accordingly:

```
[Server:server-one] 14:02:34,649 INFO [org.wildfly.extension.undertow] (MSC service thread 1-2) WFLYUT0012: Started server default-server.
[Server:server-one] 14:02:34,651 INFO [org.wildfly.extension.undertow] (MSC service thread 1-2) WFLYUT0018: Host default-host starting
[Server:server-one] 14:02:34,794 INFO [org.wildfly.extension.undertow] (MSC service thread 1-4) WFLYUT0006: Undertow HTTP listener default listening on 172.25.254.250:8180
```

- 4.2. Start **server-two**, **server-three**, and **server-four** and verify that the HTTP listener is bound to ports **8280**, **8380**, and **8480** respectively.

```
[domain@workstation:9990 /] /host=host2/server-config=server-two:start
{
    "outcome" => "success",
    "result" => "STARTING"
}
[domain@workstation:9990 /] /host=host3/server-config=server-three:start
{
    "outcome" => "success",
    "result" => "STARTING"
}
[domain@workstation:9990 /] /host=host3/server-config=server-four:start
{
    "outcome" => "success",
    "result" => "STARTING"
}
```



Note

Instead of starting servers one by one, you can start all of the servers in a server group with one command:

```
[domain@workstation:9990 /] /server-group=Group1:start-servers\
(blocking=true)
{
    "outcome" => "success",
    ...
}
```

► 5. Shut down of Servers, Server Groups and Host Controllers

- 5.1. You can manage the life cycle (start, stop, restart, suspend, and so on) of servers, server groups and host controllers directly from the JBoss EAP CLI. This offers flexibility in terms of managing the domain from a single access point as well as improved security because that you do not need to provide SSH login access to the host machines in the managed domain.

- 5.2. Stop **server-one** using the JBoss EAP CLI:

```
[domain@workstation:9990 /] /host=host2/server-config=server-one:stop
{
    "outcome" => "success",
    "result" => "STOPPING"
}
```

Verify that the **example** application is **NOT** accessible at <http://172.25.250.254:8180/example>. The example application should still be accessible at <http://172.25.250.254:8380/example> because **server-three** is still running.

- 5.3. Stop **server-three** using the JBoss EAP CLI:

```
[domain@workstation:9990 /] /host=host3/server-config=server-three:stop
{
    "outcome" => "success",
    "result" => "STOPPING"
}
```

Verify that the **example** application is now **NOT** accessible at URL <http://172.25.250.254:8380/example>.

- 5.4. Stop all the servers in **Group2** using the JBoss EAP CLI:

```
[domain@workstation:9990 /] /server-group=Group2:stop-servers
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

Verify that the **version** application is **NOT** accessible at both <http://172.25.250.254:8280/version> and <http://172.25.250.254:8480/version> because you have shut down the entire server group.

- 5.5. Stop both host controllers (**host2** and **host3**) on **machine2** and **machine3** using the JBoss EAP CLI. Shut down the **machine2** host controller first:

```
[domain@workstation:9990 /] /host=host2:shutdown
{
    "outcome" => "success",
    "result" => undefined
}
```

Monitor the console window of **machine2** and verify that the host controller has been shut down.

- 5.6. Stop the host controller on **machine3** using the JBoss EAP CLI:

```
[domain@workstation:9990 /] /host=host3:shutdown
{
    "outcome" => "success",
    "result" => undefined
}
```

Monitor the console window of **machine3** and verify that the host controller has been shut down.

▶ 6. Clean Up

- 6.1. After all the host controllers are shut down, shutdown the domain controller. Press **Ctrl+C** to stop the domain controller (Alternatively, you can shut down the domain controller using the JBoss EAP CLI as demonstrated in the previous step).
- 6.2. This concludes the guided exercise.

Deploying Applications on a Managed Domain

Objectives

After completing this section, students should be able to:

- Describe the different options available for application deployment on an EAP managed domain.

Application deployment on a managed domain

Deploying an application to a managed domain requires a different process from that used to deploy in standalone mode. You cannot select a specific server to deploy an application to. All applications must be deployed to a group, and all servers that belong to the specified group then deploy that application. The host controller communicates with the domain controller and ensures that deployments are synchronized across all the servers in the managed domain that are part of the server group.

There are two ways to deploy an application using the domain mode:

- The Management console
- The CLI tool

Unlike the standalone mode, it is not possible to deploy using the **deployment-scanner** subsystem. The reason for this is that is not possible to guarantee that the application will be available for the entire group (manually deployed for each server).

Application deployment using the management console

Application deployment using the management console is a two-step process, where the application (EAR, WAR, JAR, and so on) is first uploaded to the *content repository* and then deployed to selected server groups in the managed domain. The content repository is a folder on the domain controller located at **DOMAIN_BASE_DIR/domain/content**, where **DOMAIN_BASE_DIR** is the base directory where the configuration files of the managed domain are stored.

Using the **Content Repository** menu, click **Add** to upload a new deployment to the repository.

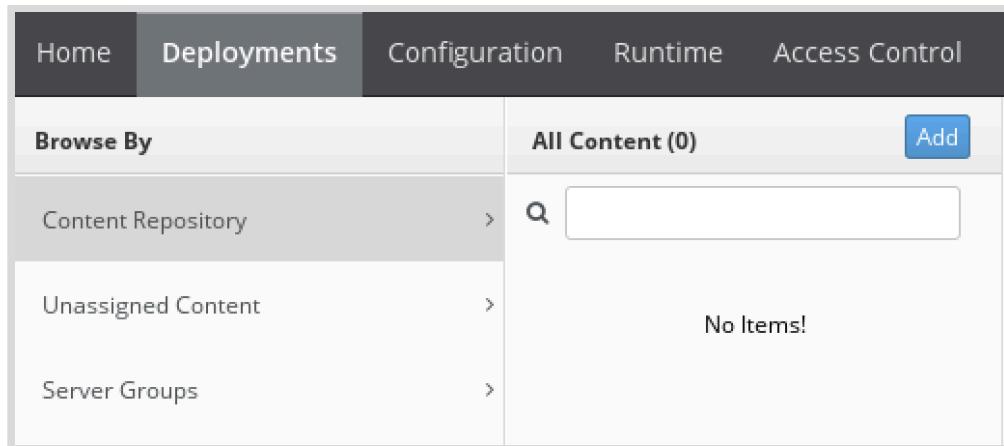


Figure 5.19: EAP Managed Console - empty Content Repository page

A wizard will start to upload a new application to the repository. The first step of the wizard asks for the type of the deployment. Two options are available:

- **Upload a new deployment:** Using this option, an application is uploaded and is available in the repository. A reference to the application is created in the `domain.xml` file.
- **Create an unmanaged deployment:** Using this option, a path should be specified to where the application archive is available. The deployment content will not be uploaded to the repository and it will be deployed directly from the specified location.



Note

To deploy an application using the unmanaged method, the application archive needs to be available in the same path for all hosts. This is NOT a recommended practice because it is easy to end up having different application files in some hosts. The unmanaged method exists for application that must be deployed as exploded applications. Also notice that such applications are in violation of the JEE specs.

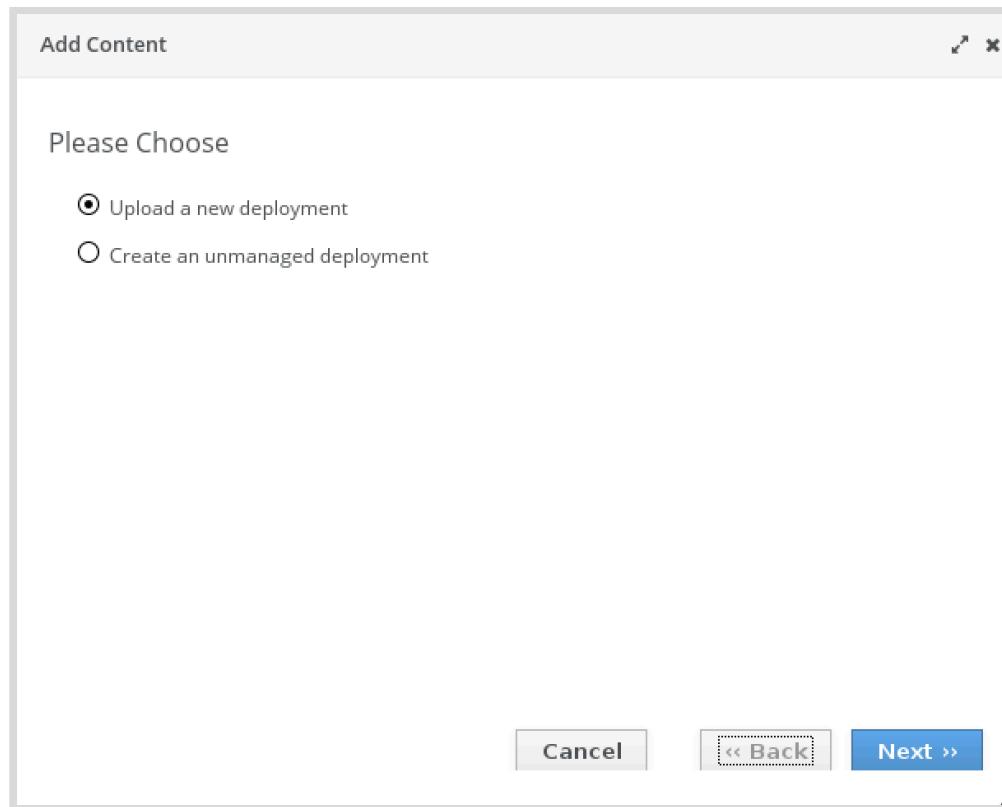


Figure 5.20: EAP Managed Console - Managed and Unmanaged deployments

Using the **Upload a new deployment** option, the second step asks for the file to upload. Click **Browse** to navigate to and select the required file.

The next step on the wizard has two options to be defined:

- **Name:** The identifier of the deployment. This must be a unique value across all deployments.
- **Runtime Name:** Defines the context of the application. The context is the name of the application in the runtime environment. If a deployment has a runtime name defined as myapp.war, it will be available on http://server:port/myapp.

The applications added to the **Content Repository** can then be assigned to one or more server groups using the **Assign** button next to the deployments.

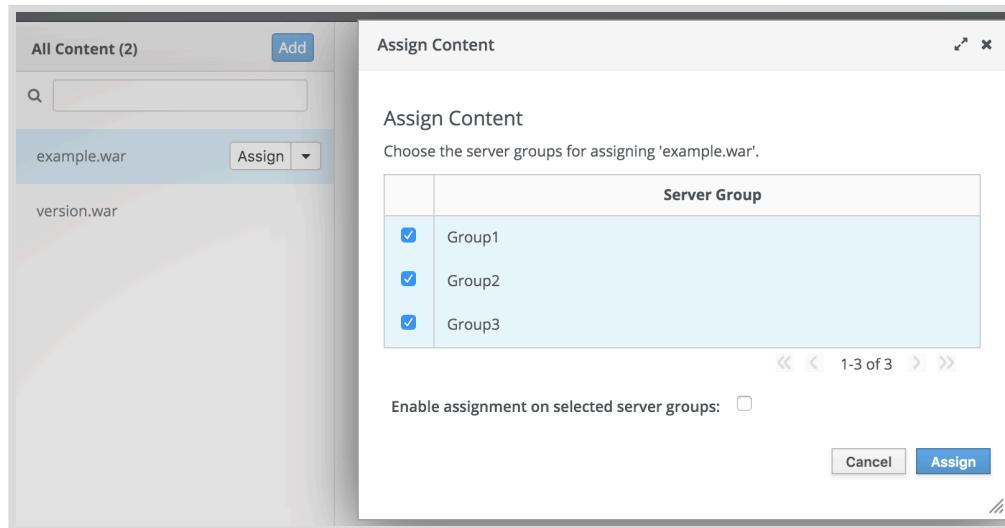


Figure 5.21: Assigning a content

Another approach is to choose a server group from the **Server Group** section of the **Deployments** tab in the management console and select applications that should be deployed to this server group.

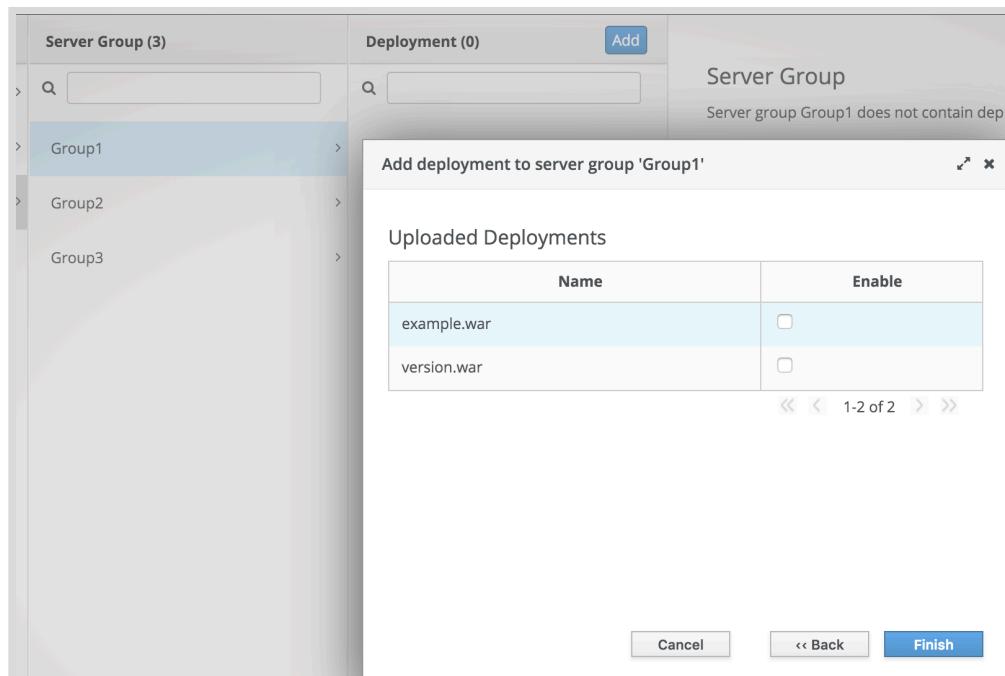


Figure 5.22: Adding a deployment to a server group

Applications can be unassigned from the server groups they were assigned to previously to undeploy applications from server groups.

The screenshot shows the JBoss EAP management console interface. The top navigation bar includes Home, Deployments (selected), Configuration, Runtime, Access Control, and Patching. Under the Deployments tab, the 'Browse By' section shows 'Content Repository' and 'Server Groups'. The 'Unassigned Content' section displays two entries: 'example.war' and 'version.war'. A context menu is open over 'example.war', with the 'Assign' option highlighted. Other options in the menu are 'Unassign', 'Replace', and 'Remove'.

Figure 5.23: Unassigning a deployment from all server groups

Alternatively, applications can be unassigned or disabled from the **Server Group** section of the **Deployments** tab in the management console.

The screenshot shows the JBoss EAP management console interface. The top navigation bar includes Home, Deployments (selected), Configuration, Runtime, Access Control, and Patching. Under the Deployments tab, the 'Browse By' section shows 'Content Repository' and 'Server Groups'. The 'Server Groups' section lists 'Group1', 'Group2', and 'Group3'. Under 'Group1', the application 'example.war' is listed. A context menu is open over 'example.war', with the 'View' option highlighted. Other options in the menu are 'Disable' and 'Unassign'. A note on the right side states 'example.war' and 'There's no server running in server' with a red warning icon.

Figure 5.24: Disabling a deployment in a server group

Application deployment using the JBoss EAP CLI

Deploying applications using the EAP CLI gives administrators the benefit of a command-line interface with the ability to create and run deployment scripts. An administrator can use this scripting ability to configure specific application deployment and management scenarios. An administrator can manage the deployments for an entire network of servers when running in a managed domain from a single point of control.

The advantage of this approach is that the steps to deploy and manage the applications can be scripted as part of an automated work flow and it is faster to deploy multiple applications to a managed domain in batched mode.

The JBoss EAP CLI offers the **deploy** and **undeploy** commands in the default top level namespace for deploying and undeploying applications in a managed domain. One of the advantages of the CLI approach is that an application can be deployed to *all* server groups using a single command:

```
[domain@workstation:9990 /] deploy /path/to/example.war --all-server-groups
```

To deploy an application to specific server groups, provide a comma-separated list of server groups as arguments to the **deploy** command:

```
[domain@workstation:9990 /] deploy /path/to/example.war \
--server-groups=Group1,Group2,Group3
```

To undeploy an application from server groups, the EAP CLI provides a convenient **--all-relevant-server-groups** option since the CLI is already aware of the deployments and keeps track of the server groups that the application has been assigned to:

```
[domain@workstation:9990 /] undeploy example.war --all-relevant-server-groups
```

An explicit list of server groups can be provided from which to undeploy an application, if undeployment of the application should not happen from all the relevant server groups:

```
[domain@workstation:9990 /] undeploy example.war --server-groups=Group1,Group2
```

Undeploy an application using the CLI

The **undeploy** command undeploys an application from the domain. To undeploy and remove the application from the entire domain, the **name** argument and the **--all-relevant-server-groups** option should be declared:

```
[domain@172.25.250.254:9990 /] undeploy myapp.war --all-relevant-server-groups
```

In the previous example, the application is also removed from the content repository. To keep the application in the repository, use the **--keep-content** argument:

```
[domain@172.25.250.254:9990 /] undeploy myapp.war --all-relevant-server-groups \
--keep-content
```

It is possible to specify only specific groups with the **--server-groups** argument:

```
[domain@172.25.250.254:9990 /] undeploy myapp.war --server-groups=main-server-
group \
--keep-content
```



Note

If the application is assigned to two or more groups, the **--keep-content** argument is required because the application content cannot be removed.

► Guided Exercise

Deployment on a Managed Domain

In this lab, you will deploy applications on the managed domain that was created earlier.

Resources	
Files:	/home/student/JB248/labs/host /home/student/JB248/labs/domain-deploy
Application URL:	http://172.25.250.254:8180/ http://172.25.250.254:8280/ http://172.25.250.254:8380/ http://172.25.250.254:8480/

Outcomes

You should be able to deploy applications on the managed domain.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and the previous lab set up a domain with two host servers at **/home/student/JB248/labs/host**:

```
[student@workstation ~]$ lab domain-deploy setup
```

► 1. Start The Domain and Hosts on the Workstation VM

- 1.1. Use the following command to start the domain controller that was configured in a previous exercise:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \
--host-config=host-master.xml
```

- 1.2. Open a new terminal window and start **host2** created in the previous exercise by using the following command:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine2/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254
```

13. Open a new terminal window and start **host3** created in the previous exercise by using the following command:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine3/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

Wait for the domain controller and two host controllers to start before proceeding.

► 2. Deploy an Application to the Group1 Server Group

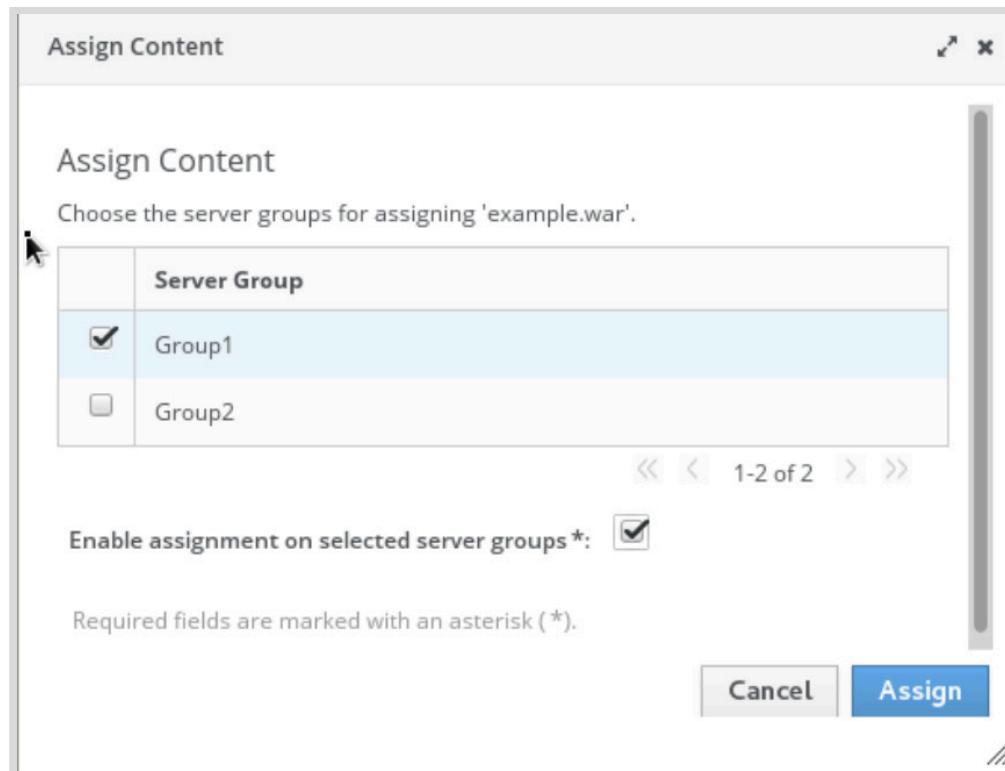
- 2.1. On the workstation VM, open the management console by pointing a web browser to 172.25.250.254:9990. The administrator user name is **jbossadm** and the password is **JBoss@RedHat123**.
- 2.2. Click **Deployments** at the top of the page and then click **Content Repository** in the first column.
- 2.3. Click **Add**, select **Upload a new deployment** and click **Next**. Click **Browse** and select the WAR file **/home/student/JB248/labs/domain-deploy/example.war** to upload.
- 2.4. On the **Verify Upload** screen, leave the **Name** and **Runtime Name** as **example.war**, and then click **Finish**.



Note

The **Runtime Name** refers to the application name, without the war extension, where the example app will be accessed.

- 2.5. Click **example.war** in the second column and then the **Assign** button.
- 2.6. Select **Group1** to deploy **example.war** to the **Group1** server group. Select **Enable assignment on selected server groups** and then click **Assign**:



► 3. Verify the Deployment

- 3.1. Look in the terminal window of **host2** and **host3**. You should see a message similar to the following stating that **example.war** was deployed:

```
[Server:server-one] 22:11:10,423 INFO  [org.jboss.as.server] (ServerService Thread Pool -- 75) WFLYSRV0010: Deployed "example.war" (runtime-name : "example.war")
```

- 3.2. Navigate to **server-one** at <http://172.25.250.254:8180/example/> and **server-three** at <http://172.25.250.254:8380/example/>. You should see the "Welcome to EAP 7" page of the **example** application.
- 3.3. Similarly, navigate to **server-two** at <http://172.25.250.254:8280/example/> and **server-four** at <http://172.25.250.254:8480/example/>. You should get a 404 error, because the **example** application is not deployed on **Group2** server group.
- 3.4. Open the file **domain.xml** in the **/home/student/JB248/labs/domain/machine1/domain/configuration** folder. Notice that **example.war** now appears as an available deployment in the **<deployments>** section. Also notice that **example.war** appears in the **<deployments>** section of the **Group1** in the **<server-group>** section.

► 4. Deploy an Application to the Group2 Server Group

- 4.1. Go back to the **Deployments** page of the Management Console of your Domain. Click **Content Repository** and then click **Add**.

- 4.2. Select **Upload a new deployment** on the first screen and then select **Next**. Click **Browse** and upload the file **/home/student/JB248/labs/domain-deploy/version.war** and click **Next**.

On the **Verify Upload** screen, leave the **Name** and **Runtime Name** as **version.war**, then click **Finish**.

- 4.3. Select the **version.war** from the **Content Repository** menu and click **Assign**. Choose the **Group2** server group, which contains your **server-two** and **server-four** server instances. Be sure select **Enable assignment on selected server groups** on the **Group2**.

► 5. Verify the Deployment

- 5.1. Look in the terminal windows of both **host2** and **host3**. The **version.war** application was deployed onto **server-two** on **host2** and **server-four** on **host3**. The output on **host2** should look similar to the following:

```
[Server:server-two] 06:37:11,608 INFO [org.jboss.as.server] (ServerService Thread Pool -- 78) WFLYSRV0010: Deployed "version.war" (runtime-name : "version.war")
```

- 5.2. Navigate to **server-two** at <http://172.25.250.254:8280/version/> and **server-four** at <http://172.25.250.254:8480/version/>. The Version Application should appear at these links.
- 5.3. Navigate to **server-one** at <http://172.25.250.254:8180/version/> and **server-three** at <http://172.25.250.254:8380/version/>. You should get a 404 error.

► 6. Undeploy the Applications

- 6.1. Go to the **Deployments** page of the Management Console for your domain controller located at <http://172.25.250.254:9990> with the user name **jbossadm** and password **JBoss@RedHat123**.
- 6.2. Click the **Server Groups** tab in the first column.
- 6.3. Notice the deployed applications are listed by server group. Click the **Group1** in the **Server Groups** list.

The screenshot shows the Red Hat JBoss Enterprise Application Platform interface. The top navigation bar includes Home, Deployments (which is selected), Configuration, Runtime, Access Control, and Patching. The Deployments page displays a tree structure under 'Browse By': Content Repository, Unassigned Content, and Server Groups. Under Server Groups, 'Group1' contains 'example.war' and 'Group2' contains nothing. On the right side, a detailed view for 'example.war' is shown, including its status as 'Deployment is disabled', last enabled date (2017-09-06 04:27:21,430 EDT), last disabled date (2017-09-06 07:21:26,151 EDT), runtime name (example.war), and reference server information. A context menu is open over 'example.war' in 'Group1', with the option 'Deployment is disabled' highlighted.

- 6.4. Click on **Group1** under the **Server Group (2)** label.
- 6.5. Click **example.war** in the list of **Deployments**, click the arrow next to **View** and select **Disable** to disable the application. Click **Confirm** in the pop-up dialog window.

This screenshot shows the same interface as the previous one, but the context menu is now open over the 'example.war' entry in the main list. The options shown are 'View', a dropdown arrow, 'Disable', and 'Unassign'. The 'Disable' option is currently selected. The detailed view on the right remains the same, showing 'example.war' is still enabled.

- 6.6. View the **example.war** application on **server-one** in your browser by navigating to <http://172.25.250.254:8180/example/> and verify that you now get a 404 error.
- 6.7. Click the same arrow in the **Deployment** column and select **Unassign** and the deployment should disappear making **example.war** no longer assigned to the **Group1**.
- 6.8. Following the same steps, disable and remove **version.war** from the **Group2**. To verify, try navigating to <http://172.25.250.254:8280/version/> for **server-two** and to <http://172.25.250.254:8480/version/> for **server-four**. You should get 404 errors.

► 7. Remove Deployments from the managed domain

- 7.1. Click the **Content Repository** tab of the **Deployments** page.

- 7.2. Select **example.war** in the **Content Repository** list and click the arrow next to **example.war**. Select **Remove**. Then click **Confirm** when the confirmation window pops up.
 - 7.3. Similarly, remove **version.war** from the **Content Repository**.
- ▶ 8. Press **Ctrl+C** to stop the domain controller and the two host controllers.

This concludes the guided exercise.

► Lab

Configuring Servers in a Managed Domain

In this lab, you will configure servers and server groups in a managed domain and deploy the bookstore application into the managed domain.

Resources	
Files	/opt/domain /home/student/JB248/labs/managed-domain-servers
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore

Outcome

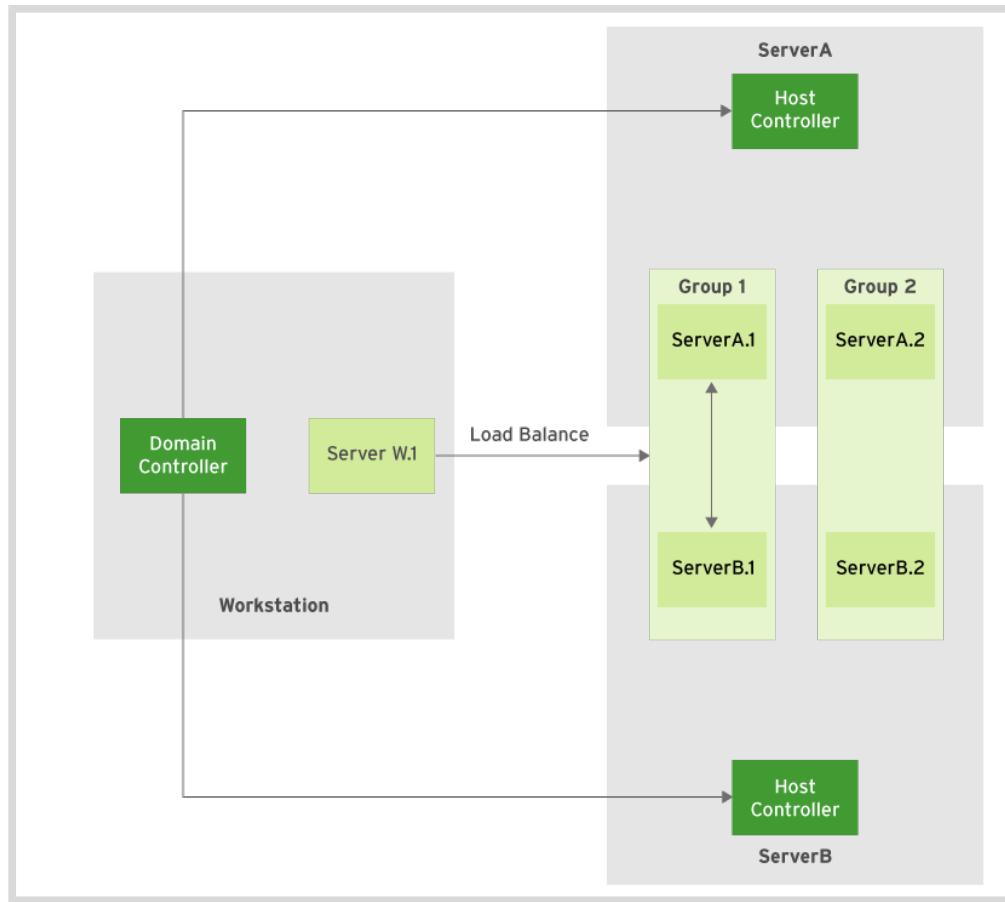
You should be able to configure an EAP 7 managed domain with two server groups and four servers and deploy the bookstore application.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been setup correctly:

```
[student@workstation ~]$ lab managed-domain-servers setup
```

1. An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively and the domain controller on **workstation** VM. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and then create new servers and server groups as per the diagram below:



As a final step, you will deploy the bookstore application WAR file on one of the newly created server groups. Bookstore is an application supporting high availability features and requires messaging subsystems and HA features from EAP. The **full-ha** profile of EAP must be used to deploy the application on EAP. You are free to choose either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

2. Briefly review the domain controller configuration on the **workstation**. Review the **/opt/domain/configuration/host-master.xml** file and the **/opt/domain/configuration/domain.xml** file to understand how the managed domain is configured. Verify that there are no server groups defined in the **domain.xml** file.
 3. Start the domain controller on **workstation**. Because the domain controller configuration files are kept in the **/opt/domain** folder on workstation, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)
- Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
4. The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

- 4.1. Briefly review the host controller configuration on **servera** and **serverb**. Review the **/opt/domain/configuration/host-slave.xml** file to understand how the hosts are configured. Verify that there are no servers defined on either host.
 - 4.2. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
 - 4.3. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
 - 4.4. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
5. Access the JBoss EAP CLI. In a new terminal window on the **workstation**, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```

6. By default, there are no server groups or servers defined in the managed domain (These were removed in the final lab of Chapter 4). To deploy the bookstore application (WAR file), you need to create server groups and define servers.

Create two new server groups named **Group1** and **Group2** in the managed domain.

- 6.1. The bookstore application uses the messaging subsystem and HA capabilities from EAP, and these features are available only in the **full-ha** profile. Create a new server group called **Group1** which is used to emulate a development environment with the following characteristics:
 - **Name: Group1**
 - **Profile: full-ha**
 - **Socket Binding Group: full-ha-sockets**
- 6.2. Create another server group called **Group2** to use for deploying bookstore in a production environment with the same characteristics as the development environment:
 - **Name: Group2**

- **Profile: full-ha**
 - **Socket Binding Group: full-ha-sockets**
- 6.3. Verify the server groups that you created in the steps above using either the management console or the JBoss EAP CLI.
7. You need to define four servers (two each on **servera** and **serverb**). Create the servers and assign them to the appropriate groups. Make sure that when you run multiple servers on a host, you configure the port offsets correctly to avoid port clashes. Also ensure that the servers are set to automatically start when the host controller is started or restarted.
- 7.1. Create a new server called **servera.1** on **servera** with the following characteristics:
- **Name: servera.1**
 - **Server Group: Group1**
 - **Socket Binding Port Offset: Leave at default value of 0**
 - **Auto Start: true**
- 7.2. Create a new server called **servera.2** on **servera** with the following characteristics:
- **Name: servera.2**
 - **Server Group: Group2**
 - **Socket Binding Port Offset: 100**
 - **Auto Start: true**
- 7.3. Create a new server called **serverb.1** on **serverb** with the following characteristics:
- **Name: serverb.1**
 - **Server Group: Group1**
 - **Socket Binding Port Offset: Leave at default value of 0**
 - **Auto Start: true**
- 7.4. Create a new server called **serverb.2** on **serverb** with the following characteristics:
- **Name: serverb.2**
 - **Server Group: Group2**
 - **Socket Binding Port Offset: 100**
 - **Auto Start: true**
8. You have now created four servers and associated them with the appropriate server groups. You will now start the newly-created servers. Instead of starting servers one-by-one, you can start all of the servers in a server group together.
- 8.1. Start the servers in **Group1**
- 8.2. Start the servers in **Group2**

- 8.3. Verify that the servers started successfully by looking at the console windows where you started the host controllers. Verify that there are no port clashes and that the ports are bound according to the port offset values defined in the previous steps.
- 8.4. Because a firewall is running on **servera** and **serverb**, you must unlock all ports used by **servera.1**, **servera.2**, **serverb.1**, and **serverb.2** for public access. To unlock them, run the following command in a new terminal window from **servera**:

```
[student@servera ~]$ sudo firewall-cmd --zone=public --add-port 8080/tcp \
--permanent
[student@servera ~]$ sudo firewall-cmd --zone=public --add-port 8180/tcp \
--permanent
[student@servera ~]$ sudo firewall-cmd --reload
```

- 8.5. To unlock the firewall ports on **serverb**, run the following commands in a new terminal window from **serverb**:

```
[student@serverb ~]$ sudo firewall-cmd --zone=public --add-port 8080/tcp \
--permanent
[student@serverb ~]$ sudo firewall-cmd --zone=public --add-port 8180/tcp \
--permanent
[student@serverb ~]$ sudo firewall-cmd --reload
```

- 8.6. Verify that you can access the default welcome page of all four servers you created in the above steps. Access the following URLs and verify that you can see the EAP 7 Welcome page:

- **servera.1**: <http://172.25.250.10:8080>
- **servera.2**: <http://172.25.250.10:8180>
- **serverb.1**: <http://172.25.250.11:8080>
- **serverb.2**: <http://172.25.250.11:8180>

9. You are now ready to deploy the bookstore application WAR file in the managed domain. The **bookstore.war** file is available in the **/tmp** folder on the **workstation**.

- 9.1. Application deployments in a managed domain are always done at the server groups level. Each application belongs to one or more server groups. Deploy the **bookstore.war** file to **Group1**.

Verify that the **bookstore.war** file is deployed on **servera.1** and **serverb.1** because they are part of the **Group1** server group. Monitor the console window of **servera** and **serverb** for any error messages or warnings.

- 9.2. Verify that you can access the bookstore application at <http://172.25.250.10:8080/bookstore> and <http://172.25.250.11:8080/bookstore>.

- 9.3. Verify that you can **NOT** access the bookstore application at <http://172.25.250.10:8180/bookstore> or <http://172.25.250.11:8180/bookstore> because these servers are part of **Group2** and you have not deployed the application on **Group2**.

10. Using either the administration console or the JBoss EAP CLI, undeploy the bookstore application and shut down the servers, server groups, host controllers, and the entire managed domain.
 - 10.1. Stop **servera.1**.
 - 10.2. Verify that the bookstore application is no longer accessible from **servera.1** but still accessible from **serverb.1**.
 - 10.3. Undeploy the bookstore application.
 - 10.4. Stop all servers in **Group1**. Verify that the bookstore application is no longer accessible.
 - 10.5. Stop the servers in **Group2**.
 - 10.6. Shutdown the host controller on **servera**. Observe the console window of **servera** and verify that the host controller has been shut down.
 - 10.7. Shutdown the host controller on **serverb**. Observe the console window of **serverb** and verify that the host controller has been shut down.
11. **Clean Up and Grading**
 - 11.1. Press **Ctrl+C** to stop the domain controller (Alternatively, you can shut down the domain controller using the JBoss EAP CLI).
 - 11.2. Run the following command from the **workstation** to grade the exercise:

```
[student@workstation bin]$ lab managed-domain-servers grade
```

This concludes the lab.

► Solution

Configuring Servers in a Managed Domain

In this lab, you will configure servers and server groups in a managed domain and deploy the bookstore application into the managed domain.

Resources	
Files	/opt/domain /home/student/JB248/labs/managed-domain-servers
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore

Outcome

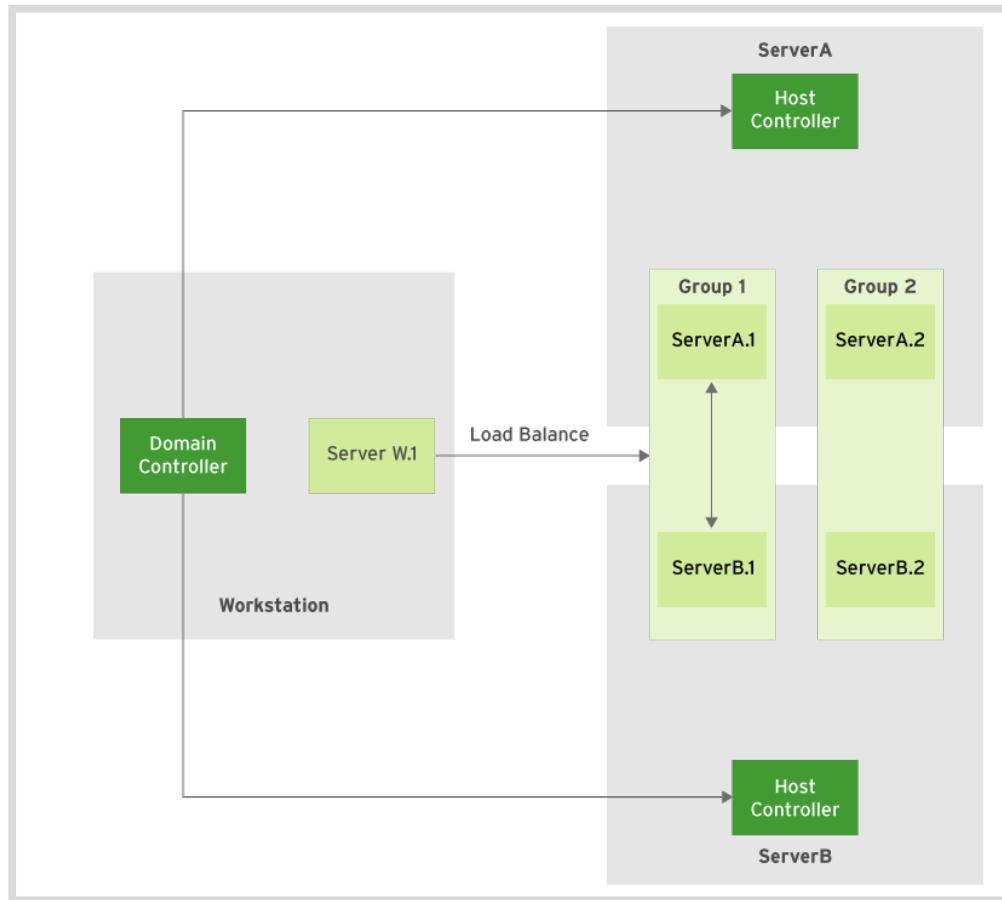
You should be able to configure an EAP 7 managed domain with two server groups and four servers and deploy the bookstore application.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been setup correctly:

```
[student@workstation ~]$ lab managed-domain-servers setup
```

1. An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively and the domain controller on **workstation** VM. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and then create new servers and server groups as per the diagram below:



As a final step, you will deploy the bookstore application WAR file on one of the newly created server groups. Bookstore is an application supporting high availability features and requires messaging subsystems and HA features from EAP. The **full-ha** profile of EAP must be used to deploy the application on EAP. You are free to choose either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

2. Briefly review the domain controller configuration on the **workstation**. Review the **/opt/domain/configuration/host-master.xml** file and the **/opt/domain/configuration/domain.xml** file to understand how the managed domain is configured. Verify that there are no server groups defined in the **domain.xml** file.
3. Start the domain controller on **workstation**. Because the domain controller configuration files are kept in the **/opt/domain** folder on workstation, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

```
[student@workstation bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ --host-config=host-master.xml
```

4. The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.
 - 4.1. Briefly review the host controller configuration on **servera** and **serverb**. Review the **/opt/domain/configuration/host-slave.xml** file to understand how the hosts are configured. Verify that there are no servers defined on either host.
 - 4.2. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Open a new terminal window on the **servera** VM and run the following command:

```
[student@servera bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 4.3. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Open a new terminal window on the **serverb** VM and run the following command:

```
[student@serverb bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 4.4. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
5. Access the JBoss EAP CLI. In a new terminal window on the **workstation**, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```

6. By default, there are no server groups or servers defined in the managed domain (These were removed in the final lab of Chapter 4). To deploy the bookstore application (WAR file), you need to create server groups and define servers.

Create two new server groups named **Group1** and **Group2** in the managed domain.

- 6.1. The bookstore application uses the messaging subsystem and HA capabilities from EAP, and these features are available only in the **full-ha** profile. Create a new server group called **Group1** which is used to emulate a development environment with the following characteristics:

- **Name:** **Group1**
- **Profile:** **full-ha**
- **Socket Binding Group:** **full-ha-sockets**

```
[domain@172.25.250.254:9990 /] /server-group=Group1:add\
(profile=full-ha,socket-binding-group=full-ha-sockets)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

- 6.2. Create another server group called **Group2** to use for deploying bookstore in a production environment with the same characteristics as the development environment:

- **Name:** **Group2**
- **Profile:** **full-ha**
- **Socket Binding Group:** **full-ha-sockets**

```
[domain@172.25.250.254:9990 /] /server-group=Group2:add\
(profile=full-ha,socket-binding-group=full-ha-sockets)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

- 6.3. Verify the server groups that you created in the steps above using either the management console or the JBoss EAP CLI.

```
[domain@172.25.250.254:9990 /] /server-group=Group1:read-resource
{
    "outcome" => "success",
    "result" => {
        "profile" => "full-ha",
        "socket-binding-group" => "full-ha-sockets",
        ...
    }
}
```

```
[domain@172.25.250.254:9990 /] /server-group=Group2:read-resource
{
    "outcome" => "success",
    "result" => {
        "profile" => "full-ha",
        "socket-binding-group" => "full-ha-sockets",
        ...
    }
}
```

7. You need to define four servers (two each on **servera** and **serverb**). Create the servers and assign them to the appropriate groups. Make sure that when you run multiple servers on a host, you configure the port offsets correctly to avoid port clashes. Also ensure that the servers are set to automatically start when the host controller is started or restarted.

- 7.1. Create a new server called **servera.1** on **servera** with the following characteristics:

- **Name:** **servera.1**
- **Server Group:** **Group1**
- **Socket Binding Port Offset:** Leave at default value of 0
- **Auto Start:** **true**

```
[domain@172.25.250.254:9990 /] /host=servera/server-config=servera.1:add\
(auto-start=true,group=Group1,socket-binding-port-offset=0)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```

- 7.2. Create a new server called **servera.2** on **servera** with the following characteristics:

- **Name:** **servera.2**
- **Server Group:** **Group2**
- **Socket Binding Port Offset:** **100**
- **Auto Start:** **true**

```
[domain@172.25.250.254:9990 /] /host=servera/server-config=servera.2:add\
(auto-start=true,group=Group2,socket-binding-port-offset=100)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```

- 7.3. Create a new server called **serverb.1** on **serverb** with the following characteristics:

- **Name:** `serverb.1`
- **Server Group:** `Group1`
- **Socket Binding Port Offset:** Leave at default value of 0
- **Auto Start:** `true`

```
[domain@172.25.250.254:9990 /] /host=serverb/server-config=serverb.1:add\
(auto-start=true,group=Group1,socket-binding-port-offset=0)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```

7.4. Create a new server called `serverb.2` on `serverb` with the following characteristics:

- **Name:** `serverb.2`
- **Server Group:** `Group2`
- **Socket Binding Port Offset:** `100`
- **Auto Start:** `true`

```
[domain@172.25.250.254:9990 /] /host=serverb/server-config=serverb.2:add\
(auto-start=true,group=Group2,socket-binding-port-offset=100)
{
    "outcome" => "success",
    "result" => {
        ...
    }
}
```

8. You have now created four servers and associated them with the appropriate server groups. You will now start the newly-created servers. Instead of starting servers one-by-one, you can start all of the servers in a server group together.

8.1. Start the servers in **Group1**

```
[domain@172.25.250.254:9990 /] /server-group=Group1:\nstart-servers(blocking=true)
{
    "outcome" => "success",
    ...
}
```

8.2. Start the servers in **Group2**

```
[domain@172.25.250.254:9990 /] /server-group=Group2:\nstart-servers(blocking=true)\n{\n    "outcome" => "success",\n    ...\n}
```

- 8.3. Verify that the servers started successfully by looking at the console windows where you started the host controllers. Verify that there are no port clashes and that the ports are bound according to the port offset values defined in the previous steps.

On the **servera** VM, successful startup of the servers should look like:

```
[Host Controller] 03:27:22,221 INFO  [org.jboss.as.host.controller]\n(Controller Boot Thread) WFLYHC0148: Connected to master host controller at\nremote://172.25.250.254:9999\n[Host Controller] 03:27:22,252 INFO  [org.jboss.as.host.controller] (Controller\nBoot Thread) WFLYHC0023: Starting server servera.1\n03:27:22,309 INFO  [org.jboss.as.process.Server:servera.1.status]\n(ProcessController-threads - 3) WFLYPC0018: Starting process 'Server:servera.1'\n[Server:servera.1] 03:27:22,957 INFO  [org.jboss.modules] (main) JBoss Modules\nversion 1.5.1.Final-redhat-1\n[Server:servera.1] 03:27:23,170 INFO  [org.jboss.msc] (main) JBoss MSC version\n1.2.6.Final-redhat-1\n[Server:servera.1] 03:27:23,246 INFO  [org.jboss.as] (MSC service thread 1-1)\nWFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) starting\n[Server:servera.1] 03:27:23,332 INFO  [org.xnio] (MSC service thread 1-4) XNIO\nversion 3.3.6.Final-redhat-1\n[Server:servera.1] 03:27:23,337 INFO  [org.xnio.nio] (MSC service thread 1-4) XNIO\nNIO Implementation Version 3.3.6.Final-redhat-1\n[Server:servera.1] 03:27:23,364 INFO  [org.jboss.remoting] (MSC service thread\n1-4) JBoss Remoting version 4.0.18.Final-redhat-1\n\n[Host Controller] 03:27:23,801 INFO  [org.jboss.as.host.controller] (management\ntask-2) WFLYHC0021: Server [Server:servera.1] connected using connection [Channel\nID 15ed583d (inbound) of Remoting connection 146a0d36 to /127.0.0.1:58069]\n[Host Controller] 03:27:23,854 INFO  [org.jboss.as.host.controller] (Controller\nBoot Thread) WFLYHC0023: Starting server servera.2\n[Host Controller] 03:27:23,885 INFO  [org.jboss.as.host.controller] (server-\nregistration-threads - 1) WFLYHC0020: Registering server servera.1\n03:27:23,929 INFO  [org.jboss.as.process.Server:servera.2.status]\n(ProcessController-threads - 3) WFLYPC0018: Starting process 'Server:servera.2'\n\n[Server:servera.2] 03:27:24,974 INFO  [org.jboss.modules] (main) JBoss Modules\nversion 1.5.1.Final-redhat-1\n[Server:servera.2] 03:27:25,308 INFO  [org.jboss.msc] (main) JBoss MSC version\n1.2.6.Final-redhat-1\n[Server:servera.2] 03:27:25,429 INFO  [org.jboss.as] (MSC service thread 1-4)\nWFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) starting\n...
```

On the **serverb** VM, successful startup of the servers should look like:

```

Host Controller] 03:30:30,534 INFO [org.jboss.as.host.controller]
(Controller Boot Thread) WFLYHC0148: Connected to master host controller at
remote://172.25.250.254:9999
[Host Controller] 03:30:30,562 INFO [org.jboss.as.host.controller] (Controller
Boot Thread) WFLYHC0023: Starting server serverb.1
03:30:30,637 INFO [org.jboss.as.process.Server:serverb.1.status]
(ProcessController-threads - 3) WFLYPC0018: Starting process 'Server:serverb.1'
[Server:serverb.1] 03:30:31,111 INFO [org.jboss.modules] (main) JBoss Modules
version 1.5.1.Final-redhat-1
[Server:serverb.1] 03:30:31,307 INFO [org.jboss.msc] (main) JBoss MSC version
1.2.6.Final-redhat-1
[Server:serverb.1] 03:30:31,383 INFO [org.jboss.as] (MSC service thread 1-1)
WFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) starting
[Server:serverb.1] 03:30:31,453 INFO [org.xnio] (MSC service thread 1-2) XNIO
version 3.3.6.Final-redhat-1
[Server:serverb.1] 03:30:31,458 INFO [org.xnio.nio] (MSC service thread 1-2) XNIO
NIO Implementation Version 3.3.6.Final-redhat-1
[Server:serverb.1] 03:30:31,479 INFO [org.jboss.remoting] (MSC service thread
1-2) JBoss Remoting version 4.0.18.Final-redhat-1

[Host Controller] 03:30:31,878 INFO [org.jboss.as.host.controller] (management
task-2) WFLYHC0021: Server [Server:serverb.1] connected using connection [Channel
ID 02443242 (inbound) of Remoting connection 146a0d36 to /127.0.0.1:34993]
[Host Controller] 03:30:31,905 INFO [org.jboss.as.host.controller] (Controller
Boot Thread) WFLYHC0023: Starting server serverb.2
[Host Controller] 03:30:31,928 INFO [org.jboss.as.host.controller] (server-
registration-threads - 1) WFLYHC0020: Registering server serverb.1

03:30:31,980 INFO [org.jboss.as.process.Server:serverb.2.status]
(ProcessController-threads - 3) WFLYPC0018: Starting process 'Server:serverb.2'
[Server:serverb.2] 03:30:32,830 INFO [org.jboss.modules] (main) JBoss Modules
version 1.5.1.Final-redhat-1
[Server:serverb.2] 03:30:33,260 INFO [org.jboss.msc] (main) JBoss MSC version
1.2.6.Final-redhat-1
[Server:serverb.2] 03:30:33,401 INFO [org.jboss.as] (MSC service thread 1-4)
WFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) starting
...

```

- 8.4. Because a firewall is running on **servera** and **serverb**, you must unlock all ports used by **servera.1**, **servera.2**, **serverb.1**, and **serverb.2** for public access. To unlock them, run the following command in a new terminal window from **servera**:

```

[student@servera ~]$ sudo firewall-cmd --zone=public --add-port 8080/tcp \
--permanent
[student@servera ~]$ sudo firewall-cmd --zone=public --add-port 8180/tcp \
--permanent
[student@servera ~]$ sudo firewall-cmd --reload

```

- 8.5. To unlock the firewall ports on **serverb**, run the following commands in a new terminal window from **serverb**:

```
[student@serverb ~]$ sudo firewall-cmd --zone=public --add-port 8080/tcp \
--permanent
[student@serverb ~]$ sudo firewall-cmd --zone=public --add-port 8180/tcp \
--permanent
[student@serverb ~]$ sudo firewall-cmd --reload
```

- 8.6. Verify that you can access the default welcome page of all four servers you created in the above steps. Access the following URLs and verify that you can see the EAP 7 Welcome page:
 - **servera.1:** `http://172.25.250.10:8080`
 - **servera.2:** `http://172.25.250.10:8180`
 - **serverb.1:** `http://172.25.250.11:8080`
 - **serverb.2:** `http://172.25.250.11:8180`
9. You are now ready to deploy the bookstore application WAR file in the managed domain. The **bookstore.war** file is available in the `/tmp` folder on the **workstation**.
 - 9.1. Application deployments in a managed domain are always done at the server groups level. Each application belongs to one or more server groups. Deploy the **bookstore.war** file to **Group1**.

```
[domain@172.25.250.254:9990 /] deploy \
/tmp/bookstore.war --server-groups=Group1
```

Verify that the **bookstore.war** file is deployed on **servera.1** and **serverb.1** because they are part of the **Group1** server group. Monitor the console window of **servera** and **serverb** for any error messages or warnings.

- 9.2. Verify that you can access the bookstore application at `http://172.25.250.10:8080/bookstore` and `http://172.25.250.11:8080/bookstore`.
- 9.3. Verify that you can **NOT** access the bookstore application at `http://172.25.250.10:8180/bookstore` or `http://172.25.250.11:8180/bookstore` because these servers are part of **Group2** and you have not deployed the application on **Group2**.
10. Using either the administration console or the JBoss EAP CLI, undeploy the bookstore application and shut down the servers, server groups, host controllers, and the entire managed domain.

10.1. Stop **servera.1**.

```
[domain@172.25.250.254:9990 /] /host=servera/server-config=servera.1:stop
{
    "outcome" => "success",
    "result" => "STOPPING"
}
```

- 10.2. Verify that the bookstore application is no longer accessible from **servera.1** but still accessible from **serverb.1**.

Chapter 5 | Configuring Servers in a Managed Domain

Verify that the **bookstore** application is **NOT** accessible at URL `http://172.25.250.10:8080/bookstore`. The **bookstore** application should still be accessible at `http://172.25.250.11:8080/bookstore` because **serverb.1** is still running.

- 10.3. Undeploy the bookstore application.

```
[domain@172.25.250.254:9990 /] undeploy bookstore.war \
--all-relevant-server-groups
```

- 10.4. Stop all servers in **Group1**. Verify that the bookstore application is no longer accessible.

```
[domain@172.25.250.254:9990 /] /server-group=Group1:stop-servers
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

- 10.5. Stop the servers in **Group2**.

```
[domain@172.25.250.254:9990 /] /server-group=Group2:stop-servers
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => undefined
}
```

- 10.6. Shutdown the host controller on **servera**. Observe the console window of **servera** and verify that the host controller has been shut down.

```
[domain@172.25.250.254:9990 /] /host=servera:shutdown
{
    "outcome" => "success",
    "result" => undefined
}
```

- 10.7. Shutdown the host controller on **serverb**. Observe the console window of **serverb** and verify that the host controller has been shut down.

```
[domain@172.25.250.254:9990 /] /host=serverb:shutdown
{
    "outcome" => "success",
    "result" => undefined
}
```

11. Clean Up and Grading

- 11.1. Press **Ctrl+C** to stop the domain controller (Alternatively, you can shut down the domain controller using the JBoss EAP CLI).

11.2. Run the following command from the **workstation** to grade the exercise:

```
[student@workstation bin]$ lab managed-domain-servers grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- An EAP managed domain is made up of a domain controller, host controllers, server groups, and servers.
- Server groups are configured at the domain controller level in **domain.xml**, whereas servers are configured at the host controller level in **host.xml**.
- A managed domain can have any number of servers and server groups. A server belongs to a single server group.
- A host can be configured to run any number of servers provided it has the hardware capacity to run the server instances.
- Applications (EAR, WAR, JAR files, and so on) are always deployed to server groups.
- Server groups and servers can be configured and managed from the EAP management console as well as the EAP CLI. The underlying **domain.xml** and **host.xml** files are automatically updated and the tools reflect the changes immediately.
- If multiple servers are configured to run on a single host, port offsets need to be used to avoid network port collisions.
- A server group is associated with a profile, which defines the subsystems, configuration, and network sockets that will be used by applications assigned to the server group.

Chapter 6

Configuring Data Sources

Overview

Goal Configure JDBC database drivers and data sources.

- Objectives**
- Describe the purpose and architecture of the data source subsystem.
 - Deploy a JDBC driver as a module and configure the driver in the data source subsystem.
 - Configure a data source.
 - Configure an XA data source.

Sections

- Exploring the Datasource Subsystem (and Quiz)
- Configuring JDBC Drivers (and Guided Exercise)
- Configuring Datasources (and Guided Exercise)
- Configuring an XA Datasource (and Guided Exercise)

Lab

- Configuring Data Sources

Exploring the Data Source Subsystem

Objectives

After completing this section, students should be able to:

- Describe the database subsystem and connection pools.

Data sources

Most Java EE applications involve a database that contains the information users need to access and perform the basic CRUD operations (create, read, update, and delete). A **data source** is a facility provided by an application server that is responsible for accessing databases without providing sensitive information, such as credentials and the database location, to an application. To connect to a database, the application uses a name bound to the data source via the application server. For a JEE application server, the name is defined by the Java Naming and Directory Interface (JNDI) specification.

Using a Java Database Connectivity (JDBC) API specific to a user's relational database management system, users can fine tune a data source to maximize efficiency and manage load on the database.

Data sources are configured through the *Datasource Subsystem*. Declaring a new data source consists of two separate steps:

1. Installing the JDBC driver for a particular database server.
2. Define a data source within the **datasource** subsystem of the configuration file.

Database connection pools

One obstacle to overcome when managing a data source is being able to handle traffic and concurrent connections to the database. It can be a potential bottleneck and a performance concern during peak loads for the deployed applications.

To improve the performance of connecting to a database, EAP creates a *database connection pool* for each database connection that is used by the applications. The pool contains open connections to the database, so that when the application needs to access the database, an open connection is already available, avoiding the overhead of opening and closing a connection for every single request. The size of the connection pool can be configured for each data source.



Note

Each database used by an application requires its own connection pool.

► Quiz

Data Sources

Based on the previous lecture, choose the correct answer to the following questions:

- ▶ 1. **Which validation method, by default, causes the largest strain on the database load? (Choose one)**
 - a. Validate on Match
 - b. Background Validation
- ▶ 2. **Which items listed are required for creating a data source? (Choose two)**
 - a. Installing the relevant JDBC Driver.
 - b. Configuring connection validation.
 - c. Defining the data source in the EAP configuration file.
 - d. Adjusting the size of the connection pool.
- ▶ 3. **Which term describes a facility provided by a Java EE application server to open multiple database connections? (Choose one)**
 - a. Datasource
 - b. JDBC
 - c. Database Connection Pool
 - d. JNDI
- ▶ 4. **Which Java API is responsible for defining names to a data source? (Choose one)**
 - a. JDBC
 - b. JNDI
 - c. JBDS
 - d. JPA
- ▶ 5. **Which statements describe advantages for using a connection pool? (Choose three)**
 - a. It provides a simple naming convention for accessing the database.
 - b. Connection pools improve performance by maintaining a pool of available connections, avoiding creating a new physical connection each time one is needed.
 - c. The size of the connection pool is customizable to maximize the database efficiency for each data source.
 - d. Verifying the database connections can be accomplished either in the background or before each new connection.
 - e. Connection pools protect secure connection credentials.

► Solution

Data Sources

Based on the previous lecture, choose the correct answer to the following questions:

- ▶ 1. Which validation method, by default, causes the largest strain on the database load? (Choose one)
 - a. Validate on Match
 - b. Background Validation

- ▶ 2. Which items listed are required for creating a data source? (Choose two)
 - a. Installing the relevant JDBC Driver.
 - b. Configuring connection validation.
 - c. Defining the data source in the EAP configuration file.
 - d. Adjusting the size of the connection pool.

- ▶ 3. Which term describes a facility provided by a Java EE application server to open multiple database connections? (Choose one)
 - a. Datasource
 - b. JDBC
 - c. Database Connection Pool
 - d. JNDI

- ▶ 4. Which Java API is responsible for defining names to a data source? (Choose one)
 - a. JDBC
 - b. JNDI
 - c. JBDS
 - d. JPA

- ▶ 5. Which statements describe advantages for using a connection pool? (Choose three)
 - a. It provides a simple naming convention for accessing the database.
 - b. Connection pools improve performance by maintaining a pool of available connections, avoiding creating a new physical connection each time one is needed.
 - c. The size of the connection pool is customizable to maximize the database efficiency for each data source.
 - d. Verifying the database connections can be accomplished either in the background or before each new connection.
 - e. Connection pools protect secure connection credentials.

Configuring JDBC Drivers

Objectives

After completing this section, students should be able to:

- Deploy a JDBC Driver as a module.
- Configure a driver in the data source subsystem.

Deploying a JDBC Driver as a Module

A **JDBC driver** is a component that Java applications use to communicate with a database. A JDBC driver is packaged in a JAR (Java Archive) file, and contains a class file that contains the driver definition. JDBC drivers are available from database vendors, such as from MySQL or PostgreSQL. In order to use a JDBC driver in EAP 7, it first needs to be installed as a module on the server.

Adding the JDBC driver module can be done either manually or much more simply with the EAP CLI. The module add command from CLI requires the user to provide:

- Name: A name based on a Java package name that will be used by EAP to store the JAR file and a configuration file called module. It should be unique and cannot conflict with existing libraries available at **\$JBOSS_HOME/modules** directory.
- Resources: Refers to the location that stores the JAR file.
- Dependencies: Identify which Java libraries are used by the JDBC driver and where they are located in EAP.

The syntax for the **module add** command is:

```
[disconnected /] module add  
--name=<module_name>  
--resources=<JDBC_Driver>  
--dependencies=<library1>,<  
library2>,...
```

For example, the following command in the EAP CLI will create the JDBC MySQL 5.5 module by using the file **mysql-connector-java.jar** that was downloaded from the database vendor:

```
[disconnected /] module add --name=com.mysql \  
--resources=/opt/jboss-eap-7.0/bin/mysql-connector-java.jar \  
--dependencies=javax.api,javax.transaction.api
```



Note

Because this command does not actually modify the **standalone.xml** or **domain.xml** files, the EAP server does not need to be running and the CLI does not need to be in connected mode.

After executing this command, a new directory is created at `/opt/jboss-eap-7.0/modules/com/mysql/main`. The new directory contains the provided JAR file as well as a `module.xml` that is generated based on the driver's given dependencies.

The following `module.xml` file defines a module for the MySQL JDBC driver:

```
<?xml version="1.0" ?>

<module xmlns="urn:jboss:module:1.1" name="com.mysql">

    <resources>
        <resource-root path="mysql-connector-java.jar"/>
    </resources>

    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```



Note

To remove a module from EAP, the server must be stopped and the following command can be used in the CLI:

```
[disconnected /] module remove --name=<module_name>
```

Defining drivers

After adding the driver as a module, the next step is to create a `<driver>` definition in the `<drivers>` section of the data source subsystem in the EAP configuration file.

A CLI operation can be used to create it easily, without touching the XML file. The following command can be executed on a stand-alone server:

```
/subsystem=datasources/jdbc-driver=<driver_name>:add\
(driver-module-name=<module_name>,driver-name=<unique_driver_name>)
```

In the driver definition command, the following fields are required:

- **driver-name**: A unique name for the driver.
- **driver-module-name**: The unique name from the module installed at `$JBOSS_HOME/modules` directory.

For the managed domain mode, the syntax is as follows:

```
/profile=<profile_name>/subsystem=datasources/jdbc-driver=<
driver_name>:add\
(driver-module-name=<module_name>,driver-name=<unique_driver_name>)
```

For example, the CLI command to define the MySQL driver in the default profile in a managed domain looks like the following:

```
[domain@172.25.250.254 /] /profile=default/subsystem=datasources/jdbc-
driver=mysql:add(\n
    driver-module-name=com.mysql,\n
    driver-name=mysql\n
)
```

**Note**

The `domain.xml/standalone.xml` configuration file will have an additional `driver` tag similar to the following after the driver is defined using the CLI:

```
<drivers>
    <driver name="mysql" module="com.mysql"/>
</drivers>
```

After defining the `<driver>` within the `<drivers>` section of the data source subsystem, the driver is referenced by its `name` attribute. After completing this step, users can create data sources that utilize the driver.

► Guided Exercise

Configuring JDBC Drivers

In this lab, you will install a MySQL JDBC driver as a module and enable it in the server configuration.

Resources	
Files:	/usr/share/java/mysql-connector-java.jar /home/student/JB248/labs/standalone
Application URL:	N/A

Outcomes

You should be able to install a MySQL JDBC driver as a module with the EAP CLI.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to set up a new base directory for EAP at **/home/student/JB248/labs/standalone**:

```
[student@workstation ~]$ lab datasource-driver setup
```

► 1. Start the EAP CLI

Before enabling the MySQL JDBC driver, you need to create a module for the driver. You can do this in EAP 7 using the CLI, as follows:

```
[student@workstation ~]$ cd
/opt/jboss-eap-7.0/bin
[student@workstation bin]$ sudo -u jboss ./jboss-cli.sh
```

► 2. Create the Module

Located in the **/usr/share/java** directory from the workstation VM is the MySQL JDBC JAR file. It is a driver provided by Red Hat's **yum** repository and it was installed during the VM installation by running **yum -y install mysql-connector-java** as root.

By installing it as a module, it becomes available as a driver in any EAP instances based on this installation in order to connect to MySQL databases.

Use the following command within the EAP CLI to create the module by pointing to the JDBC JAR file, listing the JAR's dependencies and the MySQL driver vendor ID as the name:

```
[disconnected /] module add --name=com.mysql \
--resources=/usr/share/java/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

Evaluate if the driver was correctly installed as a module by checking if a directory was created with **module.xml** and **mysql-connector-java.jar** using the following command from the terminal window:

```
[student@workstation ~]$ ls /opt/jboss-eap-7.0/modules/com/mysql/main  
module.xml  mysql-connector-java.jar
```

Also evaluate if the **module.xml** file was correctly generated by comparing it with the following listing:

```
<?xml version="1.0" ?>  
  
<module xmlns="urn:jboss:module:1.1" name="com.mysql">  
  
    <resources>  
        <resource-root path="mysql-connector-java.jar"/>  
    </resources>  
  
    <dependencies>  
        <module name="javax.api"/>  
        <module name="javax.transaction.api"/>  
    </dependencies>  
</module>
```

Open it from **JBOSS_HOME/modules/com/mysql/main** with your preferred text editor.

► 3. Define the MySQL Driver

To define the MySQL Driver, the EAP CLI needs to be connected to a running instance of EAP.

- 3.1. Open a new terminal window and start a standalone instance with **/home/student/JB248/labs/standalone** as the base directory:

```
[student@workstation ~]$ cd  
/opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

The server should start up successfully with an output similar to the following:

```
17:03:30,497 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP  
7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3313ms -Started 261 of  
509 services (332 services are lazy, passive or on-demand)
```

- 3.2. Return to the previous terminal that is running the EAP CLI in disconnected mode. Use the following command to connect it to the now running instance of EAP:

```
[disconnected /] connect localhost:9990
```

If prompted, provide the username **jbossadm** and the password **JBoss@RedHat123**. The prompt should change to:

```
[standalone@localhost:9990 /]
```

- 3.3. Use the following command to define the MySQL driver by specifying the EAP module installed in the previous step:

```
[standalone@localhost:9990 /]
/subsystem=datasources\
/jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql)
```

Notice that the **driver-module-name** of **com.mysql** matches the name given to the module in the previous step. The following output confirms the driver was correctly configured:

```
{"outcome" => "success"}
```

▶ 4. Verify the Driver

Use the following command to inspect the new MySQL JDBC driver:

```
[standalone@localhost:9990 /] /subsystem=datasources/jdbc-driver=mysql:read-
resource
```

The output should appear as follows:

```
{
  "outcome" => "success",
  "result" => {
    "deployment-name" => undefined,
    "driver-class-name" => undefined,
    "driver-datasource-class-name" => undefined,
    "driver-major-version" => undefined,
    "driver-minor-version" => undefined,
    "driver-module-name" => "com.mysql",
    "driver-name" => "mysql",
    "driver-xa-datasource-class-name" => undefined,
    "jdbc-compliant" => undefined,
    "module-slot" => undefined,
    "profile" => undefined,
    "xa-datasource-class" => undefined
  }
}
```

▶ 5. Clean Up

- 5.1. Exit the EAP CLI:

```
[standalone@localhost:9990 /] exit
```

- 5.2. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the guided exercise.

Configuring Datasources

Objectives

After completing this section, students should be able to:

- Configure a data source.
- Configure connection validity for data sources.

Creating a data source

In EAP 7, a data source is configured in the server's configuration file (**domain.xml** or **standalone.xml**) within the **datasource** subsystem.

The structure of the data source subsystem is as follows:

```
<subsystem xmlns="urn:jboss:domain:datasources:4.0">
<datasources>
    <datasource jndi-name="..." ❶ jta="true or false" ❷ pool-name="..." ❸
        enabled="true or false" ❹ use-java-context="true or false" ❺
            <connection-url> connection URL </connection-url>❻
            <driver> the driver module </driver>
            <pool> ❼
                Connection pool settings
            </pool>
            <security> ❽
                Username and password for connecting to the database
            </security>
            <validation>validation settings</validation>❾
            <timeout>timeouts</timeout>❿
            <statement>SQL statements</statement>❻
        </datasource>
        <drivers>
            <driver name="..." module="module_name"/>⫋
        </drivers>
    </datasources>
</subsystem>
```

- The JNDI name used for looking up the data source.
- Dictates whether JTA integration is enabled or disabled.
- The name of the pool referenced in the **<pool>** child element.
- Dictates whether the data source is enabled or disabled.
- Dictates whether to bind the data source to global JNDI.
- The JDBC driver connection URL.
- The connection pool settings.
- Security credentials for connecting to the database.
- The validation strategies for the data source connections.
- Contains child elements which are timeout settings.
- Describes the configuration for the behavior of prepared statements.

- ⑫ The driver definition.

Any number of **<datasource>** entries within the **<datasources>** subsystem can be defined. Each data source requires a unique JNDI name, assigned using the **jndi-name** attribute. Components that need to obtain a database connection from a pool use the **jndi-name** value to look up the pool in the JNDI naming service. A valid JNDI name for a data source must start with "java:" or "java:/jboss", for example, **java:/jboss/datasources/bookstore**.

Within the **<pool>** section are various configuration settings for the connection pool. Some of the pool settings include **<min-pool-size>**, **<max-pool-size>**, and **<prefill>**, which attempts to populate the connection pool when the pool is initially created.

The **<validation>** section allows for various settings for checking the validity of a database connection in the pool. Connection validation will be discussed later in this section.

The **<statement>** section is for configuring the behavior of prepared statements. The available options are **<prepared-statement-cache-size>** for specifying the number of prepared statements to cache per connection, and **<share-prepared-statements>** for specifying if non-closed prepared statements can be reused.



Note

Another **<statement>** configuration is **<track-statements>**, which can be set to **true**, **false** or **nowarn**. This is a useful option because, when set to **true** or **nowarn**, it closes database connections and result sets that were not explicitly closed by the application when the connection is returned to the pool.

The **<drivers>** section is for defining JDBC drivers that have been deployed as modules. The name attribute for the driver corresponds to the value used in the **<datasource>** definition. The **module** attribute must be assigned to a deployed module on the server.



Note

The XSD schema file for the **Datasources Subsystem** section is located in the **docs/schema** folder of the EAP installation in a file named **wildfly-datasources_4_0.xsd**.

Database connection validation

EAP 7 provides the capability to manage outages, network problems, database maintenance, or any other event that can cause the server to lose connection to the database. Using one of several validation methods, users can configure and customize when and how to validate the database connection and how to handle the loss of connection.

The first step for enabling validation is selecting one of the following validation methods and setting the value to **true** within the **<datasource>** section of the server configuration file in the **<validate>** section:

- **validate-on-match**

If the **validate-on-match** value is set to true, every time a connection is pulled from the connection pool, it will be validated.

The following is an example of using **validate-on-match**:

```
<datasources>
  <datasource ...>
    <validation>
      <validate-on-match>true or false</validate-on-match>①
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>②
    </validation>
  </datasource>
</datasources>
```

- ①** Can be **true** or **false** to enable or disable the validate on match.
- ②** The class name for the exception sorter, specific to the database vendor.

• **background-validation**

If the **background-validation** value is set to true, the connection will be validated based on the value set in the **<background-validation-millis>**.

The following is an example of using **background-validation**:

```
<datasources>
  <datasource ...>
    <validation>
      <background-validation>true or false</background-validation>①
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>②
    </validation>
  </datasource>
</datasources>
```

- ①** Can be **true** or **false** to enable or disable the background validation.
- ②** The class name for the exception sorter, specific to the database vendor.



Note

Both validation methods can be defined, however only one can be enabled at a time.

After selecting the validation method, users need to select a validation mechanism to describe how the connection will be validated. Select one of the following mechanisms for validating the connection:

• **valid-connection-checker**

EAP 7 provides several classes that can be used to validate a database connection that is specific to the user's relational database management system. For example, for a MySQL validation class checker, appears as follows:

```
<valid-connection-checker
  class-
  name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
```

• **check-valid-connection-sql**

Chapter 6 | Configuring Data Sources

This validation mechanism requires the user to provide a simple SQL statement that will be executed to validate the connection. The following is an example for a MySQL database connection:

```
<check-valid-connection-sql>select 1</check-valid-connection-sql>
```

- **exception-sorter**

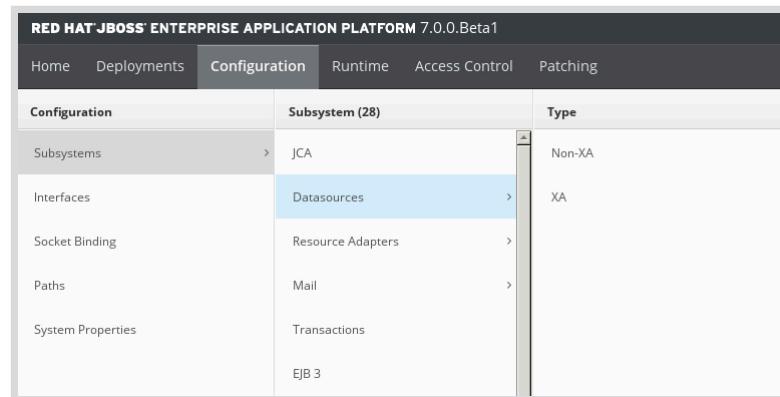
Using the **exception-sorter** allows users to provide a class to properly detect and clean up after fatal connection exceptions. EAP 7 provides several exception sorter classes based on the relational database management system being used. The following is an example for a MySQL exception sorter:

```
<exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
```

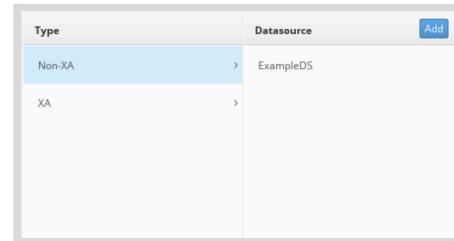
Creating a data source with the management console

The EAP Management Console provides an easy way to create a data source using templates that are preconfigured for different database vendors. The following steps can be used to create a Non-XA data source with the management console in a standalone server.

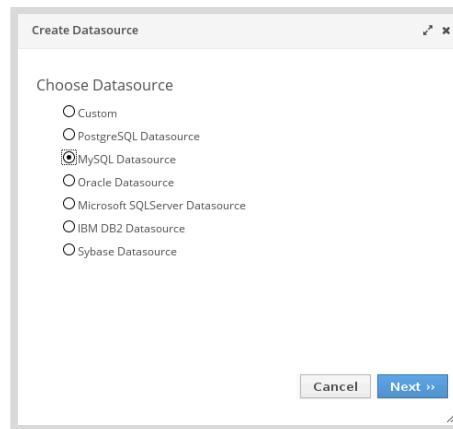
1. Select **Configuration** from the top of the management console and select **Subsystems** and then **Datasources** to access the data source subsystem.



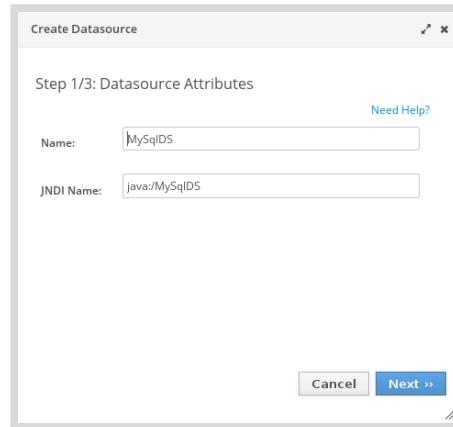
2. Click **Non-XA** and then click **Add** in order to open the **Create Datasource** wizard.



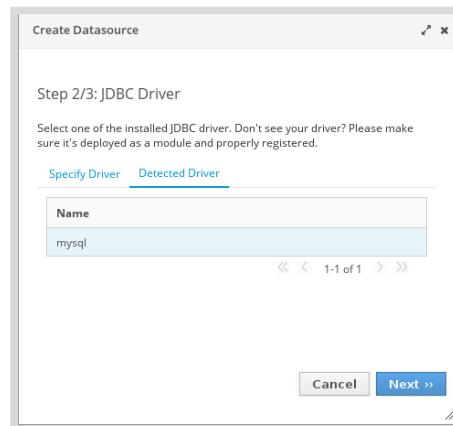
3. Select the database that should be used by the Java application and click **Next**. If none of them are valid, then select the **Custom** option and look for the JDBC driver documentation to complete the following steps.



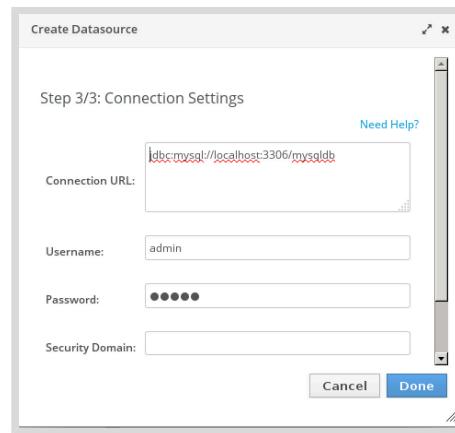
4. After entering the **Datasource Attributes**, click **Next** to select the driver.



5. Click the **Detected Driver** option and select the **mysql** driver that was installed as a module and select **Next**.



6. On the final step, notice that the connection URL is formatted for the correct MySQL syntax. Click **Done** to finish creating the data source.



Testing connection pool

The management console provides a **Test Connection** button to check that the connections from a connection pool can access the database. It can be accessed clicking the data source name and accessing the Test Connection button.

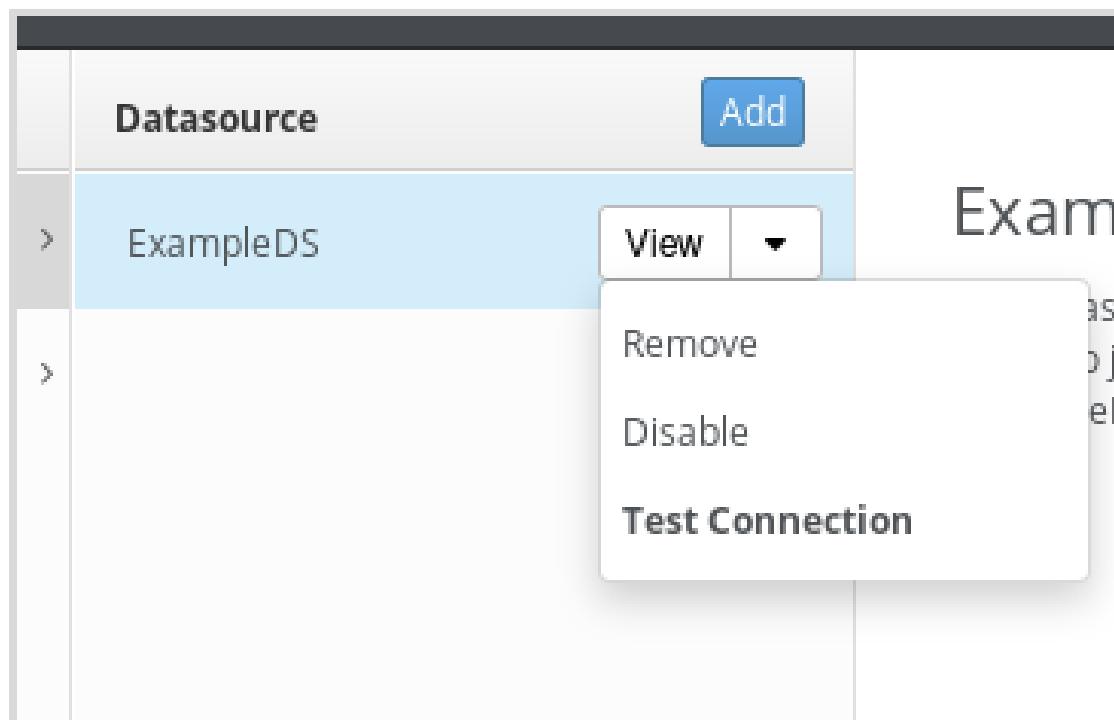


Figure 6.7: Test connection button

Likewise, the CLI can be used to test if the data source was correctly configured. To check it use the following command:

```
[standalone@localhost /] /subsystem=datasources/data-source=datasource_name:test-connection-in-pool
```

Data source example

The following data source configuration is for a MySQL database named **MySqlDS**.

```
<datasources>
  <datasource jndi-name="java:jboss/MySqlDS"① pool-name=" MySqlDS">
    <connection-url>jdbc:mysql://localhost:3306/jbosssdb</connection-url>
    <driver>mysql</driver>②
    <security>③
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>④
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
</datasources>
```

- ① The JNDI name that would be used to look up the data source by a component on the EAP server.
- ② The name that refers to the driver that is defined in the **<drivers>** section below the data source.
- ③ The credentials used to access the MySQL database.
- ④ Both validate on match and background validation are defined, but only one of them can be enabled. In this case, the data source will validate each connection as it is pulled from the connection pool using the class **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker**.

The corresponding CLI command to define this data source is:

```
[standalone@localhost] data-source add --name=MySqlDS --jndi-name=java:jboss/
MySqlDS \
--driver-name=mysql \
--connection-url=jdbc:mysql://localhost:3306/jbosssdb \
--user-name=admin --password=admin \
--validate-on-match=true --background-validation=false \
--valid-connection-checker-class-name=\
org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker \
--exception-sorter-class-name=\
org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

A similar command can be used when in domain mode in the CLI to add a data source to a specific profile:

```
[domain@localhost] data-source add --name=MySqlDS --profile=full-ha \
--jndi-name=java:jboss/MySqlDS --driver-name=mysql \
--connection-url=jdbc:mysql://localhost:3306/jbossdb \
--user-name=admin --password=admin \
--validate-on-match=true --background-validation=false \
--valid-connection-checker-class-name=\
org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker \
--exception-sorter-class-name=\
org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLEceptionSorter
```

► Guided Exercise

Configuring a Data Source

In this lab, you will create and configure a MySQL data source, and deploy a Java application to test the data source.

Resources	
Files:	/home/student/JB248/labs/standalone /tmp/
Application URL:	http://localhost:8080/dstest
Resources	dstest.war

Outcomes

You should be able to create and test a data source based on the MySQL driver installed in the previous exercise.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, to verify that the previous guided exercise was completed correctly, and to download the **dstest.war** application:

```
[student@workstation ~]$ lab datasource-configure setup
```

► 1. Start the Standalone EAP Instance

Use the following command to start an EAP instance to enable access to the management console:

```
[student@workstation ~]$ cd  
/opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Wait for the server to finish starting before proceeding.

► 2. Accessing the MySQL Database

Create a datasource to connect to a MySQL instance running locally on the workstation as a service. This database contains information used by applications that will be used throughout this course.

- 2.1. In a new terminal window, run the following command to verify that the database is started and running:

```
[student@workstation ~]$ sudo systemctl status mariadb
```

The output should look similar to the following:

```
mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; vendor
   preset: disabled)
   Active: active (running) since Mon 2016-04-11 08:40:09 EDT; 11h ago
```

2.2. MySQL has the following credentials for the **bookstore** application database:

- User name: **bookstore**
- Password: **redhat**

Use the following command to connect to the server running locally with the **mysql** client:

```
[student@workstation ~]$ mysql -ubookstore -predhat
```

2.3. To check which databases are available, use the following command:

```
MariaDB [(none)]> show databases;
```

The output should appear as follows:

```
+-----+
| Database      |
+-----+
| information_schema |
| bookstore      |
| test           |
+-----+
3 rows in set (0.00 sec)
```

2.4. Use the following command to select the bookstore database:

```
MariaDB [(none)]> use bookstore;
```

2.5. To list the tables in the **bookstore** database, run the following command:

```
MariaDB [bookstore]> show tables;
```

The output should appear as follows:

```
+-----+
| Tables_in_bookstore |
+-----+
| Address           |
+-----+
```

```
| CatalogItem          |
| Contact             |
| Customer            |
| OrderItem           |
| Payment              |
| Promotion            |
| UserToken            |
| WishList             |
| WishListItem         |
| hibernate_sequence  |
| order_               |
+-----+
12 rows in set (0.00 sec)
```

2.6. To quit the client, run the following command:

```
MariaDB [bookstore]> exit;
```

▶ 3. Configure the Data source

3.1. Navigate to the management console at `127.0.0.1:9990`. Go to the **Configuration** page.



Note

The admin user name is **jbossadm** and the password is **JBoss@RedHat123**.

3.2. Navigate to the Datasources subsystem by clicking **Subsystems** and then **Datasources**.

3.3. Select the **Non-XA** data source type and then click **Add**.

3.4. On the first window, select **MySQL Datasource** and click **Next**.

- Enter **bookstore** for the **Name**.
- Enter **java:jboss/datasources/bookstore** for the **JNDI Name**.

Click **Next**.

3.5. On Step 2 of the selection menu, click **Detected Driver** and select the driver named **mysql**. This is the driver that was installed in the previous exercise.

Click **Next**.

3.6. JDBC uses a standard format to connect to a database that is provided by the driver documentation. For a Java application to connect to a MySQL instance, the address format is: **jdbc:mysql://<IP>:<port>/<databaseName>**

- The **Connection URL** is **jdbc:mysql://localhost:3306/bookstore**.
- Enter **bookstore** for the **Username** and **redhat** for the **Password**.

Click **Finish** and the **bookstore** data source should appear in the fourth column.

► 4. Verify the Data source Configuration

- 4.1. In the terminal window where the server is running, look for the following log event:

```
09:44:10,769 INFO [org.jboss.as.connector.subsystems.datasources] (MSC service thread 1-4) WFLYJCA0001: Bound data source [java:jboss/datasources/bookstore]
```

- 4.2. Start the EAP CLI in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh
```

- 4.3. Use the following command to see the newly added data source as well as the other fields that can be modified:

```
[disconnected /] connect  
[standalone@localhost:9990] /subsystem=datasources/data-source=\  
bookstore:read-resource
```

The output should look like the following:

```
...  
"connection-properties" => undefined,  
  "connection-url" => "jdbc:mysql://localhost:3306/bookstore",  
  "datasource-class" => undefined,  
  "driver-class" => "com.mysql.jdbc.Driver",  
  "driver-name" => "mysql",  
  "enabled" => true,  
  "enlistment-trace" => true,  
  "exception-sorter-class-name" =>  
"org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter",  
  "exception-sorter-properties" => undefined,  
  "flush-strategy" => undefined,  
  "idle-timeout-minutes" => undefined,  
  "initial-pool-size" => undefined,  
  "jndi-name" => "java:jboss/datasources/bookstore",  
  "jta" => true,  
  "max-pool-size" => undefined,  
  "mcp" => "org.jboss.jca.core.connectionmanager.pool.mcp.\  
SemaphoreConcurrentLinkedDequeManagedConnectionPool",  
  "min-pool-size" => undefined,  
...  
...
```

► 5. Test the Data source

- 5.1. Deploy the **dtest.war** file found in the **/tmp/** directory using the already opened EAP CLI window.

```
[standalone@localhost:9990]  
deploy \  
/tmp/dtest.war
```

- 5.2. Navigate your browser to `http://127.0.0.1:8080/dstest/` to access the **dstest** application.
- 5.3.
 - Enter **java:jboss/datasources/bookstore** for the JNDI name.
 - Enter **bookstore.CatalogItem** for the table name.

Click **List** to test the data source.



Note

No credentials are required to access the data source as they were already configured during the data source configuration. Developers only need to know the JNDI name for the particular data source in order to access the database.

- 5.4. Read the results page and verify that the data source lookup was successful. You should see the contents of the **CatalogItem** table in the bookstore database.

▶ 6. Modify the Data source

In this step, you will configure some of the available settings of the connection pool using the CLI and the management console to improve the database performance.

- 6.1. Enter the following commands to view the current settings of the **bookstore** data source:

```
[standalone@localhost:9990] cd /subsystem=datasources/data-source=bookstore  
[standalone@localhost:9990 data-source=bookstore] :read-resource(recursive=true)
```

Notice that several of the attributes of the **bookstore** data source are undefined.

- 6.2. Enter the following command, which sets the minimum pool size of the **bookstore** data source to 5:

```
[standalone@localhost:9990 data-source=bookstore] :write-attribute\  
(name=min-pool-size,value=5)
```

- 6.3. Verify the change was made:

```
[standalone@localhost:9990 data-source=bookstore] :read-resource(recursive=true)
```

The output should now contain the following:

```
...  
"min-pool-size" => 5,  
...
```

- 6.4. Return to the **Configuration** page of the management console.
- 6.5. Select the **bookstore** data source by clicking **Subsystem**, then **Datasources** and then **Non-XA**. Click **View** next to **bookstore**.
- 6.6. Click **Disable** in order to be able to make changes to the data source and dismiss the notification regarding a server restart.

- 6.7. Click the **Pool** tab. You should see that **Min Pool Size** is 5.
- 6.8. Click **Edit** and set the **Max Pool Size** to 20.
- 6.9. Click **Save** to save your changes, then click **Enable** to enable the **bookstore** data source again.
- 6.10. Verify that the changes you made using the CLI and the management console appear in the **/home/student/JB248/labs/standalone/configuration/standalone.xml** configuration file.

► 7. Clean Up

- 7.1. Undeploy the **dtest.war** application:

```
[standalone@localhost:9990 data-source=bookstore] undeploy dtest.war
```

- 7.2. Exit the EAP CLI:

```
[standalone@localhost:9990 data-source=bookstore] exit
```

- 7.3. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the guided exercise.

Configuring an XA Data Source

Objectives

After completing this section, students should be able to:

- Configure an XA data source.

Configuring an XA data source

An XA data source is a data source that can participate in a transaction that spans multiple resources (**such as a database, a message queue, or legacy systems**) and roll back any changes made if a failure is detected in one of the resources. This strategy can help avoid data loss if any aspect of the transaction was to fail.

The following XML demonstrates a MySQL XA data source:

```
<datasources>
  <xa-datasource jndi-name="..." pool-name="...">
    <xa-datasource-property name="ServerName">
      localhost①
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      bookstore②
    </xa-datasource-property>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
  </xa-datasource>
  <drivers>
    <driver name="..." module="...">
      <xa-datasource-class>
        com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
      </xa-datasource-class>③
    </driver>
  </drivers>
</datasources>
```

^① The host name for the database server.

^② The database name for the data source.

^③ The XA data source class for the JDBC driver provided by the JDBC driver's documentation.

The following CLI command creates an XA data source:

```
[connected /] xa-data-source add --name=<internal_datasource_name> \
-jndi-name=<jndi_name> --driver-name=<driver_name> --user-name=<login> \
--password=<password>--xa-datasource-properties={"ServerName"=>"<server>", \
"DatabaseName"=>"<db_name>"}
```

Defining an XA data source compared to a non-XA data source is largely the same except for a few minor differences. The **ServerName** and **DatabaseName** are both required as properties, rather than requiring the user to provide the connection URL.

The driver for the database can be used for both XA and non-XA data sources, however an **xa-datasource-class** must be defined in the **<driver>** definition. This information is provided by the JDBC driver documentation.

The following CLI command registers the driver with a specified XA data source class:

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=<name>,driver-module- \
name=<module>,driver-xa-datasource-class-name=<xaclass>)
```

The following is an example CLI command for a MySQL driver configured to work with an XA data source:

```
[connected /]
/subsystem=datasources/jdbc-driver=mysql:add\
(driver-name=mysql,driver-module-name\
=com.mysql,driver-xa-datasource-class- \
name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource)
```

► Guided Exercise

Configuring an XA Datasource

In this lab, you will create and configure a MySQL XA data source, and deploy a Java application to test the data source.

Resources	
Files:	/home/student/JB248/labs/standalone /tmp/
Application URL:	http://localhost:8080/dstest
Resources	dstest.war

Outcomes

You should be able to create and test an XA data source based on the MySQL driver installed in the previous exercise.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, to verify that the driver was correctly installed, and to download the **dstest.war** application:

```
[student@workstation ~]$ lab datasource-configurexa setup
```

► 1. Start the Standalone EAP Instance

Use the following command to start an EAP instance to enable access to the management console:

```
[student@workstation ~]$ cd  
/opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Wait for the server to finish starting before proceeding.

► 2. Configure the XA JDBC driver. Open a new terminal window, and run the following commands to connect to EAP via CLI:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation ~]$ ./jboss-cli.sh --connect
```



Note

The admin username is **jbossadm** and the password is **JBoss@RedHat123**.

In XA, the driver and the datasource class are different from the EAP, and they should be passed explicitly as a parameter from the command line. The expected parameters for the datasource are:

- *Datasource name*: **bookstorexa**
- *JNDI name*: **java:jboss/datasources/bookstorexa**
- *driver name*: **mysql**
- *XA Datasource class*: **com.mysql.jdbc.jdbc2.optional.MysqlXADataSource**
- *username*: **bookstore**
- *password*: **redhat**
- XA datasource parameters:
 - *ServerName*: **localhost**
 - *DatabaseName*: **bookstore**

To create a datasource following these parameters, execute the following command from the CLI session:

```
[standalone@localhost:9990] xa-data-source add --name=bookstorexa \
--jndi-name=java:jboss/datasources/bookstorexa --driver-name=mysql \
--xa-datasource-class=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource \
--user-name=bookstore --password=redhat \
--xa-datasource-properties=[{"ServerName"=>"localhost"}, \
{"DatabaseName"=>"bookstore"}]
```

► 3. Test the Data source

- 3.1. Deploy the **dtest.war**. The **dtest.war** window.

```
[standalone@localhost:9990] deploy \
/tmp/dtest.war
```

- 3.2. Navigate your browser to <http://127.0.0.1:8080/dtest/> to access the **dtest** application.
- 3.3.
 - Enter **java:jboss/datasources/bookstorexa** for the JNDI name.
 - Enter **bookstore.CatalogItem** for the table name.

Click **List** to test the data source.
- 3.4. Read the results page and verify that the data source lookup was successful. You should see the contents of the **CatalogItem** table in the bookstore database.

► 4. Clean Up

- 4.1. Undeploy the **dtest.war** application:

```
[standalone@localhost:9990] undeploy dtest.war
```

- 4.2. Exit the EAP CLI:

```
[standalone@localhost:9990] exit
```

- 4.3. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the guided exercise.

▶ Lab

Configuring Data Sources

In this lab, you will configure EAP to enable a database access via a MySQL database to the bookstore application, instead of using an embedded database.

Resources	
Files	/opt/domain
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore
Resources	bookstore.war mysql-connector-java.jar

Outcomes

You should be able to install a MySQL JDBC module and create a data source in a managed domain.

Before You Begin

Use the following command to verify that no instances of EAP are running, that the managed domain is correctly configured based on the previous final chapter labs, and to download the **bookstore.war** file:

```
[student@workstation ~]$ lab  
datasources-lab setup
```

1. Add the MySQL Module to Each Host

In previous labs, the bookstore application was not using MySQL. In order to align with the final solution, it is required that the bookstore is backed by a MySQL database. In a managed domain, each host needs to add the module for the MySQL JDBC driver individually.

1. On the **workstation**, **servera** and **serverb**, copy the MySQL module located at **/usr/share/java/mysql-connector-java.jar** to the **jboss** user owned **/opt/jboss-eap-7.0/bin/** directory. This is done to avoid permissions issues when adding the module as user **jboss**.
2. Using the EAP CLI without connecting, create a new module on the workstation for the MySQL JDBC with dependencies **javax.api** and **javax.transaction.api** and with the name **com.mysql**. In the previous step, the driver was copied to **/opt/jboss-eap-7.0/bin**.
3. On **servera**, use the EAP CLI without connecting and create a new module for the MySQL JDBC with dependencies **javax.api** and **javax.transaction.api** and with the name **com.mysql**.

Chapter 6 | Configuring Data Sources

- 1.4. Repeat the command on **serverb** in order to create a new module for the MySQL JDBC.
- 1.5. Evaluate if the driver was correctly installed as a module on the workstation, **servera**, and **serverb** by checking if a directory was created with **module.xml** and **mysql-connector-java.jar**.

2. Define the MySQL Driver

After installing the MySQL JDBC module, the driver needs to be defined to be utilized by a data source. Use the following steps to define the MySQL Driver.

- 2.1. Start the domain controller on the workstation:
- 2.2. Start the host controller on **servera**.
- 2.3. Start the host controller on **serverb**.
- 2.4. Use the previously started EAP CLI and connect to the domain controller on the workstation and register the JDBC driver for the **full-ha** profile. Name the driver **mysql**.

3. Create the Bookstore MySQL Data source

Using the already running EAP CLI on the workstation, create the bookstore MySQL data source for the full-ha profile with the following properties:

- Name: **bookstore**
- JNDI Name: **java:jboss/datasources/bookstore**
- Driver Name: **mysql**
- Connection URL: **jdbc:mysql://172.25.250.254:3306/bookstore**.
- User name: **bookstore**
- Password: **redhat**

4. Test the Data source

Use the management console to test the connection to the database is valid, ensuring that the driver was installed correctly and that the bookstore data source was created with the correct values.

5. Deploy the Bookstore Application

The bookstore application deployed in previous labs was not configured to support a MySQL database. This version of the **bookstore.war** was updated to support MySQL. Use the EAP CLI to deploy **/tmp/bookstore.war** on server group **Group1**.

6. Verify the Bookstore Application

After the application is deployed to the server group **Group1**, visit the application on each of the servers in the server group.

7. Clean Up and Grading

- 7.1. Press **Ctrl+C** to stop the EAP CLI and the domain and host controllers.
- 7.2. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab datasources-lab grade
```

This concludes the lab.

► Solution

Configuring Data Sources

In this lab, you will configure EAP to enable a database access via a MySQL database to the bookstore application, instead of using an embedded database.

Resources	
Files	/opt/domain
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore
Resources	bookstore.war mysql-connector-java.jar

Outcomes

You should be able to install a MySQL JDBC module and create a data source in a managed domain.

Before You Begin

Use the following command to verify that no instances of EAP are running, that the managed domain is correctly configured based on the previous final chapter labs, and to download the **bookstore.war** file:

```
[student@workstation ~]$ lab
datasources-lab setup
```

1. Add the MySQL Module to Each Host

In previous labs, the bookstore application was not using MySQL. In order to align with the final solution, it is required that the bookstore is backed by a MySQL database. In a managed domain, each host needs to add the module for the MySQL JDBC driver individually.

- 1.1. On the **workstation**, **servera** and **serverb**, copy the MySQL module located at **/usr/share/java/mysql-connector-java.jar** to the **jboss** user owned **/opt/jboss-eap-7.0/bin/** directory. This is done to avoid permissions issues when adding the module as user **jboss**.

```
[student@workstation ~]$ sudo cp /usr/share/java/mysql-connector-java.jar \
/opt/jboss-eap-7.0/bin
```

```
[student@servera ~]$ sudo cp /usr/share/java/mysql-connector-java.jar \
/opt/jboss-eap-7.0/bin
```

```
[student@serverb ~]$ sudo cp /usr/share/java/mysql-connector-java.jar \
/opt/jboss-eap-7.0/bin
```

- 1.2. Using the EAP CLI without connecting, create a new module on the workstation for the MySQL JDBC with dependencies **javax.api** and **javax.transaction.api** and with the name **com.mysql**. In the previous step, the driver was copied to **/opt/jboss-eap-7.0/bin**.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ sudo -u jboss ./jboss-cli.sh
[disconnected /] module add --name=com.mysql \
--resources=/opt/jboss-eap-7.0/bin/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

- 1.3. On **servera**, use the EAP CLI without connecting and create a new module for the MySQL JDBC with dependencies **javax.api** and **javax.transaction.api** and with the name **com.mysql**.

```
[student@servera ~]$ cd /opt/jboss-eap-7.0/bin
[student@servera bin]$ sudo -u jboss ./jboss-cli.sh
[disconnected /] module add --name=com.mysql \
--resources=/opt/jboss-eap-7.0/bin/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

- 1.4. Repeat the command on **serverb** in order to create a new module for the MySQL JDBC.

```
[student@serverb ~]$ cd /opt/jboss-eap-7.0/bin
[student@serverb bin]$ sudo -u jboss ./jboss-cli.sh
[disconnected /] module add --name=com.mysql \
--resources=/opt/jboss-eap-7.0/bin/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

- 1.5. Evaluate if the driver was correctly installed as a module on the workstation, **servera**, and **serverb** by checking if a directory was created with **module.xml** and **mysql-connector-java.jar**.

Use the following command from the terminal window:

```
[student@workstation ~]$ ls /opt/jboss-eap-7.0/modules/com/mysql/main
module.xml  mysql-connector-java.jar
```

2. Define the MySQL Driver

After installing the MySQL JDBC module, the driver needs to be defined to be utilized by a data source. Use the following steps to define the MySQL Driver.

- 2.1. Start the domain controller on the workstation:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ --host-config=host-master.xml
```

2.2. Start the host controller on **servera**.

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

2.3. Start the host controller on **serverb**.

```
[student@serverb bin]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

2.4. Use the previously started EAP CLI and connect to the domain controller on the workstation and register the JDBC driver for the **full-ha** profile. Name the driver **mysql**.

```
[disconnected /] connect 172.25.250.254:9990
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=datasources\
/jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql)
```

The following output confirms that the driver was installed correctly for each of the four servers:

```
{
  "outcome" => "success",
  "result" => undefined,
  "server-groups" => {
    "Group1" => {"host" => {
      "servera" => {"servera.1" => {"response" => {
        "outcome" => "success",
        "result" => undefined
      }}},
      "serverb" => {"serverb.1" => {"response" => {
        "outcome" => "success",
        "result" => undefined
      }}}
    },
    "Group2" => {"host" => {
      "servera" => {"servera.2" => {"response" => {
        "outcome" => "success",
        "result" => undefined
      }}},
      "serverb" => {"serverb.2" => {"response" => {
        "outcome" => "success",
        "result" => undefined
      }}}
    }
  }
}
```

```
        }}}
```

3. Create the Bookstore MySQL Data source

Using the already running EAP CLI on the workstation, create the bookstore MySQL data source for the full-ha profile with the following properties:

- Name: **bookstore**
- JNDI Name: **java:jboss/datasources/bookstore**
- Driver Name: **mysql**
- Connection URL: **jdbc:mysql://172.25.250.254:3306/bookstore**.
- User name: **bookstore**
- Password: **redhat**

```
[domain@172.25.250.254:9990 /] data-source add \
--profile=full-ha --name=bookstore --driver-name=mysql \
--jndi-name=java:jboss/datasources/bookstore \
--connection-url=jdbc:mysql://172.25.250.254:3306/bookstore \
--user-name=bookstore --password=redhat
```

4. Test the Data source

Use the management console to test the connection to the database is valid, ensuring that the driver was installed correctly and that the bookstore data source was created with to correct values.

- 4.1. Access the management console by visiting <http://172.25.250.254:9990>. Use the user name **jbossadm** and the password **JBoss@RedHat123**.
- 4.2. At the top of the page, click **Configuration** and then **Profiles** and select the **full-ha** profile.
- 4.3. Click **Subsystems** and **Datasources** to access the data source subsystem.
- 4.4. Select **Non-XA** under data sources and then select the **bookstore** data source. Click **View** next to the data source.
- 4.5. In the overview page for the **bookstore** data source, click **Connection** and then click **Test Connection**. A pop-up should confirm that the connection is valid. If not, review the previous steps and verify that the connection matches the given properties and that the driver module was correctly installed.

5. Deploy the Bookstore Application

The bookstore application deployed in previous labs was not configured to support a MySQL database. This version of the **bookstore.war** was updated to support MySQL. Use the EAP CLI to deploy **/tmp/bookstore.war** on server group **Group1**.

```
[domain@172.25.250.254:9990 /] deploy \
/tmp/bookstore.war --server-groups=Group1
```

6. Verify the Bookstore Application

After the application is deployed to the server group **Group1**, visit the application on each of the servers in the server group.

The bookstore application can be accessed on **servera.1** at <http://172.25.250.10:8080/bookstore> and **serverb.1** at <http://172.25.250.11:8080/bookstore>.

7. Clean Up and Grading

- 7.1. Press **Ctrl+C** to stop the EAP CLI and the domain and host controllers.
- 7.2. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab datasources-lab grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- Creating a data source first requires installing the relevant JDBC driver.
- The management console provides database vendor specific templates to facilitate data source creation.
- A connection pool prevents manual creation of a connection each time one is needed, instead opting to open many connections initially and pulling one from the "pool" as needed.
- A JDBC driver can be installed using the EAP CLI command **module add**.
- One JDBC driver can be used for multiple relevant XA and non-XA data sources.
- The connection pool can be configured by adjusting the **<min-pool-size>**, **<max-pool-size>**, and **<prefill>** settings.
- Connections can be validated either as they are pulled from the pool (Validate-on-match) or at a certain time interval (background-validation).
- XA data sources provide the capability to roll back multi-resource transactions that partially fail.

Chapter 7

Configuring the Logging Subsystem

Overview

Goal Configure log handlers and loggers.

Objectives

- Configure various types of logging handlers.
- Configure ROOT and category loggers.

Sections

- Configuring Logging Handlers (and Guided Exercise)
- Configuring Loggers (and Guided Exercise)

Lab

- Configuring the Logging Subsystem

Configuring Logging Handlers

Objectives

After completing this section, students should be able to:

- Configure various types of logging handlers.

Overview of the logging subsystem

Logging is one of the most crucial tasks performed by an application server. Gathering and analyzing log files is an important task for an EAP administrator to help diagnose, troubleshoot and analyze issues in production. Sometimes, there may also be a business requirement to store logging information from applications to meet regulatory and compliance related laws.

Consistent with the modular design of EAP, logging is handled by a subsystem. The logging configuration is represented by the *logging subsystem*, and is configured in the **domain.xml** (in managed domain mode) and **standalone.xml** (in a stand-alone server) configuration files.

The EAP logging subsystem has three main components:

- Handlers:** a handler determines "where" and "how" an event will be logged. EAP comes with several handlers out-of-the-box that can be used to log messages from applications.
- Loggers:** a logger (also called log category) organizes events into logically related categories. A category usually maps to a certain package namespace that is running within the Java Virtual Machine (Could be both EAP as well as custom application package names) and defines one or more handlers that will process and log the messages.
- Root Logger:** Loggers are organized in a parent-child hierarchy and the root logger resides at the top of the logger hierarchy.

A **handler** is activated only if one or more **loggers** reference it. Loggers can reference more than one handler (or none at all). The **root logger** captures all log messages of the specified log level, that are not captured by a log category.

By default, the log level of EAP is **INFO**, and two handlers are enabled:

- CONSOLE:** a handler that logs events to the console window, and
- FILE:** a handler that logs events to a file named **server.log**.

Default log file locations

Earlier versions of EAP had a single log folder. EAP 7 has multiple log file locations depending on whether EAP is running as a standalone server or managed domain mode.

For example, if EAP is running as a standalone server, then the log files are found in:

```
$BASE_DIR/standalone/log
```

where **BASE_DIR** is the base EAP folder. The **standalone/log** folder contains two log files:

- **gc.log.0.current**: for logging the Java Virtual Machine flags that EAP was started with and memory management events while the server is running. The settings for configuring this file are found in the **BASE_DIR/bin/standalone.sh** file. The **gc.log.0.current** only shows log events that occurred during the most recent startup of EAP.
- **server.log**: for the server's log events. Log events are appended to this file, so typically the server will need to roll this file over based on the file size or length of time (e.g., hourly or daily). The settings for configuring **server.log** are found in the logging subsystem section of **standalone.xml**.

If EAP is running in managed domain mode, then logging occurs in multiple locations (relative to the EAP base folder):

- **domain/log/host-controller.log**: contains log events during the bootup of the host controller (or domain controller if this particular host is the domain controller). It also registers events such as lost connection with domain controller or new host controllers registered and debug or runtime errors in the host controller.
- **domain/log/process-controller.log**: contains log events related to the starting and stopping of processes on the Host. (Note that the host controller runs in its own process, and also remember that each server on the host runs in its own process.)
- **domain/servers/<server-name>/log/server.log**: contains this server's log events.

Notice that each server on a host has its own folder for log files. For example, if a host has a server on it named **dev-server-one**, then the log file for that server is:

domain/servers/dev-server-one/log/server.log

The **<server-name>/log/server.log** is analogous to **server.log** the one from a standalone server, and is a good place to start when looking for issues on a server. This file is appended to each time the server starts, so it is a good practice to rotate **server.log** on an hourly or daily basis, or when the file reaches a certain size. The configuration of log files will be discussed next, and a rotating log file based on size will be configured in the guided exercise later in this chapter.



Note

The default location of the log folders can be changed by setting the **jboss.server.log.dir** property at runtime. Remember that the **jboss.server.log.dir** folder is relative to **jboss.domain.base.dir**, so if **jboss.domain.base.dir** is defined, a different value for **jboss.server.log.dir** is not needed unless there are specific requirements for the managed domain.

The built-in handlers

EAP comes with six built-in handlers. The built-in handlers are:

- **Console handler**: Defines a handler that writes to the console. It is configured using the **<console-handler>** element.
- **File handler**: Defines a handler that writes to a file. It is configured using the **<file-handler>** element.

- **Periodic-rotating file handler:** Defines a handler that writes to a file, rotating the log after a time period derived from the given suffix string. It is configured using the `<periodic-rotating-file-handler>` element.
- **Size-rotating file handler:** Defines a handler that writes to a file, rotating the log after a the size of the file grows beyond a certain point and keeping a fixed number of backups. It is configured using the `<size-rotating-file-handler>` element.
- **Async handler:** Defines a handler that writes to the sub-handlers in an asynchronous thread. This is useful in scenarios where the application generates a large number of log messages in parallel threads and where the file system's I/O throughput becomes a bottleneck. It is configured using the `<async-handler>` element.
- **Syslog handler:** Defines a handler which sends events to a remote syslog server. It is configured using the `<syslog-handler>` element.



Note

Handlers specify "where" and "how" an event is logged, but defining a handler does not automatically turn it on. That is done in either the `<logger>` or `<root-logger>` tags, which are discussed later in this chapter.



Note

A custom handler can be developed for logging events to any type of resource and in any format required. This can be achieved by writing a Java class that extends the base EAP logging framework API and then defining a **custom-handler** in the EAP configuration file. Writing a custom handler is beyond the scope of this course.

Handlers are configured within the logging subsystem section of the server configuration file. For example, `standalone.xml` and most of the profiles in `domain.xml` come preconfigured with a `<console-handler>` defined. The CLI definition looks like:

```
{  
    "outcome" => "success",  
    "result" => {  
        "autoflush" => true,  
        "enabled" => true,  
        "encoding" => undefined,  
        "filter" => undefined,  
        "filter-spec" => undefined,  
        "formatter" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n",  
        "level" => "INFO",  
        "name" => "CONSOLE",  
        "named-formatter" => "COLOR-PATTERN",  
        "target" => "System.out"  
    }  
}
```

The corresponding XML looks like:

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
    <console-handler name="CONSOLE" autoflush="true">
        <level name="INFO"/>
        <formatter>
            <named-formatter name="COLOR-PATTERN"/>
        </formatter>
    </console-handler>

    ...
</subsystem>
```

- The **<level>** element specifies the minimum log level for this handler.
- The **<formatter>** element let you specify a named pattern to use when logging events. The **COLOR-PATTERN** formatter colorizes the log output for different levels and makes it easy to locate errors when parsing large log files. For example: log messages with **ERROR** or **FATAL** level severity are colorized in red to draw the administrator's attention when viewing the logs.

The **CONSOLE** handler can also be configured using the EAP management console in the **Configuration → Subsystems → Logging** section. Click **Logging**, then select the **Handler** tab, then click the **Console** link:

The screenshot shows the 'Configuration: Subsystems > Subsystem: Logging' screen. The 'HANDLER' tab is selected. Under the 'Console Handler' section, the 'Name' field is set to 'CONSOLE'. The 'Attributes' table includes the following entries:

Attribute	Value
Autoflush:	true
Enabled:	true
Encoding:	
Filter spec:	
Formatter:	%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n
Level:	INFO

Figure 7.1: The Console Handler

Periodic rotating file handlers

Use the **periodic-rotating-file-handler** to automatically rotate the log files based on a certain amount of elapsed time. For example, the default **FILE** handler is set to roll over the **server.log** file every 24 hours. The definition of the handler is as follows:

```
<periodic-rotating-file-handler name="FILE" autoflush="true">
    <formatter>
        <named-formatter name="PATTERN"/>
    </formatter>
    <file relative-to="jboss.server.log.dir" path="server.log"/>
    <suffix value=".yyyy-MM-dd"/>
    <append value="true"/>
</periodic-rotating-file-handler>
```

- The **name** of this handler is **FILE** and is the handler EAP uses by default.
- The value of **suffix** determines how often the log file rotates. In the example above, suffix is **yyyy-MM-dd**, which represents a day format. Therefore, this log will rotate once a day (at midnight).
- The value of **relative-to** is either a **path** element (defined in this configuration file) or a Java property. In this case we are using the predefined property **jboss.server.log.dir**, which points to the **BASE_DIR/standalone/log** folder in a standalone server.

The suffix value is a date-time format, and the pattern specifies how often the log file rotates. For example, if a log file should rotate every hour, set the suffix to:

```
yyyy-MM-dd.HH
```

Similarly, monthly rotation could be accomplished by setting suffix to:

```
yyyy-MM
```

The **periodic-rotating-file-handler** can be configured and managed using the EAP management console in the **Configuration → Subsystems → Logging** section. Click **Logging**, then the **Handler** tab, then click the **Periodic** link to view, edit or add a **periodic-rotating-file-handler**:

Attributes		File
Append:	true	
Autoflush:	true	
Enabled:	true	
Encoding:		
Formatter:	%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n	
Level:	ALL	
Named formatter:	PATTERN	

Figure 7.2: The Periodic Rotating File Handler

You can also use the EAP CLI to define a new handler using the **add** operation. For example, the following CLI command adds a new **periodic-rotating-file-handler** named **MY_HANDLER**:

```
/profile=default/subsystem=logging/periodic-rotating-file-handler=MY_HANDLER:add(  
    autoflush=true,  
    append=true,  
    named-formatter=PATTERN,  
    file={  
        path=mylog.log,  
        relative-to=jboss.server.log.dir  
    },  
    level=INFO,  
    suffix=.yyyy-DD-mm-HH  
)
```

Size rotating file handlers

Use the **size-rotating-file-handler** to automatically rotate log files based on the size of the log file. This handler is useful in scenarios where the number of log files generated within a period is very large and could lead to performance bottlenecks if the log file is several gigabytes in size. The handler can be configured to rotate the log file after a set size (**rotate-size** value). Here is a sample definition of this handler:

```
<size-rotating-file-handler name="FILE_BY_SIZE"①>  
    <level name="INFO"② />  
    <formatter>  
        <named-formatter name="PATTERN"③ />  
    </formatter>  
    <file relative-to="jboss.server.log.dir"④ path="production-server.log"⑤ />  
    <rotate-size value="1m"⑥ />  
    <max-backup-index value="3"⑦ />  
</size-rotating-file-handler>
```

- ① The **name** attribute is required and is the name of the handler. It must be unique within a profile (in managed domain mode).
- ② The **level** attribute sets the default log level for this handler. The handler handles all log messages at this level and below in the logger hierarchy.
- ③ The **formatter** attribute references a valid **named-formatter** reference defined in the configuration file. The pattern can also be directly provided by using the **pattern-formatter** tag and defining a pattern template.
- ④ The **relative-to** attribute references either the full path where the log file should be created, or it can refer to a logical **path** reference defined in the configuration file. The default value is the pre-defined property **jboss.server.log.dir** which points to the **BASE_DIR/standalone/log** folder in a stand-alone server and **BASE_DIR/domain/servers/<server-name>/log** folder in managed domain mode.
- ⑤ The **path** attribute provides a path to the log file which is relative to the **relative-to** path value. For example, if the path value of **mylog.log** is provided, then the log file is created at **BASE_DIR/standalone/log/mylog.log** in a standalone server and **BASE_DIR/domain/servers/<server-name>/log/mylog.log** in managed domain mode.
- ⑥ The **rotate-size** value is the size of the log file after which the handler rotates the file. The rolled over file is suffixed with a numeric index. The rotate size value can be given in

megabytes by suffixing the value with an "m". For example, to rotate the log file after 5 MB, the **rotate-size** value should be given as **5m**.

- 7 The **max-backup-index** attribute determines how many backup copies of the rotated log files are maintained before the files are reused. For example, for a log file named **mylog.log** with a **max-backup-index** value of three, the handler will rotate the log file three times (after log file reaches the **rotate-size** value) into log files called **mylog.log.1**, **mylog.log.2** and **mylog.log.3**. On the next rotation, **mylog.log** will be overwritten and the old log messages are purged. Make sure the **max-backup-index** value is defined appropriately based on the rate at which the application generates logs.

Size rotating file handlers can be defined in the management console in the **Configuration → Subsystems → Logging** section. Click **Logging**, then the **Size** link under the **Handler** tab.

The instructor will now demonstrate how to define a size rotating file handler by using the EAP management console:

Demonstration: Configuring size rotating file handlers

Review the video to follow along with the steps. Replay it as often as needed.

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command to create the lab directory and verify that EAP is installed and not currently running:

```
[student@workstation ~]$ demo  
logging-srfh setup
```

2. Often users want to be able to run multiple standalone instances without needing to install EAP multiple times on the same host. The previous guided exercise created an alternative directory in order to customize the EAP instance and leave the original installation untouched.

Verify that you can see the following three folders under the **/home/student/JB248/labs/standalone** folder:

- **configuration**
- **deployments**
- **lib**

3. The variable **jboss.server.base.dir** allows users to pass in a directory path to use it for an alternative base directory for the server content. Run the following command to start the EAP server using the **standalone.sh** script in the original EAP installation while using the new EAP configuration files:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

The server should start successfully with the following message:

```
17:03:30,497 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP  
7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3313ms -Started 261 of  
509 services (332 services are lazy, passive or on-demand)
```

Chapter 7 | Configuring the Logging Subsystem

4. On the workstation, navigate to `http://localhost:9990` to access the EAP management console.
Log in with username **jbossadm** and password **JBoss@RedHat123**.
5. In the EAP management console, the **Logging** subsystem is configured under the **Configuration → Subsystems → Logging** section.

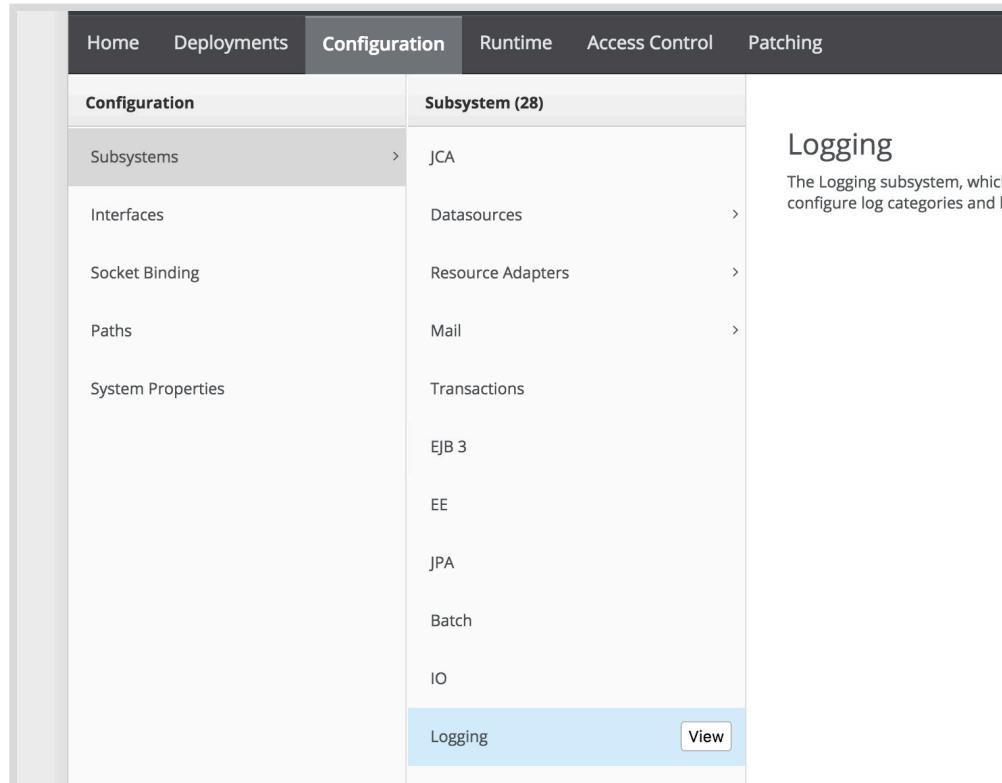


Figure 7.3: The Logging subsystem

- Navigate to **Configuration → Subsystems → Logging** in the EAP management console and click **View** next to **Logging** to bring up the Logging subsystem configuration page.
6. In the logging subsystem configuration page, there are four tabs available in the top menu providing the following configuration options:
 - **ROOT LOGGER:** The default logging configuration used by EAP and all deployed applications and subsystems.
 - **LOG CATEGORIES:** A filter used to obtain logs from certain applications/subsystems.
 - **HANDLER:** Defines where and how log files are generated and configurations that can be used by EAP and all applications/subsystems.
 - **FORMATTER:** Configures the output generated by a handler.
 7. Click the **HANDLER** tab in the logging subsystem configuration page and then click the **Size** link in the left sidebar menu.

The screenshot shows the 'Configuration: Subsystems > Subsystem: Logging' screen. The 'HANDLER' tab is selected. On the left, a sidebar lists various handler types: Console, File, Periodic, Periodic Size, **Size** (which is selected and highlighted with a blue border), Async, Custom, and Syslog. The main panel is titled 'Size Handler' and contains a description: 'Defines a handler which writes to a file, rotating the log after the size of the file grows beyond a certain point and keeping a fixed number of backups.' Below this is a table with a single row labeled 'Name'. A message 'No Items!' is displayed below the table. At the bottom of the panel are buttons for 'Add' and 'Remove', and a 'Need Help?' link. On the right side of the main panel, there are tabs for 'Attributes' (which is selected) and 'File'. Under the 'Attributes' tab, several configuration options are listed: 'Append:', 'Autoflush:', 'Enabled:', 'Encoding:', and 'Filter spec:'.

Figure 7.4: The Size Rotating File Handler

8. Click **Add** in the size handler configuration page to add a new size rotating file handler.
9. In the resulting pop-up window, enter **FILE_BY_SIZE** as the value in the **Name** field and click **Next**. Note that the name of the handler must be unique because there can be multiple handlers defined in the EAP configuration file.

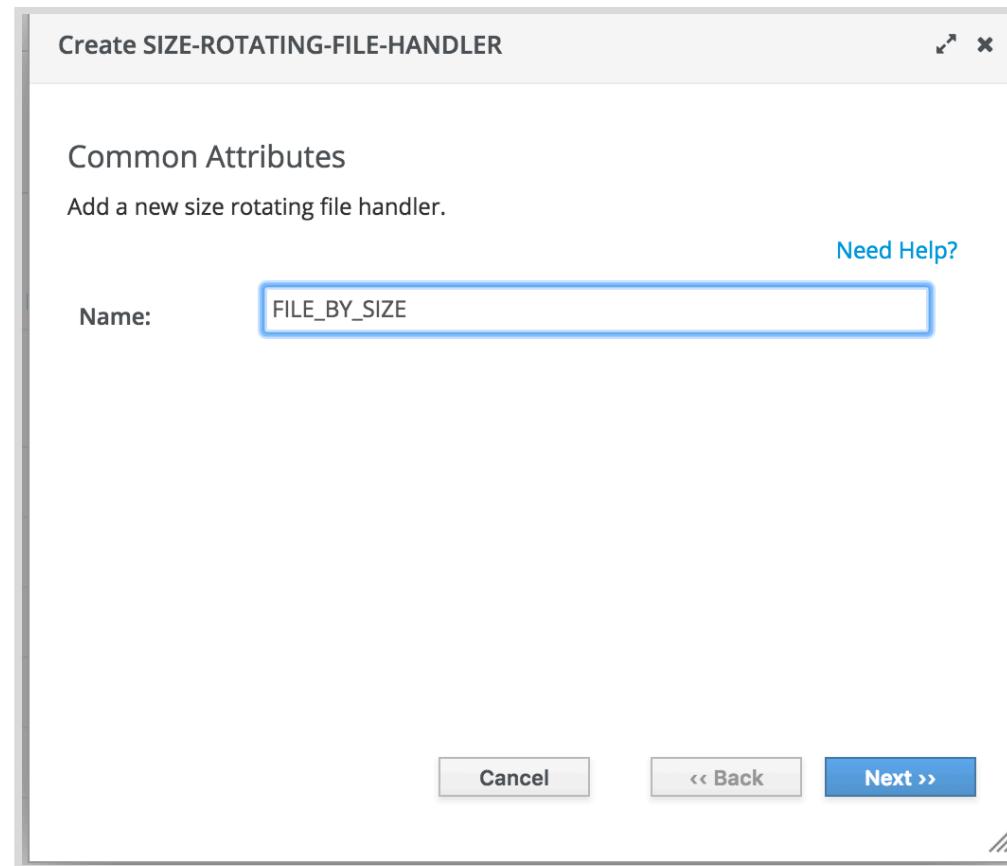


Figure 7.5: Adding a new Size Rotating File Handler

In the next page, enter **production-server.log** as the value in the **Path** field and **jboss.server.log.dir** as the value in the **Relative Path** field. Finally, click **Finish** to create the handler.



Note

jboss.server.log.dir is an EAP defined system variable that points to **\${jboss.server.base.dir}/log** by default.

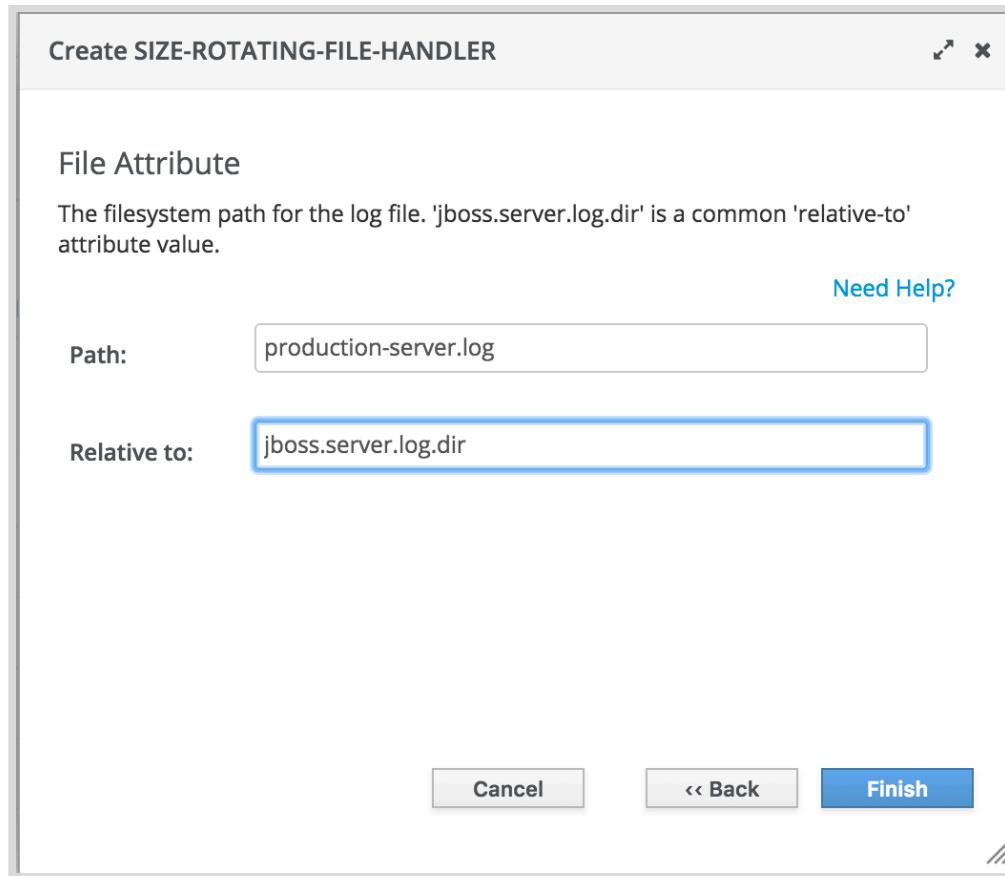


Figure 7.6: Configuring the attributes of a Size Rotating File Handler

You should now see a new size rotating file handler called **FILE_BY_SIZE** listed in the size handler configuration page. Observe that the **Level** field is set to **ALL** and the **Rotate size**

Chapter 7 | Configuring the Logging Subsystem

field is set to **2m** (2 MB). Setting the level to **ALL** will capture log messages at all the levels across all loggers deployed on EAP.

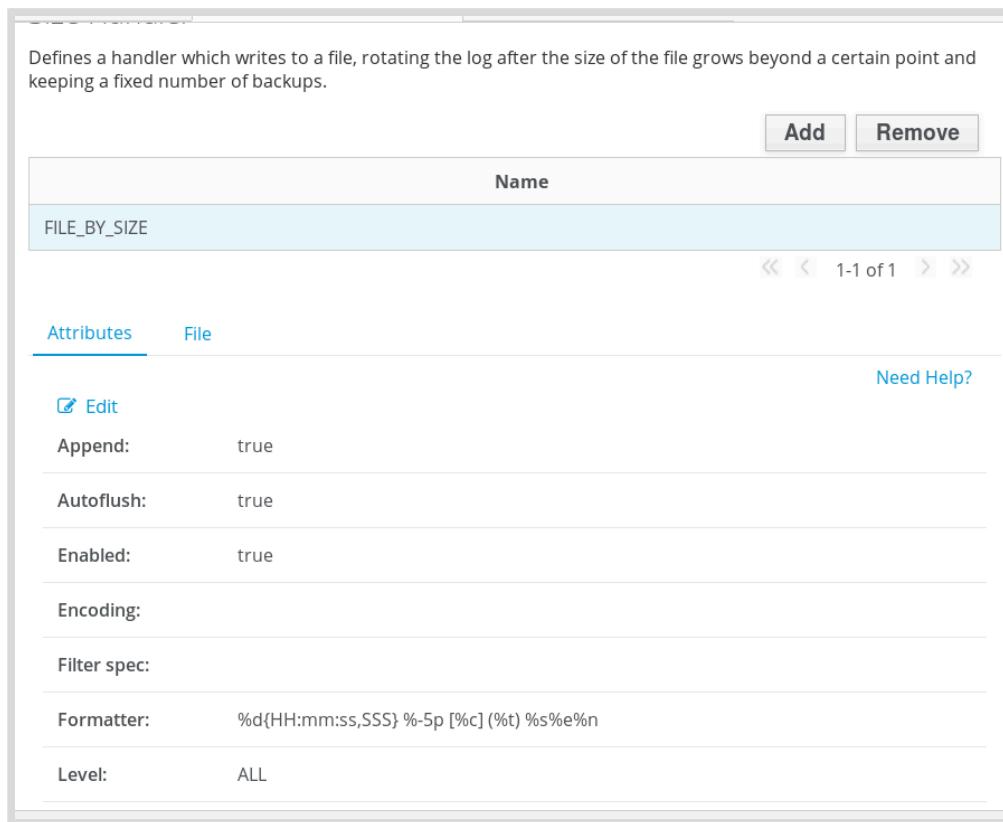


Figure 7.7: The new FILE_BY_SIZE handler

10. Edit the attributes of the **FILE_BY_SIZE** handler by clicking the blue **Edit** link below the **Attributes** tab.

Change the **Level** attribute to **INFO** in the drop-down field next to the **Level** label and change the value of the **Rotate size** field to **1m** (1 MB). Finally, Click **Save** to modify the attributes of the handler.

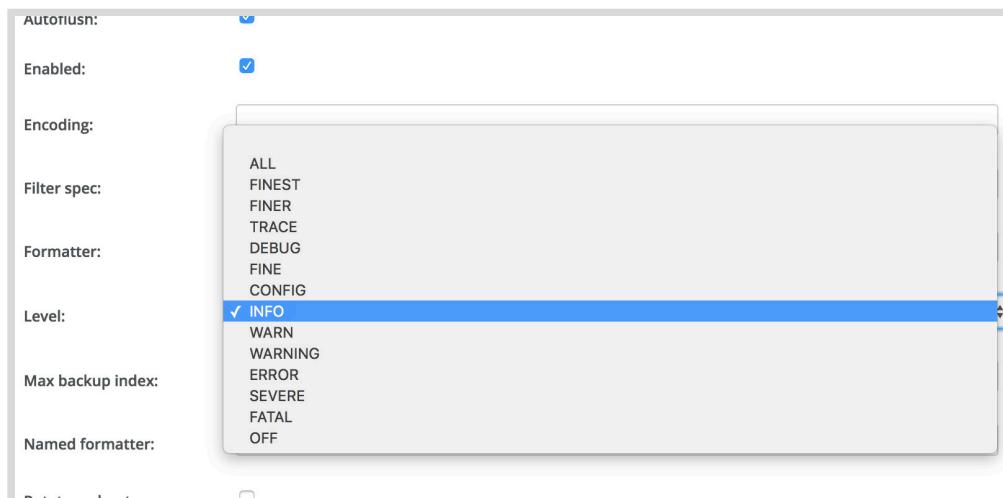


Figure 7.8: Changing the log level and rotate size of the handler

11. Open the `/home/student/JB248/labs/standalone/configuration/standalone.xml` file and verify that a new size rotating file handler called **FILE_BY_SIZE** is visible:

```
<size-rotating-file-handler name="FILE_BY_SIZE">
    <level name="INFO"/>
    <file relative-to="jboss.server.log.dir" path="production-server.log"/>
    <rotate-size value="1m"/>
</size-rotating-file-handler>
```

We will use this **FILE_BY_SIZE** handler in the next demonstration to set up asynchronous logging.

12. Verify that a file called **production-server.log** was created under the `/home/student/JB248/labs/standalone/log` folder. The file rotates when the file size is greater than the **rotate-size** attribute. The rotated file will be kept and the index appended to the name moving previously rotated file indexes up by 1 until the **max-backup-index** is reached. Once the **max-backup-index** is reached, the indexed files will be overwritten. The default value of **max-backup-index** is 1.
13. Stop EAP by pressing **Ctrl+C** on the terminal window that is running EAP.

This concludes the demonstration.

Asynchronous logging

The **async-handler** provides a simple way to accomplish *asynchronous logging*. The **async-handler** does not actually write log events to a file or the console (or any other resource). Instead, the **async-handler** sits in front of an existing handler and adds the asynchronous behavior. Use the `<subhandler>` element to define which handler the asynchronous behavior is being added to. A sample definition for this handler is as follows:

```
<async-handler name="ASYNC">
    <level name="INFO"/>
    <queue-length value="1024"/>
    <overflow-action value="BLOCK"/>
    <subhandlers>
        <handler name="FILE"/>
    </subhandlers>
</async-handler>
```

- The name of this handler is **ASYNC** and the log level is **INFO**.
- The maximum number of log messages that can be held in the queue is 1024.
- The possible values of **overflow-action** are **BLOCK** and **DISCARD**. If the queue of log events is full when a new log event is attempted to be added, then **BLOCK** means the current thread will block and wait for room in the queue, and **DISCARD** means the log event will simply be discarded.
- The `<subhandlers>` section can contain multiple handlers. In this case, the asynchronous behavior is added to the default **FILE** handler.

Adding the asynchronous behavior to logging can improve performance on high-volume servers. If a request is made to the server that causes a log event to occur, a response can be sent back to the client without making the client wait for the actual writing of the log event to the handler.

The instructor will now demonstrate how to add asynchronous logging to the **FILE_BY_SIZE** handler defined in the previous section:

Demonstration: Configuring asynchronous logging

Review the video to follow along with the steps. Replay it as often as needed.

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command to create the lab directory and verify that EAP is installed and not currently running:

```
[student@workstation ~]$ demo logging-async setup
```

2. Often users want to be able to run multiple stand-alone instances without needing to install EAP multiple times on the same host. The previous demonstration in this chapter created an alternative directory to customize the EAP instance and leave the original installation untouched.

Verify that you can see the following three folders under the **/home/student/JB248/labs/standalone** folder:

- **configuration**
 - **deployments**
 - **lib**
3. The variable **jboss.server.base.dir** allows users to pass in a directory path to use as an alternative base directory for the server content. Run the following command to start the EAP server using the **standalone.sh** script in the original EAP installation while using the new EAP configuration files:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

The server should start successfully with the following message:

```
17:03:30,497 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP  
7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) started in 3313ms -Started 261 of  
509 services (332 services are lazy, passive or on-demand)
```

4. We will use the JBoss EAP CLI in this demonstration to configure the asynchronous handler. Run the following command to log in to the CLI:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh --connect
```

5. The asynchronous handler does not actually write log events to a file or the console (or any other resource). Instead, it sits in front of an existing handler and adds the asynchronous

Chapter 7 | Configuring the Logging Subsystem

behavior. In this demonstration, we will add asynchronous behavior to the **FILE_BY_SIZE** handler we created in the previous demonstration. First, verify that the **FILE_BY_SIZE** handler has been defined correctly using the EAP CLI:

```
[standalone@localhost:9990 /] /subsystem=logging\
  /size-rotating-file-handler=FILE_BY_SIZE:read-resource
{
  "outcome" => "success",
  "result" => {
    "append" => true,
    "autoflush" => true,
    "enabled" => true,
    "encoding" => undefined,
    "file" => {
      "relative-to" => "jboss.server.log.dir",
      "path" => "production-server.log"
    },
    "filter" => undefined,
    "filter-spec" => undefined,
    "formatter" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n",
    "level" => "INFO",
    "max-backup-index" => 1,
    "name" => "FILE_BY_SIZE",
    "named-formatter" => undefined,
    "rotate-on-boot" => false,
    "rotate-size" => "1m",
    "suffix" => undefined
  }
}
```

6. The asynchronous handler has several configuration attributes:
 - **level:** The default logging level that this handler will use (**ALL**, **INFO**, **DEBUG**, **ERROR**, **WARN** etc.)
 - **queue-length:** The maximum number of log messages to be held in memory while log files are flushed to disk asynchronously.
 - **overflow-action:** Action that the handler should take when queue is full (**DISCARD** discards the log messages, **BLOCK** blocks the thread writing the log message until there is space left in the queue).
 - **subhandler:** An existing handler to which asynchronous behavior is to be added.

Create a new asynchronous handler called **ASYNC** with a level set to **INFO**, queue length value of **1024**, overflow-action set to **BLOCK**, and assign the **FILE_BY_SIZE** handler as a subhandler to it:

```
[standalone@localhost:9990 /]
/subsystem=logging/async-handler=ASYNC:add\
  (level=INFO,queue-length=1024,overflow-action=BLOCK,subhandlers=[FILE_BY_SIZE])
{"outcome" => "success"}
```

7. Add the **ASYNC** handler to the **ROOT** logger:

```
[standalone@localhost:9990 /]
/subsystem=logging/root-logger=ROOT:\n
add-handler(name=ASYNC)
{"outcome" => "success"}
```

8. Open the **/home/student/JB248/labs/standalone/configuration/standalone.xml** file and verify that an **async-handler** called **ASYNC** is visible:

```
<async-handler name="ASYNC">
  <level name="INFO"/>
  <queue-length value="1024"/>
  <overflow-action value="block"/>
  <subhandlers>
    <handler name="FILE_BY_SIZE"/>
  </subhandlers>
</async-handler>
```

9. You can also verify the new **async-handler** using the EAP CLI:

```
[standalone@localhost:9990 /] /subsystem=logging/async-handler=ASYNC:read-resource
{
  "outcome" => "success",
  "result" => {
    "enabled" => true,
    "filter" => undefined,
    "filter-spec" => undefined,
    "level" => "INFO",
    "name" => "ASYNC",
    "overflow-action" => "BLOCK",
    "queue-length" => 1024,
    "subhandlers" => ["FILE_BY_SIZE"]
  }
}
```

10. Open a new terminal window on the workstation VM and observe the **/home/student/JB248/labs/standalone/log/production-server.log** file:

```
[student@workstation ~]$ tail -f \
/home/student/JB248/labs/standalone/log/production-server.log
```

11. Stop EAP by pressing **Ctrl+C** on the terminal window that is running EAP.
12. Observe that when you shut down EAP, log messages are logged to the **/home/student/JB248/labs/standalone/log/production-server.log** file.
13. Restart the EAP standalone instance again and verify that log messages are seen in the **/home/student/JB248/labs/standalone/log/production-server.log** file.

Although you have not directly referenced the **FILE_BY_SIZE** handler, the **ASYNC** handler adds asynchronous behavior to the **FILE_BY_SIZE** handler and proxies log messages to it, which results in the log messages being logged to the **production-server.log** file by the **FILE_BY_SIZE** handler.

14. Shut down EAP by pressing **Ctrl+C** again on the terminal window that is running EAP.

This concludes the demonstration.

Syslog handler

The **syslog-handler** provides a simple way to send events to a remote logging server. This allows multiple applications to send their log messages to a central syslog server, where they can all be stored, analyzed, and archived simultaneously to simplify backups. A sample definition of this handler appears as follows:

```
<syslog-handler name="SYSLOG">
    <level name="INFO"/>
    <server-address value="172.25.2.9"/>
    <hostname value="workstation.lab.example.com"/>
    <port value="514"/>
    <app-name value="MY_APP"/>
</syslog-handler>
```

- The name of this handler is **SYSLOG** and the log level is **INFO**.
- This handler sends log messages to the remote logging server located at 172.25.2.9 and listens on TCP port 514.
- This handler formats the log message according to the RFC5424 (syslog protocol) specification. Any syslog compliant remote server can accept these log messages.
- The name of the host name that the messages are being sent from is set to **workstation.lab.example.com** to uniquely identify this host because multiple hosts can send syslog messages to the remote server. The log messages on the remote server can be parsed and filtered based on this attribute.

Syslog handlers can be defined using the EAP management console in the **Configuration → Subsystems → Logging** section. Click **Logging**, then the **Handler** tab, then click the **Syslog** link to view, edit or add a **syslog-handler**:

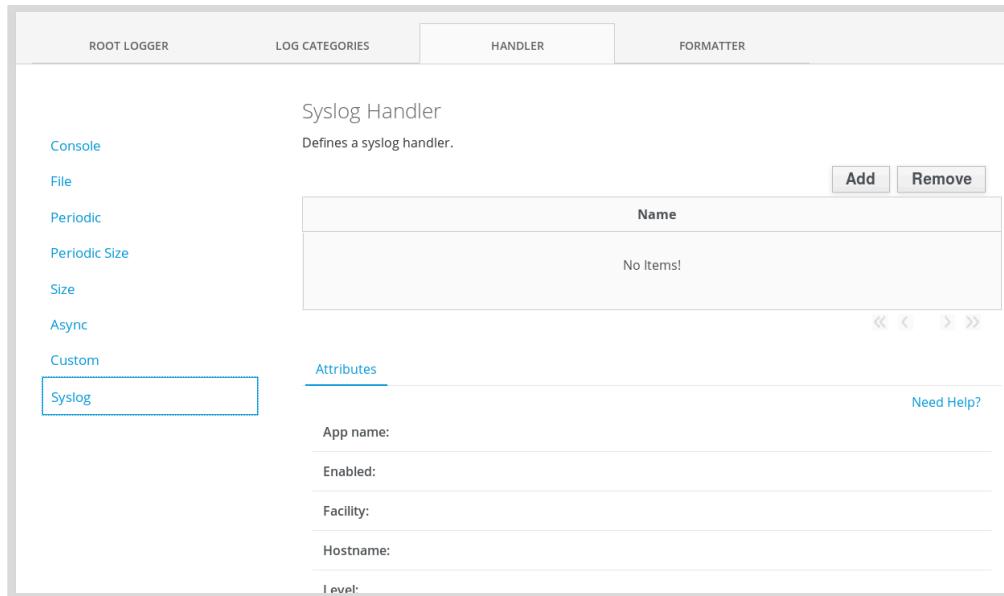


Figure 7.9: The Syslog handler

► Guided Exercise

Configuring Logging Handlers

In this lab, you will create a size rotating file handler and deploy an application that uses this handler to log messages.

Resources	
Files:	/home/student/JB248/labs/standalone /home/student/JB248/labs/logging-handlers
Application URL:	http://localhost:8080/logtest
Resources	/home/student/JB248/labs/logging-handlers/ logtest.war /home/student/JB248/labs/logging-handlers/ add_sizerotating_log.cli

Outcomes

You should be able to create a size rotating file handler and view log messages generated by the application in the EAP server log files.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, that the previous guided exercise has been completed, and to download the **logtest.war** application:

```
[student@workstation ~]$ lab logging-handlers setup
```

► 1. Start the Stand-alone EAP Server

Use the following command to start an EAP instance using the **/home/student/JB248/labs/standalone** folder as the base directory:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Wait for the server to finish starting before proceeding.

► 2. You will use a JBoss EAP CLI script file to create a new **size-rotating-file-handler** and deploy the **logtest.war** file. Briefly review the file **add_sizerotating_log.cli** located under the **/home/student/JB248/labs/logging-handlers** directory.

2.1. The commands in the **add_sizerotating_log.cli** file should be as follows:

```
batch

/subsystem=logging/size-rotating-file-handler=FILE_BY_SIZE_ROTATING/:add\
(file={"path">"production-server.log", \
"relative-to">"jboss.server.log.dir"}, \
formatter="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n", \
level=INFO,max-backup-index=3,name=FILE_BY_SIZE_ROTATING, \
rotate-size=1m)

/subsystem=logging/logger=com.redhat.training.view:add\
(category=com.redhat.training.view,handlers=["FILE_BY_SIZE_ROTATING"])

deploy /home/student/JB248/labs/logging-handlers/logtest.war

run-batch
```

The script will configure the logging subsystem to use the handler called **FILE_BY_SIZE_ROTATING** to capture all the logs generated by the category **com.redhat.training.view** (which represents a Java package where the logging source code will be executed), and it will capture all the logs generated with the **INFO** level in a file called **production-server.log** which will be located under the **/home/student/JB248/labs/standalone/log** folder.

After the log file reaches 1 MB in size, the logging subsystem will rotate the log file to a new log file with a numbered suffix. There will be a maximum of three (3) log files before the contents of the log files are overwritten in an iterative manner.

The application called logtest is deployed on the server. It is a Java web application with all the source code in the **com.redhat.training.view** package.

- 3. Run the CLI script in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh -c \
--file=/home/student/JB248/labs/logging-handlers/add_sizerotating_log.cli
The batch executed successfully
```

Check for the above message to verify if the batch commands executed successfully. If you observe errors during execution, you can try and insert the commands line by line into the CLI prompt and debug the errors.

- 4. Use the following CLI command to verify that the handler has been added successfully:

```
[student@workstation bin] ./jboss-cli.sh --connect
[standalone@localhost:9990] /subsystem=logging/\
size-rotating-file-handler=FILE_BY_SIZE_ROTATING:read-resource
```

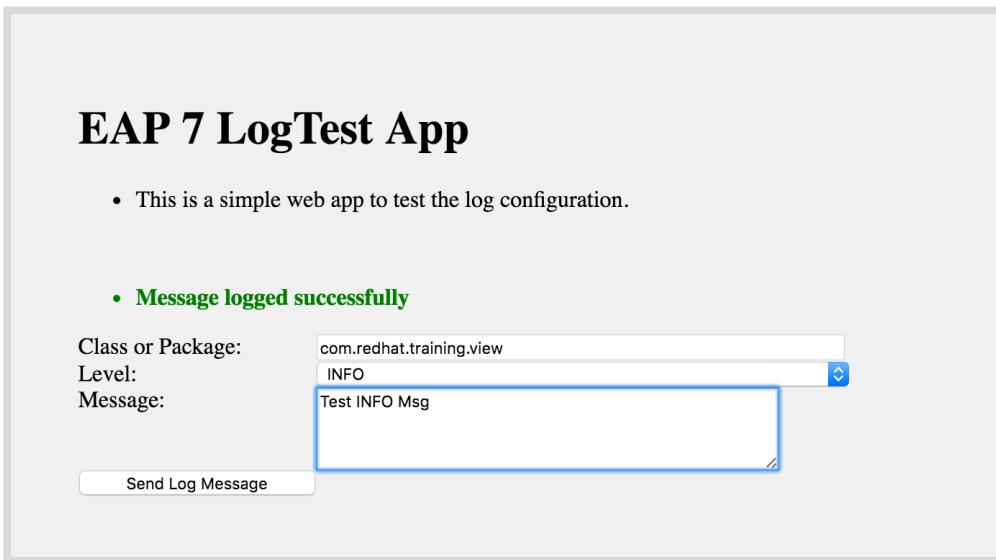
The output should appear as follows:

```
{
  "outcome" => "success",
  "result" => {
    "append" => true,
```

```
"autoflush" => true,  
"enabled" => true,  
"encoding" => undefined,  
"file" => {  
    "relative-to" => "jboss.server.log.dir",  
    "path" => "production-server.log"  
},  
"filter" => undefined,  
"filter-spec" => undefined,  
"formatter" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n",  
"level" => "INFO",  
"max-backup-index" => 3,  
"name" => "FILE_BY_SIZE_ROTATING",  
"named-formatter" => undefined,  
"rotate-on-boot" => false,  
"rotate-size" => "1024m",  
"suffix" => undefined  
}  
}
```

▶ 5. Test Logging

- 5.1. Navigate to <http://127.0.0.1:8080/logtest/> on the workstation VM to access the logtest application.



- 5.2.
 - Enter **com.redhat.training.view** in the **Class or Package** field.
 - Select **INFO** level in the **Level** drop-down list.
 - Enter the text **Test INFO Msg** in the **Message** field.

Click **Send Log Messages** to send the log message to the handler you created in the previous step.
- 5.3. Open a new terminal window to view the log file at **/home/student/JB248/labs/standalone/log/production-server.log** and verify that you can see the messages logged by the application.

```
[student@workstation ~]$ tail -f /home/student/JB248/labs/standalone/log/production-server.log  
01:52:05,392 INFO [com.redhat.training.view] (default task-10) Test INFO Msg
```

- 5.4. Send a few more **INFO** messages from the application and verify that the messages appear in the log file.
- 5.5. Change the **Class or Package** field to **com.redhat.training** in the logtest application and verify that the messages DO NOT appear in the log file. This is because the handler is configured to process log messages only from the **com.redhat.training.view** package by the logger. The messages will be displayed, however, in the console window where you started EAP because the **CONSOLE** handler is a child of the **ROOT** logger which logs all **INFO** level messages from any package.

► 6. Clean Up

- 6.1. Undeploy the **logtest.war** application:

```
[standalone@localhost:9990 /] undeploy logtest.war
```

- 6.2. Exit the EAP CLI:

```
[standalone@localhost:9990 /] exit
```

- 6.3. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

Stop the **tail** command by pressing **Ctrl+C** in the terminal window that is running the **tail** command.

This concludes the guided exercise.

Configuring Loggers

Objectives

After completing this section, students should be able to:

- Configure category loggers.
- Configure the root logger.

Defining loggers

Java developers use logging extensively in their source code as a means of communicating the status of the application based on certain events. Not all log events are to be considered as signs of an issue with the application: for example, a log event can be used to communicate that something worked successfully. In a running EAP server, every single log event that appears within the Java code may or may not be important. By using loggers, these events can be filtered by:

- specifying the log level, and
- specifying a category name.

The logger **category** defines the source of the event. It is the standard convention to use a fully qualified Java class name as the logger category in Java source code, but the category name could be any string that makes sense to the developer.

The logger **level** defines the severity of the event. For example, it represents a fatal error the application cannot recover from, or it provides just debugging information to the developer.

A logger is defined using the **<logger>** tag within the logging subsystem configuration and the format is as follows:

```
<logger category="java_class_and/or_package_name">
  <level name="log_level"/>
</logger>
```

Logger categories follow a hierarchical rule. If a category is defined for a certain package, then all classes in that package and any subpackages are included in the category. For example, the following logger sets the log level to **WARN** for any Java class in the **org.jboss.jca** package:

```
<logger category="org.jboss.jca">
  <level name="WARN"/>
</logger>
```

The CLI command to define this logger is:

```
/profile=default/subsystem=logging/logger=org.jboss.jca:add(\n  category=org.jboss.jca, level=WARNING)
```

Chapter 7 | Configuring the Logging Subsystem

The category can be as specific as desired, including configuring a logger for a single class. For example, the following logger sets the log level to **DEBUG** for just the **org.jboss.jca.core.naming.JndiBinder** class:

```
<logger category="org.jboss.jca.core.naming.JndiBinder">
    <level name="DEBUG"/>
</logger>
```

The CLI command to define this logger is:

```
/profile=default/subsystem=logging/logger=org.jboss.jca.core.naming.JndiBinder:\n    add(category=org.jboss.jca.core.naming.JndiBinder, level=DEBUG)
```

Loggers can be configured and managed using the EAP management console in the **Configuration → Subsystems → Logging** section. Click **Logging**, then the **Log Categories** tab to view, edit, or add loggers identified by their **category** attributes:

Name	Level
com.arjuna	WARN
org.jboss.as.config	DEBUG
sun.rmi	WARN

Figure 7.11: Log categories

A logger can also specify which handler to use. Consider the following example:

```
<logger category="org.jboss.as" ①>
    <level name="INFO" ②/>
    <handlers>
        <handler name="FILE" ③/>
    </handlers>
</logger>
```

- ① The **category** attribute represents the name of the logger and maps to a fully qualified class or package name, for example, if the application resides in the

- `com.mycompany.myapp` package, a new logger can be defined by adding `<logger category="com.mycompany.myapp">`.
- ❷ The `level` attribute sets the default log level for this logger. The logger accepts all log events that are equal or higher in priority.
 - ❸ The `handler` attribute denotes a valid `handler` reference defined in the configuration file. Multiple handlers can be listed within the `handlers` tag. The logger sends all relevant messages to the defined handlers and `ALSO` to handlers inherit from ancestor loggers.

Understanding the logger hierarchy

Loggers are organized in a tree-like hierarchy based on their category setting. The dot (.) is used as a separator between hierarchy levels, following Java package naming rules. If a logger is defined for a subpackage of another logger, it becomes a child of that logger. For example, `com.mycompany.onlinestore` is a child logger of `com.mycompany`.

Applications and EAP components always instantiate at runtime the specific child logger they require. Following the previous example, an application Java class named `com.mycompany.onlinestore.CheckOut` creates a logger using its name as the category.

If a logger is NOT explicitly configured, its parent logger configuration is used. If the parent logger is also not configured, the grandparent configuration is used, and so on. The Root logger serves as default configuration for all loggers that were not explicitly configured not any of its ancestors.

Most times a specific logger is only configured when it (or its child loggers) require a different level setting or a different handler. If not, the settings from an ancestor will be used.

The following example shows a configuration where generic EAP log events are filtered at a very high level (`WARN`) but the JBeret subsystem gets a lower level (`INFO`) so it has more events sent to whatever handler both inherited from the root logger:

```
<logger category="org.wildfly">
    <level name="WARN"/>
</logger>
<logger category="org.wildfly.jberet">
    <level name="INFO"/>
</logger>
```

Logger configurations like the previous example inherits all handlers from the root logger. Most administrators assume it would be enough to configure a different handler in any logger to have its events sent ONLY to that handler, but this is NOT true: A logger always inherits all parent handlers UNLESS this behavior is turned off by changing the `use-parent-handlers` attribute of the logger.

For example, the following example sends all events from the online store application to the handler named `STORE_APP_HANDLER` and `also` to the handlers configured by the Root logger:

```
<logger category="com.mycompany.onlinestore">
    <level name="INFO"/>
    <handlers>
        <handler name="STORE_APP_HANDLER"/>
    </handlers>
</logger>
```

That is, log events generated by the application classes might be sent to multiple handlers even if the logger configuration names only a single handler.

Chapter 7 | Configuring the Logging Subsystem

If the intended result was having all log events from the application being sent to only the **STORE_APP_HANDLER** and **not** to the handlers configured by the root logger, set the **use-parent-handlers** attribute to **false**:

```
<logger category="com.mycompany.onlinestore" use-parent-handlers="false">
    <level name="INFO"/>
    <handlers>
        <handler name="STORE_APP_HANDLER"/>
    </handlers>
</logger>
```

Configuring the root logger

The loggers have a hierarchy where settings from parent loggers are inherited by the child logger. At the top of the logger hierarchy is a unique logger called the *root logger*. The root logger is sort of a catch-all in that its settings are associated with every single log event that is not specifically associated with a child logger. Because it resides at the top of the logger hierarchy, the settings of the root logger are inherited by all other loggers, and all other loggers can override these settings.

In the EAP server configuration files, the root logger is configured as:

```
<root-logger>
    <level name="INFO"/>
    <handlers>
        <handler name="CONSOLE"/>
        <handler name="FILE"/>
    </handlers>
</root-logger>
```

Notice the default log level of all events is **INFO**, and these events get logged at the console and with the **FILE** handler, which is the **server.log** file.

The root logger can be configured for a particular need. For example, suppose on the production server the log is at the **ERROR** level and it should disable the logging to the console, the following configuration will satisfy this requirement:

```
<root-logger>
    <level name="ERROR"/>
    <handlers>
        <handler name="FILE"/>
    </handlers>
</root-logger>
```

The logging subsystem has several CLI operations pertaining to the root logger. These are available in the **/subsystem=logging/root-logger=ROOT**: namespace of the CLI:

- **change-root-log-level**: changes just the logging level attribute of the root logger.
- **set-root-logger**: assigns a new root-logger definition.
- **remove-root-logger**: removes the entire root-logger definition.
- **root-logger-assign-handler**: defines the handlers for the root logger.
- **root-logger-unassign-handler**: removes a handler from the list of root logger handlers.

Chapter 7 | Configuring the Logging Subsystem

For example, the following command changes the root logger level to **ERROR** in the default profile:

```
/profile=default/subsystem=logging/root-logger=ROOT:change-root-log-level(level=ERROR)
```

Of course, root logger and other logger objects **level** and **handler** attributes can be changed directly, without resorting to these specialized operations. They exist to make it simpler to make changes like adding or removing a single handler.

The root logger can also be configured and managed using the EAP management console in the **Configuration → Subsystems → Logging** section. Click on **Logging** button, then the **ROOT LOGGER** tab to view, edit or add root loggers:

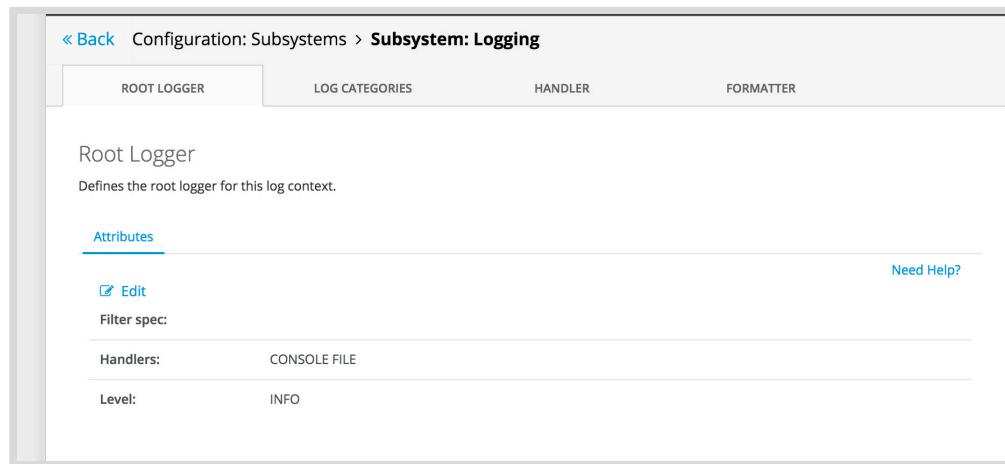


Figure 7.12: The ROOT logger

Log level priority

Log levels are arranged in decreasing order of verbosity (levels lower in the priority ordering result in more verbose logs). Each log event is tagged with a specific level. Loggers and handlers can be configured with the minimum level not to be discarded. The log level priorities are, from higher to lower:

Log Level	Description
OFF	A special level that disables all logging.
FATAL	A fatal event means that the service cannot continue.
SEVERE	A severe event indicates a serious failure where one or more EAP services have failed.
ERROR	A problem that may prevent the service to continue running.
WARNING	A warning event indicates a potential problem.
WARN	A potentially harmful event has occurred.
INFO	Information about an event. Does not indicate any type of error.
CONFIG	Config is a message level for static configuration messages.

Log Level	Description
FINE	Fine is a message level providing tracing information.
DEBUG	Extra information useful for debugging errors.
TRACE	Allows for deep probing of the JBoss server behavior.
FINER	Indicates a fairly detailed tracing message that is more detailed than FINE .
FINEST	Indicates a highly detailed tracing message that is more detailed than FINER .
ALL	Indicates that all messages should be logged.

The recommended approach in production servers is to use **ERROR** or **WARN** level and **NOT** use **INFO** level and below as much as possible to avoid verbose logging. Whenever there is an issue to be investigated, the log levels can be set to a lower level for a short period of time to collect and observe log messages and then turned up to **ERROR** or **WARN** level when the troubleshooting is complete.

If possible, add category loggers corresponding to the application package namespace or the subsystem of EAP under investigation rather than change the root logger level to avoid too much verbose logging which could cause the application to slow down due to I/O bottlenecks.



Note

EAP server restart is not required after changing log levels. Log messages at the appropriate level start appearing in the log files automatically depending on the logger and handler configuration.

► Guided Exercise

Controlling Logging Levels

In this lab, you will manipulate the logging levels of the size rotating file handler that you created in the previous lab and deploy an application that uses this handler to log messages.

Resources	
Files:	/home/student/JB248/labs/standalone /home/student/JB248/labs/logging-levels
Application URL:	http://localhost:8080/logtest
Resources	/home/student/JB248/labs/logging-levels/ logtest.war

Outcomes

You should be able to change the log levels of the handler and view log messages generated by the application in the EAP server log files.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, that the previous guided exercise is complete, that a size rotating file handler is defined, and to download the **logtest.war** application:

```
[student@workstation ~]$ lab logging-levels setup
```

► 1. Start the Standalone EAP Server

Use the following command to start an EAP instance using the **/home/student/JB248/labs/standalone** folder as the base directory:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Wait for the server to finish starting before proceeding.

► 2. Verify that a size rotating file handler called **FILE_BY_SIZE_ROTATING** has been defined using the EAP CLI. Open a new terminal window and run the following commands:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect
[standalone@localhost:9990] /subsystem=logging/
size-rotating-file-handler=FILE_BY_SIZE_ROTATING:read-resource
```

The output should appear as follows:

```
{  
    "outcome" => "success",  
    "result" => {  
        "append" => true,  
        "autoflush" => true,  
        "enabled" => true,  
        "encoding" => undefined,  
        "file" => {  
            "relative-to" => "jboss.server.log.dir",  
            "path" => "production-server.log"  
        },  
        "filter" => undefined,  
        "filter-spec" => undefined,  
        "formatter" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n",  
        "level" => "INFO",  
        "max-backup-index" => 3,  
        "name" => "FILE_BY_SIZE_ROTATING",  
        "named-formatter" => undefined,  
        "rotate-on-boot" => false,  
        "rotate-size" => "1024m",  
        "suffix" => undefined  
    }  
}
```

- 3. You will now deploy the **logtest.war** file on EAP to generate messages at various log levels.

Run the following command:

```
[standalone@localhost:9990] deploy \  
/home/student/JB248/labs/logging-levels/logtest.war
```

► 4. Test Logging Levels

- 4.1. Navigate to <http://127.0.0.1:8080/logtest/> to access the logtest application.

EAP 7 LogTest App

- This is a simple web app to test the log configuration.

- Message logged successfully**

Class or Package:	com.redhat.training
Level:	INFO
Message:	INFO Log Level Test
<input type="button" value="Send Log Message"/>	

4.2.

- Enter **com.redhat.training.view** in the **Class or Package** field.
- Select **INFO** level in the **Level** drop-down list.
- Enter the text **INFO Log Level Test** in the **Message** field.

Click **Send Log Messages** to send the log message to the handler you created in the previous step.

4.3. Open a new terminal window to view the log file at **/home/student/JB248/labs/standalone/log/production-server.log** and verify that you can see the messages logged by the application.

```
[student@workstation ~]$ tail -f \
/home/student/JB248/labs/standalone/log/production-server.log
...
04:09:53,623 INFO [com.redhat.training.view] (default task-27) INFO Log Level
Test
```

- Send a few more **INFO** messages from the application and verify that the messages appear in the log file.
- Change the **Level** field to **DEBUG** in the logtest application and verify that the messages DO NOT appear in the log file. This is because the handler is configured to only process **INFO** and all levels below **INFO** in the logger hierarchy, whereas **DEBUG** is higher than **INFO** in the logger hierarchy.
- Change the **Level** field to **WARN** in the logtest application and verify that the messages appear in the log file. This is because the handler is configured to process **INFO** and all levels below **INFO** in the logger hierarchy.
- Now change the default log level of the **FILE_BY_SIZE_ROTATING** handler to **DEBUG**.

Run the following command:

```
[standalone@localhost:9990] /subsystem=logging/\
size-rotating-file-handler=FILE_BY_SIZE_ROTATING:\\
write-attribute(name=level,value=DEBUG)
```

- 4.8. Change the **Level** field to **DEBUG** in the logtest application and verify that the messages now appear in the log file. This is because the handler is now configured to process **DEBUG** and all levels below **DEBUG** in the logger hierarchy.
- 4.9. Change the **Level** field to **INFO** in the logtest application and verify that the messages appear in the log file. This is because **INFO** is below **DEBUG** in the logger hierarchy.
- 4.10. Change the **Level** field to **TRACE** in the logtest application and verify that the messages DO NOT appear in the log file. This is because **TRACE** is above **DEBUG** in the logger hierarchy and the handler will not process messages at this level.

► 5. Clean Up

- 5.1. Undeploy the **logtest.war** application:

```
[standalone@localhost:9990 /] undeploy logtest.war
```

- 5.2. Exit the EAP CLI:

```
[standalone@localhost:9990 /] exit
```

- 5.3. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

Stop the **tail** command by pressing **Ctrl+C** in the terminal window that is running the **tail** command.

This concludes the guided exercise.

▶ Lab

Configuring the Logging Subsystem

In this lab, you will configure the logging subsystem in an EAP managed domain and deploy the bookstore application to verify logging.

Resources	
Files	/opt/domain /var/log/jboss
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore

Outcome

You should be able to configure the logging subsystem of an EAP 7 managed domain and deploy the bookstore application to verify that application logs are generated in a single centralized location.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab logging-lab-final setup
```

1. You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, but the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively and the domain controller on the workstation. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and configure the **logging** subsystem of the **full-ha** profile in the managed domain.

As a final step, you will deploy the bookstore WAR file to the managed domain and verify that application logs from all the server instances in the managed domain are generated under the **/var/log/jboss** folder from an NFS mount point already configured and pointing to the workstation VM.
2. Start the domain controller on the workstation. Because the domain controller configuration files are kept in the **/opt/domain** folder on workstation, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

3. The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.
 - 3.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
 - 3.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
 - 3.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
4. By default, the logging subsystem is configured with two handlers: a **console-handler** that logs messages to the console; and a **periodic-rotating-file-handler** that logs messages to a file under the **/opt/domain/servers/<SERVER_NAME>/log** folder on both **servera** and **serverb**. These log files are rotated on a daily basis.

To easily back up and archive EAP log files, the system administrator has requested you to configure a size-based rotating file handler that stores all server logs under a single central location for the entire managed domain. This central logging location is the **/var/log/jboss** folder on **servera** and **serverb** VMs. This folder is configured as an NFS remote share mounted from the workstation VM. Because the names of the server instances in the managed domain are unique, the logs from a particular server instance should be stored under the appropriately named folder. For example, if a server is named **server-one**, the logs must be stored under the **/var/log/jboss/server-one/** folder.

Create a new **size-rotating-file-handler** to handle this requirement. You can use either the EAP management console or the EAP CLI to achieve your objective.

- 4.1. Recall from the demonstration and the guided exercises that a **size-rotating-file-handler** accepts a **relative-path** value and a file name as input. Usually, the relative paths point to the **log** directory under the EAP base directory. In this lab, you need to change this **relative-path** value to point to the **/var/log/jboss** folder.

We might change the path of the centralized logging folder in the future and hence referring to the full path must be avoided and instead the EAP concept of a **path** variable can be used. A **path** variable refers to logical name for a file system path. The **domain.xml**, **host.xml** and **standalone.xml** configurations all include a section where paths can be declared. Other sections of the configuration can then reference those paths by their logical name, rather than having to include the full details of the path.

Access the JBoss EAP CLI. In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller as the **jboss** user:

Create a new *path* variable called **custom.log.dir** and set its value to **/var/log/jboss**. In the EAP management console, this can be done in the **Configuration → Paths** section.

- 4.2. After you have defined a new path variable, you can refer to it when creating handlers. Create a new **size-rotating-file-handler** called **BOOKSTORE_LOG_HANDLER** with the following characteristics:
 - **Name:** **BOOKSTORE_LOG_HANDLER**
 - **Path (name of log file):** \${jboss.server.name}/bookstore.log
 - **Relative To (relative path variable):** **custom.log.dir**
 - **Append, Autoflush, Enabled:** Enable all these attributes (if using the CLI, set to 'true')
 - **Level:** **DEBUG**
 - **Rotate Size:** 1m (1 MB)
 - **Max Backup Index:** 5
- 4.3. Verify the new handler that you created in the step above using either the management console or the JBoss EAP CLI.
5. In this lab, you are going to deploy only the bookstore application on the managed domain. You have already set up a new handler to capture the log files from the application. In production environments, the recommended approach is to set the **ROOT** logger to a very low level such as **ERROR** or **WARN** to avoid too much verbose log output, and to set application specific logger categories to **INFO** or **DEBUG** level to gather and analyze application specific issues.

Create a new logger category called **com.redhat.training** set to **DEBUG** level by default, which is the package hierarchy for the **bookstore** application. Associate this logger with the **BOOKSTORE_LOG_HANDLER** you created in the previous step to have bookstore related log messages show up in the **bookstore.log** file.

Change the **ROOT** logger level to **WARN**.
6. Reload all servers configuration in the domain to apply the log changes made so far.
 - 6.1. Reload the configuration from all servers in the **Group1** server group.

Verify that the **bookstore.war** is deployed on **servera.1** and **serverb.1** because they are part of the **Group1** server group. Observe the console window of **servera** and **serverb** for any error messages or warnings. Because you reset the **ROOT** logger to **WARN** level, you should only see **WARN** and **ERROR** level messages in the console window of **servera** and **serverb**.
 - 6.2. Verify that you can access the bookstore application at <http://172.25.250.10:8080/bookstore> and <http://172.25.250.11:8080/bookstore>.
 - 6.3. Verify that you can view the logs for the bookstore application under the **/var/log/jboss/servera.1** and **/var/log/jboss/serverb.1** folders. Note how each server instance has its own folder under which the log files are created. After the log file

Chapter 7 | Configuring the Logging Subsystem

reaches 1 MB (rotate size), additional log files with a numbered suffix are created and rotated until the number of files reaches the **max-backup-index** value.

You should see the following **DEBUG** level logs in the **bookstore.log** of **servera.1** and **serverb.1**:

```
15:06:18,563 DEBUG [com.redhat.training.utils.DatabasePopulator] (ServerService Thread Pool -- 90) Loaded catalog successfully!!!
15:06:18,647 DEBUG [com.redhat.training.utils.DatabasePopulator] (ServerService Thread Pool -- 90) Loaded Customers successfully!!!
15:06:18,649 DEBUG [com.redhat.training.utils.DatabasePopulator] (ServerService Thread Pool -- 90) Loaded Promotions successfully!!!
```

7. Using either the administration console or the JBoss EAP CLI, undeploy the bookstore application and shut down the servers, server groups, host controllers, and the entire managed domain.
 - 7.1. Undeploy the bookstore application.
 - 7.2. Verify that the bookstore application is no longer accessible from **servera.1** and **serverb.1**.
 - 7.3. Stop all servers in **Group1**. Verify that the bookstore application is no longer accessible.
 - 7.4. Shut down the host controller on **servera**. Observe the console window of **servera** and verify that the host controller has been shut down.
 - 7.5. Shut down the host controller on **serverb**. Observe the console window of **serverb** and verify that the host controller has been shut down.

8. Clean Up and Grading

- 8.1. Press **Ctrl+C** to stop the domain controller. Alternatively, you can shut down the domain controller using the JBoss EAP CLI command **/host=master:shutdown()**.
- 8.2. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab logging-lab-final grade
```

This concludes the lab.

► Solution

Configuring the Logging Subsystem

In this lab, you will configure the logging subsystem in an EAP managed domain and deploy the bookstore application to verify logging.

Resources	
Files	/opt/domain /var/log/jboss
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore

Outcome

You should be able to configure the logging subsystem of an EAP 7 managed domain and deploy the bookstore application to verify that application logs are generated in a single centralized location.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab logging-lab-final setup
```

1. You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, but the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively and the domain controller on the workstation. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and configure the **logging** subsystem of the **full-ha** profile in the managed domain.

As a final step, you will deploy the bookstore WAR file to the managed domain and verify that application logs from all the server instances in the managed domain are generated under the **/var/log/jboss** folder from an NFS mount point already configured and pointing to the workstation VM.
2. Start the domain controller on the workstation. Because the domain controller configuration files are kept in the **/opt/domain** folder on workstation, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ --host-config=host-master.xml
```

3. The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

- 3.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.

Open a new terminal window on the **servera** VM and run the following command:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 3.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.

Open a new terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 3.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
4. By default, the logging subsystem is configured with two handlers: a **console-handler** that logs messages to the console; and a **periodic-rotating-file-handler** that logs messages to a file under the **/opt/domain/servers/<SERVER_NAME>/log** folder on both **servera** and **serverb**. These log files are rotated on a daily basis.

To easily back up and archive EAP log files, the system administrator has requested you to configure a size-based rotating file handler that stores all server logs under a single central location for the entire managed domain. This central logging location is the **/var/log/jboss** folder on **servera** and **serverb** VMs. This folder is configured as an NFS remote

share mounted from the workstation VM. Because the names of the server instances in the managed domain are unique, the logs from a particular server instance should be stored under the appropriately named folder. For example, if a server is named **server-one**, the logs must be stored under the **/var/log/jboss/server-one/** folder.

Create a new **size-rotating-file-handler** to handle this requirement. You can use either the EAP management console or the EAP CLI to achieve your objective.

- 4.1. Recall from the demonstration and the guided exercises that a **size-rotating-file-handler** accepts a **relative-path** value and a file name as input. Usually, the relative paths point to the **log** directory under the EAP base directory. In this lab, you need to change this **relative-path** value to point to the **/var/log/jboss** folder.

We might change the path of the centralized logging folder in the future and hence referring to the full path must be avoided and instead the EAP concept of a *path* variable can be used. A *path* variable refers to logical name for a file system path. The **domain.xml**, **host.xml** and **standalone.xml** configurations all include a section where paths can be declared. Other sections of the configuration can then reference those paths by their logical name, rather than having to include the full details of the path.

Access the JBoss EAP CLI. In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss \
/opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```

Create a new *path* variable called **custom.log.dir** and set its value to **/var/log/jboss**. In the EAP management console, this can be done in the **Configuration → Paths** section.

```
[domain@172.25.250.254:9990 /] /path=custom.log.dir:\nadd(path=/var/log/jboss/)\n{\n    "outcome" => "success",\n    ...\n}
```

- 4.2. After you have defined a new path variable, you can refer to it when creating handlers. Create a new **size-rotating-file-handler** called **BOOKSTORE_LOG_HANDLER** with the following characteristics:

- **Name:** **BOOKSTORE_LOG_HANDLER**
- **Path (name of log file):** \${jboss.server.name}/bookstore.log
- **Relative To (relative path variable):** **custom.log.dir**
- **Append, Autoflush, Enabled:** Enable all these attributes (if using the CLI, set to 'true')
- **Level:** **DEBUG**
- **Rotate Size:** **1m (1 MB)**
- **Max Backup Index:** **5**

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging/\
size-rotating-file-handler=BOOKSTORE_LOG_HANDLER:add\
(file={"relative-to"=>"custom.log.dir", \
"path"=>"${jboss.server.name}/bookstore.log"}, \
enabled=true, append=true, autoflush=true, \
rotate-size=1m, max-backup-index=5, \
level=DEBUG)
{
    "outcome" => "success",
    ...
}
```

- 4.3. Verify the new handler that you created in the step above using either the management console or the JBoss EAP CLI.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging/\
size-rotating-file-handler=BOOKSTORE_LOG_HANDLER:read-resource
{
    "outcome" => "success",
    ...
}
```

5. In this lab, you are going to deploy only the bookstore application on the managed domain. You have already set up a new handler to capture the log files from the application. In production environments, the recommended approach is to set the **ROOT** logger to a very low level such as **ERROR** or **WARN** to avoid too much verbose log output, and to set application specific logger categories to **INFO** or **DEBUG** level to gather and analyze application specific issues.

Create a new logger category called **com.redhat.training** set to **DEBUG** level by default, which is the package hierarchy for the **bookstore** application. Associate this logger with the **BOOKSTORE_LOG_HANDLER** you created in the previous step to have bookstore related log messages show up in the **bookstore.log** file.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging/\
logger=com.redhat.training:add\
(category=com.redhat.training, level=DEBUG, handlers=["BOOKSTORE_LOG_HANDLER"])
{
    "outcome" => "success",
    ...
}
```

Change the **ROOT** logger level to **WARN**.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging/\
root-logger=ROOT:write-attribute(name=level,value=WARNING)
{
    "outcome" => "success",
    ...
}
```

6. Reload all servers configuration in the domain to apply the log changes made so far.

- 6.1. Reload the configuration from all servers in the Group1 server group.

```
[domain@172.25.250.254:9990 /] /server-group=Group1:reload-servers
```

Verify that the **bookstore.war** is deployed on **servera.1** and **serverb.1** because they are part of the **Group1** server group. Observe the console window of **servera** and **serverb** for any error messages or warnings. Because you reset the **ROOT** logger to **WARN** level, you should only see **WARN** and **ERROR** level messages in the console window of **servera** and **serverb**.

- 6.2. Verify that you can access the bookstore application at <http://172.25.250.10:8080/bookstore> and <http://172.25.250.11:8080/bookstore>.
- 6.3. Verify that you can view the logs for the bookstore application under the **/var/log/jboss/servera.1** and **/var/log/jboss/serverb.1** folders. Note how each server instance has its own folder under which the log files are created. After the log file reaches 1 MB (rotate size), additional log files with a numbered suffix are created and rotated until the number of files reaches the **max-backup-index** value.

You should see the following **DEBUG** level logs in the **bookstore.log** of **servera.1** and **serverb.1**:

```
15:06:18,563 DEBUG [com.redhat.training.utils.DatabasePopulator] (ServerService Thread Pool -- 90) Loaded catalog successfully!!!
15:06:18,647 DEBUG [com.redhat.training.utils.DatabasePopulator] (ServerService Thread Pool -- 90) Loaded Customers successfully!!!
15:06:18,649 DEBUG [com.redhat.training.utils.DatabasePopulator] (ServerService Thread Pool -- 90) Loaded Promotions successfully!!!
```

7. Using either the administration console or the JBoss EAP CLI, undeploy the bookstore application and shut down the servers, server groups, host controllers, and the entire managed domain.

- 7.1. Undeploy the bookstore application.

```
[domain@172.25.250.254:9990 /] undeploy bookstore.war \
--all-relevant-server-groups
```

- 7.2. Verify that the bookstore application is no longer accessible from **servera.1** and **serverb.1**.
Verify that the bookstore application is **NOT** accessible at URL <http://172.25.250.10:8080/bookstore> and <http://172.25.250.11:8080/bookstore>.
 - 7.3. Stop all servers in **Group1**. Verify that the bookstore application is no longer accessible.

```
[domain@172.25.250.254:9990 /] /server-group=Group1:\nstop-servers(blocking=true)\n{\n    "outcome" => "success",\n    ...\n}
```

Chapter 7 | Configuring the Logging Subsystem

- 7.4. Shut down the host controller on **servera**. Observe the console window of **servera** and verify that the host controller has been shut down.

```
[domain@172.25.250.254:9990 /] /host=servera:shutdown()  
{  
    "outcome" => "success",  
    ...  
}
```

- 7.5. Shut down the host controller on **serverb**. Observe the console window of **serverb** and verify that the host controller has been shut down.

```
[domain@172.25.250.254:9990 /] /host=serverb:shutdown()  
{  
    "outcome" => "success",  
    ...  
}
```

8. Clean Up and Grading

- 8.1. Press **Ctrl+C** to stop the domain controller. Alternatively, you can shut down the domain controller using the JBoss EAP CLI command **/host=master:shutdown()**.
- 8.2. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab logging-lab-final grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- The EAP logging subsystem is made up of three components:
 - Handlers: Determine "where" and "how" an event will be logged. EAP comes with several in-built handlers that can be used to log messages from applications.
 - Loggers: Organize events into logically related categories. A category usually maps to a certain package namespace that is running within the JVM.
 - Root Logger: Loggers are organized in a parent-child hierarchy and the root logger resides at the top of the logger hierarchy.
- EAP comes with a number of useful in-built handlers that can be used to configure the logging subsystem.
- By default, in EAP there are two handlers defined: a **CONSOLE** handler, which logs messages to the system console (terminal); and a **FILE** handler, which logs to a file called **server.log**. Both of these handlers handle log messages at the **INFO** level by default.
- Logger categories can be used for fine grained control over what events to log. A logger can reference one or more handlers.
- Log levels are also arranged in a hierarchical manner that determines the verbosity of the logging. Logging occurs at different levels, and you can configure the minimum level for your particular needs. The **root-logger** sits at the top of the logger hierarchy and its settings are associated with every log event that is not specifically associated with a child logger. The settings of root logger are inherited by all other loggers.
- The logging subsystem components like handlers, loggers and the ROOT logger can be configured and managed in multiple ways using the EAP management console, the EAP CLI or manually editing the XML configuration files (not recommended).
- The log levels can be changed dynamically at runtime to assist with troubleshooting issues in production without requiring a server restart.

Chapter 8

Configuring the Messaging Subsystem

Overview

- Goal** Describe the messaging architecture and configure various messaging resources.
- Objectives**
- Describe the Java Messaging Service and the architecture of the messaging subsystem.
 - Configure connection factories and destinations.
 - Configure messaging journals and other messaging subsystem settings.
- Sections**
- Exploring the Messaging Subsystem (and Quiz)
 - Configuring Messaging Resources (and Guided Exercise)
 - Configuring Journals and Other Settings (and Guided Exercise)
- Lab**
- Configuring the Messaging Subsystem

Exploring the Messaging Subsystem

Objectives

After completing this section, students should be able to:

- Describe the Java Messaging Service and the architecture of the messaging subsystem.

Messaging concepts

Messaging middleware, also known as **Message-Oriented Middleware (MOM)** is responsible for managing application messages and delivering them to the intended recipient. This kind of messaging has no relationship to popular forms of Internet user messaging like email and instant messaging. MOM middleware usually provide features such as security, routing, guaranteed delivery, classes of services, and transactions.

The MOM middleware can be embedded inside other products, much like the way EAP 7 embeds a MOM based on **ActiveMQ Artemis**. It can also be, and usually is, a standalone middleware server, requiring its own specialized management, tuning and clustering.

Messaging is the basis of many **Enterprise Integration Patterns (EIP)**. Architecting applications around messaging has been popular for decades in banking and other industries. Do not consider messaging as an "old" technology; it is also a popular pattern for online stores and other kinds of web applications. Messaging importance is on the rise because of cloud and mobile applications.

MOMs and messaging patterns provide a loosely-coupled communication mechanism for software components that makes changing business processes easier and also makes the resulting applications more scalable and resilient. As an example use case for messaging, consider the checkout process for a web store. It triggers several actions:

- Authorize and confirm payment.
- Remove the goods from inventory.
- Schedule the goods for delivery.

All those actions take time to process and doing them as part of the checkout request means users may have to wait for a long time until their order are confirmed. But they do NOT need to be performed right away; the checkout action could just send a message to a payment application, and return to the user instantly. When the payment application gets confirmation of the credit to the store, it could send a message to an inventory component. When the goods are available and sorted, the inventory application could send a message to a delivery component.

Each action could be handled by a different back-end system, easing capacity planning. If one component cannot perform its action right away (e.g., there are items missing in inventory) the action can be either postponed without canceling the entire process, or an alternative process flow can be triggered. If something outside the organization control is offline (for example the credit card operator system) the process do not need to be canceled either; a new attempt can be done at a later time.

Overview of the Java Messaging Service (JMS)

The **Java Message Service (JMS)** is a specification for accessing messaging middleware, allowing Java components to send and receive messages without being directly connected to each other.

JMS provides a standardized API for accessing MOM services, much like JDBC provides a standardized API for accessing relational databases. It also defines some common semantics the MOM is expected to follow so that applications are not dependent on particular product idiosyncrasies.

The JMS API is based around three main resource types:

ConnectionFactory

Encapsulates the connection to a MOM. JEE 7 states that a default **ConnectionFactory** is provided by the application server, so that applications requiring messaging can be deployed right away, without knowledge of connection parameters. Additional **ConnectionFactories** can be configured by the administrator to server-specific application needs or point to external MOMs from different vendors.

Queue

Messages are retrieved in a first-in, first-out basis. A producer places a message in the queue, and a single consumer removes the message.

Topic

Implement a publish-subscribe method where messages published to the topic are delivered to multiple consumers that are subscribed to the topic. If a consumer is offline, the MOM keeps messages for later delivery to that particular consumer.

JMS **Queues** and **Topics** are both called **destinations** by JMS, and many operations apply to both, such as sending a message. Applications refer to them not by the internal MOM identifier, but by using a **JNDI name**. This name is under the control of the application server administrator, creating an indirection level that allows redirecting applications to produce and consume from different destinations without touching Java source code, allowing easier changes to the business processes implemented using messaging.

JMS **ConnectionFactories** are also referenced by using a JNDI name. This way applications can be directed to different MOMs, even different products when requirements change. It is very common to use this feature to connect to the application server embedded MOM during development, but connect to an external MOM for production.

Like JDBC, the JMS API can be used by standalone Java SE applications, outside a JEE application server. In this case, the application needs to embed the **JMS provider** specific to the MOM and there is no standard way to package and load this provider.

Inside an application server, the JMS API collaborates with the **Java Connection Architecture (JCA)** API to provide a portable way to package the JMS provider as a **Resource Adapter Archive (RAR)** that can be deployed to any certified JEE application server.

RAR files are generic middleware drivers, much like JDBC drivers, but not tied to a specific kind of middleware. Different vendors provide RAR files for connecting JEE application servers to their middleware products, such as ERP systems and Transaction Monitors. The JMS specification builds over this JCA capability instead of providing similar features, unlike JDBC, whose origins predate Java EE.

A JMS overview is not complete without presenting the message structure. A JMS message is made of three parts:

- **Headers:** are usually created and processed by the MOM. The JMS API defines a few headers that applications may use, but they are usually treated as part of the MOM infrastructure and in most cases should not be relied upon by application logic.
- **Properties:** are defined by the producer as a way to pass metadata to both the MOM and consumers. One example is routing a message to different consumers based on the value of a property. The previous example of a web store could use properties to deliver orders to different payment processing systems.
- **Body:** contains application-specific data that a MOM should not try to handle, just pass along. Components that perform message transformations are, from the JMS point of view, regular client applications and not part of the MOM infrastructure.

An introduction to EAP built-in messaging

Apache ActiveMQ Artemis is an open source project to build a multiprotocol, embeddable, very high performance, clustered, asynchronous messaging system. The **Artemis** subproject is the next generation **Apache ActiveMQ** messaging and Enterprise Integration Patterns server, based on the Red Hat code donation of the **HornetQ** project.

ActiveMQ was already very popular because of its multiprotocol support and native clients for a range of programming languages and Operating Systems. HornetQ was also very popular because of its network and disk I/O performance but supported only Java clients. The Artemis project brings the strengths of both offering into the same product.

Red Hat bought **FuseSource** in 2012, the company behind ActiveMQ commercial support, and rebranded the product JBoss A-MQ server. This left Red Hat with two competing MOM products, both strong among Java EE developers.

In 2014 Red Hat and the HornetQ community decided to join efforts with the ActiveMQ community and to facilitate this Red Hat donated the HornetQ sources to the Apache Software Foundation, creating the Artemis project.

The current **Red Hat JBoss A-MQ** releases, at the time of EAP 7.0.0 GA, was still based on the original ActiveMQ project. It is expected a future release will be based on the Artemis project. This future release will feature the external appearance of the current A-MQ product (administration interfaces based on Fabric, protocol supported, and native client libraries) but with many of the internals from HornetQ (asynchronous I/O engine and high performance journaled disk storage).

The messaging server embedded in EAP 7 leaves out from ActiveMQ Artemis the components that provide support for protocols such as STOMP and AMQP. Native client libraries for languages other than Java are also absent in EAP 7.

To understand EAP messaging versus A-MQ from a product standpoint: the first is positioned by Red Hat as a general messaging and integration platform, and the second is positioned as a complete JMS messaging server solution.

Some Java EE application server vendors provide an embedded JMS server that are mainly useful only for development and small scale production use. This is NOT the case with EAP 7; although there are missing protocols, EAP 7 JMS server is fully supported and capable of dealing with high performance and mission critical production scenarios. It is just not meant to be used outside the Java EE world.

Customers requiring backward compatibility with EAP 6 messaging, based on the HornetQ project, have native protocol support. EAP 6 servers can publish and consume messages from EAP 7 servers, and vice versa.

To use Red Hat JBoss A-MQ instead or EAP built-in messaging, install the A-MQ JCA Resource Adaptor into EAP 7 and configure JMS **ConnectionFactories** using it. The same procedure (use the product -specific JCA Resource Adaptor) is expected to be used when a future A-MQ release becomes based on Artemis.

The activemq-messaging subsystem

ActiveMQ Artemis embedded into JBoss EAP 7 is the **messaging-activemq** subsystem and managed using EAP 7 CLI and Management Console. The subsystem syntax and managed resources follow a different syntax compared to the EAP 6 **messaging** subsystem but administrators will recognize many of the same features and concepts.

Activating the **messaging-activemq** subsystem implies using either the **standalone-full.xml** or **standalone-full-ha.xml** configuration files for standalone server mode, or either the **full** or **full-ha** profiles for managed domains.

The subsystem has two child objects in the default EAP 7 configuration, as can be seen by the following CLI command:

```
[domain@127.0.0.1:9990 /] /profile=full/subsystem=messaging-activemq:read-resource
{
    "outcome" => "success",
    "result" => {
        "jms-bridge" => undefined,
        "server" => {"default" => undefined}
    }
}
```

The child objects are:

- **jms-bridge**: contains bridges for transferring messages automatically between the embedded JMS server and any other JMS server known by the EAP 7 domain, including servers from other vendors. No bridge comes predefined in the standard configuration files.
- **server=default**: the embedded JMS server. It contains a number of attributes and child objects that configure all MOM aspects, from JMS resources to networking, storage and clustering. The most common configurations are the subject of the remainder of this chapter.

Clustering with ActiveMQ is different than general EAP 7 clustering, supporting a number of different approaches, such as in-memory replication and shared storage. This course does not intend to detail all those alternative configurations.

For now it suffices to say that the default ActiveMQ clustering configuration, which is the same in the **standalone-full-ha.xml** configuration file and the **full-ha** profile, does NOT use **Infinispan** for replication. Instead the ActiveMQ server automatically creates bridges connecting cluster members, and employs a smart message distribution algorithm, where cluster members without consumers do not receive any replication traffic.

Creating those bridges requires shared authentication credentials stored in the **cluster-password** attribute. The default EAP configuration files set this attribute from the value of the **jboss.messaging.cluster.password** system property.

Applications deployed into an EAP 7 server instance connect by default to the local messaging server, to avoid network overhead. They are not affected by clustering configurations, in the sense that, if their EAP server fails, their messaging server has also failed. Nothing prevents

administrators from configuring JMS **ConnectionFactories** that connect to remote servers, and in that case the EAP server works just like any other remote client.

Remote JMS clients learn about all messaging cluster members the same way cluster members find each other, using **JGroups**. Connections are distributed among cluster members, so no single messaging server becomes a bottleneck. Remote JMS client connections automatically fail over to a surviving cluster member if their current connection fails, and no message is lost in the process.

General EAP clustering is the subject of a following chapter in this book and it presents JGroups configurations. For more information about ActiveMQ configuration and tuning, not limited to clustering, see the EAP 7 and A-MQ product documentation.



References

For the JMS specification:

JSR 343 at JCP

<https://jcp.org/aboutJava/communityprocess/final/jsr343/index.html>

For information about the open source projects ActiveMQ and Artemis:

ActiveMQ project page at Apache Foundation

<http://activemq.apache.org/>

ActiveMQ Artemis project page

<http://activemq.apache.org/artemis/>

For specific information about ActiveMQ embeded in EAP 7:

EAP 7 product documentation: Configuring Messaging

<https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0/configuring-messaging/configuring-messaging>

For information about the Red Hat JBoss A-MQ product:

A-MQ product page at Red Hat

<https://www.redhat.com/en/technologies/jboss-middleware/amq>

A-MQ product documentation

<https://access.redhat.com/documentation/en/red-hat-jboss-a-mq/>

Course JB437 - Red Hat JBoss A-MQ Development and Deployment

<https://www.redhat.com/en/services/training/jb437-red-hat-jboss-mq-development-and-deployment>

► Quiz

The Messaging Subsystem

Choose the correct answer to the following questions:

► 1. Which of the following is the messaging server embedded into EAP 7? (Choose one.)

- a. HornetQ
- b. ActiveMQ Artemis
- c. ActiveMQ Apollo
- d. JBossMQ
- e. Apache Qpid
- f. Red Hat MRG
- g. RabbitMQ

► 2. An administrator needs to connect EAP 7 to an external stand-alone MOM. Which JMS resource is needed to implement that? (Choose one.)

- a. **ConnectionFactory**
- b. **DataSource**
- c. **Queue**
- d. **Topic**
- e. **SecurityDomain**
- f. **Destination**

► 3. Which of these JEE APIs focus on client applications instead of server-side components? (Choose three.)

- a. JMS
- b. JCA
- c. EJB
- d. JDBC
- e. Servlets

► 4. Which of the following use cases would be good matches for JMS Queues? (Choose two.)

- a. Sending an employee expense report for refund by the organization HR department.
- b. Sending updated airplane locations to other airports traffic control systems.
- c. Sending the customer an email notification about a loan credit analysis results.
- d. Delivering news to social media sites.

- ▶ **5. Which of the following use cases would be good matches for JMS Topics? (Choose two.)**
- a. Delivering an insurance application for risk analysis by an automated system.
 - b. Getting goods out of inventory for delivery to a customer.
 - c. Sending updates about stock quotes prices to brokers.
 - d. Notifying project members about a new meeting invitation.
- ▶ **6. Which part of a JMS message should an application insert data to influence how the message is routed to specific consumers? (Choose one.)**
- a. Body
 - b. Headers
 - c. Trailers
 - d. Properties
- ▶ **7. How does an application reference JMS resources such as ConnectionFactories and Queues? (Choose one.)**
- a. By following a name convention where the application deployment name is part of the resource name.
 - b. By using the name assigned to the resource in the application server JNDI tree.
 - c. By using the name assigned to the resource by the MOM.
 - d. By looking for a configuration directive in the application deployment descriptor file **web.xml**.
- ▶ **8. Which of the following CLI paths would match the EAP 7 embedded messaging server in the default configuration files? (Choose two.)**
- a. `/profile=full/subsystem=messaging-activemq/server=default`
 - b. `/profile=full/subsystem=messaging-activemq/server=artemis`
 - c. `/profile=full-ha/subsystem=messaging/server=default`
 - d. `/subsystem=messaging-activemq/server=artemis`
 - e. `/subsystem=messaging/server=default`
 - f. `/subsystem=messaging-activemq/server=default`

► Solution

The Messaging Subsystem

Choose the correct answer to the following questions:

► 1. Which of the following is the messaging server embedded into EAP 7? (Choose one.)

- a. HornetQ
- b. ActiveMQ Artemis
- c. ActiveMQ Apollo
- d. JBossMQ
- e. Apache Qpid
- f. Red Hat MRG
- g. RabbitMQ

► 2. An administrator needs to connect EAP 7 to an external stand-alone MOM. Which JMS resource is needed to implement that? (Choose one.)

- a. **ConnectionFactory**
- b. **DataSource**
- c. **Queue**
- d. **Topic**
- e. **SecurityDomain**
- f. **Destination**

► 3. Which of these JEE APIs focus on client applications instead of server-side components? (Choose three.)

- a. JMS
- b. JCA
- c. EJB
- d. JDBC
- e. Servlets

► 4. Which of the following use cases would be good matches for JMS Queues? (Choose two.)

- a. Sending an employee expense report for refund by the organization HR department.
- b. Sending updated airplane locations to other airports traffic control systems.
- c. Sending the customer an email notification about a loan credit analysis results.
- d. Delivering news to social media sites.

- ▶ **5. Which of the following use cases would be good matches for JMS Topics? (Choose two.)**
- a. Delivering an insurance application for risk analysis by an automated system.
 - b. Getting goods out of inventory for delivery to a customer.
 - c. Sending updates about stock quotes prices to brokers.
 - d. Notifying project members about a new meeting invitation.
- ▶ **6. Which part of a JMS message should an application insert data to influence how the message is routed to specific consumers? (Choose one.)**
- a. Body
 - b. Headers
 - c. Trailers
 - d. Properties
- ▶ **7. How does an application reference JMS resources such as ConnectionFactories and Queues? (Choose one.)**
- a. By following a name convention where the application deployment name is part of the resource name.
 - b. By using the name assigned to the resource in the application server JNDI tree.
 - c. By using the name assigned to the resource by the MOM.
 - d. By looking for a configuration directive in the application deployment descriptor file `web.xml`.
- ▶ **8. Which of the following CLI paths would match the EAP 7 embedded messaging server in the default configuration files? (Choose two.)**
- a. `/profile=full/subsystem=messaging-activemq/server=default`
 - b. `/profile=full/subsystem=messaging-activemq/server=artemis`
 - c. `/profile=full-ha/subsystem=messaging/server=default`
 - d. `/subsystem=messaging-activemq/server=artemis`
 - e. `/subsystem=messaging/server=default`
 - f. `/subsystem=messaging-activemq/server=default`

Configuring Messaging Resources

Objectives

After completing this section, students should be able to:

- Configure JMS connection factories and destinations.
- Describe ActiveMQ connectors and acceptors.

ActiveMQ connectors and acceptors

During the remainder of this book the EAP 7 embedded ActiveMQ Artemis is referred to as ActiveMQ. Its network configuration is based on two components:

- Acceptors** define networking protocols and parameters for accepting connections from messaging clients.
- Connectors** define networking protocols and parameters for connecting to ActiveMQ servers.



Note

Note that ActiveMQ use the term **connector** to refer to a **client-side** component, while other EAP subsystems usually refer to a **server-side** component when using this term.

Because EAP 7 acts as both a server and a client for ActiveMQ, the default EAP 7 configuration files includes ready-to-use definitions for both component types. Two kinds of ActiveMQ acceptors and connectors are provided:

- http**: uses the native ActiveMQ protocol tunneled through an HTTP connection. The HTTP connection is accepted by the **undertow** subsystem which is the subject of a dedicated chapter in this book. This way an EAP 7 server instance does NOT require opening additional firewall ports to accept connections from remote messaging clients.
- in-vm**: allows messaging clients running under the same JVM as the ActiveMQ server to connect without networking overhead.

Networking parameters for an ActiveMQ connector are defined indirectly, by referring to an acceptor element. The connector connects to whatever IP address and TCP port the acceptor is configured to accept connections. This indirection is related to the way ActiveMQ remote clients use a discovery mechanism to find servers to connect to. The following listing shows the connectors and acceptors in the default EAP 7 configuration files:

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0">
  <server name="default">
    ...
    <http-connector name="http-connector" socket-binding="http"
      endpoint="http-acceptor" ①/>
    ...
    <in-vm-connector name="in-vm" server-id="0"/> ②
  </server>
</subsystem>
```

```

<http-acceptor name="http-acceptor"③ http-listener="default"④ />
...
<in-vm-acceptor name="in-vm" server-id="0"/>
...
</server>
</subsystem>

```

Callouts in the previous listing highlight the relationship between connectors and acceptors:

- ① ③ The **<http-connector>** element named **http-connector** references the **<http-acceptor>** element named **http-acceptor** by using the **endpoint** attribute.
- ② The **<in-vm-connector>** element does not need to reference any **<in-vm-acceptor>** element because local clients do NOT need discovery to find the messaging server.
- ④ The **<http-acceptor>** element references the **http-listener** named **default**, which is defined by the **undertow** subsystem.

A third connector and acceptor type is supported by EAP 7: the **remote** type. It allows remote clients to connect to the messaging server using the ActiveMQ native protocol instead of HTTP tunneling. The native protocol is required by remote clients running outside an EAP 7 server and also to connect to standalone ActiveMQ servers.

Creating a new **<remote-connector>** and **<remote-acceptor>** pair on EAP 7 also requires creating a new **<socket-binding>** to provide the IP address and TCP port to be used by the **<remote-acceptor>**. For more information refer to the EAP product documentation.

The remote connector and acceptor type also supports connections to EAP 6 and standalone HornetQ servers but additional configurations are required. Refer to the EAP 7 product documentation for details.

Defining JMS ConnectionFactories and PooledConnectionFactories

JMS **ConnectionFactory** resources defined inside the **messaging-activemq** subsystem refer to an ActiveMQ connector defined in the same subsystem. The following listing shows the **connection-factory** resources in the default EAP 7 configuration files:

```

<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0">
    <server name="default">
        ...
        <connection-factory name="InVmConnectionFactory" connectors="in-vm"
            entries="java:/ConnectionFactory"/>
        <connection-factory name="RemoteConnectionFactory" connectors="http-
            connector" entries="java:jboss/exported/jms/RemoteConnectionFactory"/>
        ...
    </server>
</subsystem>

```

The previous listing shows two **<connection-factory>** elements, each referring to a different **connector** and bound to different JNDI names using the **entries** attribute:

- The first **<connection-factory>** element is named **InVmConnectionFactory** and refers to the **<in-vm-connector>** element named **in-vm**. It serves applications using the JMS API in a Java SE-style but deployed to EAP 7. It should NOT be used by enterprise components such as Servlets and EJBs.

- The second **<connection-factory>** element is named **RemoteConnectionFactory** and refers to the **<http-connector>** element named **http-connector**. It serves remote JMS clients connecting to the embedded ActiveMQ in EAP 7.

Java EE applications should use NONE of those JMS **ConnectionFactory** resources. Instead they should use a JMS **PooledConnectionFactory** which provides:

- Reuse of connections to the messaging server.
- XA transaction integration.
- JAAS security context propagation.

The following listing shows the **pooled-connection-factory** resource in the default EAP 7 configuration files:

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0">
    <server name="default">
        ...
        <pooled-connection-factory name="activemq-ra" transaction="xa"
            connectors="in-vm"
            entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory"/>
        ...
    </server>
</subsystem>
```

This one will be used by Java EE applications when no JMS **ConnectionFactory** is declared. Notice it refers to the **in-vm** connector. If a different messaging server will be used by default, it should either be changed to refer to a different connector element or replaced by one referring to a JCA Resource Adapter.

An application may refer to a JMS **ConnectionFactory** by a JNDI name that might not exist. Thus, the administrator must create a new **pooled-connection-factory** resource under the same JNDI referring to the correct connector. For example, to use the JNDI name **java:/MyCF** and point to the embedded messaging server, add the following to the **messaging-activemq** subsystem:

```
<pooled-connection-factory name="mycf" transaction="xa" connectors="in-vm"
    entries="java:/MyCF"/>
```

The following CLI command creates the resource from the previous listing under the **full** profile for an EAP 7 managed domain:

```
[domain@localhost:9990 /] cd /profile=full/subsystem=messaging-activemq/
server=default
[domain@localhost:9990 server=default] ./pooled-connection-factory=mycf:add(\n
connectors=[in-vm], entries=[java:/jms/MyCF])
```

If an application refers to a non-existing JNDI name it might deploy without errors, and throw Java **NamingException** errors only at run-time, when the application tries to actually connect to the messaging server.

**Note**

The first EAP 7 release only allowed plain **connection-factory** resources to be configured using the web management interface. The CLI had to be used to create or configure **pooled-connection-factory** resources.

The reasoning behind using a JMS **PooledConnectionFactory** rather than a plain JMS **ConnectionFactory** is the same as a JDBC **DataSource** versus a **DriverManager**. Although JMS **ConnectionFactory** and **DriverManager** can be used by Java EE applications they are not efficient because:

- Either the application includes additional code to manage sharing connections between multiple threads, but loses scalability by doing so, or,
- The application creates and destroys connections for each user request, which are expensive operations.

Both JDBC **DataSource** and JMS **PooledConnectionFactory** objects bring connection pools to improve application performance and scalability. The difference is that, with JDBC, the developer has to code to the right interface, but for JMS only the administrator has to create the correct resource type: a JMS **PooledConnectionFactory** is also a JMS **ConnectionFactory**.

**Note**

Creating a JMS **PooledConnectionFactory** that refers to an external, non-ActiveMQ MOM, requires a number of steps:

1. Either package the MOM vendor JMS provider as a JBoss module and add it to the EAP installation or deploy the RAR package if provided by the vendor.
2. Define a JNDI external context to refer to the JMS provider JNDI tree, in the **naming** subsystem.
3. Add a JCA resource adapter configuration to the **resource-adapters** subsystem.
4. Configure the **ejb** subsystem to use the new resource adapter for MDBs.

The EAP 7 product documentation includes examples of those configurations for popular MOMs, for example Tibco EMS and WebSphere MQ.

Defining JMS destinations

When using the embedded MOM from EAP 7, creating the JMS destinations resources also creates the underlying MOM objects. This facility is provided by the **messaging-activemq** subsystem that closely integrates EAP 7 with the embedded ActiveMQ.

**Note**

If using an external MOM, JMS destinations are configured as part of the JMS Provider configuration in the **naming** and **resource-adapters** subsystems, but creating them does NOT creates the underlying objects in the MOM. This course deals only with the embedded messaging server.

Chapter 8 | Configuring the Messaging Subsystem

To create a destination using the CLI, create either a **jms-queue** or a **jms-topic** resource. For example, the following commands create a JMS Topic named stocks bound to the JNDI name **java:/jms/broker/StockUpdates**:

```
[domain@localhost:9990 server=default] ./jms-topic=stocks:add(\nentries=[java:/jms/broker/StockUpdates])
```

The previous command adds the following to the domain mode configuration files:

```
<jms-topic name="stocks" entries="java:/jms/broker/StockUpdates"/>
```

The **jms-topic** resource attributes are:

- **name**: the name of the managed object. This name is only useful for administration, it is NOT used by applications using the JMS API.
- **entries**: provide one or more JNDI names for the resource. These are the names visible to applications.
- **legacy-entries**: allow this destination to be used by EAP 6 and HornetQ remote clients.

A **jms-queue** resource can have all attributes from a **jms-topic** resource add also:

- **durable**: if true, messages in this queue are saved to disk and are not lost on MOM restart. If false, messages are kept only in memory and can be discarded in situations such as out-of-memory conditions.
- **selector**: provides a filter for messages accepted by this queue. For more information refer to the JMS specification.

JMS destinations for the embedded MOM in EAP 7 can also be created and configured using the management console. In managed domain mode, enter **Configuration > Profiles > full (or full-ha) > Messaging-ActiveMQ > default**, to enter the **Messaging Provider** page, as illustrated by the following figure:

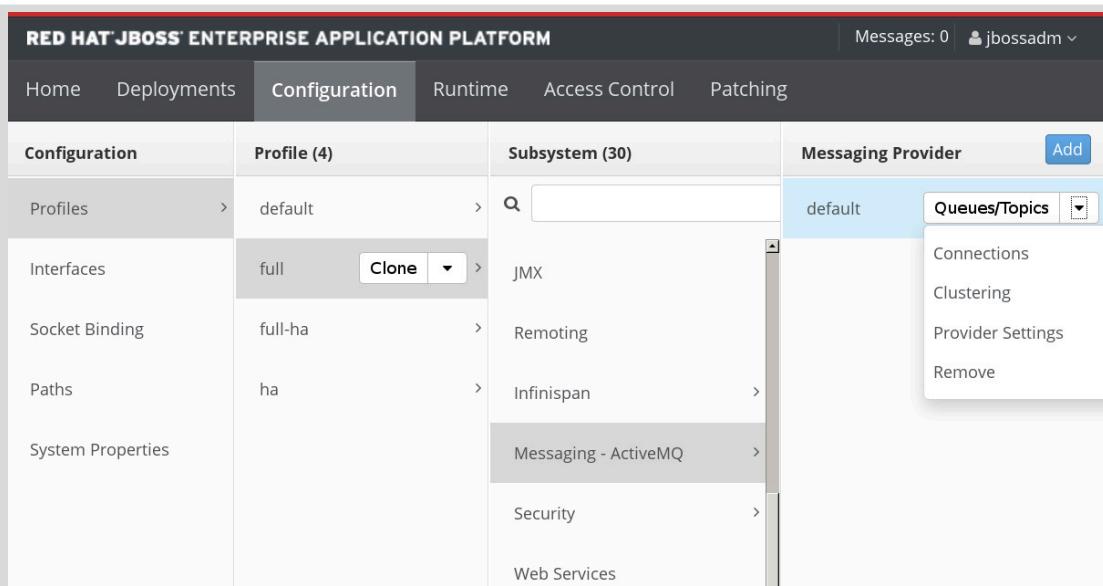


Figure 8.1: EAP 7 Messaging Provider selection page.

In the **Messaging Provider** page, click **Queues/Topics** to see the **JMS Endpoints** configuration page, as illustrated by the following figure:

The screenshot shows the 'MESSAGING DESTINATIONS' section of the EAP 7 configuration interface. Under 'Queues/Topics', the 'Queues' tab is selected. A table lists two entries:

Name	JNDI
ExpiryQueue	[java:/jms/queue/ExpiryQueue]
DLQ	[java:/jms/queue/DLQ]

Below the table, detailed configuration for the 'ExpiryQueue' row is shown:

- Name: ExpiryQueue
- JNDI Names: java:/jms/queue/ExpiryQueue
- Durable?: true
- Selector:

Figure 8.2: EAP 7 JMS Endpoints page

From that page new destinations can be created and existing ones can be changed or removed. The links on the left also allow configuring other messaging parameters and are explained later in this chapter.

Monitoring JMS destinations

JMS destinations in the embedded ActiveMQ from EAP 7 can be monitored using a few object run-time attributes and operations. Use the CLI operations **read-resource-description** and **read-resource-operations** to learn about all attributes and operations made available by EAP 7 to monitor JMS destinations. Only a few of them are presented in this course.

In standalone server mode, the monitoring attributes and operations are available from the same object that defines the resource as part of the server profile. For example, to check the number of messages waiting to be consumed in the **jms-queue** named **MyQueue**, use the following command to read the **message-count** attribute:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] ./jms-queue=MyQueue:read-attribute(
  name=message-count)
```

Another interesting attribute is the **messages-added** attribute, which counts the number of messages published to the destination since the MOM was started; that is, since the EAP server instance was last started or reloaded.

In managed domain server mode, the same attributes and operations are available from a running server instance and NOT from the subsystem configuration. For example, to list messages waiting

to be consumed in the **jms-queue** named **MyQueue** from the **srv1** server in the **devhost** host, use the following command to invoke the **list-messages** operation:

```
[domain@localhost:9990 /] cd /host=devhost/server=srv1
[domain@localhost:9990 server=srv1] cd ./subsystem=messaging-activemq/
server=default
[domain@localhost:9990 server=default] ./jms-queue=MyQueue:list-messages
```

Messages are listed using DMR syntax but for each message only headers and properties are shown as object attributes. There is no way to see a message body using the EAP CLI. The following is a sample output of the previous operation:

```
{
  "outcome" => "success",
  "result" => [
    {
      "JMSPriority" => 4,
      "JMSMessageID" => "ID:b921a30-12e4-11e6-ae18-597e323e1397",
      "address" => "jms.queue.TestQueue",
      "JMSExpiration" => 0,
      "__AMQ_CID" => "f51cfb7a-12e4-11e6-ae18-597e323e151397",
      "MyAppProperty" => "MyAppValueXYZ",
      "JMSTimestamp" => 1462468442322L,
      "messageID" => 5,
      "JMSDeliveryMode" => "PERSISTENT"
    },
    {
      "JMSPriority" => 4,
      "JMSMessageID" => "ID:f5136421-12e4-11e6-ae18-597e323e2def",
      "address" => "jms.queue.TestQueue",
      "JMSExpiration" => 0,
      "__AMQ_CID" => "3d0cfb7a-12e4-11e6-ae18-597e323e2def",
      "OtherAppProperty" => "OtherValueASD",
      "JMSTimestamp" => 1462468442339L,
      "messageID" => 9,
      "JMSDeliveryMode" => "PERSISTENT"
    }
  ]
}
```

In the previous listing, the **result** attribute is an array containing two objects and each object a single message. Attributes of each message object are headers and properties of the JMS message. Some of them, for example those with the **JMS** prefix, are defined by the JMS specification. Other headers, for example **address** and **messageID**, are defined by the MOM. Finally, the **MyAppProperty** and **OtherAppProperty** properties are defined by the application and have no meaning for either the JMS API or the MOM.

**Note**

There is no way to infer, just from the **list-messages** operation output, which attributes are message headers and which are message properties. Neither is it possible to sort which ones are defined by the MOM or not. Check the EAP and A-MQ product documentation and the JMS specification for information about specific headers and properties. Also check with the application developer about application-defined properties.

Some of the destination attributes can be seen using the management console. Select **Runtime** in the top-level navigation menu and navigate to **Messaging-ActiveMQ** inside a running server. Click **View** to see the **Messaging Statistics** page, and select the **default** messaging provider. Then select a queue (or topic) to see the statistics. Usually more information is available through the CLI.

The following figure illustrates navigating the management console in managed domain mode to get to the **Messaging Statistics** page:

The screenshot shows the Red Hat JBoss Enterprise Application Platform management console interface. The top navigation bar includes links for Home, Deployments, Configuration, Runtime (which is selected), Access Control, and Patching. On the right side of the header, there are 'Messages: 0' and a user dropdown for 'jbossadm'. Below the header is a search bar and a 'Refresh' button. The main content area is a tree-based navigation pane. It starts with 'Hosts' under 'Browse Domain', then 'master', then 'Server Groups'. Under 'Server Groups', there is a 'JVM' dropdown set to 'servera'. From 'servera', two servers are listed: 'servera.1 (Group1)' and 'servera.2 (Group2)'. 'servera.1' has a 'View' button next to it. To the right of the tree, there are several tabs labeled 'Monitor' and 'Subsystem'. Under 'Monitor', there are links for 'JVM', 'Batch', 'Environment', 'Log Files', and 'Subsystems'. Under 'Subsystems', there is a link for 'Messaging - ActiveMQ' with a 'View' button. Other tabs include 'Datasources', 'JPA', 'JNDI View', and 'Transactions'.

Figure 8.3: Navigating to the Messaging-ActiveMQ monitoring page.

Of course, monitoring attributes and operations is different for **jms-queue** and **jms-topic** destinations. For example, only topics have subscribers.

► Guided Exercise

Configuring Messaging Resources

In this lab, you will configure JMS resources required by an application.

Resources	
Files:	/home/student/JB248/labs/messaging-resources
Application URL:	http://localhost:9990 http://localhost:8080/messaging-client

Outcomes

You should be able to create a JMS **ConnectionFactory** and a **Queue** and deploy applications that make use of them.

Before You Begin

This guided exercise makes use of two simple test applications:

1. **messaging-client.war** is a web application that connects to the JNDI address `java:/jms/CustomCF` JMS **ConnectionFactory** and publishes messages to the JNDI address `java:/jms/queue/JB248TestQueue` JMS **Queue**. It accepts text from the user to use as message properties and the message body.
2. **messaging-mdb.jar** is an application with a **Message-Driven Bean** (MDB) that connects to the same JMS **ConnectionFactory** and consumes messages from same JMS **Queue** as the previous application. It shows the message properties and body in the EAP server log.

Both applications will be deployed to a standalone server instance created in the first guided exercise about **Datasources**.

Before beginning the guided exercise, run the following command to verify that EAP is installed at `/opt/jboss-eap-7.0`, that no instance of EAP is running, and to download the files for the exercise:

```
[student@workstation ~]$ lab messaging-resources setup
```

- 1. Start an EAP 7 standalone server instance.

Start an EAP instance using the `/home/student/JB248/labs/standalone` folder as the base directory but using the `standalone-full.xml` configuration file.

Use the following commands:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh --server-config=standalone-full.xml \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Leave this terminal running the EAP standalone server instance.

► 2. Create the JMS **ConnectionFactory** using the JNDI name expected by the applications.

- 2.1. Open another terminal window and start the CLI:

```
[student@workstation ~]$ /opt/jboss-eap-7.0/bin/jboss-cli.sh --connect
```

- 2.2. Create a **pooled-connection-factory** referring to the default **in-vm** ActiveMQ connector and using the JNDI name **java:/jms/CustomCF**.

Run the following command:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default  
[standalone@localhost:9990 server=default] ./pooled-connection-factory=\  
custom:add(connectors=[in-vm], entries=[java:/jms/CustomCF])
```

Leave this terminal running the CLI and connected. We will come back to it later in this exercise.

► 3. Create the JMS **Queue** using the JNDI name expected by the applications.

- 3.1. Open a web browser and access the management console at <http://localhost:9990>
- 3.2. Click **Configuration** on the top-level menu bar, and follow to **Subsystems > Messaging - ActiveMQ > default**. Then click **Queues/Topics** on the combo box to enter the **Messaging Destinations page**.
- 3.3. Click **Add** to open the **Create JMS Queue** form and complete it as follows:

- **Name:** TestQueue
- **JNDI Name:** java:/jms/queue/JB248TestQueue
- **Durable:** checked.
- **Selector:** empty.

Click **Save** to create the JMS **Queue**.

Leave the management console browser tab open because we will come back to it a few times during this exercise.

► 4. Deploy the message producer test application.

- 4.1. Click **Back** in the management console to leave the **Messaging Destinations** page. Then click **Deployments** on the top-level menu bar.
- 4.2. Click **Add** to enter the **Add Deployment** assistant. Select **Upload a new deployment** and click **Next**.

- 4.3. Click **Browse** and choose the **messaging-client.war** file from the **/home/student/JB248/labs/messaging-resources** folder. Then click **Next**.
- 4.4. Do NOT change any field and click **Finish** to upload and enable the producer test application.
- 4.5. Go back to the terminal where EAP was started and check for log messages confirming deployment was successful.

You should see messages similar to:

```
13:08:15,111 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 77) WFLYUT0021: Registered web context: /messaging-client  
13:08:15,128 INFO [org.jboss.as.server] (XNIO-1 task-7) WFLYSRV0010: Deployed "messaging-client.war" (runtime-name : "messaging-client.war")
```



Note

Beware that deployment may be successful even if you made an error in the previous steps, and the JMS resources were created using JNDI names different from expected by the application.

Also leave the application browser tab open because we will come back to it a few times during this exercise.

- 5. Publish two test messages.
- 5.1. Open another browser tab and enter the producer test application using the following URL: <http://localhost:8080/messaging-client>
 - 5.2. Complete the web form as follows:
 - **Message Count: 2**
 - **Message Label: test**
 - **Message to send: testing message sending**Click **Send Message** to produce (or send) the message.
 - 5.3. The web form should refresh and display the following message in green: **Messages sent successfully.**
- 6. Verify the messages were queued inside EAP.
- 6.1. Go back to the browser tab where the management console is open.
Click **Runtime** in the top-level menu bar. Then follow to **Standalone Server > Subsystems > Messaging - ActiveMQ** and click **View** to enter the **Messaging Statistics** page.
 - 6.2. Click **View** on the **default** Messaging provider table row to see the **JMS Queue Metrics** page.
 - 6.3. Click the **TestQueue** row and notice both the **Message Count** and **Messages Added** counter are two. All other counters should be zero.

The following figure illustrates what you should see:

JMS Queue Metrics: Provider 'default'

Metrics for JMS queues.

Name	JNDI
ExpiryQueue	[java:jms/queue/ExpiryQueue]
DLQ	[java:jms/queue/DLQ]
TestQueue	[java:jms/queue/JB248TestQueue]

Queue Metrics

Consumer Count	0
Message Count	2
Messages Added	2
Scheduled Count	0

Figure 8.4: JMS Queue Metrics page after two messages are produced (or sent).

If you see different values on the counters they should reflect other actions you performed using the producer test application. But if your actions are NOT reflected by the counters, there was an error in a previous step. For example: if you publish a message but the Messages Added counter does NOT increase, this mean the publisher application is NOT being able to publish messages, maybe because of JMS resources configured incorrectly.

**Note**

If you created the JMS **ConnectionFactory** or the JMS **Queue** using JNDI names different than expected by the test application, sending a message will generate a **NameNotFoundException** in the EAP server log. For example:

```
16:07:49,811 INFO  [stdout] (default task-7) java.lang.RuntimeException:  
java.lang.RuntimeException: javax.naming.NameNotFoundException: jms/CustomCF  
-- service jboss.naming.context.java.jms.CustomCF
```

If you see errors like the previous one, go back and correct the JNDI names for the JMS resources. Then refresh the application in the browser tab. There should be no need to redeploy the producer test application.

► 7. Inspect the headers and properties of the messages in the queue.

- 7.1. Go back to the terminal where the EAP CLI is connected and check the queue message counters.

Run the following command:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue\  
:read-resource(include-runtime=true)
```

The expected output is similar to:

```
{  
    "outcome" => "success",  
    "result" => {  
        "consumer-count" => 0,  
        "dead-letter-address" => "jms.queue.DLQ",  
        "delivering-count" => 0,  
        "durable" => true,  
        "entries" => ["java:/jms/queue/JB248TestQueue"],  
        "expiry-address" => "jms.queue.ExpiryQueue",  
        "legacy-entries" => undefined,  
        "message-count" => 2L,  
        "messages-added" => 2L,  
        "paused" => false,  
        "queue-address" => "jms.queue.TestQueue",  
        "scheduled-count" => 0L,  
        "selector" => undefined,  
        "temporary" => false  
    }  
}
```

The values for **consumer-count**, **messages-count**, **messages-added** and **scheduled-count** should match those displayed by the management console.

- 7.2. Inspect the headers and properties for messages published but not yet consumed.

Run the following command:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected output is similar to:

```
{  
    "outcome" => "success",  
    "result" => [  
        {  
            "JMSPriority" => 4,  
            "JMSMessageID" => "ID:c3811a30-12e4-11e6-ae18-597e323e15a8",  
            "address" => "jms.queue.TestQueue",  
            "JMSExpiration" => 0,  
            "__AMQ_CID" => "c37cfb7a-12e4-11e6-ae18-597e323e15a8",  
            "Copy" => 1,  
            "JMSTimestamp" => 1462468442322L,  
            "Label" => "test",  
            "messageID" => 17,  
            "JMSDeliveryMode" => "PERSISTENT"  
        },  
        {  
        }
```

```
"JMSPriority" => 4,  
"JMSMessageID" => "ID:c3836421-12e4-11e6-ae18-597e323e15a8",  
"address" => "jms.queue.TestQueue",  
"JMSExpiration" => 0,  
"__AMQ_CID" => "c37cfb7a-12e4-11e6-ae18-597e323e15a8",  
"Copy" => 2,  
"JMSTimestamp" => 1462468442339L,  
"Label" => "test",  
"messageID" => 18,  
"JMSDeliveryMode" => "PERSISTENT"  
}  
]  
}
```

Notice that each message is enclosed by curly brackets { ... } and, for each message, the attributes **Copy** and **Labels** match those typed by the user using the publisher test application. The message body is not visible using the **list-messages** operation.

► **8.** Deploy the message consumer test application.

- 8.1. Go back to the browser tab where the management console is open.
Click **Back** to leave the **JMS Queue Metrics** page, and then click **Deployments** on the top-level menu bar.
- 8.2. Click **Add** to enter the **Add Deployment** assistant. Select **Upload a new deployment** and click **Next**.
- 8.3. Click **Browse** and choose the **messaging-mdb.jar** file from the **/home/student/JB248/labs/messaging-resources** folder. Click **Next**.
- 8.4. Do NOT change any field and click **Finish** to upload and enable the consumer test application.
- 8.5. Go back to the terminal where EAP was started and check for the log messages confirming that deployment was successful.

You should see a log entry similar to the following:

```
15:38:35,555 INFO [org.jboss.as.server] (ServerService Thread Pool -- 76)  
WFLYSRV0010: Deployed "messaging-mdb.jar" (runtime-name : "messaging-mdb.jar")
```



Note

Be aware that deployment may be successful even if you made an error in previous steps, and the JMS resources were created using different names from those expected by the application.

For example, if the JMS **ConnectionFactory** JNDI name is wrong, a **NameNotFoundException** will be seen in the server logs but the deployment operation will NOT fail. The server logs would appear as follows:

```
16:07:35,914 WARN [org.apache.activemq.artemis.ra] (default-threads - 2) AMQ152005: Failure in broker activation
  org.apache.activemq.artemis.ra.inflow.ActiveMQActivationSpec(
    ra=org.apache.activemq.artemis.ra.ActiveMQResourceAdapter@78712571
    destination=jms/queue/JB248TestQueue destinationType=javax.jms.Queue
    ack=Auto-acknowledge durable=false clientID=null user=null
    maxSession=15): javax.naming.NameNotFoundException: jms/CustomCF -- service
    jboss.naming.context.java.jms.CustomCF
16:07:35,925 INFO [org.jboss.as.server] (ServerService Thread Pool -- 84) WFLYSRV0010: Deployed "messaging-mdb.jar" (runtime-name : "messaging-mdb.jar")
```

After an error such as the previous one the consumer test application will NOT be able to consume any messages and will have to be redeployed after the JNDI names are fixed.

- ▶ 9. Verify that all queued messages were consumed.
 - 9.1. The consumer test application includes an MDB that should immediately start consuming messages, and will show each message body and properties in the server log.

The expected log entries are similar to:

```
15:38:35,767 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
  (Thread-1 (ActiveMQ-client-global-threads-427178549)) Message Properties: Copy #1
  [test]
15:38:35,767 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
  (Thread-1 (ActiveMQ-client-global-threads-427178549)) Message Body: testing
  message sending
15:38:35,777 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
  (Thread-0 (ActiveMQ-client-global-threads-427178549)) Message Properties: Copy #2
  [test]
15:38:35,778 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
  (Thread-0 (ActiveMQ-client-global-threads-427178549)) Message Body: testing
  message sending
```

- 9.2. Go back to the browser tab where the Management Console is open. Click **Runtime** in the top-level menu bar. Then follow to **Standalone Server > Subsystems > Messaging - ActiveMQ** and click **View** to enter the **Messaging Statistics** page.

- 9.3. Click **View** on the **default** Messaging provider table row to see the **JMS Queue Metrics** page.
- 9.4. Click the **TestQueue** row and notice the **Messages Added** counter is still at two, but the **Message Count** counter is at zero because there are no messages waiting to be consumed.

JMS Queue Metrics: Provider 'default'

Metrics for JMS queues.

Name	JNDI
ExpiryQueue	[java:jms/queue/ExpiryQueue]
DLQ	[java:jms/queue/DLQ]
TestQueue	[java:jms/queue/JB248TestQueue]

Queue Metrics

Consumer Count 15
Message Count 0
Messages Added 2
Scheduled Count 0

Figure 8.5: JMS Queue Metrics page after two messages are consumed.



Note

The **Consumer Count** counter at 15 means EAP started 15 copies of the consumer MDB to process messages in parallel. Check the EAP 7 product documentation for information on MDB configuration settings such as the number of instances created.

- 10. Publish a third test message.

10.1. Go back to the browser tab where the producer test application is open.

10.2. Complete the form as follows:

- **Message Count: 1**
- **Message Label: third**
- **Message to send: another test message**

Click **Send Message** to publish the message.

10.3. The form should refresh and display the following message in green: **Messages sent successfully.**

- 11. Verify that the message was consumed immediately.

11.1. Go back to the terminal where the EAP standalone server instance is running.

It should show log messages similar to the following:

```
18:25:45,583 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
(Thread-2 (ActiveMQ-client-global-threads-427178549)) Message Properties: Copy #1
[third]
18:25:45,583 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
(Thread-2 (ActiveMQ-client-global-threads-427178549)) Message Body: another test
message
```

11.2. Go back to the terminal where the CLI is running and check the queue message counters again.

Run the following command: (**Hint:** use the arrow keys to rerun the command from history.)

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue\
:read-resource(include-runtime=true)
```

The expected output is similar to:

```
{
  "outcome" => "success",
  "result" => {
    "consumer-count" => 15,
    "dead-letter-address" => "jms.queue.DLQ",
    "delivering-count" => 0,
    "durable" => true,
    "entries" => ["java:/jms/queue/JB248TestQueue"],
    "expiry-address" => "jms.queue.ExpiryQueue",
    "legacy-entries" => undefined,
    "message-count" => 0L,
    "messages-added" => 3L,
    "paused" => false,
    "queue-address" => "jms.queue.TestQueue",
    "scheduled-count" => 0L,
    "selector" => undefined,
    "temporary" => false
  }
}
```

Note that the **message-count** attribute is still at zero, but the **messages-added** counter increased to three.

- 12. Stop the EAP standalone server instance, but leave the test applications and JMS resources, as they will be required by the next guided exercise.

Use the CLI to stop the server instance:

```
[standalone@localhost:9990 server=default] /:shutdown
```

This concludes the guided exercise.

Configure Journals and Other Settings

Objectives

After completing this section, students should be able to:

- Configure messaging journals and other messaging subsystem settings.

Configuring the messaging journals: NIO versus AIO

Persistence in JMS refers to the ability to guarantee delivery of messages in a topic or queue, even if the JMS provider fails. Typically, JMS implementations use a database for implementing persistence. However, ActiveMQ takes the unique approach of using the file system, and the files that store the messages are referred as journals.

ActiveMQ message persistence applies only to queues marked as durable and all topics (because of subscriptions). Other data such as the active subscriber list for each topic is also saved to the file system. There are a few file sets maintained by ActiveMQ but in this course we deal only with the messaging journal.

The ActiveMQ journals are optimized for handling a large volume of messages quickly and reliably. The journal on-disk data structure is similar to the transaction logs used by databases such as PostgreSQL and metadata on file systems such as Linux ext4 and XFS.

Journals have the following properties:

- The journal consists of a set of files on your machine's file system.
- Each file is created a fixed size and filled with padding to maximize disk access performance.
- Messages are appended to the journal. The set of journal files is read and written as a ring buffer, so space from messages already consumed is reused by new published messages.
- When one journal file is full, ActiveMQ moves on to the next one. If there are no remaining empty journal files, ActiveMQ creates a new one.
- ActiveMQ also has a compaction algorithm which recovers space used by messages that were already consumed and eliminates fragmentation inside messaging journals. This might leave a few journal files empty and eligible to be removed.
- The journal also fully supports transactional operations if required, supporting both local and XA transactions.

To maximize ActiveMQ on-disk performance, avoid creating new journal files and running the compaction algorithm. When files have to be created, expanded or deleted, the OS must perform additional I/O operations on file system metadata. If those operations are avoided, the OS can perform the minimal number of physical I/O operations to read and write application data.

To achieve those results, the administrator can configure the **server=default** embedded MOM instance inside the **messaging-activemq** subsystem with the optimum number of messaging journal files and the optimum size for each file. Those numbers are obtained from either capacity planning or load tests.

Chapter 8 | Configuring the Messaging Subsystem

The following listing shows all attributes related to the messaging journal from the default EAP 7 configuration in managed domain mode:

```
[domain@127.0.0.1:9990 /] cd /profile=full/subsystem=messaging-activemq/
server=default
[domain@127.0.0.1:9990 server=default] :read-resource
{
    "outcome" => "success",
    "result" => {
        ...
        "journal-buffer-size" => undefined,
        "journal-buffer-timeout" => undefined,
        "journal-compact-min-files" => 10,
        "journal-compact-percentage" => 30,
        "journal-file-size" => 10485760,
        "journal-max-io" => undefined,
        "journal-min-files" => 2,
        "journal-pool-files" => -1,
        "journal-sync-non-transactional" => true,
        "journal-sync-transactional" => true,
        "journal-type" => "ASYNCIO",
        ...
    }
}
```

As the previous listing shows, the default messaging journal has two (**journal-min-files**) 10 MiB (**journal-file-size**) files. All journal files have the same size and each one can store messages from multiple durable queues and topics.

If the administrator wants to change this to **nine** files **50 MiB** each, the following commands would be used, this time in standalone server mode:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] :write-attribute(\n
    name=journal-min-files, value=9)
[standalone@localhost:9990 server=default] :write-attribute(\n
    name=journal-file-size, value=52428800)
```

The main journal attributes can also be changed using the management console. In managed domain mode, enter **Configuration > Profiles > full (or full-ha) > Messaging-ActiveMQ > default**, to enter the **Messaging Provider** page. Then select **Provider Settings** to show the **Provider Settings** pop-up window and click **Journal** for the messaging journal attributes.

The following figure shows an example with default settings, in managed domain mode. The pop-up window was resized and scrolled down to show all the attributes shown by the current book section.

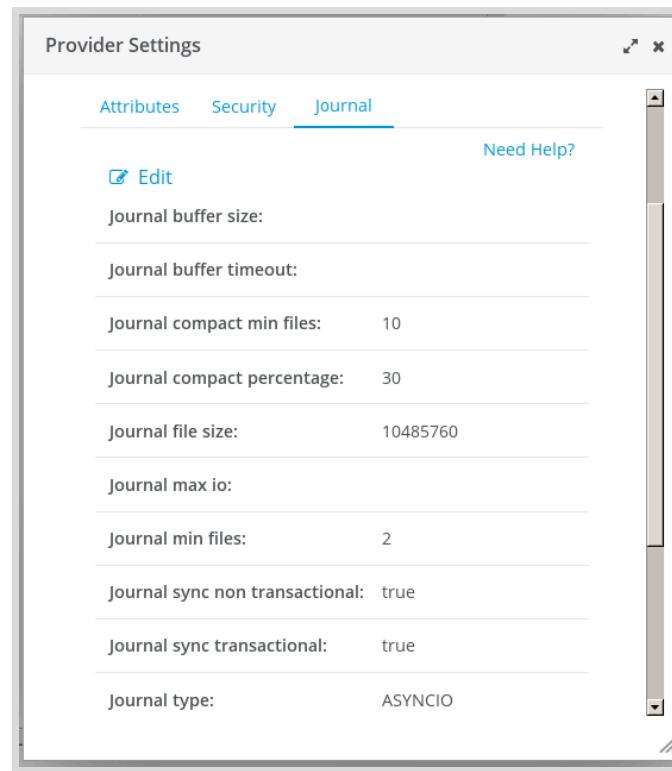


Figure 8.6: EAP 7 JMS Provider window, messaging journal settings

This course does NOT cover any other tuning parameters related to ActiveMQ memory and disk usage, except for the important attribute **journal-type**. This attribute accepts two values:

- **NIO**: uses the standard Java NIO API to interface with the file system. This provides extremely good performance and runs on any platform where there is a Java 6+ runtime. EAP 7 requires at least Java 8.
- **ASYNCIO**: uses the native Linux Asynchronous I/O library (AIO). Using AIO will typically provide even better performance than using Java NIO.

The AIO library requires Linux kernel 2.6 or later and the RPM package *libaio*. Both are installed by default in Red Hat Enterprise Linux 7. For other Linux distributions, check with the OS vendor.

The default value for the **journal-type** is **ASYNCIO** but if the AIO native library cannot be loaded then ActiveMQ falls back to **NIO**. This way EAP 7 embedded messaging uses the best alternative available and there is no need to change the **journal-type** attribute for most installations.

If the administrator wants to make sure EAP 7 is using the AIO library, check the server instance logs for a message similar to the following:

```
2016-05-23 15:28:48,994 INFO [org.apache.activemq.artemis.core.server]
(ServerService Thread Pool -- 68) AMQ221012: Using AIO Journal
```

Other messaging settings

As shown early in this chapter, ActiveMQ destinations have very few attributes. But there are many settings an administrator expects to be able to configure in a production-level MOM. For example: security, flow control, redelivery, and so on.

ActiveMQ puts those settings outside the destinations, in dedicated configuration elements, because most real-world applications require many destinations with similar settings. This is done to ease the administrator work load and lower the risk of having inconsistent settings for destinations from the same application.

There are two configuration elements: **address-settings** and **security-settings**. The first one contains most settings, while the second one contains only access control attributes.

Both element names rely on a specific wildcard syntax to match a destination name to the element name. The wildcard metacharacters are:

- # (pound sign): matches anything. For example: the element `<address-setting name="#>` matches any destination name, be it a queue or a topic. Another example: `stock.us.#` matches `stock.us.rht` and `stock.us.nasdaq.rht` but not `stock.emea.xyz`.
- . (a single period): matches the space between words in a wildcard expression. Most of the time it can be taken as a literal period.
- * (asterisk): matches a single word. For example: `stock.us.*` matches `stock.us.rht` but not `stock.us.nasdaq.rht`. Another example: `stock.*.rht` matches `stock.us.rht` and `stock.emea.rht` but not `stock.us.nasdaq.rht`.

To make an **address-settings** and **security-settings** match a single destination name, use the full destination name as the element name.

Be aware that the destination name is NOT the JNDI name. It is the name used **internally** by ActiveMQ to refer to the destination. For an embedded ActiveMQ inside EAP 7 the internal name is the object name prefixed by `jms.queue.` for a **jms-queue** resource or by `jms.topic.` for a **jms-topic** resource. Notice the dot at the end of both prefixes.

A **jms-topic** also gets a suffix based on the subscriber ID, so an **address-setting** referring to a **jms-topic** should always add the suffix `#`.

The following table shows the name of a destination resource and the name for an **address-setting** that match that destination specifically:

ActiveMQ destination name	ActiveMQ address-setting name
<code>jms-queue=MyQueue</code>	<code>address-settings=jms.queue.MyQueue</code>
<code>jms-topic=MyTopic</code>	<code>address-settings=jms.topic.MyTopic.#</code>

An **address-setting** may configure a number of different features, check the EAP product documentation for description and example usages for each of its attributes. The following examples are included only as syntax examples and they use some of the attributes related to flow control.

- The following example matches only the queue named `jms.queue.Volatile` and limits memory usage for all messages to 100 KiB. If more messages are published that would increase memory use above that threshold, the new messages are silently discarded:

```
[domain@localhost:9990 /] cd /profile=full/subsystem=messaging-activemq/
server=default
[domain@localhost:9990 server=default] ./address-setting=jms.queue.Volatile:add(\n
max-size-bytes=102400, address-full-policy=DROP)
```

- The following example matches all destinations from the sensors application (they all follow a naming convention having the **app.sensors** prefix) and limits memory usage for each destination to 150 KiB. But instead of discarding messages, publishers are prevented from publishing more messages until a few messages are consumed and memory is released:

```
[domain@localhost:9990 /] cd /profile=full/subsystem=messaging-activemq/
server=default
[domain@localhost:9990 server=default] ./address-
setting=jms.queue.app.sensors.*:add(\n
max-size-bytes=153600, address-full-policy=BLOCK)
```

Two important caveats when configuring ActiveMQ **address-settings** are:

- Many independent features are configured by the same object. When creating an **address-setting** object focused on a single feature, such as in the previous examples, attributes related to other features get their default values. This may have unexpectedly side effects, such as a feature that was enabled for all destinations in the default EAP settings become disabled for a few destinations.
- Multiple **address-settings** objects can match the same destination. The most specific match usually overrides less specific ones. Test carefully whether the merged configuration has the expected behavior.

The management console provides a page to create and configure **address-setting** objects alongside the **Queues/Topics** page and is illustrated by the following figure:

Pattern	Dead Letter Address	Expiry Address	Redelivery Delay	Max Delivery Attempts
#	jms.queue.DLQ	jms.queue.ExpiryQueue	0	10

Figure 8.7: EAP 7 Messaging Provider - Address Settings page

Messaging security

Messaging security in EAP 7 deals with both the embedded MOM, that is, ActiveMQ, and the JMS API:

Chapter 8 | Configuring the Messaging Subsystem

- The MOM settings define how to authenticate local and remote clients, and which permissions to grant those clients for different destinations.
- The JMS API is concerned with how to pass credentials from the client to the MOM.

Security for the embedded MOM is configured as attributes for the **server=default** object inside the **messaging-activemq** subsystem. The **security-domain** attribute selects the EAP security domain to be used to authenticate client user credentials and fetch its roles.

The default configuration points to the **other** security domain, which validates user credentials using the **ApplicationRealm** configured in either the **standalone.xml** or the **host.xml** file. For more information about configuring users and roles in the **ApplicationRealm**, see the **Securing JBoss EAP** chapter later in this book.

Security is NOT enforced for applications connecting to the embedded ActiveMQ because the default settings have the **override-in-vm-security** attribute set to **true**. Unless this attribute is changed to **false**, only remote clients have to provide authentication credentials and are subject to authorization checks.

Most production environments require changing default **messaging-activemq** subsystem configurations to use a different security realm, backed by a database or LDAP server. They probably also require enabling authentication from local clients.

To configure permissions for messaging destinations, the administrator configures **security-setting** objects. Those objects work much the same way as **address-setting** objects by using their names as wildcards to match internal ActiveMQ destination names.

A **security-setting** has child objects of type **role** whose attributes define the permissions held by the named role. The standard EAP configuration defines a single **security-setting=#** that contains a single **role=guest** child that allows any user to publish and consume messages on any destination.

The following command inspects the default **security-setting** to show the roles specified by the object:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] ./security-setting=#:read-resource
{
    "outcome" => "success",
    "result" => {"role" => {"guest" => undefined}}
}
```

The previous output shows that only the **guest** role was assigned permissions by the default **security-setting**.

To show the permissions assigned to the role, inspect the **role=guest** child, as in the following example:

```
[standalone@localhost:9990 server=default] ./security-setting=#/role=guest:read-
resource
{
    "outcome" => "success",
    "result" => {
        "consume" => true,
        "create-durable-queue" => false,
        "create-non-durable-queue" => true,
```

```
"delete-durable-queue" => false,  
"delete-non-durable-queue" => true,  
"manage" => false,  
"send" => true  
}  
}
```

An alternative is to pass the **recursive=true** argument to the **read-resource** operation on the **security-setting**:

```
[standalone@localhost:9990 server=default] ./security-setting=#:read-resource(\n  recursive=true)\n{\n    "outcome" => "success",\n    "result" => {"role" => {"guest" => {\n        "consume" => true,\n        "create-durable-queue" => false,\n        "create-non-durable-queue" => true,\n        "delete-durable-queue" => false,\n        "delete-non-durable-queue" => true,\n        "manage" => false,\n        "send" => true\n    }}}\n}
```

Be ware that a single **security-setting** object may provide permissions for many roles, and multiple **security-setting** objects may match the same destination. Different from an **address-setting** object, where multiple matches may affect the same destination, the most specific **security-setting** is used alone. This way a more specific **security-setting** may deny permissions granted by a more generic one.

Production environments probably require the removal of the default permissions assigned to the **guest** role to implement the **secure by default** philosophy. The following command does that:

```
[standalone@localhost:9990 server=default] ./security-setting=#/role=guest:remove
```

The following example shows how to add permissions to a specific role:

```
[standalone@localhost:9990 server=default] ./security-setting=#/role=sender:add(\n  send=true)
```

The previous example shows how to add the **sender** role permission to publish messages to any destination, and nothing else.

Specific permissions, such as **send** and **consume**, and their meanings, are presented later in this course in the **Securing JBoss EAP** chapter.

Handling message delivery failures

One important aspect of MOM management is handling failures to consume messages. Delivery failures might impact performance as a consumer may become stuck in a loop trying to consume

the same message over and over, and generating errors for each attempt to process the message contents.

This scenario might happen because messages that were not acknowledged as consumed are NOT discarded; the messages are left in the destination, to be consumed again at a later time. This is similar to a database rolling back a failed transaction to ensure data ; consistency the message reading operation is undone because its enclosing transaction was aborted.

Most times this is the right thing to do because the failure may be related to something that was supposed to work, such as an offline database server, and discarding the message may mean losing important business transactions, such as an order payment. Trying again at a later time allows the application to recover from unexpected failures.

If the failure happens only with some messages and not others, however, it is helpful to get the failed messages out of the way. This saves hardware resources to process the messages that were not affected by the current issues and allows those messages that generated failures to be dealt with at a later time, after the issues are fixed, by an exception work flow.

For example, an application that processes payments from an online store may accept three payment options: credit card, bank transfers, and gift cards. Processing credit card payments requires calling an external web service. This service sometimes experiences technical difficulties, but other payment options are not affected and they do not need to wait until the credit card service is back online.

ActiveMQ **address-setting** objects allow the administrator to define:

- A time delay to redeliver a failed message, in the hope that the issue does not happen on the next attempt, by using the **redelivery-delay** attribute.
- A destination to forward the message after a number of failed delivery attempts, by using the **max-delivery-attempts** and **dead-letter-address** attributes.

The default **messaging-activemq** settings include the following address setting:

```
[standalone@localhost:9990 server=default] ./address-setting=#:read-resource
{
    "outcome" => "success",
    "result" => {
        ...
        "dead-letter-address" => "jms.queue.DLQ",
        ...
        "max-delivery-attempts" => 10,
        ...
        "redelivery-delay" => 0L,
        ...
    }
}
```

This means that for all destinations (the **address-setting** object name is #), messages that fail to be consumed after 10 attempts (**max-delivery-attempts**) are moved to the **jms.queue.DLQ** destination (**dead-letter-address**). There is zero delay between delivery attempts (**redelivery-delay**).



Important

Remember an important caveat is: the destination names used to match an **address-setting** and for the **dead-letter-address** attribute are not JNDI names as seen by the JEE application. They also are not the name of the destination resources as seen by the CLI. They are internal ActiveMQ destination addresses.

The rules for generating ActiveMQ internal names were presented earlier in this chapter. Another way to get a destination internal name is by using the **jms-queue** or **jms-topic** objects' **queue-address** runtime attribute.

Sorting messages from many applications in the same default **dead-letter queue (DLQ)** is probably not easy and most applications require a custom **address-setting** object specifying a different redelivery policy.

For example, to have a 5-second delay between delivery attempts from the **jms-queue=MyQueue** destination and moving messages to **jms-queue=MyDLQ** after three attempts, use the following command:

```
[standalone@localhost:9990 server=default] ./address-
setting=jms.queue.MyQueue:add(\n
    redelivery-delay=5000, dead-letter-address=jms.queue.MyDLQ, max-delivery-
    attempts=3)
```

A concept closely related to DLQ is the **expiry** concept. It means some messages might not be useful, or they might require processing by different business rules, if they are not consumed in a certain time frame.

ActiveMQ allows an administrator to configure an expiry queue using the **expiry-address** attribute from the **address-setting** object. The JMS API specifies how an application may provide a message expiry interval. An administrator may also use the **expiry-delay** attribute to define a default interval for messages to expire. This default interval does NOT override the one specified by the application.

For example, to have a 20-minute default expiry interval (20 minutes x 60 seconds x 1000 ms = 1200000ms) for messages from the **jms-queue=MyQueue** destination and move messages that took longer to consume to **jms-queue=MyExpiry**, use the following command:

```
[standalone@localhost:9990 server=default] ./address-
setting=jms.queue.MyQueue:add(\n
    expiry-delay=1200000, expiry-address=jms.queue.MyExpiry)
```

If both redelivery and expiry options from the two previous examples are required, a single **address-setting** object has to be created merging all attributes: **address-setting** names must be unique, even if multiple **address-setting** object names can match the same destination internal address.

It was already shown in this chapter that **address-setting** objects can be configured using the management console, and this way redelivery and expiry settings can be configuring using the web management interface instead of the CLI.

► Guided Exercise

Configuring the Journals and Other Settings

In this lab, you will configure a custom dead letter queue and change journal parameters.

Resources	
Files:	NA
Application URL:	http://localhost:9990 http://localhost:8080/messaging-client

Outcomes

You should be able to configure a custom dead letter queue for an application queue and change the number and size of message journal files.

Before You Begin

This guided exercise makes use of two simple test applications:

1. **messaging-client.war** is a web application that connects to the **java:/jms/CustomCF** JMS **ConnectionFactory** and publishes messages to the **java:/jms/queue/JB248TestQueue** JMS **Queue**. It accepts text from the user to use as message properties and the message body.
2. **messaging-mdb.jar** is a **Message-Driven Bean** (MDB) that connects to the same JMS **ConnectionFactory** and consumes messages from the same JMS **Queue** as the previous application. It shows the message properties and body in the EAP server log.

Both applications were deployed by the previous guided exercise, into a standalone server instance.

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, the JMS resources are configured, and the test applications are deployed:

```
[student@workstation ~]$ lab messaging-persistence setup
```

- 1. Start an EAP 7 standalone server instance.
Start an EAP instance using the **/home/student/JB248/labs/standalone** folder as the base directory but using the **standalone-full.xml** configuration file.
Use the following commands:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation ~]$ ./standalone.sh --server-config=standalone-full.xml \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Leave this terminal running the EAP standalone server instance.

► 2. Create a new JMS **Queue** to be used as a DLQ.

2.1. Open another terminal window and start the CLI:

```
[student@workstation ~]$ /opt/jboss-eap-7.0/bin/jboss-cli.sh --connect
```

2.2. Create a **jms-queue** named **TestDLQ** mapped to the JNDI name **java:/jms/JB248TestDLQ**:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default  
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:add(\  
entries=[java:/jms/JB248TestDLQ])
```

2.3. Inspect **TestDLQ** to get its internal ActiveMQ address:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:\  
read-attribute(name=queue-address)
```

The expected output is:

```
{  
    "outcome" => "success",  
    "result" => "jms.queue.TestDLQ"  
}
```

► 3. Configure the new queue as the DLQ for **TestQueue**.

3.1. Inspect **TestQueue** to get its internal ActiveMQ address:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:\  
read-attribute(name=queue-address)
```

The expected output is:

```
{  
    "outcome" => "success",  
    "result" => "jms.queue.TestQueue"  
}
```

3.2. Configure **TestDLQ** as the DLQ for **TestQueue** alone, and to move messages after a single failed delivery attempt.

Use the internal addresses queried in the previous steps to identify the queues when creating the **address-setting** configuration:

```
[standalone@localhost:9990 server=default] ./address-setting=\
jms.queue.TestQueue:add(dead-letter-address=jms.queue.TestDLQ,\
max-delivery-attempts=1)
```

- 3.3. Inspect the newly-created **address-setting** to see that, although it defines dead letter and related redelivery attributes, it does NOT specify any expiration queue and delay:

```
[standalone@localhost:9990 server=default] ./address-setting=\
jms.queue.TestQueue:read-resource
```

The expected output is similar to:

```
{
  "outcome" => "success",
  "result" => {
    "address-full-policy" => "PAGE",
    "auto-create-jms-queues" => false,
    "auto-delete-jms-queues" => false,
    "dead-letter-address" => "jms.queue.TestDLQ",
    "expiry-address" => undefined,
    "expiry-delay" => -1L,
    "last-value-queue" => false,
    "max-delivery-attempts" => 1,
    "max-redelivery-delay" => 0L,
    "max-size-bytes" => -1L,
    "message-counter-history-day-limit" => 0,
    "page-max-cache-size" => 5,
    "page-size-bytes" => 10485760L,
    "redelivery-delay" => 0L,
    "redelivery-multiplier" => 1.0,
    "redistribution-delay" => -1L,
    "send-to-dla-on-no-route" => false,
    "slow-consumer-check-period" => 5L,
    "slow-consumer-policy" => "NOTIFY",
    "slow-consumer-threshold" => -1L
  }
}
```

Note the values for **dead-letter-address**, **max-delivery-attempts**, **max-redelivery-delay**, **redelivery-delay** and **redelivery-multiplier**. They are all related to dead letter handling but this book does NOT go into detail about them.

Note also the values of the **expiry-delay** and **expiry-address** attributes. This shows how defaults for a new **address-setting** may be unexpected. In this case, there is no expiry handling, in spite of having dead letter handling.

Leave this terminal running the CLI. We will come back to it later in this exercise.

► 4. Send a message known to be processed without issues.

- 4.1. Open a web browser and enter the producer test application using the following URL:
<http://localhost:8080/messaging-client>

- 4.2. Complete the web form as follows:

- **Message Count: 1**
- **Message Label: first**
- **Message to send: test message - to be consumed**

Click **Send Message** to produce (or send) the message.

- 4.3. The web form should refresh and display the following message in green: **Messages sent successfully.**

► 5. Send a message known to trigger processing errors.

- 5.1. Use the same browser tab as the previous step and the web form as follows:

- **Message Count: 1**
- **Message Label: second**
- **Message to send: test message - will FAIL**

It is important that you type "FAIL" in uppercase, or processing will NOT generate errors.

Click **Send Message** to produce (or send) the message.

- 5.2. The web form should refresh and display the following message in green: **Messages sent successfully.** This message is supposed to fail but not during publishing, only when consuming.

Leave the producer test application browser tab open because we will come back to it a few times during this exercise.

► 6. Disable the consumer test application so that new messages are kept in the queue.

- 6.1. Undeploy the **messaging-mdb.jar** deployment. Go back to the terminal running the CLI and run the following command:

```
[standalone@localhost:9990 server=default] /deployment=messaging-mdb.jar:\nundeploy
```

- 6.2. Verify the consumer test application is stopped:

```
[standalone@localhost:9990 server=default] /deployment=messaging-mdb.jar:\nread-attribute(name=status)
```

The expected output is:

```
{\n    "outcome" => "success",\n    "result" => "STOPPED"\n}
```

► 7. Send another message known to be processed without issues.

This message will NOT be processed, just queued, because the consumer application was disabled during the previous step.

- 7.1. Go back to the browser tab running the producer test application and complete the web form as follows:

- **Message Count: 1**
- **Message Label: third**
- **Message to send: test message - to be queued**

Click **Send Message** to produce (or send) the message.

- 7.2. The web form should refresh and display the following message in green: **Messages sent successfully.**

Note the consumer has no way to know how long the message will be waiting to be processed.

- 8. Verify the fist message was consumed, the second message was moved to the queue created at **Step 2.2**, and the third messages is still in **TestQueue**.

- 8.1. Go back to the terminal running the EAP server and verify there are log entries related to the processing of the first message. There should be entries similar to:

```
09:14:19,706 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
(Thread-0 (ActiveMQ-client-global-threads-991436930)) Message Properties: Copy #1
[first]
09:14:19,717 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
(Thread-0 (ActiveMQ-client-global-threads-991436930)) Message Body: test message
- to be consumed
```

Remember those log entries are generated by the MDB when consuming a message.

Note the log entry about "Message Properties" property has the value "**first**" as provided in **Step 4.2**.

- 8.2. Verify also in the EAP server logs the entries related to the second message processing failure.

There should be a couple multiline log entries, each one embedding a Java stack trace, similar to:

```
09:14:19,706 ERROR [org.jboss.as.ejb3.invocation] (Thread-1 (ActiveMQ-
client-global-threads-991436930)) WFLYEJB0034: EJB Invocation
failed on component JB248TestQueueMDB for method public void
com.redhat.training.messaging.mdb.MessageReceiver.onMessage(javax.jms.Message):
javax.ejb.EJBTransactionRolledbackException: Message processing failed for no
reason
...
09:14:19,715 ERROR [org.apache.activemq.artemis.ra] (Thread-1 (ActiveMQ-
client-global-threads-991436930)) AMQ154004: Failed to deliver message:
javax.ejb.EJBTransactionRolledbackException: Message processing failed for no
reason
...
```

A single message processing failure generated both entries. The first one is from the MDB and the second one from the ActiveMQ server. Remember from **Step 3.3** the **address-setting** is configured to perform a single redelivery attempt (**max-delivery-attempts=1**), so there should be a single pair of errors matching the previous listing.

There should also be a warning message stating the message was moved from the application queue to the configured DLQ:

```
09:14:19,745 WARN [org.apache.activemq.artemis.core.server] (Thread-20 (ActiveMQ-server-org.apache.activemq.artemis.core.server.impl.ActiveMQServerImpl$2@3da34eba-1936375109)) AMQ222149: Message Reference[181]:RELIABLE:ServerMessage[messageID=181,durable=true,userID=8ac5181d-15e7-11e6-b7ce-f31b4b7807d0,priority=4, bodySize=512, timestamp=Mon May 09 09:11:29 EDT 2016,expiration=0, durable=true, address=jms.queue.TestQueue,properties=TypedProperties[Label=second, _AMQ_CID=a011e476-15e5-11e6-b7ce-f31b4b7807d0,Copy=1]]@322081305 has reached maximum delivery attempts, sending it to Dead Letter Address jms.queue.TestDLQ from jms.queue.TestQueue
```

8.3. Go back to the terminal running the CLI and list all messages in **TestDLQ**:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:list-messages
```

The expected output is similar to:

```
{
  "outcome" => "success",
  "result" => [
    {
      "JMSMessageID" => "ID:8ac5181d-15e7-11e6-b7ce-f31b4b7807d0",
      "address" => "jms.queue.TestDLQ",
      "JMSExpiration" => 0,
      "JMSTimestamp" => 1462799489118L,
      "Label" => "second",
      "messageID" => 200,
      "JMSDeliveryMode" => "PERSISTENT",
      "_AMQ_ORIG_MESSAGE_ID" => 181,
      "JMSPriority" => 4,
      "_AMQ_CID" => "a011e476-15e5-11e6-b7ce-f31b4b7807d0",
      "_AMQ_ORIG_QUEUE" => "jms.queue.TestQueue",
      "Copy" => 1,
      "_AMQ_ORIG_ADDRESS" => "jms.queue.TestQueue"
    }
}
```

Note the message property "**Label**" with the value "**second**" as provided in **Step 5.1**.

8.4. Go back to the terminal running the CLI and list all messages in **TestQueue**:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected output is similar to:

```
{  
    "outcome" => "success",  
    "result" => [{  
        "JMSPriority" => 4,  
        "JMSMessageID" => "ID:81eda5bd-15e8-11e6-b7ce-f31b4b7807d0",  
        "address" => "jms.queue.TestQueue",  
        "JMSExpiration" => 0,  
        "__AMQ_CID" => "a011e476-15e5-11e6-b7ce-f31b4b7807d0",  
        "Copy" => 1,  
        "JMSTimestamp" => 1462799903781L,  
        "Label" => "third",  
        "messageID" => 232,  
        "JMSDeliveryMode" => "PERSISTENT"  
    }]  
}
```

Note the message property **"Label"** with the value **"third"** as provided in **Step 7.1**.

► **9.** Inspect the current journal files, created according to the default EAP configuration.

- 9.1. In the terminal running the CLI, inspect the **default** ActiveMQ server attributes related to the journal files:

```
[standalone@localhost:9990 server=default] :read-attribute(name=\  
journal-min-files)  
[standalone@localhost:9990 server=default] :read-attribute(name=\  
journal-file-size)
```

The expected values are **2** and **10485760**, respectively.

- 9.2. Open another terminal and list all files under **data/messaging** at the server instance base folder, that is, **/home/student/standalone/data/messaging**:

```
[student@workstation ~]$ ls -lh ~/JB248/labs/standalone/data/activemq/journal/
```

The expected output is similar to:

```
total 21M  
-rw-rw-r--. 1 student student 10M May  5 20:34 activemq-data-1.amq  
-rw-rw-r--. 1 student student 10M May  6 17:06 activemq-data-2.amq  
-rw-rw-r--. 1 student student  19 May  5 12:44 server.lock
```

Note that the number of data files and the size of each file matches the attribute values inspected in the previous step.

► **10.** Reload the server instance and verify the second message is still in **TestQueue**.

- 10.1. In the terminal running the CLI, reload the standalone server:

```
[standalone@localhost:9990 server=default] /:reload
```

- 10.2. Check the log entries in the terminal running the EAP server and wait for the messages that show the server was restarted successfully.

- 10.3. Go back to the terminal running the CLI and list all messages in **TestQueue**:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected output is the same as **Step 8.4**.

- 11. Change the journal file size and number of files.

- 11.1. In the terminal running the CLI, increase the **journal-min-files** attribute value to 5 and the **journal-file-size** attribute value to 20 MB:

```
[standalone@localhost:9990 server=default] :write-attribute(\n  name=journal-min-files, value=5)\n[standalone@localhost:9990 server=default] :write-attribute(\n  name=journal-file-size, value=20971520)
```

- 11.2. Changing those attributes requires reloading or restarting the server instance:

```
[standalone@localhost:9990 server=default] /:reload
```

- 11.3. Check the log entries in the terminal running the EAP server and wait for the messages that show the server was restarted successfully.

- 12. Verify the second message is still in **TestQueue** and the journal files were changed.

- 12.1. Go back to the terminal running the CLI and list all messages in **TestQueue**:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected output is the same as **Step 8.4**.

- 12.2. Go back to the idle terminal and check there are now five journal files, 20 MB each:

```
[student@workstation ~]$ ls -lh ~/JB248/labs/standalone/data/activemq/journal/
```

The expected output is similar to:

```
total 91M\n-rw-rw-r-- 1 student student 20M May  9 09:31 activemq-data-10.amq\n-rw-rw-r-- 1 student student 20M May  9 09:31 activemq-data-11.amq\n-rw-rw-r-- 1 student student 20M May  9 09:31 activemq-data-12.amq\n-rw-rw-r-- 1 student student 20M May  9 09:31 activemq-data-13.amq\n-rw-rw-r-- 1 student student 10M May  9 09:18 activemq-data-1.amq\n-rw-rw-r-- 1 student student  19 May  5 12:44 server.lock
```

The numbers in each log file name are not important, only the number of files and their sizes.

- 13. Disable persistence for the messaging subsystem as a whole.

- 13.1. Go back to the terminal running the CLI and change the **persistence-enabled** attribute to **false**:

```
[standalone@localhost:9990 server=default] :write-attribute(\n  name=persistence-enabled,value=false)
```

- 13.2. Changing this attributes requires reloading or restarting the server instance:

```
[standalone@localhost:9990 server=default] /:reload
```

- 13.3. Check the log entries in the terminal running the EAP server and wait for the messages that show the server was restarted successfully.

► **14.** Verify the third message was lost.

- 14.1. Go back to the terminal running the CLI and list all messages in **TestQueue**:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected output is similar to:

```
{\n    "outcome" => "success",\n    "result" => []\n}
```

This shows the queue is empty.



Note

Curious students may check the journal files folder, repeating the commands from **Step 12.2**, and be surprised the journal files are still there. This is expected, but the files are NOT being used by EAP any more.

► **15. Clean up:** remove the test applications and JMS resources created by this and the previous exercise, and enable messaging persistence.

- 15.1. Go back to the terminal running the CLI and remove the producer and consumer applications:

```
[standalone@localhost:9990 server=default] /deployment=messaging-mdb.jar\\n:remove\n[standalone@localhost:9990 server=default] /deployment=messaging-client.war\\n:remove
```

- 15.2. Remove the custom **address-setting**:

```
[standalone@localhost:9990 server=default] ./address-setting=\\n  jms.queue.TestQueue:remove
```

- 15.3. Remove the JMS custom **ConnectionFactory** and **Queue** resources:

```
[standalone@localhost:9990 server=default] ./pooled-connection-factory=\
custom:remove
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:remove
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:remove
```

15.4. Re-enable persistence for the ActiveMQ server:

```
[standalone@localhost:9990 server=default] :write-attribute(\
name=persistence-enabled,value=true)
```

15.5. Stop the standalone server instance:

```
[standalone@localhost:9990 server=default] /:shutdown
```

This concludes the guided exercise.

▶ Lab

Configuring the Messaging Subsystem

In this lab, you will configure JMS resources for a future version of the **bookstore** application. While the updated application is not available, test applications are provided to test the JMS configuration is working as required.

Resources	
Files:	/opt/domain
	/home/student/JB248/labs/messaging-lab
	/tmp/messaging-client.war
	/tmp/messaging-mdb.jar
Application URL:	http://172.25.250.10:8080/messaging-client

Outcome

You should be able to create a JMS **ConnectionFactory** and a **Queue** in a managed domain, and deploy test applications that make use of them.

Before You Begin

You can choose either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively and the domain controller on the **workstation** VM. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines.

This lab makes use of two applications:

- **messaging-client.war** is the message producer: a web application that accepts text from the user and publishes as JMS messages properties and body.
- **messaging-mdb.jar** is the message consumer: an application with a **Message-Driven Bean** (MDB) that consumes JMS messages and shows the message properties and body in the EAP server log.

Both test applications make use of the following JMS resources:

- **custom** JMS **ConnectionFactory**, with JNDI address **java:/jms/CustomCF**. It should point to the ActiveMQ **in-vm** connector.
- **TestQueue** JMS **Queue** with JNDI address **java:/jms/queue/JB248TestQueue**. There is no need to configure any message settings for this queue.

The applications should be deployed to the **Group1** server group, which uses the clustered profile **full-ha**. Fortunately a fellow sysadmin provided you with the **artemis-**

firewalld-rules.sh shell script that configures Linux kernel buffers and firewall rules to match default EAP7 settings for this profile. This script is available at **/home/student/JB248/labs/messaging-lab** on the workstation VM.

It is important to make sure that all application server instances in the **Group1** server group are consuming messages. This means message content processing is distributed among the EAP hosts and the application is scalable.

Before beginning the chapter review lab, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, no instance of EAP is running, the managed domain is configured, and to download the JEE application package files to the **/tmp** folder:

```
[student@workstation ~]$ lab messaging-lab setup
```

1. Configure the host OSes for EAP multicast clustering.
 - 1.1. Download the **artemis-firewalld-rules.sh** shell script from the workstation VM and run it on the **servera** VM.
 - 1.2. Inspect the **artemis-firewalld-rules.sh** shell script to check the TCP and UDP ports it opens. Verify the new firewall rules were applied using the **firewall-cmd** command.
 - 1.3. Verify the new kernel settings were applied using the **sysctl** command .
 - 1.4. Repeat **Step 1.1**, **Step 1.2**, and **Step 1.3** on the **serverb** VM.
2. Start the EAP managed domain.
 - 2.1. Start the domain controller on the **workstation** VM using **/opt/domain** as the base folder and **host-master.xml** as the host configuration file. All EAP processes should be owned by the **jboss** user.
 - 2.2. Start the host controller on the **servera** VM using **/opt/domain** as the base folder and **host-slave.xml** as the host configuration file. All EAP processes should be owned by the **jboss** user.

You also need to provide the machine IP address 172.25.250.10 as the argument to the **-bprivate** command-line option. Without that, EAP server instances will not find each other to form a cluster.
 - 2.3. Start the host controller on **serverb** using the same parameters given in the previous step for **servera**.

Remember to replace the IP address given to the **-bprivate** command-line option with the **serverb** IP address 172.25.250.11.
 - 2.4. Verify that both host controllers connect to the domain controller and form a managed domain.
3. Verify that all **four** server instances connected to form an ActiveMQ cluster.

The managed domain contains another server group with two more server instances. This group is also configured to use the **full-ha** profile, so it gets the same clustering parameters as **Group1** and its ActiveMQ embedded server instances will also join the same messaging cluster.

- 3.1. Enter any of the running server instances **messaging-activemq** subsystem.

Configure the logging subsystem to increase the logging level by running the following command:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging\
/root-logger=ROOT:write-attribute(name=level,value=INFO)
```

- 3.2. Verify the **topology** attribute of the **cluster-connection** object shows all four server instances in the domain.

4. Create the JMS resources on the **full-ha** profile.

- 4.1. Create the **custom** JMS **ConnectionFactory** using **java:/jms/CustomCF** as the JNDI name.

- 4.2. Create the **TestQueue** JMS **Queue** using **java:/jms/queue/JB248TestQueue** as the JNDI name.

5. Deploy the consumer and producer applications.

- 5.1. Deploy the message producer application to **Group1**.

- 5.2. Deploy the message consumer application to **Group1**.

- 5.3. Verify both deployments were successful.

6. Publish a few copies of a test message and verify BOTH server instances from the **Group1** server group consumed messages.

Also verify the embedded messaging servers in both **servera.1** and **serverb.1** server instances received messages to their local queues.

- 6.1. Publish a test message. Enter the message producer application running on **servera.1** at <http://172.25.250.10:8080/messaging-client> and complete the form as follows:

- **Message Count: 5**
- **Message Label: testmsg**
- **Message to send: jms test**

- 6.2. Check the server logs for the entries generated by the consumer application. Servers **servera.1** and **serverb.1** should have log entries generated by the consumer application.

Server instances from **Group2** should NOT consume messages because they have no consumer application deployed.

- 6.3. Verify that messages were queued to different EAP server instances; that is, the message load was shared by different ActiveMQ server instances. Only **servera.1** and **serverb.1** should have any activity in their local queues.

The messaging servers embedded in **Group2** servers should NOT queue any messages because ActiveMQ default settings do not deliver messages to cluster members that have no active consumers. This is done to avoid unnecessary network traffic.

7. Clean up and grading.

- 7.1. Stop the slave hosts.
- 7.2. Stop the domain controller.
- 7.3. Run the following command from the workstation to grade this chapter review lab:

```
[student@workstation bin]$ lab messaging-lab grade
```

This concludes the lab.

► Solution

Configuring the Messaging Subsystem

In this lab, you will configure JMS resources for a future version of the **bookstore** application. While the updated application is not available, test applications are provided to test the JMS configuration is working as required.

Resources	
Files:	/opt/domain
	/home/student/JB248/labs/messaging-lab
	/tmp/messaging-client.war
	/tmp/messaging-mdb.jar
Application URL:	http://172.25.250.10:8080/messaging-client

Outcome

You should be able to create a JMS **ConnectionFactory** and a **Queue** in a managed domain, and deploy test applications that make use of them.

Before You Begin

You can choose either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively and the domain controller on the **workstation** VM. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines.

This lab makes use of two applications:

- **messaging-client.war** is the message producer: a web application that accepts text from the user and publishes as JMS messages properties and body.
- **messaging-mdb.jar** is the message consumer: an application with a **Message-Driven Bean** (MDB) that consumes JMS messages and shows the message properties and body in the EAP server log.

Both test applications make use of the following JMS resources:

- **custom** JMS **ConnectionFactory**, with JNDI address **java:/jms/CustomCF**. It should point to the ActiveMQ **in-vm** connector.
- **TestQueue** JMS **Queue** with JNDI address **java:/jms/queue/JB248TestQueue**. There is no need to configure any message settings for this queue.

The applications should be deployed to the **Group1** server group, which uses the clustered profile **full-ha**. Fortunately a fellow sysadmin provided you with the **artemis-firewalld-rules.sh** shell script that configures Linux kernel buffers and firewall rules

to match default EAP7 settings for this profile. This script is available at **/home/student/JB248/labs/messaging-lab** on the workstation VM.

It is important to make sure that all application server instances in the **Group1** server group are consuming messages. This means message content processing is distributed among the EAP hosts and the application is scalable.

Before beginning the chapter review lab, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, no instance of EAP is running, the managed domain is configured, and to download the JEE application package files to the **/tmp** folder:

```
[student@workstation ~]$ lab messaging-lab setup
```

- Configure the host OSes for EAP multicast clustering.

- Download the **artemis-firewalld-rules.sh** shell script from the workstation VM and run it on the **servera** VM.

Open a terminal window on the **workstation** VM to copy the script to the **servera** machine:

```
[student@workstation ~]$ scp \n\n~/JB248/labs/messaging-lab/artemis-firewalld-rules.sh servera:~
```

Open another terminal window on the **servera** machine to run the script using **sudo**:

```
[student@servera ~]$ sudo sh artemis-firewalld-rules.sh
```

- Inspect the **artemis-firewalld-rules.sh** shell script to check the TCP and UDP ports it opens. Verify the new firewall rules were applied using the **firewall-cmd** command.

Check TCP ports 54200 and 54300, and UDP ports 45688, 55200, and 55300 are open:

```
[student@servera ~]$ firewall-cmd --list-all --zone=public
```

The expected output is similar to:

```
public (default, active)
  interfaces: eth0
  sources:
  services: dhcpcv6-client ssh
  ports: 54300/tcp 8180/tcp 55300/udp 8080/tcp 55200/udp 54200/tcp 45688/udp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
```

- Verify the new kernel settings were applied using the **sysctl** command.

Check **net.core.rmem_max** and **net.core.wmem_max** are both set to 1 MiB (1073741824 bytes):

```
[student@servera ~]$ sysctl net.core.rmem_max net.core.wmem_max
```

The expected output is:

```
net.core.rmem_max = 1073741824  
net.core.wmem_max = 1073741824
```

- 1.4. Repeat **Step 1.1**, **Step 1.2**, and **Step 1.3** on the **serverb** VM.
2. Start the EAP managed domain.
 - 2.1. Start the domain controller on the **workstation** VM using **/opt/domain** as the base folder and **host-master.xml** as the host configuration file. All EAP processes should be owned by the **jboss** user.
Open a terminal window on the **workstation** VM and run the following command:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \  
-Djboss.domain.base.dir=/opt/domain --host-config=host-master.xml
```
 - 2.2. Start the host controller on the **servera** VM using **/opt/domain** as the base folder and **host-slave.xml** as the host configuration file. All EAP processes should be owned by the **jboss** user.
You also need to provide the machine IP address 172.25.250.10 as the argument to the **-bprivate** command-line option. Without that, EAP server instances will not find each other to form a cluster.
Open a terminal window on the **servera** VM and run the following command:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \  
-Djboss.domain.base.dir=/opt/domain --host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254 \  
-bprivate=172.25.250.10
```
 - 2.3. Start the host controller on **serverb** using the same parameters given in the previous step for **servera**.
Remember to replace the IP address given to the **-bprivate** command-line option with the **serverb** IP address 172.25.250.11.
Open a terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \  
-Djboss.domain.base.dir=/opt/domain --host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254 \  
-bprivate=172.25.250.11
```
 - 2.4. Verify that both host controllers connect to the domain controller and form a managed domain.
Check the terminal window where you started the domain controller to verify that both **servera** and **serverb** are registered as slaves to the domain controller.
The domain controller log should include entries similar to the following:

```
...
[Host Controller] 17:15:49,131 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 34) WFLYHC0019: Registered remote slave host "servera", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
...
[Host Controller] 17:16:04,294 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads - 35) WFLYHC0019: Registered remote slave host "serverb", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
...
```

3. Verify that all **four** server instances connected to form an ActiveMQ cluster.

The managed domain contains another server group with two more server instances. This group is also configured to use the **full-ha** profile, so it gets the same clustering parameters as **Group1** and its ActiveMQ embedded server instances will also join the same messaging cluster.

- 3.1. Enter any of the running server instances **messaging-activemq** subsystem.

Open a terminal window on the **workstation** VM and connect the CLI:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254
```

Configure the logging subsystem to increase the logging level by running the following command:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging\
/root-logger=ROOT:write-attribute(name=level,value=INFO)
```

Enter any of the running server instances **messaging-activemq** subsystem. In the following command the **servera.1** server from host **servera** was chosen:

```
[domain@172.25.250.254:9990 /] cd /host=servera/server=servera.1
[domain@172.25.250.254:9990 server=servera.1] cd subsystem=messaging-activemq
```

- 3.2. Verify the **topology** attribute of the **cluster-connection** object shows all four server instances in the domain.

Use the following command. Remember you can not **cd** to a run-time resource, in this case **server=default**:

```
[domain@172.25.250.254:9990 subsystem=messaging-activemq] ./server=default\
/cluster-connection=my-cluster:read-attribute(name=topology)
```

The expected output is similar to:

```

...
    "result" => "topology on
Topology@2f00ce83[owner=ClusterConnectionImpl@1089898802
[nodeUUID=66010359-1888-11e6-bf74-bba07d6c198b,
 connector=TransportConfiguration(name=http-connector,
 factory=org.apache.activemq.artemis.core.remoting.impl.netty-
NettyConnectorFactory) ?httpUpgradeEnabled=true&httpPgradeEndpoint=http-
acceptor&port=8080&host=172-25-250-10, address=jms,
 server=ActiveMQServerImpl::serverUUID=66010359-1888-11e6-bf74-bba07d6c198b]":
771766a0-1888-11e6-a6cd-735504754596 => TopologyMember[
id = 771766a0-1888-11e6-a6cd-735504754596,
connector=Pair[a=TransportConfiguration(name=http-connector,
factory=org.apache.activemq.artemis.core.remoting.impl.netty-
NettyConnectorFactory) ?httpUpgradeEnabled=true&httpPgradeEndpoint=http-
acceptor&port=8180&host=172-25-250-11, b=null], backupGroupName=null,
scaleDownGroupName=null]
74943ed3-1888-11e6-80c1-ef9418706af0 => TopologyMember[
id = 74943ed3-1888-11e6-80c1-ef9418706af0,
connector=Pair[a=TransportConfiguration(name=http-connector,
factory=org.apache.activemq.artemis.core.remoting.impl.netty-
NettyConnectorFactory) ?httpUpgradeEnabled=true&httpPgradeEndpoint=http-
acceptor&port=8080&host=172-25-250-11, b=null], backupGroupName=null,
scaleDownGroupName=null]
684f38d4-1888-11e6-a0bc-6f1ea2c1a877 => TopologyMember[
id = 684f38d4-1888-11e6-a0bc-6f1ea2c1a877,
connector=Pair[a=TransportConfiguration(name=http-connector,
factory=org.apache.activemq.artemis.core.remoting.impl.netty-
NettyConnectorFactory) ?httpUpgradeEnabled=true&httpPgradeEndpoint=http-
acceptor&port=8180&host=172-25-250-10, b=null], backupGroupName=null,
scaleDownGroupName=null]
66010359-1888-11e6-bf74-bba07d6c198b => TopologyMember[
id = 66010359-1888-11e6-bf74-bba07d6c198b,
connector=Pair[a=TransportConfiguration(name=http-connector,
factory=org.apache.activemq.artemis.core.remoting.impl.netty-
NettyConnectorFactory) ?httpUpgradeEnabled=true&httpPgradeEndpoint=http-
acceptor&port=8080&host=172-25-250-10, b=null], backupGroupName=null,
scaleDownGroupName=null]
nodes=4 members=4",
...

```

Note that each **TopologyMember** resource has a different **id** and points to a different **http-acceptor port** and **host** pair.

4. Create the JMS resources on the **full-ha** profile.

- 4.1. Create the **custom** JMS **ConnectionFactory** using **java:/jms/CustomCF** as the JNDI name.

Enter the **messaging-activemq** subsystem in the **full-ha** profile and create a **pooled-connection-factory**:

```
[domain@172.25.250.254:9990 /] cd /profile=full-ha  
[domain@172.25.250.254:9990 profile=full-ha] cd \  
subsystem=messaging-activemq/server=default  
[domain@172.25.250.254:9990 server=default] ./pooled-connection-factory=\  
custom:add(connectors=[in-vm], entries=[java:/jms/CustomCF])
```

- 4.2. Create the **TestQueue** JMS Queue using **java:/jms/queue/JB248TestQueue** as the JNDI name.

Use the CLI to create a **jms-queue**:

```
[domain@172.25.250.254:9990 server=default] ./jms-queue=TestQueue:add(\  
entries=[java:/jms/queue/JB248TestQueue])
```

5. Deploy the consumer and producer applications.

- 5.1. Deploy the message producer application to **Group1**.

Use the CLI to deploy the **/tmp/messaging-client.war** JEE package:

```
[domain@172.25.250.254:9990 server=default] deploy \  
/tmp/messaging-client.war --server-groups=Group1
```

- 5.2. Deploy the message consumer application to **Group1**.

Use the CLI to deploy the **/tmp/messaging-mdb.jar** JEE package:

```
[domain@172.25.250.254:9990 server=default] deploy \  
/tmp/messaging-mdb.jar --server-groups=Group1
```

- 5.3. Verify both deployments were successful.

Check each slave host server logs. They should have entries similar to the following:

```
...  
[Server:servera.1] 18:41:55,697 INFO [org.jboss.as.server] (ServerService Thread  
Pool -- 108) WFLYSRV0010: Deployed "messaging-client.war" (runtime-name :  
"messaging-client.war")  
...  
[Server:servera.1] 18:44:50,425 INFO [org.jboss.as.server] (ServerService  
Thread Pool -- 113) WFLYSRV0010: Deployed "messaging-mdb.jar" (runtime-name :  
"messaging-mdb.jar")  
...
```

6. Publish a few copies of a test message and verify BOTH server instances from the **Group1** server group consumed messages.

Also verify the embedded messaging servers in both **servera.1** and **serverb.1** server instances received messages to their local queues.

- 6.1. Publish a test message. Enter the message producer application running on **servera.1** at <http://172.25.250.10:8080/messaging-client> and complete the form as follows:

- **Message Count: 5**

- **Message Label:** testmsg
- **Message to send:** jms test

Open a web browser and navigate to <http://172.25.250.10:8080/messaging-client>.

Complete the web form and click **Send Message** to send the message.

The web form should refresh and display the following message in green: **Messages sent successfully.**

- 6.2. Check the server logs for the entries generated by the consumer application. Servers **servera.1** and **serverb.1** should have log entries generated by the consumer application.

Server instances from **Group2** should NOT consume messages because they have no consumer application deployed.

There should be log entries similar to the following on the **servera** terminal window:

```
[Server:servera.1] 19:54:56,317 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-11 (ActiveMQ-client-global-threads-904575248)) Message Properties: Copy #3 [testmsg]
[Server:servera.1] 19:54:56,318 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-12 (ActiveMQ-client-global-threads-904575248)) Message Properties: Copy #5 [testmsg]
[Server:servera.1] 19:54:56,318 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-11 (ActiveMQ-client-global-threads-904575248)) Message Body: jms test
[Server:servera.1] 19:54:56,319 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-12 (ActiveMQ-client-global-threads-904575248)) Message Body: jms test
[Server:servera.1] 19:54:56,322 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-10 (ActiveMQ-client-global-threads-904575248)) Message Properties: Copy #1 [testmsg]
[Server:servera.1] 19:54:56,323 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-10 (ActiveMQ-client-global-threads-904575248)) Message Body: jms test
```

There should be log entries similar to the following on the **serverb** terminal window:

```
[Server:serverb.1] 19:54:58,282 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-9 (ActiveMQ-client-global-threads-1005879511)) Message Properties: Copy #4 [testmsg]
[Server:serverb.1] 19:54:58,285 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-8 (ActiveMQ-client-global-threads-1005879511)) Message Properties: Copy #2 [testmsg]
[Server:serverb.1] 19:54:58,285 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-9 (ActiveMQ-client-global-threads-1005879511)) Message Body: jms test
[Server:serverb.1] 19:54:58,285 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-8 (ActiveMQ-client-global-threads-1005879511)) Message Body: jms test
```

Remember it is normal having messages consumed out-of-order because EAP runs many MDB instances in parallel. It is also normal having log entries from different messaging threads mixed in the server log.

The message distribution between servers you see may be different, but you should see both server instances from **Group1** (**servera.1** and **serverb.1**) getting messages.

- 6.3. Verify that messages were queued to different EAP server instances; that is, the message load was shared by different ActiveMQ server instances. Only **servera.1** and **serverb.1** should have any activity in their local queues.

The messaging servers embedded in **Group2** servers should NOT queue any messages because ActiveMQ default settings do not deliver messages to cluster members that have no active consumers. This is done to avoid unnecessary network traffic.

Go back to the terminal window on the **workstation** VM that is connected to the CLI and check the message counters on **servera.1**:

```
[domain@172.25.250.254:9990 server=default] cd /host=servera\  
/server=servera.1  
[domain@172.25.250.254:9990 server=servera.1] cd \  
.subsystem=messaging-activemq  
[domain@172.25.250.254:9990 subsystem=messaging-activemq] ./server=default\  
/jms-queue=TestQueue:read-attribute(name=messages-added)
```

The expected output is similar to:

```
{  
    "outcome" => "success",  
    "result" => 3L  
}
```

Check the message counters on **serverb.1**:

```
[domain@172.25.250.254:9990 server=default] cd /host=serverb\  
/server=serverb.1  
[domain@172.25.250.254:9990 server=serverb.1] cd \  
.subsystem=messaging-activemq  
[domain@172.25.250.254:9990 subsystem=messaging-activemq] ./server=default\  
/jms-queue=TestQueue:read-attribute(name=messages-added)
```

The expected output is similar to:

```
{  
    "outcome" => "success",  
    "result" => 2L  
}
```

The attribute values you see should match the log entries from the MDB. That is, the MDB will consume messages from its local ActiveMQ server.

7. Clean up and grading.

- 7.1. Stop the slave hosts.

Run the following CLI command to each slave hosts in the managed domain:

```
[domain@172.25.250.254:9990 messaging-activemq] /host=servera:shutdown  
[domain@172.25.250.254:9990 messaging-activemq] /host=serverb:shutdown
```

- 7.2. Stop the domain controller.

Run the following CLI command to stop the master host:

```
[domain@172.25.250.254:9990 /] /host=master:shutdown
```

- 7.3. Run the following command from the workstation to grade this chapter review lab:

```
[student@workstation bin]$ lab messaging-lab grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- EAP 7 embeds the latest **ActiveMQ** Message-Oriented Middleware (MOM) based on the **Artemis** project. It is a high-performance, highly-available MOM suited for very demanding environments.
- The application server administrator configures JMS **ConnectionFactory** and **Destination** resources using JNDI names expected by applications.
 - Usually a **PooledConnectionFactory** is preferred over a plain **ConnectionFactory**.
 - A JMS **Destination** can be either a **Queue** (single consumer per message) or a **Topic** (publish/subscriber with multiple consumers per message).
- Many MOM features, for example message redelivery and flow control, are configured inside an **address-settings** that can match multiple destinations.
- The main ActiveMQ persistence component is the message journal files. They are tuned for production to avoid the need for file system metadata operations.

Chapter 9

Securing JBoss EAP

Overview

Goal Apply security configurations to enhance the security of applications deployed on JBoss EAP.

- Objectives**
- Configure a security domain based on the database login module.
 - Configure a security domain based on the LDAP login module.
 - Configure role-based access to topics and queues in the messaging subsystem.
 - Protect passwords that are stored in server configuration files by using the password vault.

- Sections**
- Configuring a Database Security Domain (and Guided Exercise)
 - Configuring an LDAP Security Domain (and Guided Exercise)
 - Securing a JMS Destination (and Quiz)
 - Configuring the Password Vault (and Guided Exercise)

- Lab**
- Securing JBoss EAP

Configuring a Database Security Domain

Objectives

After completing this section, students should be able to:

- Configure a security domain based on the database login module.

The security subsystem

Consistent with the modular design of EAP, security is handled by a subsystem. The security configuration is represented by the *security subsystem*, and is configured in the **domain.xml** (in managed domain mode) and **standalone.xml** (in a standalone server) configuration files.

The EAP security subsystem consists of a number of **security-domains**, which define how applications are authenticated and authorized. Users can define their own security domain backed by a system that stores identity information (database, LDAP, and so on) and configure applications to use the newly defined security-domain in the application's deployment descriptors.

The core principle behind the design of the security subsystem is to avoid embedding the logic for authentication and authorization inside the applications and capture the implementation details inside reusable components (security-domains) that can then be referenced by the applications at runtime. For example, administrators can change the authentication and authorization mechanism for an application from a database to LDAP without making any source code changes in the application.



References

The security subsystem in EAP 6 was developed under the **Picketlink** project. In EAP 7, **Picketlink** has been replaced by the **Elytron** project. For more details, see

The Elytron project

<https://developer.jboss.org/wiki/WildFlyElytron-ProjectSummary>

Security domains

A security domain is a set of Java Authentication and Authorization Service (JAAS) security configurations that one or more applications use to control authentication, authorization, auditing, and mapping.

A security domain consists of configurations for authentication, authorization, security mapping, and auditing. Users can create as many security domains as needed to accommodate application requirements.

By default, EAP has four security domains defined:

- jboss-ejb-policy**: The default authorization security domain for EJB modules in applications that have not defined a security-domain explicitly.
- jboss-web-policy**: The default authorization security domain for web modules in applications that have not defined a security-domain explicitly.

- **other**: This security domain is used internally within JBoss EAP for authorization and is required for correct operation.
- **jaspitest**: The jaspitest security domain is a simple Java Authentication Service Provider Interface for Containers (JASPI) security domain included for development purposes.

 **References**

For more details on **JASPI**, see
JASPI
<https://jaspic.zeef.com/>

Security domains can be defined and configured by using the EAP CLI or in the Management Console in the **Configuration → Subsystems → Security** section.

Home	Deployments	Configuration	Runtime	Access Control	Patching
Configuration		Subsystem (28)	Security Domain		Add
	Subsystems	>	<input type="text"/>	jaspitest	
	Interfaces	IO		jboss-ejb-policy	
	Socket Binding	Logging		jboss-web-policy	
	Paths	Deployment Scanners		other	
	System Properties	JMX			
		Remoting			
		Infinispan	>		
		Security	>		
		Web Services			

Figure 9.1: Configuring security domains in the management console

The format of a **security-domain** in the **standalone.xml** or **domain.xml** definition looks like the following:

```
<security-domains>
  <security-domain>
    <authentication>
      <login-module>
        ...
      </login-module>
    </authentication>
  </security-domain>
</security-domains>
```

EAP includes several built-in login modules that can be used for authentication within a security domain. The security subsystem offers some core login modules that can read user information from a relational database, an LDAP server, or flat files. In addition to these core login modules, EAP provides other login modules that can be customized depending on the security requirements of the application.



References

For a detailed listing of all login modules shipped with EAP refer to

EAP 7 Login Modules Reference

<https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0/login-module-reference/login-module-reference>

Defining a database security domain

Users can define a security domain backed by a database, which stores the users and role mapping for an application using the **Database** login module. It uses a named **datasource** reference defined in the EAP configuration file and defines queries for fetching the users and roles from the database. The recommended approach is to store passwords in an encrypted format by configuring the **hashAlgorithm** attribute for improved security in the database.

A database security domain can be created with the EAP CLI using the following command:

```
[standalone@localhost:9990] /subsystem=security/security-domain= \
  bksecurity-domain:add(cache-type=default)
```

The following command associates the security domain with a database for authentication purposes:

```
[standalone@localhost:9990] /subsystem=security/security-domain= \
  db-domain/authentication=classic:add \
  (login-modules=[{"code"=>"Database", "flag"=>"required", \
  "module-options"=>[("dsJndiName"=>"java:jboss/datasources/test-ds"), \
  ("principalsQuery"=>"SELECT password FROM users WHERE username = ?"), \
  ("rolesQuery"=>"SELECT role, 'Roles' FROM roles WHERE username = ?"), \
  ("hashAlgorithm"=>"SHA-256"), ("hashEncoding"=>"base64")]])
```

The corresponding XML definition of the database security domain is as follows:

```
<security-domain name="db-domain"① cache-type="default">
  <authentication>
    <login-module code="Database"② flag="required">
      <module-option name="dsJndiName"③
        value="java:jboss/datasources/test-ds"/>
      <module-option name="principalsQuery"④
        value="SELECT password FROM users WHERE username = ?"/>
      <module-option name="rolesQuery"⑤
        value="SELECT role, 'Roles' FROM roles WHERE username = ?"/>
      <module-option name="hashAlgorithm"⑥ value="SHA-256"/>
      <module-option name="hashEncoding"⑦ value="base64"/>
    </login-module>
  </authentication>
</security-domain>
```

- ① The **name** module option is required and is the name of the security domain. It must be unique within the **standalone.xml** file (in a standalone server) or within a profile (in a managed domain).
- ② The **code** module option is a unique string that identifies the login module that should be used for this security domain. In this case, the in-built **Database** login module is being used.
- ③ The **dsJndiName** module option references a valid datasource reference defined in the configuration file.
- ④ The **principalsQuery** module option defines the SQL query that retrieves a unique row from the database that contains the details of the user who is logging in to the application. In this case, a unique **username** (primary key) from the **users** table is being fetched.
- ⑤ The **rolesQuery** module option defines the SQL query that retrieves a list of roles for a particular user. In this case, a list of roles for a particular user from the **roles** table is being fetched. Additionally, the "**Roles**" column may refer to another string to create sub-roles for an existing role. For most applications, the Roles string can be used.
- ⑥ The **hashAlgorithm** module option is the hash algorithm that should be used to encrypt the user passwords. Possible options are **MD5**, **SHA-1**, **SHA-256**, and **SHA-512**.
- ⑦ The **hashEncoding** module option defines the encoding of the hashed password. Possible options are **hex** and **base64**.

When a request is made to this EAP server that requires a user to be authenticated using the **db-domain**, the following steps occur:

1. The user name is retrieved from the request.
2. A database connection is retrieved from the connection pool named **java:jboss/datasources/test-ds**.
3. The **principalsQuery** is executed, with the placeholder (?) being replaced with the user name from the request.
4. If the password retrieved from the database does not match the password sent in the request, then the security domain notifies the EAP container that this user is not authenticated. The request is not processed further.
5. If the password retrieved from the database matches the password sent in the request, the **rolesQuery** is executed, with the placeholder (?) being replaced with the user name from the request.
6. If the user does not belong to a required role of the requested resource, then the security domain notifies the EAP container that this user is not authorized. The request is not processed further.
7. If the user does belong to one of the required roles of the requested resource, then the user is authorized and the request is passed on to the requested resource.

Configuring a security domain in an application

In order to utilize a security domain, users need to add a reference to the security domain in the **jboss-web.xml**. The following is an example of the **<security-domain>** tag in the **jboss-web.xml**:

```
<security-domain>Security_Domain_Name</security-domain>
```

The **web.xml** file allows users to configure the security of the application, by creating security restrictions to specific paths, restricting based on roles, and defining the authentication type.

In the **<security-constraint>** portion of the **web.xml** file is the **<web-resource-collections>** section that allows users to define which URL path requires authorization. The **<url-pattern>** tag can contain any path on the server or simply use a * to secure all of the pages and resources. The **<auth-constraint>** within the same section specifies which roles are allowed to access the restricted aspects of the application.

```
...
<security-constraint>
    <web-resource-collection>
        <web-resource-name>All resources</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>*</role-name>
    </auth-constraint>
</security-constraint>
...
```

The **<login-config>** tag within the **web.xml** defines the type of authentication to be used. **BASIC** refers to an authentication requirement wherein the server requests a user name and password from the web client which is validated against the database. Defining these aspects of the **web.xml** is outside the scope of this course.

```
...
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
```

► Guided Exercise

Securing an Application

In this lab, you will use a database security scheme to secure the Example application.

Resources	
Files:	/home/student/JB248/labs/security-dbrealm
Application URL:	http://127.0.0.1:8080/example

Outcomes

You should be able to configure an application to check the user name and password using a database backed security module.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and that the MySQL driver is installed:

```
[student@workstation ~]$ lab securing-dbrealm setup
```

- 1. Start the standalone instance of EAP by running the following command with the base directory located at **/home/student/JB248/labs/standalone**:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

- 2. Connect to the Security Data Source

In this step, you will configure the non-XA data source that connects to the database storing the credentials used by the example application. The datasource will be used later by the security subsystem to fetch the user name and password.

The database is a MySQL database running on **localhost** and it is named **bksecurity**. The credentials to access the database are **bkadmin/redhat**. To access it, use the JNDI name **java:jboss/datasource/bksecurity-ds**.

- 2.1. Access the management console by navigating to 127.0.0.1:9990. Go to the **Configuration** page.



Note

The admin user name is **jbossadm** and the password is **JBoss@RedHat123**.

- 2.2. Navigate to the Data Sources subsystem by clicking **Subsystems** and then **Data Sources**.
- 2.3. Select the **Non-XA** datasource type and then click **Add**.
- 2.4. On the first window, select **MySQL Data Source** and click **Next**.
- 2.5. Use the following parameters to complete the form:
 - **Name:** **bksecurity-ds**
 - **JNDI Name:** **java:jboss/datasources/bksecurity-ds**Click **Next**.
- 2.6. On Step 2 of the selection menu, click **Detected Driver** and select the driver named **mysql**. This is the driver that was installed in the previous exercise.
Click **Next**.
- 2.7. Use the following information for the database connection:
 - **Connection URL:** **jdbc:mysql://localhost:3306/bksecurity**
 - **Username:** **bkadmin**
 - **Password:** **redhat**Click **Next** and then **Finish** and the **bksecurity** datasource should appear in the fourth column.

► 3. Configure the Security Domain

Create a security domain that will use the datasource added in the previous step to obtain credentials used by the example application from a database. The security domain should have a default cache type and it will be used for authentication purposes.

- 3.1. Go back to the **Subsystems** column.
- 3.2. Click **Security** and click **Add** in the **Security Domain** column to add a new security domain named **bksecurity**.
- 3.3. Enter the name of the security domain as **bksecurity** and set the **Cache Type** to **default** and then click **Save**.
- 3.4. In order to configure the authentication modules, start the EAP CLI in a new terminal and connect to the standalone server. Enter **jbossadm** as the user name and **JBoss@RedHat123** as the password.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh --connect
```

- 3.5. Use the following command to create an **Authentication Module**.

```
[standalone@localhost:9990] /subsystem=security/security-domain\  
=bksecurity/authentication\  
=classic:add
```

**Note**

The output will indicate that a server reload is required, however this should be ignored for now as the server will be reloaded after the login module configuration is completed.

- 3.6. Create a login module that will connect to the datasource created in the previous step. It should query the table's users and roles to identify the credentials. The password is encrypted with a SHA-256 hash algorithm and the hash uses a base64 encoding.

Use the following values when creating the login module:

- Name: **database**
- Code: **Database**
- Flag: **required**
- These module options:
 - *dsJndiName*: **java:jboss/datasources/bksecurity-ds**
 - *principalsQuery*: **select password from users where username=?**
 - *rolesQuery*: **select role, "Roles" from roles where username=?**
 - *hashAlgorithm*:**SHA-256**
 - *hashEncoding*:**base64**

Use the following CLI command to create the login module:

```
[standalone@localhost:9990] /subsystem=security/security-domain=bksecurity/\
authentication=classic/login-module=database:add( \
  code=Database, \
  flag=required, \
  module-options=[ \
    ("dsJndiName"=>"java:jboss/datasources/bksecurity-ds"), \
    ("principalsQuery"=>"select password from users where username=?"), \
    ("rolesQuery"=>"select role, 'Roles' from roles where username=?"), \
    ("hashAlgorithm"=>"SHA-256"), \
    ("hashEncoding"=>"base64") \
  ])
```

**Note**

This command can be copied and pasted from **/home/student/JB248/labs/security-dbrealm/login-module**.

- 3.7. Reload the server to allow the changes to take place:

```
[standalone@localhost:9990] :reload
```

3.8. Verify your settings closely by running the following CLI command:

```
[standalone@localhost:9990] /subsystem=security/security-domain=bksecurity\
/authentication=classic/login-module=database:read-resource
```

The response should appear as follows:

```
{
  "outcome" => "success",
  "result" => {
    "code" => "Database",
    "flag" => "required",
    "module" => undefined,
    "module-options" => {
      ("rolesQuery" => "select role, \"Roles\" from roles where username=?"),
      ("principalsQuery" => "select password from users where username=?"),
      ("dsJndiName" => "java:jboss/datasources/bksecurity-ds"),
      ("hashAlgorithm" => "SHA-256"),
      ("hashEncoding" => "base64"),
    }
  }
}
```



Note

If a mistake is found in the **module-options**, each module option can be updated using the **map-put** command in the EAP CLI. For example, if there is an error with the **dsJndiName** value, the following command can update the module option:

```
[standalone@localhost:9990] /subsystem=security/security-domain\
=bksecurity/authentication=classic/login-module=database\
:map-put(name=module-options, key="dsJndiName", \
value="java:jboss/datasources/bksecurity-ds")
```

► 4. Configure the Application Security

To secure an application deployed on EAP, the first step is to modify the application so that it requires credentials to access it.

- 4.1. Using a text editor, open the **web.xml** file in the **/home/student/JB248/labs/security-dbrealm/example/src/main/webapp/WEB-INF** folder.
- 4.2. After the **<welcome-file-list>** section in the **web.xml** file, add the following XML, which defines a security constraint on all URLs to the application and requires a user to be authenticated. You can copy and paste this XML from the file **/home/student/JB248/labs/security-dbrealm/security-constraint.xml**.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>All resources</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
```

```
<role-name>*</role-name>
</auth-constraint>
</security-constraint>
<security-role>
    <role-name>*</role-name>
</security-role>
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>bksecurity</realm-name>
</login-config>
```

- 4.3. Save your changes to **web.xml**.

► 5. Configure the Security Domain in the Application

- 5.1. Using a text editor, open the **jboss-web.xml** file in the **/home/student/JB248/labs/security-dbrealm/example/src/main/webapp/WEB-INF** folder.
- 5.2. Within the **<jboss-web>** tag, enter the following **<security-domain>** tag:

```
<security-domain>bksecurity</security-domain>
```

bksecurity references the security domain that was created previously.

- 5.3. Save your changes to **jboss-web.xml**.

► 6. Package the Application

Open a new terminal and enter the following command to create a WAR file of the example application:

```
[student@workstation ~]$ cd /home/student/JB248/labs/security-dbrealm/\
example/src/main/webapp
[student@workstation webapp]$ jar -cvf example.war .
```

► 7. Deploy the Application

Using the Management Console or CLI, deploy the newly-created **example.war** file located at **/home/student/JB248/labs/security-dbrealm/example/src/main/webapp/example.war** on the standalone server.

```
[standalone@localhost:9990 /] deploy \
/home/student/JB248/labs/security-dbrealm/example/src/main/webapp/example.war
```

Monitor the server log for any errors on deployment. The application should deploy without any errors.

► 8. Verify the Security Settings

- 8.1. Navigate to **http://127.0.0.1:8080/example/**. You should be prompted to log in.
- 8.2. Enter “**admin**” for the user name and “**admin**” for the password. You should see the “Welcome to EAP 7” page after you are logged in successfully.



Note

If the authentication fails with those credentials, verify that the security domain was set up correctly with the correct values for each module option.

- 8.3. Undeploy the Example application from the standalone server using the CLI:

```
[standalone@localhost:9990] undeploy example.war
```

- 8.4. Exit the CLI and stop the running instance of EAP.

This concludes the guided exercise.

Configuring an LDAP Security Domain

Objectives

After completing this section, students should be able to:

- Configure a security domain based on the LDAP login module.

Defining an LDAP security domain

A security domain backed by an LDAP server that stores the users and role mapping for an application using the **Ldap** login module can be defined. It integrates with an LDAP server and authenticates users against the data stored in the LDAP Directory Information Tree (DIT). Users and Roles are stored under **Organizational Units (ou)** in the LDAP tree with the user name being used as the Distinguished Name (DN) to uniquely identify a user.

An LDAP security domain can be created with the EAP CLI using the following command:

```
[standalone@localhost:9990] /subsystem=security/security-domain= \
    ldap-domain:add(cache-type=default)
```

```
[standalone@localhost:9990] /subsystem=security/security-domain=ldap-domain \
    /authentication=classic:add \
    (login-modules=[{"code"=>"Ldap", "flag"=>"required", \
    "module-options"=>[("java.naming.factory.initial"=> \
    "com.sun.jndi.ldap.LdapCtxFactory"), \
    ("java.naming.provider.url"=>"ldap://instructor:389"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("principalDNPrefix"=>"uid="), ("principalDNSuffix"=>, \
    "ou=people, dc=redhat, dc=com"), ("rolesCtxDN"=>"ou=Roles,dc=redhat,dc=com"), \
    ("uidAttributeID"=>"member"), ("matchOnUserDN"=>"true"), \
    ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>"false")]]])
```

The corresponding XML definition of the LDAP security domain appears as follows:

```
<security-domain name="ldap-domain">
    <authentication>
        <login-module code="Ldap" flag="required">
            <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
            <module-option name="java.naming.provider.url" value="ldap://instructor:389"/>
            <module-option name="java.naming.security.authentication" value="simple"/>
            <module-option name="principalDNPrefix" value="uid="/>
```

```
<module-option name="principalDNSuffix" value="" ou=people, dc=redhat,
dc=com"/>
<module-option name="rolesCtxDN" value="ou=Roles,dc=redhat,dc=com"/>
<module-option name="uidAttributeID" value="member"/>
<module-option name="matchOnUserDN" value="true"/>
<module-option name="roleAttributeID" value="cn"/>
<module-option name="roleAttributeIsDN" value="false"/>
</login-module>
</authentication>
</security-domain>
```

► Guided Exercise

Configuring the LDAP Login Module

In this lab, you will use an LDAP security scheme to secure an application.

Resources	
Files:	/home/student/JB248/labs/security-ldap
Application URL:	http://127.0.0.1:8080/guessLDAP

Outcomes

You should be able to enable authentication in an application using a LDAP login module.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and to download the lab files:

```
[student@workstation ~]$ lab securing-ldap setup
```

- 1. Start the standalone instance of EAP by running the following command with the base directory located at **/home/student/JB248/labs/standalone**:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

- 2. **The guessLDAP Application**

The guessLDAP application is already configured to require authentication. Step through the following configuration files to see how it requires authentication.

- 2.1. In the **home/student/JB248/labs/security-ldap** folder is a web application named **guessLDAP.war**. Extract the application with the following command:

```
[student@workstation bin]$ cd /home/student/JB248/labs/security-ldap  
[student@workstation security-ldap]$ jar -xvf guessLDAP.war
```

- 2.2. Explore the **jboss-web.xml** file located at **/home/student/JB248/labs/security-ldap/WEB-INF/**.

```
...
<jboss-web>
    <security-domain>jb248_ldap</security-domain>
</jboss-web>
```

The **security-domain** points to a security domain used by the application that will be called **jb248_ldap**.

► 3. Configure the LDAP Security Domain

- 3.1. An LDAP server is running on the workstation VM on port 389. Run the following command to run an **ldapsearch** on the workstation VM in a new terminal window:

```
[student@workstation ~]$ ldapsearch -x -D "cn=Manager,dc=redhat,dc=com" \
-w 43etq5 -b "dc=redhat,dc=com"
```

You should see output of the LDAP users and groups with the following at the end of the output:

```
...OUTPUT OMITTED
#numResponses: 13
#numEntries: 12
```

- 3.2. Start the EAP CLI with the following commands in a new terminal window. Enter **jbossadm** as the user name and **JBoss@RedHat123** as the password.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect
```

- 3.3. Use the following commands to create the security domain named **jb248_ldap**:

```
[standalone@localhost:9990] /subsystem=security/security-domain\
=jb248_ldap:add(cache-type=default)
[standalone@localhost:9990] /subsystem=security/security-domain\
=jb248_ldap/authentication=classic:add
```

- 3.4. Create an LDAP login module within the new security domain with the following characteristics:

- Name: **ldap_login**
- Code: **Ldap**

Be careful to type the code exactly because it is case sensitive.

- Module Options:

```
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
java.naming.provider.url=ldap://localhost:389
java.naming.security.authentication=simple
```

```
principalDNPrefix=uid=
principalDNSuffix=,ou=people, dc=redhat, dc=com
rolesCtxDN=ou=Roles, dc=redhat, dc=com
uidAttributeID=member
matchOnUserDN=true
roleAttributeID=cn
roleAttributeIsDN=false
```

Use the following command to create the security domain:

```
[standalone@localhost:9990] /subsystem=security/security-domain=jb248_ldap/\nauthentication=classic/login-module=ldap_login:add( \
  code=Ldap, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"), \
    ("java.naming.provider.url"=>"ldap://localhost:389"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("principalDNPrefix"=>"uid="), \
    ("principalDNSuffix"=>, ou=people, dc=redhat, dc=com"), \
    ("rolesCtxDN"=>, ou=Roles, dc=redhat, dc=com"), \
    ("uidAttributeID"=>"member"), \
    ("matchOnUserDN"=>"true"), \
    ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>"false") \
  ])
)
```



Note

This command can be copied and pasted from **/home/student/JB248/labs/security-ldap/ldap-login-module**.

- 3.5. Enter the following command in the CLI to verify the security domain settings of **jb248_ldap**:

```
[standalone@localhost:9990] /subsystem=security/security-domain=\njb248_ldap:read-resource(recursive=true)
```

You should see the following result:

```
{
  "outcome" => "success",
  "result" => {
    ...
    "authentication" => {"classic" => {
      "login-modules" => [
        {
          "name" => "ldap_login",
          "code" => "Ldap",
          "flag" => "required",
          "module" => undefined,
          "module-options" => [
            ("java.naming.factory.initial" =>
              "com.sun.jndi.ldap.LdapCtxFactory"),
            ("java.naming.provider.url" =>
              "ldap://localhost:389"),
            ("java.naming.security.authentication" => "simple"),
            ("principalDNPrefix" => "uid="),
            ("principalDNSuffix" => ",ou=people,dc=redhat,dc=com"),
            ("rolesCtxDN" => ",ou=Roles,dc=redhat,dc=com"),
            ("uidAttributeID" => "member"),
            ("matchOnUserDN" => "true"),
            ("roleAttributeID" => "cn"),
            ("roleAttributeIsDN" => "false")
          ]
        }
      ]
    }
  }
}
```

```

        ("java.naming.provider.url" => "ldap://localhost:389"),
        ("java.naming.security.authentication" => "simple"),
        ("principalDNPrefix" => "uid="),
        ("principalDNSuffix" => ",ou=people, dc=redhat, dc=com"),
        ("rolesCtxDN" => "ou=Roles,dc=redhat,dc=com"),
        ("uidAttributeID" => "member"),
        ("matchOnUserDN" => "true"),
        ("roleAttributeID" => "cn"),
        ("roleAttributeIsDN" => "false")
    ]
},
...
}
}

```

- 3.6. Reload the server to allow the changes to take effect.

```
[standalone@localhost:9990] :reload
```

▶ 4. Test the LDAP Security Domain

- 4.1. Using the Management Console or CLI, deploy the **guessLDAP.war** file located at **/home/student/JB248/labs/security-ldap/**.

```
[standalone@localhost:9990 /] deploy \
/home/student/JB248/labs/security-ldap/guessLDAP.war
```

- 4.2. On the workstation, navigate to **http://localhost:8080/guessLDAP**. You should be prompted to log in if the login module was configured correctly.
- 4.3. Enter "**bt1**" for the user name and "**ldap1**" for the password, and you should be logged in successfully.
If authentication is successful, you should see the Guess application.
- 4.4. Undeploy the guessLDAP application from the standalone server using the CLI:

```
[standalone@localhost:9990] undeploy guessLDAP.war
```

- 4.5. Exit the CLI and stop the running instance of EAP.
This concludes the guided exercise.

Securing a JMS Destination

Objectives

After completing this section, a system administrator should be able to:

- Configure role-based access to topics and queues in the messaging subsystem.

Security domains for messaging

It was already explained in chapter **Configuring the Messaging Subsystem** that the default settings for the embedded EAP 7 MOM in the **messaging-activemq** subsystem have the following characteristics:

- Authentication is required only from remote connections.
- JMS connections are authenticated using the **other** security domain.
- All local applications have access to publish and consume messages on all destinations.

This default configuration allows developers to quickly deploy JMS applications and test them without worrying about security, and without leaving the EAP instance open to remote attacks. It also allows easy testing of remote JMS applications because the developer only needs to add users and roles to the **ApplicationRealm** security realm by using the **add-user.sh** script.

Most production environments probably require three changes:

- Enable authentication for local applications.
- Change the ActiveMQ security domain to one based on a relational database or LDAP directory.
- Remove the permissions granted to the **guest** role.

These changes are implemented by the following commands:

- To enable authentication for local applications, change the **override-in-vm-security** attribute to **false**:

```
/subsystem=messaging:/server=default:write-attribute(\n  name=override-in-vm-security,value=false)
```

- To change the ActiveMQ security domain, change the **security-domain** attribute to point to the desired security domain name. For example, to use the security domain named **production-sd**:

```
/subsystem=messaging:/server=default:write-attribute(\n  name=security-domain,value=production-sd)
```

- After making any of the previous configuration changes, reload the server. In managed domain mode, reload all server instances using the changed profile.
- There are two ways to remove the permissions granted to the **guest** role:

- Remove the **guest** role from the all-destinations (#) **security-setting**:

```
/subsystem=messaging:/server=default/security-setting#=rule=guest:remove
```

- Remove the whole all-destinations (#) **security-setting**:

```
/subsystem=messaging:/server=default/security-setting#:remove
```

After making those changes, the administrator has to either create **security-setting** objects granting the correct permissions to the roles associated with ActiveMQ application users or add new roles to the existing all-destinations **security-setting** object.

It is important to understand that the ActiveMQ users and roles have no relationship to an application users and roles. Remember that MOM is a server by itself. The ActiveMQ users and roles control application access to MOM destinations the same way a database users and roles control application access to a database tables.

The same way a database's users and roles do NOT usually mirror an application's users and roles, ActiveMQ's users and roles will NOT mirror the application's users and roles. The source of potential confusion is that ActiveMQ uses security domains the same way an application does, while databases have no knowledge of EAP security domains. Remember, the ActiveMQ security domain has no relationship to an application security domain.

Securing a Topic or Queue

The ActiveMQ MOM embedded in EAP 7 does NOT configures access rules in destination objects. Access rules are configured in **security-setting** objects which works much like the **address-setting** objects described in chapter **Configuring the Messaging Subsystem**. This eases configuring access rules for applications requiring multiple destinations in a consistent way.

A **security-setting** name is a wildcard expression that matches one or more destinations internal ActiveMQ names. The wildcard doesn't match the destination JNDI name or the destination object name given from the EAP CLI. The easiest way to find a destination name to use as the wildcard is from the runtime attribute **queue-address**.

Another way to find a destination's internal name is by following the naming convention ActiveMQ follows to generate the internal name from the destination object name. The internal name is generated by adding a prefix to the destination object name. The prefixes are:

- **jms.queue**. for **jms-queue** objects, that is, JMS **Queue** resources.
- **jms.topic**. for **jms-topic** objects, that is, JMS **Topic** resources.

A **security-setting** object that matches **jms-topic** objects also requires a wildcard suffix (either # or *) because JMS **Topic** resources are implemented internally by ActiveMQ as multiple queues, one for each subscriber. Each of those queues receives a suffix based on the subscriber ID.

A **security-setting** has child objects of type **role**. The name of this child object is the role name that is granted access, and its attributes are permissions. ActiveMQ understands the following permissions that loosely correspond to JMS API operations that can be performed on a destination:

- **send**: allows publication of messages to the destination.

- **consume**: allows consumption of messages from the destination.
- **createDurableQueue**: allows creation of durable queues.
- **deleteDurableQueue**: allows deletion of durable queues.
- **createNonDurableQueue**: allows creation of temporary queues.
- **deleteNonDurableQueue**: allows deletion of temporary queues.
- **manage**: allows management operations on the destination using a proprietary ActiveMQ API.

There are no permissions specific for JMS **Topic** resources because they are implemented by ActiveMQ as a queue for each subscriber. If a client has the **consume** permission on the **jms-topic** object, it has permission to subscribe to it.

To grant permissions to a destination for a role, create the role object inside a **security-setting** object whose name matches the destination, and set the granted permission attributes to **true**. All other permissions will be **false** by default.

For example, the following commands grant **send** and **consume** permissions to the **publisherAndConsumer** role on the **jms-topic** named **StockQuotes**:

```
/subsystem=messaging-activemq/server=default/security-setting=\
jms.topic.StockQuotes.#:add()
```

```
/subsystem=messaging-activemq/server=default/security-setting=\
jms.topic.StockQuotes.#/role=publisherAndConsumer:add('\
send=true,consume=true)
```

If multiple **security-setting** objects match the same destination, the most specific one overrides the generic ones. In the previous example, if the all-destinations (#) **security-setting** granted the **manage** permission to the **publisherAndConsumer** role the **jms.topic.StockQuotes.# security-setting** denies it.

► Quiz

Securing a JMS Destination

Choose the correct answer to the following questions:

- ▶ 1. Which of the following are default characteristics of the messaging subsystem in EAP? (Choose three.)
 - a. JMS connections without security configuration set are authenticated using the **other** security domain.
 - b. All local applications have access to publish and consume messages on all destinations.
 - c. Queues and Topics restrict **manage** rights to only users with the role **admin**.
 - d. Authentication is required only from remote connections by default.
 - e. No security realm is used until one is manually configured.

- ▶ 2. Which security setting allows users to publish messages? (Choose one.)
 - a. Send
 - b. Consume
 - c. Manage
 - d. CreateDurableQueue

- ▶ 3. Which of the following pattern values could be used to match to a queue named **jms.queue.news.europe.fr**? (Choose three.)
 - a. #
 - b. jms.queue.news.europe.#
 - c. jms.queue.news.europe.fr
 - d. jms.queue.news.europe.fr.##
 - e. news.europe.##

- ▶ 4. Which role is given to users by default if no messaging subsystem configurations are made? (Choose one.)
 - a. admin
 - b. No role is given.
 - c. guest
 - d. user

► Solution

Securing a JMS Destination

Choose the correct answer to the following questions:

- ▶ 1. Which of the following are default characteristics of the messaging subsystem in EAP? (Choose three.)
 - a. JMS connections without security configuration set are authenticated using the **other** security domain.
 - b. All local applications have access to publish and consume messages on all destinations.
 - c. Queues and Topics restrict **manage** rights to only users with the role **admin**.
 - d. Authentication is required only from remote connections by default.
 - e. No security realm is used until one is manually configured.

- ▶ 2. Which security setting allows users to publish messages? (Choose one.)
 - a. Send
 - b. Consume
 - c. Manage
 - d. CreateDurableQueue

- ▶ 3. Which of the following pattern values could be used to match to a queue named **jms.queue.news.europe.fr**? (Choose three.)
 - a. #
 - b. jms.queue.news.europe.#
 - c. jms.queue.news.europe.fr
 - d. jms.queue.news.europe.fr.##
 - e. news.europe.##

- ▶ 4. Which role is given to users by default if no messaging subsystem configurations are made? (Choose one.)
 - a. admin
 - b. No role is given.
 - c. guest
 - d. user

Configuring the Password Vault

Objectives

After completing this section, a system administrator should be able to:

- Protect passwords that are stored in server configuration files by using the password configuration.

The password vault

EAP simplifies a lot of server configuration by consolidating all subsystem configurations into a single XML file (**standalone.xml** or **domain.xml**). This, however, can expose sensitive information to any user that has access to the configuration files. For example, datasource credentials are by default stored in plain text in the XML file.

The vault encrypts sensitive strings, stores them in an encrypted keystore, and decrypts them for applications and verification systems. To avoid leaving sensitive credentials readable in the **standalone.xml** or **domain.xml** files, users can store passwords or other attributes in the vault and then reference the vault from within the server configuration file. Using a vault creates a level of abstraction and obfuscates data which could otherwise be read by anyone who has access to the configuration files.

Creating the password vault

The process for storing a password into the vault is accomplished using the following steps:

1. Create a Java keystore.
2. Initialize the EAP vault with the keystore.
3. Store the sensitive information in the vault.
4. Update the server configuration to include the vault information.
5. Reference the stored attribute in the server configuration file.

Create a Java Keystore to store sensitive strings

The vault utilizes a keystore by using the certificate stored in the keystore as an encryption key for the vault as a whole. To initialize the vault, users need to first create the JavaSE keystore. The following is the syntax used to create a private key, a certificate and to store them in a keystore:

```
keytool -genseckeyp -alias  
<alias> \  
-keyalg <algorithm> -storetype <type> -keysize size \  
-keystore <filepath>
```

- **alias**: The alias is a unique identifier for the vault or other data stored in the keystore.
- **keyalg**: The algorithm to use for encryption.

- **storetype:** The keystore type. **jceks** is the recommended type.
- **keysize** The size of the encryption key which determines the difficulty in brute forcing the key.
- **keystore:** The file path and file name in which the keystore's values are stored.

```
[student@workstation ~]$ keytool -genseckeypair -alias vault \
-keyalg AES -storetype jceks -keysize 128 \
-keystore /home/student/vault.keystore
```

After running the command, users will be prompted for a keystore password and a keystore file will be created at **/home/student/vault.keystore**.

Using the vault

The vault can be initialized either interactively, by providing each parameter one at a time, or by providing all of the parameters initially.

The following is an example of the syntax used to initialize the vault:

```
vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD \
--alias KEYSTORE_ALIAS --vault-block VAULT_BLOCK --attribute ATTRIBUTE \
--sec-attr SEC-ATTR --enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT \
--salt SALT
```

The following parameters are required to initialize the vault:

- **KEYSTORE_URL:** The path to the previously created keystore file.
- **KEYSTORE_PASSWORD:** The password to access the keystore that was used at the keystore's creation.
- **SALT:** A random string of eight characters used to encrypt the attribute stored in the vault.
- **KEYSTORE_ALIAS:** The alias that identifies the certificate stored in the keystore.
- **ITERATION_COUNT:** The number of times encryption is run.
- **ENC_FILE_DIR:** The path where the encrypted files are stored.
- **VAULT_BLOCK:** The name to be given to the block in the vault.
- **ATTRIBUTE:** The name of the attribute being stored. For example, "password" as an attribute name when storing a password value.
- **SEC-ATTR:** The value being stored.

The following is an example with the parameters populated:

```
[student@workstation ~]$ vault.sh --keystore /home/student/vault.keystore \
--keystore-password password --alias vault --vault-block bookstore \
--attribute password --sec-attr redhat --enc-dir /home/student/
--iteration 50 --salt 12345678
```

After running the **vault.sh** command, an XML definition of the vault is displayed, which needs to be added to the server configuration files. The following is an example of the XML that needs

to be added to either the **standalone.xml** or **host.xml** configuration file directly before the **<management>** section:

```
<vault>
  <vault-option name="KEYSTORE_URL" value="/home/student/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-31x/z0Xn83H4JaL0h5eK/N"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="12345678"/>
  <vault-option name="ITERATION_COUNT" value="50"/>
  <vault-option name="ENC_FILE_DIR" value="/home/student//>
</vault>
```

The final step is to replace the sensitive data with a reference to the attribute in the vault. The **vault.sh** command provides the exact syntax necessary to reference the secured attribute after all of the parameters are provided. Using the previous example, the vault command generated the following:

```
VAULT::bookstore::password::1
```

In order to use this reference, use the following syntax within the server configuration:

```
 ${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::1}
```

The following can replace the previous password in the server configuration for the bookstore datasource password:

```
 ${VAULT::bookstore::password::1}
```

After replacing the password for the datasource, the server configuration for the datasource will look similar to the following:

```
<datasource jndi-name="java:jboss/datasources/nookdyotr" ...>
  <connection-url>...</connection-url>
  <driver>mysql</driver>
  <security>
    <user-name>bkadmin</user-name>
    <password>${VAULT::bookstore::password::1}</password>
  </security>
</datasource>
```

► Guided Exercise

Encrypting a Password

In this lab, you will secure the bookstore MySQL datasource by storing the password in a vault file.

Resources	
Files:	/home/student/JB248/labs/security-encrypt/
Application URL:	http://127.0.0.1:8080/dstest

Outcomes

You should be able to keep the MySQL bookstore application database password encrypted and protected in a vault file.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, that the MySQL bookstore datasource is configured, and to download the lab files:

```
[student@workstation ~]$ lab securing-encrypt setup
```

► 1. Create a Java Keystore to Store the Database Password

1. Open a new terminal window and run the following commands to create a keystore file named **vault.keystore** at **/home/student/**:

```
[student@workstation ~]$ keytool -genkey -alias vault \
-keyalg AES -storetype jceks -keysize 128 \
-keystore /home/student/vault.keystore
```

Notice that the value of alias is **vault** and it refers to a keystore entry where the password will be stored.

2. When prompted, use "**password**" as the password for your keystore and certificate passwords.
3. Verify you now have a file named **vault.keystore** in **/home/student**.

```
[student@workstation ~]$ ls | grep vault
vault.keystore
```

► 2. Run the Vault Script to Encrypt a Password

1. Enter the following command to run the vault tool script, which is used for adding passwords to a vault:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./vault.sh
```

- 2.2. At the first prompt, enter 0 to select **Start Interactive Session**.
- 2.3. When prompted for the directory to store encrypted files, enter **/home/student/**. (Note the trailing slash is required.)
- 2.4. The **Keystore URL** is the path to the **vault.keystore** file you created in the previous step. Enter:

```
/home/student/vault.keystore
```

- 2.5. For the remaining prompts, use the following values:
 - Password: **password**
 - 8 character salt: **12345678**
 - Iteration count: **50**
 - Keystore Alias: **vault**
- 2.6. Make a note of the **Vault Configuration**.
The vault tool has now connected to your vault. Enter 0 to store a secured attribute.
- 2.7. At the prompt to enter the attribute value, enter **redhat**, which is the password to connect to the MySQL database running on your machine. You will have to enter **redhat** twice to verify.
- 2.8. At the prompt to enter a **Vault Block**, enter **bookstore**.
- 2.9. At the **Enter Attribute Name** prompt, enter **password**.
- 2.10. Make a note of the resulting information, as prompted:

```
Please make note of the following:  
*****  
Vault Block:bookstore  
Attribute Name:password  
Configuration should be done as follows:  
VAULT::bookstore::password::1  
*****
```

- 2.11. The password for the MySQL database is now in **vault.keystore**, so enter 3 to **Exit** the vault tool.

► 3. Configure the Vault

- 3.1. Open **/home/student/JB248/labs/standalone/configuration/standalone.xml** in a text editor.
- 3.2. To open the vault, an EAP instance should be configured in the **<vault>** section of the **standalone.xml** configuration file, which appears between the **<extensions>**

and <management> entries. Add the following <vault> section immediately before the <management> section in /home/student/JB248/labs/standalone/configuration/standalone.xml:

```
<vault>
  <vault-option name="KEYSTORE_URL" value="/home/student/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-31x/z0Xn83H4JaL0h5eK/N"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="12345678"/>
  <vault-option name="ITERATION_COUNT" value="50"/>
  <vault-option name="ENC_FILE_DIR" value="/home/student/"/>
</vault>
```

You can copy and paste this XML from the /home/student/JB248/labs/security-encrypt/vault.xml file.

Save your changes to standalone.xml.

- 3.3. Start the standalone instance. Within the first few lines of the log output, you should see log events showing a security vault successfully initialized and ready.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Within the first few lines of the log output, you should see log events showing a security vault successfully initialized and ready:

```
17:37:39,730 INFO [org.jboss.security] (Controller Boot Thread) PBOX00361:
Default Security Vault Implementation Initialized and Ready
```

▶ 4. Configure the Data Source

- 4.1. Navigate to the **Configuration** page of the management console at `http://localhost:9990`.
- 4.2. Click **Subsystems** and then click **Data Sources** to access the datasources subsystem.
- 4.3. Click the **bookstore** datasource in the list of **Non-XA** datasources, and then click **View**.
- 4.4. Click **Disable** to disable the **bookstore** datasource. (You cannot modify the security settings of a datasource that is currently in use.)
If prompted to reload the server, reload it and return to the bookstore datasource management page.
- 4.5. Select **Security**, then click **Edit**.
- 4.6. In the **Password** text field, remove the current password value.
- 4.7. Within a \${} notation, copy and paste the configuration entry displayed at the end of the vault tool script. The **Password** field should appear as follows:

```
 ${VAULT::bookstore::password::1}
```

**Note**

You will not be able to see what you type, so be sure to copy and paste to avoid incorrectly entering the data.

- 4.8. Click **Save** to save your changes.
- 4.9. Re-enable the **bookstore** datasource by clicking **Enable**.
- 4.10. Reload the server using the EAP CLI to allow the changes to take effect.

```
[student@workstation ~]$ /opt/jboss-eap-7.0/bin/jboss-cli.sh --connect  
[standalone@localhost:9990] :reload
```

▶ 5. Verify the Data Source

- 5.1. Deploy the **dtest.war** file located at **/tmp/dtest.war** with the EAP CLI:

```
[standalone@localhost:9990 /] deploy /tmp/dtest.war
```

- 5.2. Open the **standalone.xml** file in **/home/student/JB248/labs/standalone/configuration**. Verify the **<password>** entry for the **bookstore** is enclosed in **\${} **and contains the text you copy-and-pasted from the previous step.****
- 5.3. Navigate to **http://127.0.0.1:8080/dtest/**.
- 5.4. Enter **java:jboss/datasources/bookstore** for the JNDI name and **bookstore.CatalogItem** for the table name.
- 5.5. Click **Submit** and verify that the database was connected successfully.

▶ 6. Clean Up

- 6.1. Undeploy the **dtest.war** application:

```
[standalone@localhost:9990 /] undeploy dtest.war
```

- 6.2. Stop the running instance of EAP.

This concludes the guided exercise.

► Guided Exercise

Securing JBoss EAP

In this lab, you will secure an application, protect a datasource password with a vault, and secure a queue.

Resources	
Files:	/home/student/JB248/labs/security-lab
Application URL:	http://172.25.250.10:8080/messaging-client

Outcomes

You should be able to configure the security domain to secure the messaging client application, encrypt the datasource password for the bookstore datasource, and apply Role Based Access Control to the queue created in the previous exercise.

Before You Begin

Before beginning the guided exercise, run the following command to verify that no instance of EAP is running, the previous labs were completed, and to download the lab files:

```
[student@workstation ~]$ lab securing-lab setup
```

The first portion of this lab will focus on securing the messaging client application used in the previous chapter's final lab. Use the following steps to create a database backed security domain and to deploy the application in a managed domain.

After the application has a configured security domain, you will then secure the datasource that is used to verify users for the application. At the moment, the **bksecurity** datasource stores the password in plain text in the **domain.xml** file. Place the password in a new vault and update the datasource to reference the vault for the datasource's credentials.

Finally, update the security for the messaging queues that were created in the previous lab. Currently, any user is able to send or receive messages. Restrict sending privileges to only the **admin** users.

► 1. Configure the bksecurity Data Source in the Managed Domain

Create a new datasource in the managed domain in the **full-ha** profile for the **bksecurity** database. The datasource should have the following characteristics and should use the already installed **mysql** driver:

- **Name:** bksecurity-ds
- **JNDI Name:** java:jboss/datasources/bksecurity-ds
- **Connection URL:** jdbc:mysql://172.25.250.254:3306/bksecurity
- **Username:** bkadmin

- **Password:** redhat

Validate that the connection works properly.

1. Start the managed domain to enable the management console for creating and testing the datasource.

Start the host master on the **workstation**:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ sudo -u jboss ./domain.sh \  
-Djboss.domain.base.dir=/opt/domain/ \  
--host-config=host-master.xml
```

Start the host controller on **servera**:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \  
-Djboss.domain.base.dir=/opt/domain --host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254 \  
-bprivate=172.25.250.10
```

Start the host controller on **serverb**:

```
[student@serverb ~]$ sudo -u  
jboss /opt/jboss-eap-7.0/bin/domain.sh \  
-Djboss.domain.base.dir=/opt/domain --host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254 \  
-bprivate=172.25.250.11
```

- 1.2. Add a new non-XA datasource with the following credentials as a **MySQL Data Source**:

- **Name:** bksecurity-ds
- **JNDI Name:** java:jboss/datasources/bksecurity-ds
- **Connection URL:** jdbc:mysql://172.25.250.254:3306/bksecurity
- **Username:** bkadmin
- **Password:** redhat

- 1.3. Access the management console running at 172.25.250.254:9990 and click on **Configuration** and then the datasource subsystem in the **full-ha** profile. The user name for the management console is **jbossadm** and the password is **JBoss@RedHat123**.

- 1.4. Select the **Non-XA** data source type and then click **Add**.

- 1.5. On the first window, select **MySQL Datasource** and click **Next**.

- Enter **bksecurity-ds** for the **Name**.
- Enter **java:jboss/datasources/bksecurity-ds** for the **JNDI Name**.

Click **Next**.

- 1.6. On Step 2 of the selection menu, click **Detected Driver** and select the driver named **mysql**.
Click **Next**.
- 1.7. Finish the Datasource configuration with the following values:
 - The **Connection URL** is **jdbc:mysql://172.25.250.254:3306/bksecurity**.
 - Enter **bkadmin** for the **Username** and **redhat** for the **Password**.Click **Finish** and the **bookstore** data source should appear in the fourth column.
- 1.8. Verify the datasource is working properly by testing it in the management console.
Click **Connection** and then click **Test Connection** in the **bksecurity-ds** overview. A pop-up window should confirm that the connection is valid.

► 2. Create a Database Authentication Login Module

Create a security domain that will use the datasource added in the previous step to obtain credentials used by the example application from a database. The security domain should have a default cache type and it will be used for authentication purposes. The security domain should have the following characteristics:

- Security Domain Name: **jb248-sd**
 - Login module name: **database**
 - Code: **Database**
 - Flag: **required**
 - These module options:
 - *dsJndiName: java:jboss/datasources/bksecurity-ds*
 - *principalsQuery: select password from users where username=?*
 - *rolesQuery: select role, 'Roles' from roles where username=?*
 - *hashAlgorithm:SHA-256*
 - *hashEncoding:base64*
- 2.1. Click **Security** in the **Configuration** page for the **full-ha** profile and click **Add** in the **Security Domain** column to add a new security domain named **jb248-sd**.
 - 2.2. Enter the name of the security domain as **jb248-sd** and set the **Cache Type** to default and then click **Save**.
 - 2.3. In order to configure the authentication modules, start the EAP CLI in a new terminal and connect to the host master.

```
[student@workstation ~]$ cd  
/opt/jboss-eap-7.0/bin  
[student@workstation bin]$ sudo -u jboss ./jboss-cli.sh --connect \  
--controller=172.25.250.254:9990
```

- 2.4. Use the following command to create an authentication module.

```
[domain@172.25.250.254:9990] /profile=full-ha/subsystem=\nsecurity/security-domain\
=jb248-sd/authentication\
=classic:add
```

- 2.5. Create a login module that will connect to the data source created in the previous step. It should query the table's users and roles to identify the credentials. The password is encrypted with a SHA-256 hash algorithm and the hash uses a base64 encoding.

Use the following values when creating the login module:

- Name: **database**
- Code: **Database**
- Flag: **required**
- These module options:
 - *dsJndiName*: **java:jboss/datasources/bksecurity-ds**
 - *principalsQuery*: **select password from users where username=?**
 - *rolesQuery*: **select role, 'Roles' from roles where username=?**
 - *hashAlgorithm*:**SHA-256**
 - *hashEncoding*:**base64**

Use the following CLI command to create the login module:

```
[domain@172.25.250.254:9990] /profile=full-ha/subsystem=security\
/security-domain=jb248-sd/authentication\
classic/login-module=database:add( \
  code=Database, \
  flag=required, \
  module-options=[ \
    ("dsJndiName"=>"java:jboss/datasources/bksecurity-ds"), \
    ("principalsQuery"=>"select password from users where username=?"), \
    ("rolesQuery"=>"select role, 'Roles' from roles where username=?"), \
    ("hashAlgorithm"=>"SHA-256"), \
    ("hashEncoding"=>"base64") \
  ])
])
```

This command can be copied and pasted from **/home/student/JB248/labs/security-lab/login-module**.

- 2.6. Reload the server to allow the changes to take place:

```
[domain@172.25.250.254:9990] :reload-servers(blocking=true)
```

- 2.7. Verify your settings by running the following CLI command:

```
[domain@172.25.250.254:9990] /profile=full-ha/subsystem=security\
/securities-domain=jb248-sd/authentication=classic\
login-module=database:read-resource
```

The response should be as follows:

```
{
  "outcome" => "success",
  "result" => {
    "code" => "Database",
    "flag" => "required",
    "module" => undefined,
    "module-options" => {
      ("rolesQuery" => "select role, 'Roles' from roles where username=?"),
      ("principalsQuery" => "select password from users where username=?"),
      ("dsJndiName" => "java:jboss/datasources/bksecurity-ds"),
      ("hashAlgorithm" => "SHA-256"),
      ("hashEncoding" => "base64"),
    }
  }
}
```



Note

If there are typos or errors in the login-module, it is easier to update the module options in the management console. Access the security subsystem to fix any errors in the login-module.

3. Deploy the Secure Messaging Client

In the previous lab, you deployed a simple messaging client. Explore the source for an updated, secure version of the same application. Open the `web.xml` and `jboss-web.xml` to see that the application uses the new security domain, `jb248-sd`. Then deploy the war file located at `/tmp/messaging-client-secure.war` onto server group **Group 1**.

The application source code is located at `/home/student/JB248/labs/security-lab/messaging-client-secure/`. Verify it is secured by navigating to `http://172.25.250.10:8080/messaging-client` and using the credentials **admin** for the user name and **admin** for the password.

- 3.1. Open the `/home/student/JB248/labs/security-lab/messaging-client-secure/src/main/webapp/WEB-INF/web.xml` file. Observe the `login-config` tag with the basic authorization method and realm name `jb248-sd`.

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>jb248-sd</realm-name>
</login-config>
```

- 3.2. Open the `/home/student/JB248/labs/security-lab/messaging-client-secure/src/main/webapp/WEB-INF/jboss-web.xml` file. Within the `<jboss-web>` tag notice the `<security-domain>` is configured as `jb248-sd`:

```
<security-domain>jb248-sd</security-domain>
```

- 3.3. Using the Management Console or CLI, deploy the newly-created **messaging-client-secure.war** file located at **/tmp** on server **Group1**.

```
[domain@172.25.250.254:9990] deploy /tmp/messaging-client-secure.war --server-groups=Group1
```

Monitor the server log for any errors on deployment. The application should deploy without any errors.

- 3.4. Access the application as user **admin** and password **admin** at <http://172.25.250.10:8080/messaging-client-secure>.

▶ 4. Create a Vault Keystore

The **bksecurity** data source has the user name and password stored in plain text in the **domain.xml** file, creating a high security risk. Store the password in a vault and reference the vault in the configuration to protect the datasource's credentials.

Create a keystore with the following characteristics:

- Alias: **vault**
- keyalg: **AES**
- genseckeyp: enabled
- storetype: **jceks**
- keyszie: **128**
- keystore: **/opt/jboss-eap-7.0/vault.keystore**

When prompted, use "**password**" for the vault's password.

- 4.1. Open a new terminal window and run the following commands to create a new keystore file named **vault.keystore** at **/opt/jboss-eap-7.0/**:

```
[student@workstation ~]$ sudo -u jboss keytool \
-genseckeyp -alias vault \
-keyalg AES -storetype jceks -keyszie 128 \
-keystore /opt/jboss-eap-7.0/vault.keystore
```

- 4.2. When prompted, use "**password**" as the password for your keystore.

- 4.3. Copy the new keystore to both **servera** and **serverb**:

```
[student@workstation ~]$ sudo scp /opt/jboss-eap-7.0/vault.keystore \
servera:/opt/jboss-eap-7.0
[student@workstation ~]$ sudo scp /opt/jboss-eap-7.0/vault.keystore \
serverb:/opt/jboss-eap-7.0
```

- 4.4. Change ownership of the keystore to user **jboss** on both **servera** and **serverb**:

```
[student@servera ~]$ sudo chown jboss:jboss /opt/jboss-eap-7.0/vault.keystore
```

```
[student@serverb ~]$ sudo chown jboss:jboss /opt/jboss-eap-7.0/vault.keystore
```

► 5. Create an Entry with the bksecurity Data Source Password

Create a vault to store the bksecurity data source password. The vault should have the following characteristics and it should reference the keystore created in the previous step located at **/opt/jboss-eap-7.0/vault.keystore**.

- Password: **password**
- 8 character salt: **12345678**
- Iteration count: **50**
- Keystore Alias: **vault**
- Vault Block: **bksecurity**
- Attribute Name: **password**
- Attribute Value: **redhat**

Copy the vault file to **/opt/jboss-eap-7.0/** on both **servera** and **serverb**, and update the **/opt/domain/configuration/host-slave.xml** to include the vault definition.

- 5.1. Enter the following command to run the vault tool script, which is used for adding passwords to a vault:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ sudo -u jboss ./vault.sh
```

- 5.2. At the first prompt, enter 0 to select **Start Interactive Session**.
- 5.3. When prompted for the directory to store encrypted files, enter **/opt/jboss-eap-7.0/**. (Note the trailing slash is required.)
- 5.4. The **Keystore URL** is the path to the **vault.keystore** file you created in the previous step. Enter:

```
/opt/jboss-eap-7.0/vault.keystore
```

- 5.5. For the remaining prompts, use the following values:

- Password: **password**
- 8 character salt: **12345678**
- Iteration count: **50**
- Keystore Alias: **vault**

- 5.6. Make a note of the **Vault Configuration**.

The vault tool has now connected to your vault. Enter 0 to store a secured attribute.

At the prompt to enter attribute value, enter **redhat**, which is the password to connect to the MySQL database. You will have to enter **redhat** twice to verify.

- 5.7. At the prompt to enter a **Vault Block**, enter **bksecurity** and at the **Enter Attribute Name** prompt, enter **password**.

- 5.8. Make a note of the resulting information:

```
Please make note of the following:  
*****  
Vault Block:bksecurity  
Attribute Name:password  
Configuration should be done as follows:  
VAULT::bksecurity::password::1  
*****
```

- 5.9. The password for the MySQL database is now in **vault.keystore**. Enter 3 to **Exit** the vault tool.

- 5.10. Stop the host on **servera**, **serverb**, and **workstation**.

- 5.11. Add the following **<vault>** section immediately before the **<management>** section in the **/opt/domain/configuration/host-master.xml** on the workstation:

```
<vault>  
  <vault-option name="KEYSTORE_URL" value="/opt/jboss-eap-7.0/vault.keystore"/>  
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-31x/z0Xn83H4JaL0h5eK/N"/>  
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>  
  <vault-option name="SALT" value="12345678"/>  
  <vault-option name="ITERATION_COUNT" value="50"/>  
  <vault-option name="ENC_FILE_DIR" value="/opt/jboss-eap-7.0//"/>  
</vault>
```



Note

This vault definition can be copied and pasted from the output of the **vault.sh** command.

Save your changes to **host-master.xml**.

- 5.12. On both **servera** and **serverb**, add the same **<vault>** definition that was added to the host master to the **/opt/domain/configuration/host-slave.xml** configuration file before the management section.

```
<vault>
  <vault-option name="KEYSTORE_URL" value="/opt/jboss-eap-7.0/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-31x/z0Xn83H4JaL0h5eK/N"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="12345678"/>
  <vault-option name="ITERATION_COUNT" value="50"/>
  <vault-option name="ENC_FILE_DIR" value="/opt/jboss-eap-7.0"/>
</vault>
```

- 5.13. Copy the vault file from the **workstation** to **servera** and **serverb**:

```
[student@workstation ~]$ sudo scp /opt/jboss-eap-7.0/VAULT.dat \
servera:/opt/jboss-eap-7.0
[student@workstation ~]$ sudo scp /opt/jboss-eap-7.0/VAULT.dat \
serverb:/opt/jboss-eap-7.0
```

Change ownership of the **VAULT.dat** file to the **jboss** user:

```
[student@servera ~]$ sudo chown jboss:jboss /opt/jboss-eap-7.0/VAULT.dat
```

```
[student@serverb ~]$ sudo chown jboss:jboss /opt/jboss-eap-7.0/VAULT.dat
```

- 5.14. Start the host on **servera** and **serverb** as well as the host master on **workstation**.
- 5.15. Replace the password for the **bksecurity-ds** datasource with the vault reference to the bksecurity password.

In the management console, access the **bksecurity-ds** datasource in the datasource subsystem. Click **View** and disable the datasource. Use the following CLI command to reload the servers:

```
[domain@172.25.250.254:9990 /] /:reload-servers(blocking=true)
```

Return to the management console and click **Edit** under the security credentials. Update the password to the following and save the changes:

```
 ${VAULT::bksecurity::password::1}
```

- 5.16. Reload the servers again for the changes to take effect:

```
[domain@172.25.250.254:9990 /] /:reload-servers(blocking=true)
```

In the management console in the **bksecurity-ds** overview, click **Connection** and click **Test Connection** to verify that the connection works. A pop-up should confirm that the connection is valid. If not, use the server logs to troubleshoot the problem.

► 6. Secure the Queue

The credentials for the messaging producer and consumer are both hardcoded with the **admin** user's credentials.

Restrict access to every queue in the application to ensure that only administrators are able to send and receive messages. Create a new security restriction for the role **admin** that has send, receive, and manage privileges. It should use # for the pattern so that the settings are applied to every queue. Remove the existing **guest** security configuration.

Use the following command to set the default security domain for the messaging subsystem to be **jb248-sd**:

```
[domain@172.25.250.254:9990 server=default] /profile=full-ha/subsystem\
=messaging-activemq/server=default:\ 
write-attribute(name=security-domain,value=jb248-sd)
```

- 6.1. Navigate to the management console on the workstation and select the **full-ha** profile. Click **Messaging - ActiveMQ**, select **default**, and click **Queues/Topics**.
- 6.2. On the left side of the messaging subsystem overview, click **Security Settings**.
- 6.3. Click **Remove** to remove the current permissions for the **guest** role.
- 6.4. Click **Add** to create a new security setting. Use the following settings to allow all users with the role **admin** the ability to send, consume, and manage any queue.
 - Pattern: #
 - Role: **admin**
 - Send: Selected
 - Consume: Selected
 - Manage: SelectedClick **Save** to save the settings.
- 6.5. Using the EAP CLI, use the following command to change the default security domain for the messaging-activemq subsystem to the **jb248-sd** security domain that was created earlier in the lab that is backed by a database:

```
[domain@172.25.250.254:9990
server=default] /profile=full-ha/subsystem\
=messaging-activemq/server=default:\ 
write-attribute(name=security-domain,value=jb248-sd)
```

- 6.6. Reload all of the servers in the managed domain.

```
[domain@172.25.250.254:9990 /] /:reload-servers(blocking=true)
```

▶ 7. Test the Queue

Access the secure message client application and send a test message onto a queue when logged in as the user **admin**. Remove the send and receive privileges for this user and verify that you can no longer send a message.

- 7.1. Deploy the MDB application located at **/tmp/messaging-mdb-secure** onto server **Group1**.

```
[domain@172.25.250.254:9990 /] deploy /tmp/messaging-mdb-secure.jar \
--server-groups=Group1
```

- 7.2. Publish a test message. Enter the message producer application running on **servera.1** at <http://172.25.250.10:8080/messaging-client-secure> using the **admin/admin** credentials and fill the form as follows:

- **Message Count:** 5
- **Message Label:** testmsg
- **Message to send:** jms test

Navigate to <http://172.25.250.10:8080/messaging-client-secure>.

Complete the web form and click **Send Message** to send the message.

The web form should refresh and show the following message in green: **Messages sent successfully.**

- 7.3. Check the server logs for the entries generated by the consumer application. Servers **servera.1** and **serverb.1** should have log entries generated by the consumer application.

There should be log entries similar to the following on the **servera** terminal window:

```
[Server:servera.1] 19:54:56,317 INFO [class
com.redhat.training.messaging.mdb.MessageReceiver] (Thread-11 (ActiveMQ-client-
global-threads-904575248)) Message Properties: Copy #3 [testmsg]
[Server:servera.1] 19:54:56,318 INFO [class
com.redhat.training.messaging.mdb.MessageReceiver] (Thread-12 (ActiveMQ-client-
global-threads-904575248)) Message Properties: Copy #5 [testmsg]
[Server:servera.1] 19:54:56,318 INFO [class
com.redhat.training.messaging.mdb.MessageReceiver] (Thread-11 (ActiveMQ-client-
global-threads-904575248)) Message Body: jms test
[Server:servera.1] 19:54:56,319 INFO [class
com.redhat.training.messaging.mdb.MessageReceiver] (Thread-12 (ActiveMQ-client-
global-threads-904575248)) Message Body: jms test
[Server:servera.1] 19:54:56,322 INFO [class
com.redhat.training.messaging.mdb.MessageReceiver] (Thread-10 (ActiveMQ-client-
global-threads-904575248)) Message Properties: Copy #1 [testmsg]
[Server:servera.1] 19:54:56,323 INFO [class
com.redhat.training.messaging.mdb.MessageReceiver] (Thread-10 (ActiveMQ-client-
global-threads-904575248)) Message Body: jms test
```

There should be log entries similar to the following on the **serverb** terminal window:

```
[Server:serverb.1] 19:54:58,282 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-9 (ActiveMQ-client-global-threads-1005879511)) Message Properties: Copy #4 [testmsg]
[Server:serverb.1] 19:54:58,285 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-8 (ActiveMQ-client-global-threads-1005879511)) Message Properties: Copy #2 [testmsg]
[Server:serverb.1] 19:54:58,285 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-9 (ActiveMQ-client-global-threads-1005879511)) Message Body: jms test
[Server:serverb.1] 19:54:58,285 INFO [class com.redhat.training.messaging.mdb.MessageReceiver] (Thread-8 (ActiveMQ-client-global-threads-1005879511)) Message Body: jms test
```

- 7.4. Return to the management console and update the messaging subsystem security settings for the default server. Click **Edit** next to the **admin** role settings and remove the sending and consuming privileges.
- 7.5. Return to the application and try to send another test message with the following characteristics:
 - **Message Count:** 5
 - **Message Label:** **authorizationtest**
 - **Message to send:** **authorization test**
- 7.6. Verify in the server logs for **servera** that no message was sent because this user's role did not have proper authorization to send a message.

```
[Server:servera.1] 19:00:27,100 INFO [stdout] (default task-8)
javax.jms.JMSecurityRuntimeException: AMQ119032: User: admin does not have permission='SEND' on address jms.queue.TestQueue
```

► 8. Clean Up and Grading

- 8.1. Press **Ctrl+C** to stop the domain controller on **workstation** and the two host controllers on **servera** and **serverb**.
- 8.2. Run the following command to grade the exercise:

```
[student@workstation bin]$ lab securing-lab grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- Security domains define how applications are authenticated and authorized.
- EAP has four security domains defined by default: **jboss-ejb-policy**, **jboss-web-policy**, **other**, and **jaspitest**.
- A database login module is a security domain backed by a database, which stores the user names and role mapping to secure authentication for an application.
- A security domain can be backed by an LDAP server to be used for authorization and authentication in an application.
- The messaging subsystem by default uses the **other** security domain and the **ApplicationRealm** security realm.
- By adjusting the **security-settings**, users can restrict access and authorization for queues and topics based on roles.
- A password vault is a useful tool for obscuring sensitive data in the server configuration files, such as passwords to a database.
- The process for storing a password into the vault is accomplished using the following steps:
 1. Create a Java keystore.
 2. Initialize the EAP vault with the keystore.
 3. Store the sensitive information in the vault.
 4. Update the server configuration to include the vault information.
 5. Reference the stored attribute in the server configuration file.
- The vault can be initialized by running the **vault.sh** command in the **EAP_HOME/bin/** folder.

Chapter 10

Configuring the Java Virtual Machine

Overview

Goal Configure the JVM settings in a standalone server and in a managed domain.

- Objectives**
- Make JVM configuration settings in a standalone server.
 - Make JVM configuration settings in a managed domain.

- Sections**
- Configuring the JVM in a Standalone Server (and Quiz)
 - Configuring the JVM in a Managed Domain (and Guided Exercise)

- Lab**
- Configuring the Java Virtual Machine

Configuring the JVM in a Standalone Server

Objectives

After completing this section, students should be able to:

- Make JVM configuration settings in a standalone server.

JVM memory architecture

EAP is written in Java, a popular, multi-platform and object-oriented programming language. Java programs run on a **Java Virtual Machine (JVM)**. It is important for a JBoss system administrator to have at least a minimal understanding of how Java programs run within the JVM. In this section, configuring the memory settings of the JVM for your EAP servers will be discussed. EAP 7 needs a **JDK 1.8** compatible JVM to run. Earlier JVMs such as JDK 1.7 are no longer supported.

The JVM runs as a regular user-space process on the OS, and it strictly follows the memory limits defined by the OS. To ensure uniform multi-platform runtime support, the flags and memory settings of the JVM are identical among the different Operating Systems. During the JVM start-up process, some validations are made to guarantee that valid parameters are configured (minimum heap size less than or equal to maximum heap size, for example).

The JVM uses sophisticated memory management mechanisms and it can automatically deallocate unused Java data structures (objects) from the heap using a process called **garbage collection**.

The memory of a Java Virtual Machine (JVM) can be categorized into two areas:

- **Heap**: a dynamically growing and shrinking block of memory where Java data structures (objects) from applications reside.
- **Non-Heap**: consisting of the stack, class and method metadata, code cache, and the metaspace.

The size of these various blocks of memory can be specified as arguments to the JVM at startup. Some of the settings are shown in the following diagram:

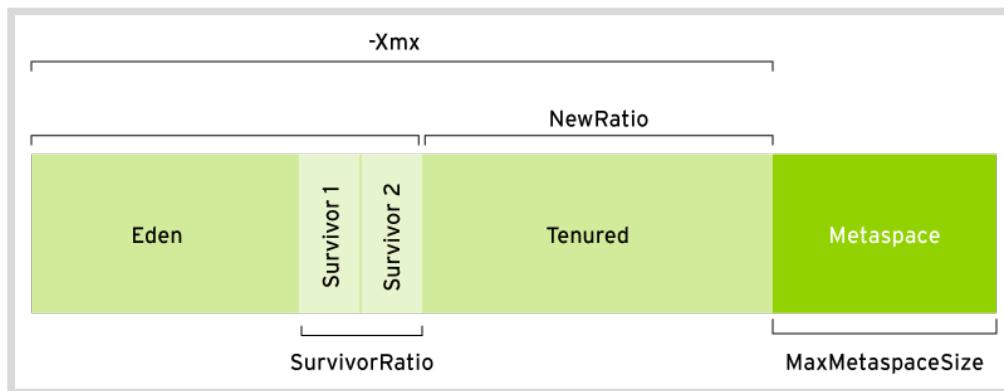


Figure 10.1: JVM memory regions

- The **Eden space** is where newly-created objects appear. If an object has a short life, it can be quickly garbage collected in Eden space.
- The **Survivor space** is where objects that have survived garbage collection cycles in the young generation are stored. There are usually two survivor spaces.
- The **Tenured space** is where "old" objects that have survived multiple garbage collection cycles in Eden and the survivor spaces are stored.
- The **Metaspace** is where class and method metadata and other JVM internal data structures are stored.



Note

One of the major changes between JDK 1.7 and JDK 1.8 is the removal of the **Permanent Generation** from JDK 1.8. The Permanent Generation has been replaced by the **Metaspace** in JDK 1.8.

There are numerous settings available to tune the performance of the JVM. Here are some of the more common ones:

- **-Xms** is used to denote the minimum heap size. It should always be less than or equal to the maximum heap size.
- **-Xmx** is used to denote the maximum heap size. If the application allocates memory greater than the maximum heap size, it will lead to the JVM process swapping and eventually cause an out-of-memory (OOM) error.
- **-XX:MaxMetaspaceSize** is used to denote the maximum size of the Metaspace.
- **-XX:NewRatio** is used to denote the ratio of the new-to-old generation sizes.
- **-XX:NewSize** is used to denote the size of the Eden generation.

Configuring the sizes of your JVM memory spaces appropriately can help avoid out-of-memory exceptions and also improve the performance of any application. Configuring the JVM is different for a standalone server versus servers in a managed domain:

- In a Standalone server, the installed JVM starts the server instance, and this process is responsible for determining the JVM settings.
- In a Managed Domain, the Host Controller starts the server JVM processes, so the Host Controller is responsible for determining the JVM settings of each individual server.

Configuring the JVM settings for a standalone server

The JVM memory settings for a standalone server are found in the **standalone.conf** file in your **JBOSS_HOME/bin** folder. The JVM settings defined in the **JAVA_OPTS** variable are passed to your system's JVM process that is running the standalone server.

The line to modify in **standalone.conf** is as follows:

```
# Specify options to pass to the Java VM.  
#  
if [ "$JAVA_OPTS" = "x" ]; then  
    JAVA_OPTS="-Xms1303m -Xmx1303m -XX:MetaspaceSize=96M -  
XX:MaxMetaspaceSize=256m ..."
```

Within the double quotes of the **JAVA_OPTS** variable, you can edit, add, or delete any of the settings regarding the JVM memory and garbage collection options for your standalone server. Changes to the JVM settings in **standalone.conf** require a server restart.

► Quiz

Configuring the JVM in a Standalone Server

Choose the correct answer to the following questions:

► 1. **How many survivor spaces are in the young generation of the JVM? (Choose one.)**

- a. 1
- b. 2
- c. 3
- d. 4

► 2. **Which of the following statements are true? (Choose two.)**

- a. The Metaspace and the permanent generation have to be equal in size in JDK 1.8.
- b. There is no permanent generation in JDK 1.8.
- c. EAP 7 can run on JDK 1.7 as well as JDK 1.8.
- d. EAP 7 can only run on JDK 1.8 and later.
- e. In JDK 1.8, the min (-Xms) and max (-Xmx) JVM heap sizes must always be set to the same value.

► 3. **Which of the following statements are false? (Choose two.)**

- a. For JDK 1.8, there are different JVM flags for different operating systems.
- b. For an EAP 7 standalone server, you cannot change the JVM settings without restarting the server.
- c. The young generation (Eden) memory size is always less than the maximum heap size (-Xmx) configured for the JVM.
- d. You cannot run more than one EAP 7 standalone server per operating system.
- e. The old generation (tenured) memory size is always less than the maximum heap size (-Xmx) configured for the JVM.

► 4. **Which of the following statements are correct? (Choose two.)**

- a. The min heap size (-Xms) of a JVM can be greater than the max heap size (-Xmx).
- b. The min heap size (-Xms) is always less than or equal to the max heap size (-Xmx).
- c. The min heap size (-Xms) is always less than and can never be equal to the max heap size (-Xmx).
- d. You can set both minimum and maximum heap size of a JVM to a value greater than the memory allocated (RAM) to the operating system.

► **5. Consider the following configuration in the standalone.conf file of an EAP 7 server**

```
JAVA_OPTS="-Xms4096m -Xmx2048m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=4098m"
```

Which of the following statements is correct? (Choose one.)

- a. The server will fail to start because the max Metaspace size is more than the max heap size (-Xmx).
- b. The server fails to start because the value of the min heap size (-Xms) is too low.
- c. The server fails to start because the min heap size (-Xms) is greater than the max heap size (-Xmx).
- d. The server starts successfully with no errors
- e. The JVM immediately allocates 2GB because the max heap size (-Xmx) is set to this value.

► Solution

Configuring the JVM in a Standalone Server

Choose the correct answer to the following questions:

► 1. **How many survivor spaces are in the young generation of the JVM? (Choose one.)**

- a. 1
- b. 2
- c. 3
- d. 4

► 2. **Which of the following statements are true? (Choose two.)**

- a. The Metaspace and the permanent generation have to be equal in size in JDK 1.8.
- b. There is no permanent generation in JDK 1.8.
- c. EAP 7 can run on JDK 1.7 as well as JDK 1.8.
- d. EAP 7 can only run on JDK 1.8 and later.
- e. In JDK 1.8, the min (-Xms) and max (-Xmx) JVM heap sizes must always be set to the same value.

► 3. **Which of the following statements are false? (Choose two.)**

- a. For JDK 1.8, there are different JVM flags for different operating systems.
- b. For an EAP 7 standalone server, you cannot change the JVM settings without restarting the server.
- c. The young generation (Eden) memory size is always less than the maximum heap size (-Xmx) configured for the JVM.
- d. You cannot run more than one EAP 7 standalone server per operating system.
- e. The old generation (tenured) memory size is always less than the maximum heap size (-Xmx) configured for the JVM.

► 4. **Which of the following statements are correct? (Choose two.)**

- a. The min heap size (-Xms) of a JVM can be greater than the max heap size (-Xmx).
- b. The min heap size (-Xms) is always less than or equal to the max heap size (-Xmx).
- c. The min heap size (-Xms) is always less than and can never be equal to the max heap size (-Xmx).
- d. You can set both minimum and maximum heap size of a JVM to a value greater than the memory allocated (RAM) to the operating system.

► **5. Consider the following configuration in the standalone.conf file of an EAP 7 server**

```
JAVA_OPTS="-Xms4096m -Xmx2048m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=4098m"
```

Which of the following statements is correct? (Choose one.)

- a. The server will fail to start because the max Metaspace size is more than the max heap size (-Xmx).
- b. The server fails to start because the value of the min heap size (-Xms) is too low.
- c. The server fails to start because the min heap size (-Xms) is greater than the max heap size (-Xmx).
- d. The server starts successfully with no errors
- e. The JVM immediately allocates 2GB because the max heap size (-Xmx) is set to this value.

Configuring the JVM in a Managed Domain

Objectives

After completing this section, students should be able to:

- Configure JVM settings in a managed domain.

Overview of JVM settings in a managed domain

In an EAP managed domain, the host controller is responsible for starting the JVM process for each server in the managed domain. The JVM settings for each individual server can be configured at three different levels:

- host controller level:** The `<jvm>` section within `host.xml`
- server group level:** The `<server-group>` section in `domain.xml`
- server level:** The `<server>` section within `host.xml`



Note

Similar to the `$JBOSS_HOME/bin/standalone.conf` file for a standalone server, there is a `domain.conf` in the same folder. The JVM memory settings in `domain.conf` refer to the memory given to the Java process that is running the Host Controller, and **NOT** the individual EAP servers within that host.

A JVM memory setting that appears at a lower level (closer to the server level) overrides any inherited settings from a higher level. More specifically:

- A JVM memory setting in the server group level of `domain.xml` overrides any settings inherited from the corresponding `<jvm>` definition in `host.xml`.
- A JVM memory setting in the server level of `host.xml` overrides any settings inherited from the `<jvm>` section and also the server group level.

Defining JVMs

JVMs can be defined and configured at all three levels using the EAP management console and the EAP CLI.

A JVM can be defined at the host controller level in the `host.xml` file within the `<jvms>` tag. A sample JVM definition in the `host.xml` looks like:

```
<jvms>
  <jvm name="small_jvm">
    <heap size="64m" max-size="128m"/>
  </jvm>
  <jvm name="production_jvm">
    <heap size="2048m" max-size="2048m"/>
    <jvm-options>
```

```

        <option value="-server"/>
    </jvm-options>
</jvm>
</jvms>

```

The corresponding CLI command to define the JVMs is as follows:

```
/host=servera/jvm=small_jvm:add(heap-size=64m,max-heap-size=128m)
```

```

/host=servera/jvm=production_jvm:add(
    heap-size=2048m,
    max-heap-size=2048m,
    jvm-options=["-server"]
)

```

In the above definition, the JVM named **small_jvm** will have a minimum heap size of 64 MB and a maximum heap size of 128 MB. The JVM named **production_jvm** will have minimum and maximum heap size of 2 GB and an option named **-server** is set for the JVM. You can pass multiple JVM options in this manner.



Note

After you have defined a JVM, it can be now be referred to in both **server-group** and **server** definitions. The specific settings of a named JVM can be overridden by the **server-group** and **server** definitions.

JVM settings in server groups

A **server-group** definition in the **domain.xml** file uses the **<jvm>** tag to specify which **<jvm>** definition to use. For example, the following **server-group** definition (in **domain.xml**) uses a JVM called **production_jvm**:

```

<server-group name="groupA" profile="default">
    <jvm name="production_jvm"/>
    <socket-binding-group ref="standard-sockets"/>
</server-group>

```

The corresponding CLI command is as follows:

```
/server-group=groupA:add \
    (profile=default,socket-binding-group=standard-sockets) \
    /server-group=groupA/jvm=production_jvm:add()
```

Any server in the **groupA** server group will inherit the JVM settings from the **production_jvm**. A **server-group** can override the settings of the **<jvm>** definition. For example, the following **server-group** definition changes the heap size of the **production_jvm**:

```
<server-group name="groupB" profile="default">
    <jvm name="production_jvm">
        <heap size="1024m" max-size="1024m"/>
    </jvm>
    <socket-binding-group ref="standard-sockets"/>
</server-group>
```

Any server in the **groupB** server group will have a min and max heap size of 1024MB, and its other JVM settings are inherited from the **production_jvm** definition.



Note

A new JVM configuration can be defined at the **server group** level without inheriting any settings from a JVM defined at the host controller level. This new JVM can be referred to at the server level and can be overridden. For example, a new JVM called **groupA-jvm** can be defined at the server-group level:

```
/server-group=groupA/jvm=groupA-jvm:add \
(heap-size=512m,max-heap-size=512mm,jvm-options=["-server"])
```

JVM settings in servers

The JVM settings can be configured at the server level within the `<server>` definition in the **host.xml** file. A new `<jvm>` can be defined , and it can also override any JVM settings that the server inherits from either its server group definition or its jvm definition at the host controller level. A sample definition at the server level is as follows:

```
<server name="test_server" group="groupB" auto-start="true">
    <socket-binding-group ref="standard-sockets" port-offset="300"/>
    <jvm name="production_jvm">
        <heap size="256m"/>
    </jvm>
</server>
```

In the previous example, **test_server** will use the settings from **production_jvm**, except its initial heap size will be 256MB instead of the 1024MB defined in the **groupB** server group (which overrides the 2048MB heap size from the **production_jvm** definition at the host controller level).

**Note**

You can also define a new JVM configuration at the **server** level without inheriting any settings from a JVM defined at the host controller level or the **server group** level. For example, a new JVM called **serverX-jvm** can be defined at the **server** level:

```
/host=host3/server-config=serverX \
/jvm=serverX-jvm:add \
(heap-size=512m,max-heap-size=1500m, \
jvm-options=["-server","-XX:ParallelGCThreads=4"])
```

► Guided Exercise

Configuring Java Virtual Machines

In this lab, you will tune the Java Virtual Machine settings of a standalone server as well as a managed domain.

Resources	
Files:	/home/student/JB248/labs/standalone /home/student/JB248/labs/domain/machine1/ domain/ /home/student/JB248/labs/host/machine2/ domain/ /home/student/JB248/labs/host/machine3/ domain/
Application URL:	NA
Resources	NA

Outcomes

You should be able to configure the JVM settings for a standalone EAP server as well as a managed domain.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, and that the previous guided exercise has been completed:

```
[student@workstation ~]$ lab jvm-settings setup
```

► 1. Configuring JVM settings for a Standalone server:

For a standalone server, the JVM settings are configured by manually editing a configuration file called **JBOSS_HOME/bin/standalone.conf** before starting the EAP server.

1. Edit the file **/opt/jboss-eap-7.0/bin/standalone.conf** using a text editor of your choice. Note that you have edit this file as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss vi /opt/jboss-eap-7.0/bin/standalone.conf
```

2. Observe the JVM settings being configured in the **\$JAVA_OPTS** variable (around line number 50). The JVM min heap size (**-Xms**) and the max heap size (**-Xmx**) are

both set to 1303m (1.3 GB) by default. These default values may not be enough for larger applications and may cause Out Of Memory (OOM) errors in the JVM due to excessive memory consumption by the application.

It is recommended to set the minimum and maximum heap sizes based on the object allocation characteristics of the application. You should also tune the garbage collection (GC) settings after profiling the application under production work loads.

- 1.3. Edit the min and max JVM heap size and increase it to 1500m (1.5 GB) as shown below:

```
JAVA_OPTS="-Xms1500m -Xmx1500m -Djava.net.preferIPv4Stack=true"
```

- 1.4. **Start the Standalone EAP Server.**

Use the following command to start an EAP instance using the **/home/student/JB248/labs/standalone** folder as the base directory:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

- 1.5. Observe the console window where EAP is starting and verify that the new JVM settings are used by EAP:

```
JBoss Bootstrap Environment  
  
JBoss_HOME: /opt/jboss-eap-7.0  
  
JAVA: java  
  
JAVA_OPTS: -server -verbose:gc -Xloggc:"/home/student/JB248/labs/standalone/log/gc.log" -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=3M -XX:-TraceClassUnloading -Xms1500m -Xmx1500m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true
```

- 1.6. You can also verify that the new settings are in effect by running the following command to view the full arguments to the **java** process representing the EAP Standalone server:

```
[student@workstation bin]$ ps -aef | grep java  
/usr/bin/java -D[Standalone] -server -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps ...  
-XX:-TraceClassUnloading -Xms1500m -Xmx1500m -Djava.net.preferIPv4Stack=true  
...
```

- 1.7. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the Standalone server JVM configuration. In the next steps, you will configure the JVM settings for a Managed Domain.

► 2. Configuring JVM settings for a Managed Domain:

The JVM settings for each server running on a managed domain can be configured using the EAP CLI or the management console at three levels:

- host controller level (the `<jvm>` section in the `host.xml`) file.
- server group level (the `<server-group>` section in the `domain.xml`) file.
- server level (the `<server>` section in the `host.xml`) file.

Recall from the lecture that the components in the lower levels can override the settings inherited from the parent levels. You will now start the managed domain and override the default JVM settings at both the **Server Group** level and the **Server** level.

- 2.1. Start the Domain Controller (machine1) and the two Host Controllers (machine2 and machine3)

Start the domain controller using a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/domain/machine1/domain/ \  
--host-config=host-master.xml
```

- 2.2. Start the host controller on **machine2**.

Run the following command from your `/opt/jboss-eap-7.0/bin` folder in a new terminal window on your workstation to start **machine2**:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine2/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

- 2.3. Start **machine3** to join the managed domain.

Run the following command from your `/opt/jboss-eap-7.0/bin` folder in a new terminal window on your workstation to start **machine3**:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./domain.sh \  
-Djboss.domain.base.dir=/home/student/JB248/labs/host/machine3/domain/ \  
--host-config=host-slave.xml \  
-Djboss.domain.master.address=172.25.250.254
```

- 2.4. Observe that all four servers in the managed domain have started with a min heap size of 64 MB and a max heap size of 256 MB. These values are the default JVM settings for all server groups and servers in the managed domain, unless they explicitly refer to the **default** JVM configuration in the `host-slave.xml` file.
(Hint: use the `ps -aef` command and grep for the appropriate server name.)

```
[student@workstation bin]$ ps -aef | grep server-one
java -D[Server:server-one] -Xms64m -Xmx256m -Djava.awt.headless=true
...
```

2.5. **Override the JVM settings for the server group Group1 and set the min and max JVM heap size value to 1024m (1 GB).**

To configure the JVM settings at the Server Group level in a Managed Domain, you can use the EAP management console:

Navigate to `http://172.25.250.254:9990/`, which is the location of the management console. You should be prompted to login. Log in as the **jbossadm** user with password **JBoss@RedHat123**.

- 2.6. Navigate to **Runtime** → **Server Groups** → **Group1** in the management console. Click the **View** drop down and select the **Stop** option to stop all servers in **Group1**.
- 2.7. When all the servers in **Group1** have stopped, click the **View** next to **Group1** to view the **Group1** configuration page. Click the **JVM Configuration** tab and click the **Edit** link to edit the JVM settings for **Group1**.
- 2.8. Configure the JVM settings as per details below:
 - **Name:** dev-group-jvm
 - **Heap Size:** 1024m
 - **Max Heap Size:** 1024m
 - **Permgen Size:** Leave Blank (Not relevant for JDK 8)
 - **Max Permgen Size:** Leave Blank (Not relevant for JDK 8)
 - **JVM Options:** -server

Click **Save** to save your settings.

- 2.9. Open the domain configuration file at **/home/student/JB248/labs/domain/machine1/domain/configuration/domain.xml** with an editor of your choice and verify that the server group **Group1** now references the **dev-group-jvm** you added above.

```
<server-group name="Group1" profile="full-ha">
  <jvm name="dev-group-jvm">
    <heap size="1024m" max-size="1024m"/>
    <jvm-options>
      <option value="-server"/>
    </jvm-options>
  </jvm>
  ...
</server-group>
```

- 2.10. Start the servers in **Group1** using the management console to verify that the JVM changes have taken effect. Navigate to **Runtime → Server Groups → Group1** in the management console. Click the **View** drop down and select the **Start** option to start all servers in **Group1**.

Observe that the two servers in **Group1 (server-one and server-three)** have started with a minimum heap size of 1024 MB and a maximum heap size of 1024 MB and the other JVM options that you provided in the previous step:

```
[student@workstation bin]$ ps -afe | grep server-one
java -D[Server:server-one] -Xms1024m -Xmx1024m -server -Djava.awt.headless=true
...
```

The servers in **Group2 (server-two and server-four)** are still running with the default JVM options because you did not override the settings for **Group2**.

- 2.11. You can also configure the JVM settings at the individual server level, which will override the JVM settings inherited from the Host level or the Server Group level.

You will now set the minimum and maximum JVM heap size of the **server-three** server to 1200m (1.2 GB) and set the **AggressiveOpts** flag to enable the JVM to perform some extra optimization on the code at runtime (this flag is not set by default). You can use the EAP CLI to achieve this.

Launch the EAP CLI in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh --connect --  
controller=172.25.250.254:9990
```

Change the JVM settings of the **server-three** server using the CLI command:

```
[[domain@172.25.250.254:9990 /] /host=host3/server-config=server-three\  
/jvm=server-three-jvm:add\  
(heap-size=1200m,max-heap-size=1200m,jvm-options=["-XX:+AggressiveOpts"])  
{  
    "outcome" => "success",  
    ...  
    "response-headers" => {  
        "operation-requires-restart" => true,  
        "process-state" => "restart-required"  
    }  
}
```

Because you changed the JVM settings for **server-three**, it requires a restart of the server:

```
[[domain@172.25.250.254:9990 /] /host=host3\  
/server-config=server-three:restart(blocking=true)  
{  
    "outcome" => "success",  
    "result" => "STARTED"  
}
```

- 2.12. Open the host configuration file for **host3** at **/home/student/JB248/labs/host/machine3/domain/configuration/host-slave.xml** with an editor of your choice and verify that the server **server-three** now defines a jvm section called **server-three-jvm** as per below.

```
<jvm name="server-three-jvm">  
    <heap size="1200m" max-size="1200m"/>  
    <jvm-options>  
        <option value="-XX:+AggressiveOpts"/>  
    </jvm-options>  
</jvm>
```

Also, observe that only **server-three** has started with the new JVM settings, while **server-one** is still running with the JVM settings inherited from the Server Group level setting for **Group1**:

```
[student@workstation bin]$ ps -aef | grep server-three
java -D[Server:server-three] -Xms1200m -Xmx1200m -XX:+AggressiveOpts -
Djava.awt.headless=true
...
```

```
[student@workstation bin]$ ps -aef | grep server-one
java -D[Server:server-one] -Xms1024m -Xmx1024m -server -Djava.awt.headless=true
...
```

► 3. Clean Up

- 3.1. Exit the EAP CLI:

```
[standalone@localhost:9990 /] exit
```

- 3.2. Stop the managed domain by pressing **Ctrl+C** in all the three terminal windows where you started EAP (machine1, machine2 and machine3), or use the EAP CLI to shut down the controllers as demonstrated in the earlier labs.

This concludes the guided exercise.

▶ Lab

Configuring the Java Virtual Machine

In this lab, you will configure the Java Virtual Machine (JVM) settings for an EAP managed domain.

Resources	
Files	/opt/domain
Application URL	NA

Outcome

You should be able to configure the Java Virtual Machine settings used by servers running in a managed domain.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab jvm-lab-final setup
```

You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively, and the domain controller on the workstation. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and configure the Java Virtual Machine settings in the managed domain.

- Start the domain controller on the **workstation** VM. Since the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**). Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.
- The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.
 - Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up

script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

- 2.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

- 2.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
3. You have been instructed by the development team to configure the JVM settings for the managed domain with the following specifications:
 - The min and max heap size of the **default** JVM on the host controller **servera** should be set to 512 MB.
 - For all the servers in the server group **Group1**, create a new JVM configuration named **group1-jvm**. Set the minimum JVM heap size to 512m (512 MB) and the maximum JVM heap size to 1024m (1 GB). Enable the **-server** flag for this server group.
 - For the server **serverb.2**, create a new JVM configuration named **serverb.2-jvm** that overrides any inherited settings. Set the minimum and maximum JVM heap size to 1024m (1 GB). Enable the **-XX:+AggressiveOpts** and **-server** flags for this server.
 - Leave all other JVM settings at default values.

Ensure that all the servers in the managed domain are started with these settings and without any errors. Verify the JVM flags with which the servers are running, using the **ps -afe** command on the respective hosts.



Note

When you start the managed domain at the beginning of the lab, all the servers are set to start automatically. Make sure you **STOP** all the servers before you make changes to the JVM settings in the managed domain.

- 3.1. Launch the EAP CLI or the management console to configure the JVM.
- 3.2. Stop the servers in the server group **Group1**.
- 3.3. Stop the servers in the server group **Group2**.
- 3.4. Configure the **default** JVM for the host controller **servera** to use the minimum amount of heap memory to 512 MB. The expected output is:Configure the maximum amount of heap memory to 512MB.The expected output is:

- 3.5. Configure the JVM minimum amount of memory for server group **Group1** to 512 MB and the maximum amount of memory to 1024 MB. Also pass the **-server** as a parameter to the JVM . The expected output is:
 - 3.6. Configure the JVM for server **serverb.2**. The minimum amount of memory for the server should be 1024 MB and the maximum amount of memory to 1024 MB. Also pass the **-XX:+AggressiveOpts** and **-server** as parameters to the JVM .
 - 3.7. Start the servers in **Group1**.
 - 3.8. Start the servers in **Group2**.
 - 3.9. Verify the start-up of servers with the configured JVM settings by using the **ps** command on each host.
4. Shut down the servers, server groups, host controllers, and the entire managed domain.
 - 4.1. Stop all servers in **Group1**.
 - 4.2. Stop the servers in **Group2**.
 - 4.3. Shut down the host controller on **servera**. Observe the console window of **servera** and verify that the host controller has been shutdown.
 - 4.4. Shut down the host controller on **serverb**. Observe the console window of **serverb** and verify that the host controller has been shutdown.
5. **Clean Up and Grading**
 - 5.1. Press **Ctrl+C** to stop the domain controller (Alternatively, you can shutdown the domain controller using the JBoss EAP CLI command **/host=master:shutdown()**).
 - 5.2. Press **Ctrl+C** to exit the EAP CLI if you used the CLI in the lab. (Alternatively, you can exit the CLI by typing **exit**.)
 - 5.3. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab jvm-lab-final grade
```

This concludes the lab.

► Solution

Configuring the Java Virtual Machine

In this lab, you will configure the Java Virtual Machine (JVM) settings for an EAP managed domain.

Resources	
Files	/opt/domain
Application URL	NA

Outcome

You should be able to configure the Java Virtual Machine settings used by servers running in a managed domain.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab jvm-lab-final setup
```

You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively, and the domain controller on the workstation. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and configure the Java Virtual Machine settings in the managed domain.

1. Start the domain controller on the **workstation** VM. Since the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ --host-config=host-master.xml
```

2. The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

- 2.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.

Open a new terminal window on the **servera** VM and run the following command:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 2.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.

Open a new terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 2.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
3. You have been instructed by the development team to configure the JVM settings for the managed domain with the following specifications:
- The min and max heap size of the **default** JVM on the host controller **servera** should be set to 512 MB.
 - For all the servers in the server group **Group1**, create a new JVM configuration named **group1-jvm**. Set the minimum JVM heap size to 512m (512 MB) and the maximum JVM heap size to 1024m (1 GB). Enable the **-server** flag for this server group.
 - For the server **serverb.2**, create a new JVM configuration named **serverb.2-jvm** that overrides any inherited settings. Set the minimum and maximum JVM heap size to 1024m (1 GB). Enable the **-XX:+AggressiveOpts** and **-server** flags for this server.
 - Leave all other JVM settings at default values.

Ensure that all the servers in the managed domain are started with these settings and without any errors. Verify the JVM flags with which the servers are running, using the **ps -aef** command on the respective hosts.



Note

When you start the managed domain at the beginning of the lab, all the servers are set to start automatically. Make sure you **STOP** all the servers before you make changes to the JVM settings in the managed domain.

- 3.1. Launch the EAP CLI or the management console to configure the JVM.

In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss\
/opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```

- 3.2. Stop the servers in the server group **Group1**.

```
[domain@workstation:9990 /] /server-group=Group1:\
stop-servers(blocking=true)
```

The expected output is:

```
{
  "outcome" => "success",
  ...
}
```

- 3.3. Stop the servers in the server group **Group2**.

```
[domain@workstation:9990 /] /server-group=Group2:\
stop-servers(blocking=true)
```

The expected output is:

```
{
  "outcome" => "success",
  ...
}
```

- 3.4. Configure the **default** JVM for the host controller **servera** to use the minimum amount of heap memory to 512 MB.

```
[domain@workstation:9990 /] /host=servera\
/jvm=default:write-attribute(name=heap-size,value=512m)
```

The expected output is:

```
{
  "outcome" => "success",
  ...
}
```

Configure the maximum amount of heap memory to 512MB.

```
[domain@workstation:9990 /] /host=servera\
/jvm=default:write-attribute(name=max-heap-size,value=512m)
```

The expected output is:

```
{
  "outcome" => "success",
  ...
}
```

- 3.5. Configure the JVM minimum amount of memory for server group **Group1** to 512 MB and the maximum amount of memory to 1024 MB. Also pass the **-server** as a parameter to the JVM .

```
[domain@workstation:9990 /] /server-group=Group1\
/jvm=group1-jvm:add\
(heap-size=512m,max-heap-size=1024m,jvm-options=["-server"])
```

The expected output is:

```
{
  "outcome" => "success",
  ...
}
```

- 3.6. Configure the JVM for server **serverb.2**. The minimum amount of memory for the server should be 1024 MB and the maximum amount of memory to 1024 MB. Also pass the **-XX:+AggressiveOpts** and **-server** as parameters to the JVM .

```
[[domain@172.25.250.254:9990 /] /host=serverb\
/server-config=serverb.2\
/jvm=serverb.2-jvm:add\
(heap-size=1024m,max-heap-size=1024m,\
jvm-options=["-XX:+AggressiveOpts","-server"])
{
  "outcome" => "success",
  ...
}
```

- 3.7. Start the servers in **Group1**.

```
[domain@workstation:9990 /] /server-group=Group1\  
:start-servers(blocking=true)
```

The expected output is:

```
{  
    "outcome" => "success",  
    ...  
}
```

3.8. Start the servers in **Group2**.

```
[domain@workstation:9990 /] /server-group=Group2\  
:start-servers(blocking=true)
```

The expected output is:

```
{  
    "outcome" => "success",  
    ...  
}
```

3.9. Verify the start-up of servers with the configured JVM settings by using the **ps** command on each host.

```
[domain@workstation:9990 /] ps -aef | grep <name of server>
```

4. Shut down the servers, server groups, host controllers, and the entire managed domain.

4.1. Stop all servers in **Group1**.

```
[domain@172.25.250.254:9990 /] /server-group=Group1:stop-servers(blocking=true)
```

The expected output is:

```
{  
    "outcome" => "success",  
    ...  
}
```

4.2. Stop the servers in **Group2**.

```
[domain@172.25.250.254:9990 /] /server-group=Group2:stop-servers(blocking=true)
```

The expected output is:

```
{  
    "outcome" => "success",  
    ...  
}
```

- 4.3. Shut down the host controller on **servera**. Observe the console window of **servera** and verify that the host controller has been shutdown.

```
[domain@172.25.250.254:9990 /] /host=servera:shutdown()
```

The expected output is:

```
{  
    "outcome" => "success",  
    ...  
}
```

- 4.4. Shut down the host controller on **serverb**. Observe the console window of **serverb** and verify that the host controller has been shutdown.

```
[domain@172.25.250.254:9990 /] /host=serverb:shutdown()  
{  
    "outcome" => "success",  
    ...  
}
```

5. Clean Up and Grading

- 5.1. Press **Ctrl+C** to stop the domain controller (Alternatively, you can shutdown the domain controller using the JBoss EAP CLI command **/host=master:shutdown()**).
- 5.2. Press **Ctrl+C** to exit the EAP CLI if you used the CLI in the lab. (Alternatively, you can exit the CLI by typing **exit**.)
- 5.3. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab jvm-lab-final grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- The EAP server as well as the applications deployed on EAP execute within a Java Virtual Machine (JVM). The JVM can be configured for optimal performance by configuring a number of command-line parameters that are used to start the JVM process.
- JVM settings for a standalone server and a managed domain are configured in different ways. The JVM settings for an EAP standalone server is configured in the **\$JBOSS_HOME/bin/standalone.conf** file.
- The JVM settings for an EAP managed domain can be configured at three different levels:
 - host controller level
 - server group level
 - server level
- JVM settings at the lower levels override the settings inherited from the higher levels. **servers** can override the settings inherited from **server groups** and **host controllers**, and **server groups** can override settings inherited from the **host controllers**.

Chapter 11

Configuring the Web Subsystem

Overview

Goal Configure connectors, servers, and other features of the web subsystem.

- Objectives**
- Describe the features of the Undertow web subsystem.
 - Configure the components of the web subsystem.
 - Configure JBoss EAP network interfaces and socket binding groups.

- Sections**
- Exploring the Features of the Web Subsystem (and Quiz)
 - Configuring the Web Subsystem (and Guided Exercise)

- Lab**
- Configuring the Web Subsystem

Exploring the Features of the Web Subsystem

Objectives

After completing this section, a system administrator should be able to:

- Describe the features of the Undertow web subsystem.

Exploring the features

The JBoss Web Server (JWS) was the default web subsystem in JBoss EAP 6. This web subsystem is based on a fork of Apache Tomcat, acting as an embedded Java EE web container.

In EAP 7, the default web subsystem was replaced with the Undertow project. Similarly to JWS, Undertow can act both as a web server and a Java EE web container. Undertow was developed because Java EE 7 specification requirements were not supported by JWS, such as:

- **Web sockets:** Provides a full-duplex communication between two peers over TCP and it is a new technology defined by W3C.
- **JSON-P:** Supports cross-domain requests avoiding the restrictions imposed by browsers.
- **HTTP/2:** Improves the performance and the payload compared to the previous HTTP protocol.

Undertow is written in Java and implements multiple improvements such as:

- **NIO web container:** Undertow is faster and supports non-blocking I/O support that is provided by the XNIO API, a framework based on the Java New I/O(NIO) API.
- **Load balancing:** The Apache HTTPD server is no longer required because now it is possible to distribute the requests among other JBoss EAP instances with the mod_cluster technology.
- **Port reduction:** Supports HTTP upgrade, which reduces the number of ports used by EAP to two. Port 990 is used for administration and port 8080 is used for applications.
- **Root web location:** Defines the default web application that should be displayed when the server receives a request.
- **Session management:** Customizes the session management for HTTP sessions.

Within the Undertow subsystem, there are five main components to configure:

1. Buffer caches
2. Server
3. Servlet container
4. Handlers
5. Filters

Buffer caches

Buffer caches are responsible for caching static content. A buffer cache consists of multiple caches (regions), and each region is split into smaller buffers. The total amount of space used can

be calculated by multiplying the buffer size by the number of buffers per region by the maximum number of regions. The buffer is defined in bytes, and the default size of a buffer cache is **10 MB**.

Server

Servers represent an instance of Undertow. It is possible to run multiple Undertow instances in a single EAP 7 instance, however, it is recommended to start a new EAP instance instead of creating a new Undertow instance because it is easy to do and not a resource-heavy operation resource-wise.

A listener handles a request according to the request protocol. Three listener types are available for each server:

- **HTTP**: This listener type are responsible for handling HTTP requests.
- **HTTPS**: This listener type are responsible for handling HTTPS requests.
- **AJP**: This listener type are responsible for handling AJP requests. The AJP protocol is responsible for communication between the EAP instance and the load balancer.

By default, the **default** and **full** profiles only define the HTTP listener, and the **ha** and **full-ha** profiles define the HTTP and AJP listeners.

Servlet container

It is possible to configure the servlets, JSP, and web-socket technologies using the servlet container. An EAP 7 instance can have multiple servlet containers, however most servers will only need a single servlet container.

Handlers

A handler adds functionality to the container. It is created using a Java class and can be used for any purpose such as security, error page handling, metrics, or virtual host support. The handlers in Undertow are chained together.

Undertow defines either synchronous or asynchronous handlers. Asynchronous handlers will provide an improved response time because it will provide asynchronous responses, thus releasing the servlet for other requests sooner.

Filters

A filter is functionally equivalent to a global valve used in previous versions of JBoss EAP. A filter filters the request allowing you to modify its contents. Some common use cases for filters include setting headers or doing GZIP compression.

The following types of filters can be defined:

- custom-filter
- error-page
- expression-filter
- gzip
- mod-cluster
- request-limit

- response-header
- rewrite



References

For more information, see the

JBoss EAP configuration guide - Configuring the Web Server (Undertow)

https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/version-7.0/configuration-guide/#configuring_the_web_server_undertow

► Quiz

Features of Web Subsystem

Match the items below to their counterparts in the table.

Asynchronous handlers

Buffer Cache

Filter

HTTP Upgrade

Handlers

JSON-P

Listeners

Server

Servlet Container

Web sockets

Component	Description
Allows you to modify a request to the server.	
Responsible for caching static content.	
Represents an instance of Undertow.	
Adds functionality to a container.	
Responsible for enabling protocols such as HTTP/HTTPS/AJP.	
Supports full-duplex communication over TCP.	
Improves response time because it releases the servlet faster.	
Allows you to configure the web technologies like JSP, servlets and websockets.	
Adds support to cross-domain requests.	
Reduces the number of ports required to access EAP 7.	

► Solution

Features of Web Subsystem

Match the items below to their counterparts in the table.

Component	Description
Allows you to modify a request to the server.	Filter
Responsible for caching static content.	Buffer Cache
Represents an instance of Undertow.	Server
Adds functionality to a container.	Handlers
Responsible for enabling protocols such as HTTP/HTTPS/AJP.	Listeners
Supports full-duplex communication over TCP.	Web sockets
Improves response time because it releases the servlet faster.	Asynchronous handlers
Allows you to configure the web technologies like JSP, servlets and websockets.	Servlet Container
Adds support to cross-domain requests.	JSON-P
Reduces the number of ports required to access EAP 7.	HTTP Upgrade

Configuring the Web Subsystem

Objectives

After completing this section, a system administrator should be able to:

- Configure the components of the web subsystem.

Configuring the root web application

Any application deployed on EAP will be published on a standard address. For example, an application called app.war will be available, by default, at `http://<server>:8080/app`. The `/app` is defined as context path according to Java EE. If no context path is provided by a HTTP request, EAP will display a **welcome** application that is deployed in the Undertow subsystem.

The Undertow subsystem is responsible for serving the JBoss welcome application.

```
<server name="default-server">
  <http-listener name="default" socket-binding="http" redirect-socket="https" />
  <host name="default-host" alias="localhost">
    <location name="/" handler="welcome-content" ① />
    <filter-ref name="server-header" />
    <filter-ref name="x-powered-by-header" />
  </host>
</server>
... More Undertow configuration ...
<handlers>
  <file name="welcome-content" path="${jboss.home.dir}/welcome-content" />
</handlers>
```

- ① The location tag defines the default application that should be displayed.

Notice that the root path (`/`) is using a handler named **welcome-content** and this is a file handler that defines the location of the JBoss welcome application.

The default application can be replaced in one of two ways:

- Changing the **welcome-content** file handler.
- Changing the **default-web-module** attribute on the **default-host**.

Changing the welcome-content file handler

The first way to replace the default application is to change the file handler named **welcome-content** to have a new path that contains the application.

It is possible to replace the **welcome-content** file handler using the CLI tool:

```
/subsystem=undertow/configuration=handler/file=welcome-content:\n  write-attribute(name=path,value="/path/to/application" ①)
```

- ① Defines the directory where a Java EE application is stored. The application should be available in the file system on all hosts that belong to the domain and for this reason, this method is not recommended.

**Note**

A reload is required for the changes to take effect.

```
/:reload-servers
```

To achieve the same goal using the managed domain, just add the **/profile=full-ha** level at the beginning of the CLI command.

Changing the default-web-module attribute

The second way uses an application that is deployed onto the standalone server or onto the managed domain.

```
/subsystem=undertow/server=default-server①/host=default-host②:\n  write-attribute(name=default-web-module③,value=myapp.war ④)
```

- ① EAP 7 provides a default server named **default-server**. Servers are discussed later in this chapter.
- ② EAP 7 also provides a default host named **default-host**. Hosts are discussed later in this chapter.
- ③ The **default-web-module** attribute is responsible for defining an application that will be displayed if a context path is not specified.
- ④ Define which deployed application should be displayed if a context path is not specified.

Disabling the default web application

In some scenarios, it is required to remove the JBoss welcome application, but without having another application as the default root web application. This can be accomplished by removing the location entry (/) for the **default-host**:

```
/subsystem=undertow/server=default-server/host=\\n  default-host/location=/:remove
```

After reloading the server, any request using the root context will receive a 404 error page.

Demonstration: Setting the root web location

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command:

```
[student@workstation ~]$ demo\nroot-web setup
```

The previous command will:

- Verify that EAP is installed.

- Verify that the guided exercise **Configuring JDBC Drivers** was executed.
 - Verify that EAP is not running.
 - Download the required files for this demonstration.
2. Start the EAP instance using the base directory located at **/home/student/JB248/labs/standalone/**:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

3. Open a new terminal window and run the following commands to start the CLI tool connection to your EAP instance:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh -c
```

**Note**

The credentials are not required because the security subsystem was not configured to request the credentials for local connections.

4. Deploy the *welcome.war* application that is available in the **/home/student/JB248/labs/root-web** folder:

```
[standalone@localhost:9990 /] deploy \  
/home/student/JB248/labs/root-web/welcome.war
```

5. Open the web browser on the workstation and navigate to `http://localhost:8080/welcome` to test if the application was deployed successfully. You should see the welcome application:

Welcome to EAP 7!

This is a simple web app that is deployed to the root context of this server

Figure 11.1: Custom welcome page

6. Notice that it is required to pass the **welcome** context path after the port to access the application. If the context path is not specified, the root application will be displayed. By default, the root application is the JBoss welcome application. Access the JBoss welcome page navigating to the `http://localhost:8080` URL:

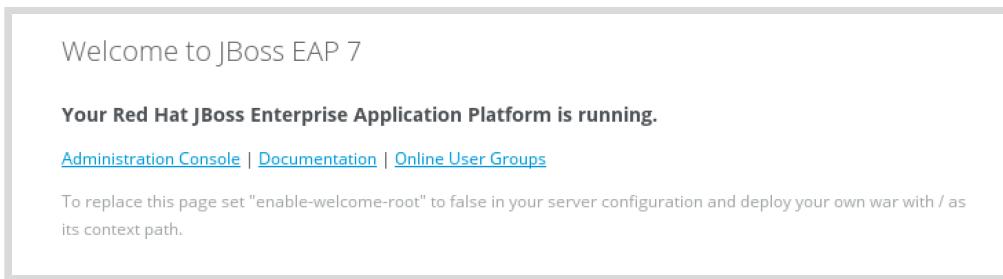


Figure 11.2: Default welcome page provided by EAP

7. It is possible to remove the default root application. To do so, return to the terminal window that has the CLI tool running and execute the following command:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/host=default-host/location=/:remove
```

The previous command will remove the / location from the **default-host**. The / location is the root application.

8. A server reload is required to determine if the root application was removed. Refresh the EAP instance configuration with the **reload** command:

```
[standalone@localhost:9990 /] reload
```

9. Open the web browser and navigate to the root application using the `http://localhost:8080` URL. You should see a **404 - Not Found** error message.
10. It is possible to define the **welcome.war** application as the root context. The Undertow subsystem must have a different web application running as the root web application, also known as the **default-web-module** for the **default-server**. In this step you will update it using the CLI.

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/host=default-host:\  
write-attribute(name=default-web-module,value=welcome.war)
```

11. To refresh the configuration with the new root application, a reload is required.

```
[standalone@localhost:9990 /] reload
```

12. Open a web browser and navigate to the root application using the `http://localhost:8080` URL. You should see the welcome application now running without the context path.

13. Exit the CLI:

```
[standalone@localhost:9990 /] exit
```

14. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the demonstration.

Configure server with listeners

The Undertow subsystem may start several web servers or web containers in a single EAP instance.

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
  ...
  <server name="default-server">
    <http-listener name="default" redirect-socket="https" socket-binding="http" />
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content" />
      <filter-ref name="server-header" />
      <filter-ref name="x-powered-by-header" />
    </host>
  </server>
  ...
</subsystem>
```

Each **server** tag starts an instance of Undertow. However, it is recommended to start a new EAP instance instead of creating a new Undertow instance because it is easy to do and not a resource intensive.

A listener handles a request according to the request protocol. Three listener types are available for each server:

- **HTTP**: This listener type is responsible for handling HTTP requests.
- **HTTPS**: This listener type is responsible for handling HTTPS requests.
- **AJP**: This listener type is responsible for handling AJP requests. The AJP protocol is responsible for communication between the EAP instance and the load balancer.

By default, the **default** and **full** profiles only define the HTTP listener and the **ha** and **full-ha** profiles define the HTTP and the AJP listeners.

Each server can define more than one listener for each listener type:

```
/subsystem=undertow/server=default-server/http-listener=\
new-http:add(socket-binding=http-n)
```

Observe the new **XML** configuration:

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
  ...
  <server name="default-server">
    <http-listener name="default" redirect-socket="https" socket-binding="http" />
    <http-listener name="new-http" socket-binding="http-n" />
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content" />
      <filter-ref name="server-header" />
      <filter-ref name="x-powered-by-header" />
    </host>
  </server>
```

```
</server>
...
</subsystem>
```

A different **socket-binding** should be defined to avoid a port conflict. In the previous configuration, the web application will be available for HTTP requests on two different sockets:**http** or **http-n**.

The host configuration within the server element is responsible for creating virtual hosts. Virtual hosts groups web applications based on the DNS names. Each server can configure a set of hosts:

```
/subsystem=undertow/server=default-server/host=\
version-host:add(alias=[version.myapp.com])
```

A default web application can be published on a host listening to multiple addresses:

```
/subsystem=undertow/server=default-server/host=\
version-host:write-attribute(name=default-web-module, value=version.war)
```

In the previous example, requests to `http://version.myapp.com` will by default serve the `version.war` application. The following XML will reflect the previous modification:

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">
  ...
  <server name="default-server">
    <http-listener name="default" redirect-socket="https" socket-binding="http" />
    <http-listener name="new-http" socket-binding="http-n" />
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content" />
      <filter-ref name="server-header" />
      <filter-ref name="x-powered-by-header" />
    </host>
    <host name="version-host" default-web-module="version.war"
          alias="version.myapp.com"/>
  </server>
  ...
</subsystem>
```



Note

A **jboss-web.xml** file should be created under the **WEB-INF** folder in the `version.war` application with the following code:

```
<jboss-web>
  <context-root>/</context-root>
  <virtual-host>version-host</virtual-host>
  <server-instance>default-server</server-instance>
</jboss-web>
```

If the new host is no longer required, delete it with:

```
/subsystem=undertow/server=default-server/host=\
version-host:remove()
```

Configure SSL connections

HTTPS is a protocol that encrypts the requests and the responses over the network. This protocol protects sensitive information that is exchanged between the web browser and the server.

To configure this protocol, a certificate file is required to encrypt and decrypt the exchanged information. A keystore should be created to store a certificate safely. The Java Development Kit (JDK) provides the **keytool** command that can create a keystore and also create a self-signed certificate if a trusted CA certificate is not provided.

To create a new keystore within a self-signed certificate, use the following command:

```
# keytool -genkeypair -alias appserver -storetype jks -keyalg RSA -keysize 2048 -\
keystore /path/identity.jks
```

The following parameters were specified:

- **genkeypair**: Responsible for defining that a new self-signed certificate should be created.
- **alias**: Defines the certificate name inside the keystore.
- **storetype**: Defines the type of the keystore.
- **keyalg**: Specifies the algorithm to be used to generate the certificate.
- **keysize**: Specifies the size of each key to be generated.
- **keystore**: Defines where the keystore should be saved.

The previous command requests two passwords. The first is the password to open the **identity** keystore and the second is to access the **appserver** certificate. Use the same value for both passwords to enable the HTTPS protocol in EAP 7.

Each host in the managed domain should have the keystore on its file system.

After creating the keystore, a new security realm should be created for each host:

```
/host=servera/core-service=management/security-realm=HTTPSRealm:add()
```

After adding the security realm, it should be configured for loading the created keystore:

```
/host=servera/core-service=management/security-realm=HTTPSRealm/server-identity=\
ssl:add(keystore-path=/path/identity.jks,keystore-\
password=changeit,alias=appserver)
```

A reload is required for the changes to take effect.

```
/:reload-servers
```

To enable the HTTPS protocol, a new HTTPS listener must be created on the default server. This listener must use the created security realm:

```
/profile=full/subsystem=undertow/server=default-server/https-listener=https:\n    add(socket-binding=https, security-realm=HTTPSSRealm)
```



Note

Remember to configure the firewall to open the HTTPS port. The default port for HTTPS is **8443**.

Other components

One of the Undertow components is the **buffer cache**.

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">\n    <buffer-cache name="default" buffer-size="1024" buffers-per-region="1024" max-\n    regions="10"\n    ...\n</subsystem>
```

It is possible to update the buffer size using the CLI:

```
/subsystem=undertow/buffer-cache=default:\n    write-attribute(name=buffer-size,value=2048)
```

If a cache is not required, it can be removed:

```
/subsystem=undertow/buffer-cache=default:remove
```

Servlet container

A servlet container instance is defined by the **servlet-container** tag:

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">\n    ...\n    <servlet-container name="default">\n        <jsp-config/>\n        <websockets/>\n    </servlet-container>\n    ...\n</subsystem>
```

To check the available attributes that can be configured for a servlet container, use the **read-resource** operation from the CLI:

```
/subsystem=undertow/servlet-container=default:read-resource
```

The following output is expected:

```
{
  "outcome" => "success",
  "result" => {
    "allow-non-standard-wrappers" => false,
    "default-buffer-cache" => "default",
    "default-encoding" => undefined,
    "default-session-timeout" => 30,
    "directory-listing" => undefined,
    "disable-caching-for-secured-pages" => true,
    "eager-filter-initialization" => false,
    "ignore-flush" => false,
    "max-sessions" => undefined,
    "proactive-authentication" => true,
    "session-id-length" => 30,
    "stack-trace-on-error" => "local-only",
    "use-listener-encoding" => false,
    "mime-mapping" => undefined,
    "setting" => {
      "jsp" => undefined,
      "websockets" => undefined
    },
    "welcome-file" => undefined
  }
}
```

Notice that it is possible to define the default buffer cache with the **default-buffer-cache** attribute.

The session can be managed in the servlet container. Notice that three properties are available for configuration:

- **default-session-timeout**: The default session timeout (in minutes) for all applications deployed in the container.
- **max-sessions**: The maximum number of sessions that can be active at one time.
- **session-id-length**: The number of characters of the generated session ID. Longer session IDs are more secure.

To set the default session timeout to one hour, use the following operation:

```
/subsystem=undertow/servlet-container=default:write-attribute(name=default-
session-timeout,value=60)
```

It is also possible to configure the JSP technology for the servlet container. To list the available attributes for the JSP configuration, use the following CLI command:

```
/subsystem=undertow/servlet-container=default/setting=jsp:read-resource
```

Handlers

EAP 7 provides two types of handlers:

- File handler

- Reverse proxy handler

File handlers serve static files. Each file handler must be attached to a location in a virtual host.

To create a new file handler, use the following CLI command:

```
/subsystem=undertow/configuration=handler/file=photos\  
:add(path=/var/photos, directory-listing=true)
```

Attach the new file handler to a location in a virtual host:

```
/subsystem=undertow/server=default-server/host=default-host/location=photos\  
:add(handler=photos)
```

This content will be served using the ip:port/context/photos.

The reverse proxy handler allows EAP to serve as a high performance reverse proxy. This topic is discussed later in this course.

Filters

By default, two filters are defined by JBoss EAP:

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">  
  ...  
  <filters>  
    <response-header name="server-header" header-value="JBoss-EAP/7" header-  
name="Server"/>  
    <response-header name="x-powered-by-header" header-value="undertow/1" header-  
name="X-Powered-By"/>  
  </filters>  
</subsystem>
```

These filters will set headers on the response. The first will define a header named **Server** with the **JBoss-EAP/7** value. The second will define a header named **X-Powered-By** with the **undertow/1** value.

These filters are applied to all requests on the **default-host** host:

```
<subsystem xmlns="urn:jboss:domain:undertow:3.0">  
  ...  
  <server name="default-server">  
    ...  
    <host name="default-host" alias="localhost">  
      <location name="/" handler="welcome-content" />  
      <filter-ref name="server-header" />  
      <filter-ref name="x-powered-by-header" />  
    </host>  
  </server>  
  ...  
</subsystem>
```

To include a new filter, use the following CLI command:

```
/subsystem=undertow/configuration=filter/gzip=gzip:add()
```

The I/O subsystem

The I/O subsystem is responsible for configuring the XNIO workers. A XNIO worker manages several kinds of threads.

The Undertow subsystem depends on the I/O subsystem. An Undertow listener defines a worker:

```
/subsystem=undertow/server=default-server/http-listener=\n  default:read-attribute(name=worker)
```

The following output is expected:

```
{\n    "outcome" => "success",\n    "result" => "default"\n}
```

The worker named **default** is defined in the I/O subsystem. Use the following command to list the attributes from the worker:

```
/subsystem=io/worker=default:read-resource
```

The following output is expected:

```
{\n    "outcome" => "success",\n    "result" => {\n        "io-threads" => undefined,\n        "stack-size" => 0L,\n        "task-keepalive" => 60,\n        "task-max-threads" => undefined\n    }\n}
```

XNIO worker instance is an abstraction layer for the Java NIO APIs. Notice that you can define the number of I/O threads to use by setting the **io-threads** attribute. The maximum number of threads can be defined for a task setting the **task-max-threads** attribute.

► Guided Exercise

Securing HTTP Connections

In this lab, you will configure a SSL connection to allow secure connections from a browser, increase the number of HTTP connections supported by EAP, and enable an AJP listener to support communication with a load balancer.

Resources	
Files:	/home/student/JB248/labs/standalone /home/student/JB248/labs/https
Application URL:	http://localhost:8080/version https://localhost:8443/version
Resources	/home/student/JB248/labs/server-listeners/version.war

Outcomes

You should be able to access EAP using a secure connection from a browser, enable AJP to allow a load balancer to distribute load with this server, and improve the responsiveness of EAP increasing the number of simultaneous HTTP connections.

Before You Begin

Before beginning the guided exercise, run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, to verify that the Configuring JDBC Drivers guided exercise was correctly executed, and to download the **version.war** application:

```
[student@workstation ~]$ lab server-listeners setup
```

► 1. Start the EAP Standalone Server

Use the following command to start an EAP instance so that you can access the management console:

```
[student@workstation ~]$ cd  
/opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone/
```

Wait for the server to finish starting before proceeding.

► 2. Deploy the version application

- 2.1. Open a new terminal window and run the following commands to start the CLI connection to your EAP instance:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh -c
```

- 2.2. Deploy the **version.war** application that is available at **/home/student/JB248/labs/server-listeners** folder:

```
[standalone@localhost:9990 /] deploy \  
/home/student/JB248/labs/server-listeners/version.war
```

- 2.3. Open a web browser on the workstation and navigate to **http://localhost:8080/version** to validate the deployment. The version application should be displayed.

► 3. Create a self signed certificate

A self-signed certificate will be used to encrypt the communication between the server and the browser. As a side effect, the certificate may raise security alerts when used because the certificate was not issued by a trusted source. Some companies provide a trusted certificate that overcomes this limitation.

In this lab, a certificate will be created to encrypt the communication between the server and the client. This certificate will be used by the HTTPS listener.

- 3.1. In a new terminal window, run the following to change to the **/home/student/JB248/labs/server-listeners** folder:

```
[student@workstation ~]$ cd /home/student/JB248/labs/server-listeners
```

- 3.2. Create the certificate with the **keytool** command with the following parameters:

- **alias:** appserver
- **storetype:** jks
- **keyalg:** RSA
- **keysize:** 2048
- **keystore:** identity.jks
- **First and last name:** System administrator
- **Organization unit:** GLS
- **Organization name:** Training
- **City:** Raleigh
- **State:** NC
- **Country:** US
- **Password:** changeit
- **Key appserver password:** changeit

```
[student@workstation server-listeners]$ keytool -genkeypair -alias appserver \
-storetype jks -keyalg RSA -keysize 2048 -keystore identity.jks
```

- 3.3. Verify that the **identity.jks** keystore was created:

```
[student@workstation server-listeners]$ ls
```

The following output is expected:

```
identity.jks version.war
```

- 3.4. Check the certificates that are available in the **identity.jks** keystore:

```
[student@workstation server-listeners]$ keytool -list -v -keystore identity.jks
```

Use **changeit** for the password. A similar output is expected:

```
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: appserver
Creation date: May 9, 2016
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=System Administrator, OU=GLS, O=Training, L=Raleigh, ST=NC, C=US
Issuer: CN=System Administrator, OU=GLS, O=Training, L=Raleigh, ST=NC, C=US

... OUTPUT OMITTED...
```

▶ 4. Create the SSL connection configuration

The SSL connection is configured by creating a new listener in the **undertow** subsystem and can be configured using the CLI. To configure this listener, a new security realm must be created to store the information about the certificate.

- 4.1. A new realm should be created to configure the HTTPS listener. Return to the CLI and in the **core-service** named **management** add a new security realm named **HTTPSRealm**:

```
[standalone@localhost:9990 /] /core-service=\
management/security-realm=HTTPSRealm:add()
```

- 4.2. Configure the **HTTPSRealm** realm to load the **/home/student/JB248/labs/server-listeners/identity.jks** keystore file. Use the password **changeit** and open the certificate named **appserver**. This keystore file was created previously in this guided exercise.

```
[standalone@localhost:9990 /] /core-service=management/security-realm=\  
HTTPSSrealm/server-identity=\  
ssl:add(keystore-path=/home/student/JB248/labs/server-listeners/identity.jks, \  
keystore-password=changeit, alias=appserver)
```

**Note**

In a production environment it is recommended to use a **vault** to protect the certificate password.

- 4.3. A reload is required to enable the new realm:

```
[standalone@localhost:9990 /] reload
```

▶ 5. Configure the HTTPS listener

- 5.1. Add a new HTTPS listener using the configured **HTTPSSrealm** realm:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/https-listener=https:\  
add(socket-binding=https, security-realm=HTTPSSrealm)
```

- 5.2. Open a web browser and navigate to `https://localhost:8443/version` to test the SSL connection.

**Note**

You should be warned about an untrusted certificate. Accept the certificate and you should see the version application over SSL. Remember that if your certificate was authorized from a trusted source, then you will not get this warning about an untrusted certificate.

- 5.3. The SSL connection is available for all applications including the root application. Navigate to `https://localhost:8443/` and test the SSL connection for the root application.

▶ 6. Enable the AJP protocol

- 6.1. To enable the AJP protocol, the AJP listener must be configured. Run the following command to enable the AJP protocol:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/ajp-listener=ajp:add(socket-binding=ajp)
```

- 6.2. Verify that the AJP listener is configured by reading its properties:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/ajp-listener=ajp:read-resource
```

A similar output is expected:

```
{  
    "outcome" => "success",  
    "result" => {  
        "allow-encoded-slash" => false,  
        "allow-equals-in-cookie-value" => false,  
        "always-set-keep-alive" => true,  
        "buffer-pipelined-data" => true,  
        "buffer-pool" => "default",  
        "decode-url" => true,  
        "disallowed-methods" => ["TRACE"],  
        "enabled" => true,  
        "max-buffered-request-size" => 16384,  
        "max-connections" => undefined,  
  
    ... OUTPUT OMMITED...
```

► 7. Tuning the HTTP listener

After analyzing the number of connections coming from the server's HTTP listener, the total number of connections has increased, causing HTTP 500 errors. Increase the maximum possible connections for the HTTP listener to prevent the errors.

- 7.1. The number of connections can be updated using the undertow subsystem's default server. Increase the maximum possible connections to 200 in the **http-listener** to improve the responsiveness:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/http-listener=default:\\  
write-attribute(name=max-connections, value=200)
```



Note

It is also possible to define the maximum connections to the **https-listener** and to the **ajp-listener** to improve the responsiveness for these protocols.

- 7.2. A reload is required to enable the new value of maximum connections:

```
[standalone@localhost:9990 /] reload
```

- 7.3. Verify that the **max-connections** attribute was changed:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\  
default-server/http-listener=default:read-attribute(name=max-connections)
```

The following output is expected:

```
{  
    "outcome" => "success",  
    "result" => 200  
}
```

► 8. Clean Up

- 8.1. Undeploy the **version.war** application:

```
[standalone@localhost:9990 /] undeploy version.war
```

- 8.2. Remove the AJP listener:

```
[standalone@localhost:9990 /] /subsystem=undertow/server=\ndefault-server/ajp-listener=ajp:remove
```

- 8.3. Exit the EAP CLI:

```
[standalone@localhost:9990 /] exit
```

- 8.4. Stop the instance of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the guided exercise.

▶ Lab

Configuring the Web Subsystem

In this lab, you will configure the Undertow web subsystem in an EAP managed domain enabling the HTTPS and the AJP protocols. You will also deploy the bookstore application making it the root web application.

Resources	
Files	/opt/domain /opt/keystore
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore https://172.25.250.10:8443/bookstore https://172.25.250.11:8443/bookstore
Resources	bookstore.war

Outcome

You should be able to configure the Undertow web subsystem of an EAP 7 managed domain enabling the HTTPS and the AJP protocols. You should also be able to deploy the bookstore application making it the root web application.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab undertow-lab-final setup
```

1. Create a self-signed certificate

An EAP administrator has set up a managed domain with two host controllers, running on **servera** and **serverb** VMs respectively, and the domain controller on the workstation.

The domain and host configuration files are stored in the **/opt/domain** folder on all three machines.

The bookstore application will be deployed onto this domain and should be accessible using the **HTTPS** protocol. To enable this protocol, you need to create a certificate using the **keytool** command. Create a self-signed certificate by creating a keystore named **identity.jks** in the **/opt/keystore** folder on **servera** and **serverb**. The keystore should have the following characteristics:

- **alias**: appserver
- **storetype**: jks
- **keyalg**: RSA

- **keysize:** 2048
- **Password:** changeit
- **Key appserver password:** changeit

Use whatever values you like for the name and location of the organization. The **jboss** user should be the keystore's owner. In the end, the keytool will confirm if the information is correct. Just type **yes**.

2. Start the Domain Controller

Start the domain controller on workstation. Because the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Also change the directory to **/opt/jboss-eap-7.0/bin** directory to avoid permission errors.

3. Start the Host Controllers

The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

3.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Also change the directory to **/opt/jboss-eap-7.0/bin** directory to avoid permission errors.

3.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Also change the directory to **/opt/jboss-eap-7.0/bin** directory to avoid permission errors.

3.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.

4. Create the SSL connection configuration

- 4.1. The SSL connection is configured by creating a new listener in the **undertow** subsystem and can be configured using the CLI. To configure this listener, a new security realm should be created for each server to store the information about the certificate.

In the management section of each host controller configuration file, create a new security realm called **HTTPSRealm**.

Because this configuration can only be created using the CLI, and the management property is not directly available in a host, you may use the core-service property instead to configure the management attribute.

- 4.2. After creating the realm, configure it to load the **/opt/keystore/identity.jks** keystore file using the password **changeit** and open the certificate named **appserver**. This keystore file was created previously in this lab.

5. Reload all the host controllers to enable the realm.

6. Configure the HTTPS listener

The **HTTPS listener** is responsible for enabling the **HTTPS** protocol in the **undertow** subsystem. Because all the servers from all host controllers are running using the **full-ha** profile, the **undertow** subsystem from that profile must be customized. Add a new **https-listener** named **https** that refers to the socket-binding named **https** (which listens to requests from port 8443) and must use the certificate associated with the **HTTPSRealm** security realm to encrypt the contents. This listener should be bound to the **default-server** from the **undertow** configuration.

7. Configuring the firewall

A firewall configuration is required to enable the communication using the **HTTPS** protocol. Open the **8443** TCP port on the **servera** and **serverb** servers.

8. Tune the HTTPS listener

To support an increased number of requests the number of **HTTPS** connections is not enough. Increase the maximum connections to **150** in the **undertow** subsystem in the **full-ha** profile.

9. Deploy the bookstore application

You are now ready to deploy the bookstore application WAR file to the managed domain. The **bookstore.war** file is available in the **/tmp/** folder on the **workstation**. Deploy the application to **Group1**. Verify that you can use the bookstore application with the HTTP and the HTTPS protocols at:

- **http://172.25.250.10:8080/bookstore**
- **http://172.25.250.11:8080/bookstore**
- **https://172.25.250.10:8443/bookstore**
- **https://172.25.250.11:8443/bookstore**

10. Define the bookstore application as the root application

By default, the root application is the JBoss welcome application. This application should be replaced by the bookstore application. Replace the root application by changing the **default-web-module** attribute from the host named **default-host**. This host is located in the **full-ha** profile, in the server named **default-server** from the **undertow** subsystem.

11. Grading and Clean Up

- 11.1. Exit from the CLI:

```
[domain@172.25.250.254:9990 /] exit
```

- 11.2. Press **Ctrl+C** to stop the domain and host controllers.

- 11.3. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab undertow-lab-final grade
```

This concludes the lab.

► Solution

Configuring the Web Subsystem

In this lab, you will configure the Undertow web subsystem in an EAP managed domain enabling the HTTPS and the AJP protocols. You will also deploy the bookstore application making it the root web application.

Resources	
Files	/opt/domain /opt/keystore
Application URL	http://172.25.250.10:8080/bookstore http://172.25.250.11:8080/bookstore https://172.25.250.10:8443/bookstore https://172.25.250.11:8443/bookstore
Resources	bookstore.war

Outcome

You should be able to configure the Undertow web subsystem of an EAP 7 managed domain enabling the HTTPS and the AJP protocols. You should also be able to deploy the bookstore application making it the root web application.

Before You Begin

Use the following command to download the relevant lab files and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab undertow-lab-final setup
```

1. Create a self-signed certificate

An EAP administrator has set up a managed domain with two host controllers, running on **servera** and **serverb** VMs respectively, and the domain controller on the workstation.

The domain and host configuration files are stored in the **/opt/domain** folder on all three machines.

The bookstore application will be deployed onto this domain and should be accessible using the **HTTPS** protocol. To enable this protocol, you need to create a certificate using the **keytool** command. Create a self-signed certificate by creating a keystore named **identity.jks** in the **/opt/keystore** folder on **servera** and **serverb**. The keystore should have the following characteristics:

- **alias**: appserver
- **storetype**: jks
- **keyalg**: RSA

- **keysize:** 2048
- **Password:** changeit
- **Key appserver password:** changeit

Use whatever values you like for the name and location of the organization. The **jboss** user should be the keystore's owner. In the end, the keytool will confirm if the information is correct. Just type **yes**.

- 1.1. On the **servera** server, open a new terminal window and change to the **/opt/keystore** folder:

```
[student@servera ~]$ cd /opt/keystore
```

- 1.2. Create the self-signed certificate:

```
[student@servera keystore]$ sudo -u jboss keytool -genkeypair \
-alias appserver \
-storetype jks -keyalg RSA -keysize 2048 -keystore identity.jks
```

- 1.3. Check the certificates that are available in the **identity.jks** keystore:

```
[student@servera keystore]$ keytool -list -v -keystore identity.jks
```

- 1.4. Copy the certificate from **servera** to **serverb** VM in the **/tmp** directory.

```
[student@servera keystore]$ scp /opt/keystore/identity.jks \
student@serverb:/tmp
```

- 1.5. Copy the file transferred to the serverb in the **/tmp** directory to the **/opt/keystore** directory.

```
[student@serverb ~]$ cp /tmp/identity.jks /opt/keystore
```

- 1.6. Change the owner of the **keystore** to the **jboss** user on the **serverb** VM.

```
[student@serverb ~]$ sudo chown jboss:jboss /opt/keystore/identity.jks
```

2. Start the Domain Controller

Start the domain controller on workstation. Because the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Also change the directory to **/opt/jboss-eap-7.0/bin** directory to avoid permission errors.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ sudo -u jboss ./domain.sh \  
-Djboss.domain.base.dir=/opt/domain/ --host-config=host-master.xml
```

3. Start the Host Controllers

The two host controllers on **servera** and **serverb** connect to the domain controller in the previous step and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

- 3.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Also change the directory to **/opt/jboss-eap-7.0/bin** directory to avoid permission errors.

Open a new terminal window on the **servera** VM and run the following commands:

```
[student@servera ~]$ cd /opt/jboss-eap-7.0/bin  
[student@servera bin]$ sudo -u jboss ./domain.sh \  
-Djboss.domain.base.dir=/opt/domain/ \  
-Djboss.domain.master.address=172.25.250.254 \  
--host-config=host-slave.xml
```

- 3.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Also change the directory to **/opt/jboss-eap-7.0/bin** directory to avoid permission errors.

Open a new terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ cd /opt/jboss-eap-7.0/bin  
[student@serverb bin]$ sudo -u jboss ./domain.sh \  
-Djboss.domain.base.dir=/opt/domain/ \  
-Djboss.domain.master.address=172.25.250.254 \  
--host-config=host-slave.xml
```

- 3.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.

4. Create the SSL connection configuration

- 4.1. The SSL connection is configured by creating a new listener in the **undertow** subsystem and can be configured using the CLI. To configure this listener, a new security realm should be created for each server to store the information about the certificate.

In the management section of each host controller configuration file, create a new security realm called HTTPSRealm.

Because this configuration can only be created using the CLI, and the management property is not directly available in a host, you may use the core-service property instead to configure the management attribute.

Access the JBoss EAP CLI. In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss \
/opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```

Add a new security realm named HTTPSRealm on the **servera** host:

```
[domain@172.25.250.254:9990 /] /host=servera/core-service=\
management/security-realm=HTTPSRealm:add()
```

Add a new security realm named HTTPSRealm on the **serverb** host:

```
[domain@172.25.250.254:9990 /] /host=serverb/core-service=\
management/security-realm=HTTPSRealm:add()
```

- 4.2. After creating the realm, configure it to load the **/opt/keystore/identity.jks** keystore file using the password **changeit** and open the certificate named **appserver**. This keystore file was created previously in this lab.

Configure the **HTTPSRealm** realm for loading the **identity.jks** keystore on the **servera** host:

```
[domain@172.25.250.254:9990 /] /host=servera/core-service=\
management/security-realm=HTTPSRealm/server-identity=\
ssl:add(keystore-path=/opt/keystore/identity.jks, \
keystore-password=changeit, alias=appserver)
```

Configure the **HTTPSRealm** realm to load the **identity.jks** keystore on the **serverb** host:

```
[domain@172.25.250.254:9990 /] /host=serverb/core-service=\
management/security-realm=HTTPSRealm/server-identity=\
ssl:add(keystore-path=/opt/keystore/identity.jks, \
keystore-password=changeit, alias=appserver)
```

5. Reload all the host controllers to enable the realm.

Reload the **servera** host to enable the realm:

```
[domain@172.25.250.254:9990 /] /host=servera:reload
```

Reload the **serverb** host to enable the realm:

```
[domain@172.25.250.254:9990 /] /host=serverb:reload
```

6. Configure the HTTPS listener

The **HTTPS listener** is responsible for enabling the **HTTPS** protocol in the undertow subsystem. Because all the servers from all host controllers are running using the full-ha profile, the undertow subsystem from that profile must be customized. Add a new **https-listener** named **https** that refers to the socket-binding named **https** (which listens to requests from port 8443) and must use the certificate associated with the **HTTPSSrealm** security realm to encrypt the contents. This listener should be bound to the **default-server** from the undertow configuration.

6.1. Add the **https-listener**:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=undertow/server=\ndefault-server/https-listener=https:\nadd(socket-binding=https, security-realm=HTTPSSrealm)
```

7. Configuring the firewall

A firewall configuration is required to enable the communication using the **HTTPS** protocol. Open the **8443** TCP port on the **servera** and **serverb** servers.

7.1. On **servera**, open the **8443** TCP port:

```
[student@servera ~]$ sudo firewall-cmd --zone=public --add-port 8443/tcp \
--permanent
```

7.2. Reload the firewall configuration on **servera**:

```
[student@servera ~]$ sudo firewall-cmd --reload
```

7.3. Verify the open ports on **servera**:

```
[student@servera ~]$ sudo firewall-cmd --list-all
```

You should see the **8443** port available in the **ports** attribute.

7.4. Open the same port on **serverb**:

```
[student@serverb ~]$ sudo firewall-cmd --zone=public --add-port 8443/tcp \
--permanent
```

7.5. Reload the firewall configuration on **serverb**:

```
[student@serverb ~]$ sudo firewall-cmd --reload
```

7.6. Verify the open ports on **serverb**:

```
[student@serverb ~]$ sudo firewall-cmd --list-all
```

- 7.7. Verify that you can access the root application with **SSL** at <https://172.25.250.10:8443> and <https://172.25.250.11:8443>.



Note

You should be warned about an untrusted certificate. Accept the certificate and you should see the version application over SSL. Keep in mind that if your certificate was authorized from a trusted source, then you will not get this warning about an untrusted certificate.

8. Tune the HTTPS listener

To support an increased number of requests the number of **HTTPS** connections is not enough. Increase the maximum connections to **150** in the undertow subsystem in the **full-ha** profile.

- 8.1. The number of connections can be updated using the **undertow** subsystem's default server in the **full-ha** profile. Increase the maximum possible connections:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=undertow/server=\ndefault-server/https-listener=https:\\\nwrite-attribute(name=max-connections, value=150)
```

- 8.2. A reload is required to enable the new value of maximum connections:

```
[domain@172.25.250.254:9990 /] /:reload-servers
```

- 8.3. Verify that the **max-connections** attribute was changed:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=undertow/server=\ndefault-server/https-listener=https:\\\nread-attribute(name=max-connections)
```

9. Deploy the bookstore application

You are now ready to deploy the bookstore application WAR file to the managed domain. The **bookstore.war** file is available in the **/tmp/** folder on the **workstation**. Deploy the application to **Group1**. Verify that you can use the bookstore application with the HTTP and the HTTPS protocols at:

- <http://172.25.250.10:8080/bookstore>
- <http://172.25.250.11:8080/bookstore>
- <https://172.25.250.10:8443/bookstore>
- <https://172.25.250.11:8443/bookstore>

- 9.1. Deploy the application:

```
[domain@172.25.250.254:9990 /] deploy \\\n/tmp/bookstore.war --server-groups=Group1
```

- 9.2. Verify that you can use the bookstore application using the provided URLs.

10. Define the bookstore application as the root application

By default, the root application is the JBoss welcome application. This application should be replaced by the bookstore application. Replace the root application by changing the **default-web-module** attribute from the host named **default-host**. This host is located in the **full-ha** profile, in the server named **default-server** from the **undertow** subsystem.

10.1. Replace the root application:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=undertow/server=\\
default-server/host=default-host:write-attribute(name=\\
default-web-module,value=bookstore.war)
```

10.2. A reload is required to enable the new root application:

```
[domain@172.25.250.254:9990 /] /:reload-servers
```

10.3. Test the root application using one of the following URLs:

- <http://172.25.250.10:8080>
- <http://172.25.250.11:8080>
- <https://172.25.250.10:8443>
- <https://172.25.250.11:8443>



Note

If the bookstore application is not displayed, try to clean the browser cache and refresh the page.

11. Grading and Clean Up

11.1. Exit from the CLI:

```
[domain@172.25.250.254:9990 /] exit
```

11.2. Press **Ctrl+C** to stop the domain and host controllers.

11.3. Run the following command from the workstation to grade the exercise:

```
[student@workstation bin]$ lab undertow-lab-final grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- JBoss Web Server was replaced by Undertow.
- Undertow is written in Java using the XNIO API.
- New features are provided by Undertow:
 - Undertow can act as a load balancer.
 - Reduced the number of ports.
 - Supports web sockets.
 - Supports JSON-P.
 - Supports HTTP/2.
- Undertow has five main components:
 - Buffer caches
 - Server
 - Servlet container
 - Handlers
 - Filters
- It is possible to define the default application in two ways:
 - Changing the welcome-content file handler.
 - Changing the default-web-module attribute.
- It is possible to configure a server with three listeners:
 - HTTP
 - HTTPS
 - AJP
- To enable an SSL connection, a certification file is required and it is possible to generate a self-signed certificate with the **keytool** command.

Chapter 12

Deploying Clustered Applications

Overview

Goal Configure various subsystems to support the deployment of clustered applications.

- Objectives**
- Describe the benefits of clustering and the JBoss EAP subsystems involved in supporting clustered application deployments.
 - Configure Infinispan and JGroups to support clusters.
 - Configure Undertow as a load balancer.
 - Configure and deploy a high availability application.

- Sections**
- Exploring Clustered Applications (and Quiz)
 - Configuring Subsystems that Support Clustered Applications (and Guided Exercise)
 - Configuring Load Balancing (and Guided Exercise)
 - Deploying HA Singleton Applications (and Quiz)

- Lab**
- Deploying Clustered Applications

Exploring Clustered Applications

Objectives

After completing this section, students should be able to:

- Describe the benefits of clustering and the JBoss EAP subsystems involved in supporting clustered application deployments.
- Describe example topologies of clustered applications.

Clustered applications

A cluster is a collection of servers that communicate with each other in such a way that they improve the availability of services by providing the following capabilities:

- **High Availability (HA)**: a service has a very high probability of being available.
- **Scalability**: a service can handle a large number of requests by spreading the workload across multiple servers.
- **Failover**: if a service fails, the client can continue processing its tasks as another cluster member takes over the client's requests.
- **Fault Tolerance**: a server can guarantee correct behavior even if fail over occurs.

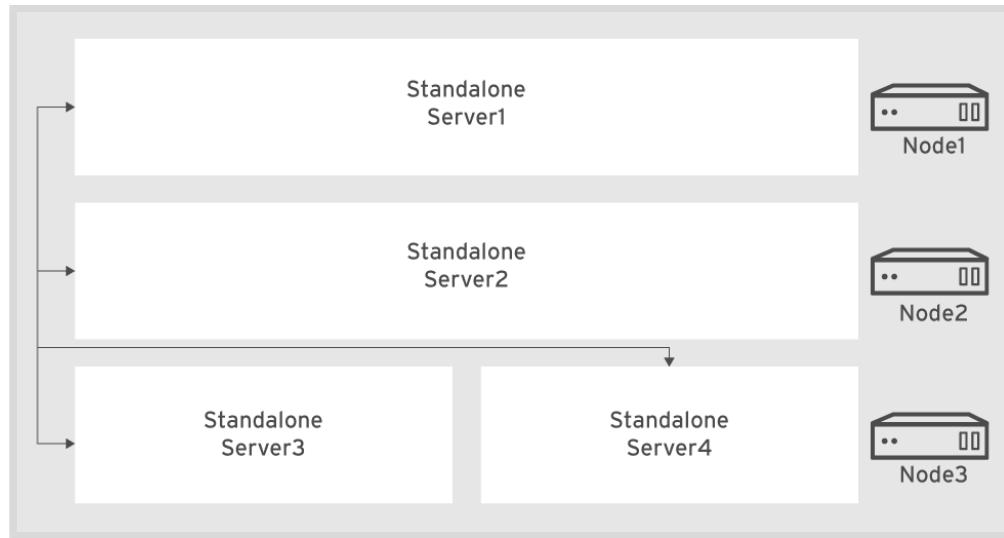
The most common way to achieve scalability and high availability is to use the following together:

- Load balancer: Often a piece of hardware, or a service like Apache httpd.
- Data replication services: A service such as memcached or a framework (Infinispan)

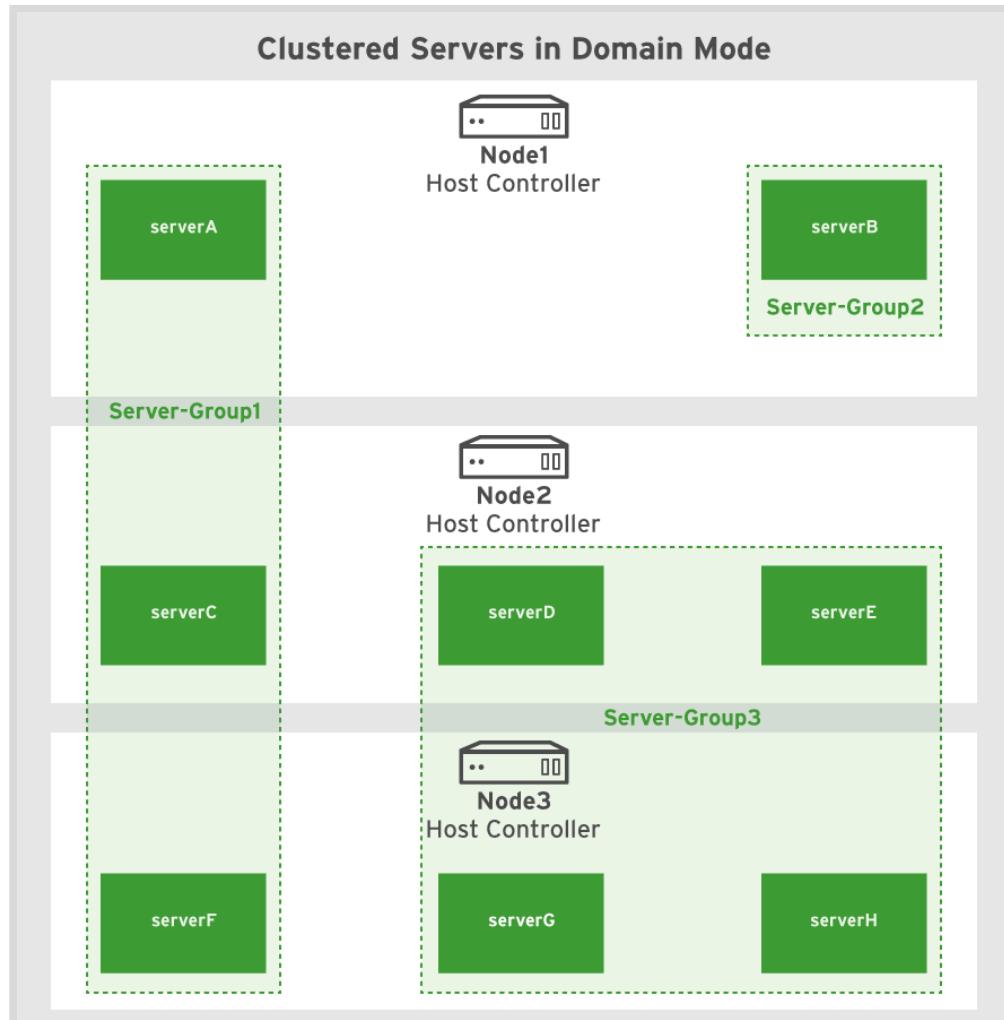
Clustering is made available to an EAP instance by two subsystems: **jgroups** and **infinispan**. The **ha** and **full-ha** profiles have these subsystems enabled by default.

In EAP, a cluster is like a group of identically-configured servers that communicate with each other to ensure that the cluster provides HA, failover, and the other clustering capabilities. Although users can run in either standalone or a managed domain, these operating modes only dictate how the servers are managed. Clustering servers can be configured in EAP in both operating modes.

In a standalone server, because each EAP instance has exactly one server, a cluster is a collection of running EAP instances:

**Figure 12.1: Standalone servers in a cluster.**

In a managed domain, a cluster is actually a collection of servers in a server group, with each server in the server group representing the nodes of the cluster:

**Figure 12.2: Clustered servers in a managed domain.**

The diagram above is an example of a clustered, managed domain. There are three nodes that host eight combined servers that are divided into three different groups.



Note

In a managed domain, a cluster is a collection of servers. In addition, a cluster can consist of two or more server groups, with all servers in the clustered server groups being the nodes of the cluster. As the above diagram shows, users can have multiple clusters within a domain, and can have servers in a server group that do not form a cluster.

Clustering architecture

Clustering is accomplished by the Infinispan and JGroups subsystems.

- Infinispan: An architecture used for caching objects and also replicating objects between caches. The Infinispan subsystem provides caching, state replication, and state distribution support.
- JGroups: A framework for nodes to communicate with each other using either UDP or TCP.



Note

Infinispan and JGroups are two large frameworks that each contain many capabilities and configuration settings. This section requires only the basic concepts of Infinispan and JGroups. Further configuration of these subsystems is outside the scope of this course.



Note

JBoss has a supported version of EAP with Infinispan backed by a NoSQL database called JBoss Data Grid (JDG). For further details, see the Red Hat course JB450.

Making an application distributable

To take advantage of the high availability features of clustered servers, an application first has to be marked as being distributable. To do so, include the `<distributable>` tag in the `web.xml` for the application. The following is an example of an application's `web.xml` that is configured for clustering:

```
<?xml version="1.0"?>
<web-app ...>
<distributable/>
</web-app>
```

In addition, users can further fine tune replication by overriding default configurations in the `jboss-web.xml` file of the application by adding a `<replication-config>` tag. The following is an example configuration that defines that the entire session is replicated (note that this is the actual default behavior for EAP):

```
<jboss-web ...>
  <replication-config>
    <replication-granularity>SESSION</replication-granularity>
  </replication-config>
</jboss-web>
```

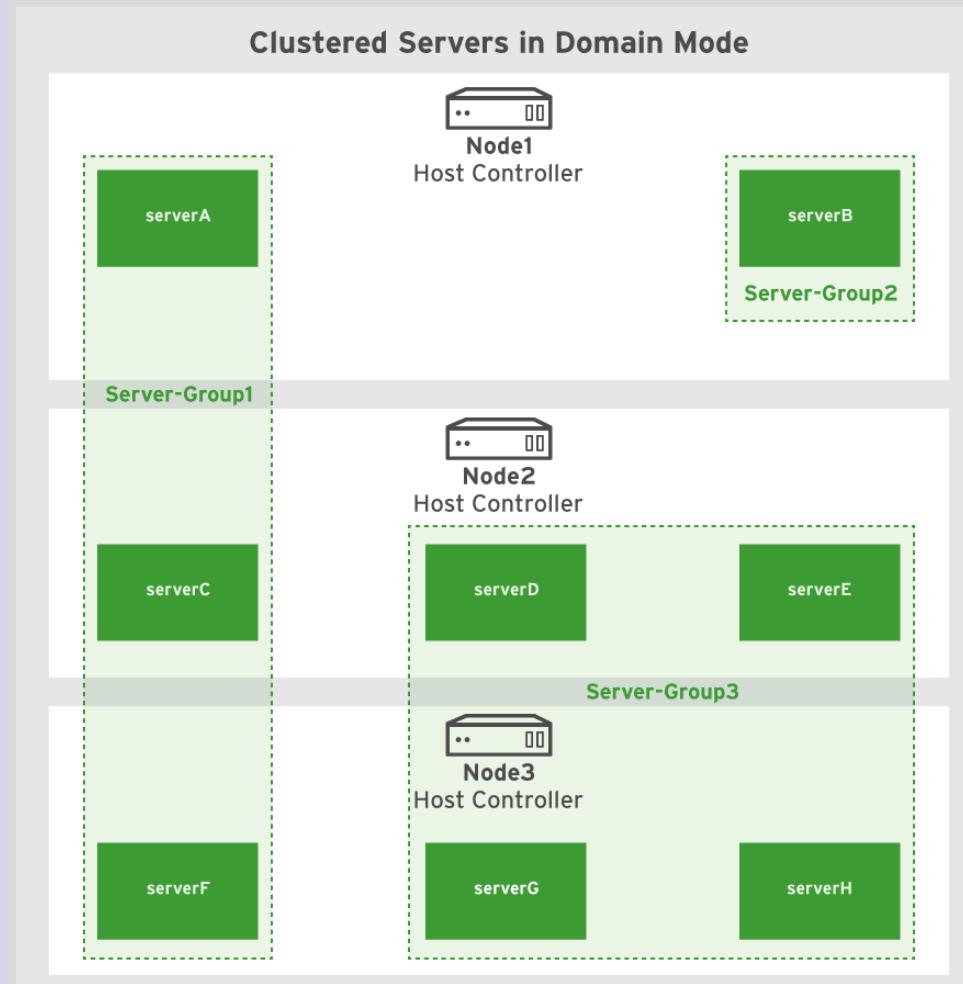
There are two possible values for **<replication-granularity>**:

- **SESSION:** The default option. A session level of replication granularity means that the entire session object is replicated whenever any attribute changes.
- **ATTRIBUTE:** Only the changed attributes in a session are replicated.

► Quiz

Exploring Clustered Applications

In managed domain mode, a cluster is actually a collection of servers in a server group, with each server in the server group representing the nodes of the cluster:



Choose the correct answer to the following questions based on the diagram of clustered servers in a managed domain:

► 1. How many server groups are in this domain? (Choose one.)

- a. One
- b. Two
- c. Three
- d. Four
- e. None

► 2. How many servers are in Server -Group1? (Choose one.)

- a. One
- b. Two
- c. Three
- d. Four
- e. None

► 3. If a distributable application is deployed onto Server -Group1 and Server -Group1 uses the ha profile, does that create a cluster between serverA, serverC, and serverF? (Choose one.)

- a. Yes. When the distributable application is deployed onto the server group, they will automatically create a cluster.
- b. Yes. But the clustering needs to be manually enabled on the server side.
- c. No. The server needs additional configuration on the *-ha profile.
- d. No. The application needs to be deployed onto an individual server, not a server group.

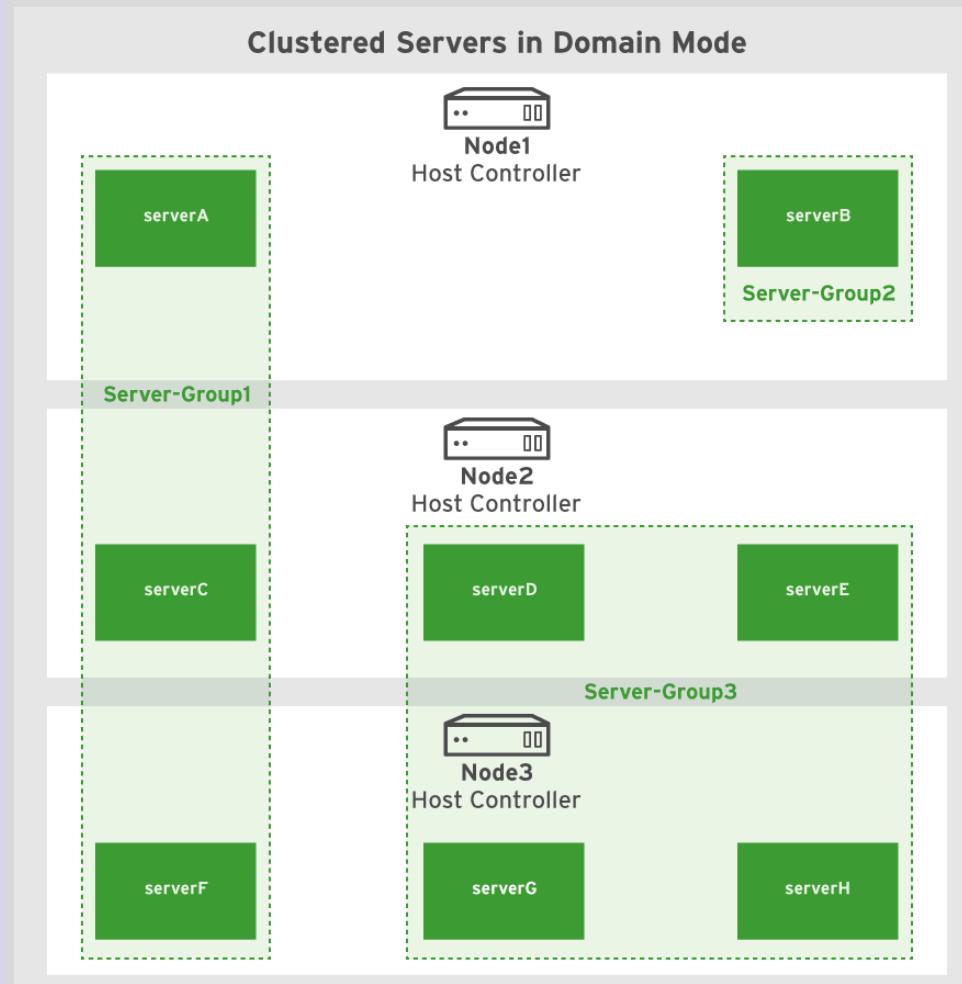
► 4. Suppose app1 is deployed onto Server -Group1 and app3 is deployed onto Server -Group3 (and both applications are distributable). What happens to these applications if node 3 fails? (Choose one.)

- a. Both applications will return errors to users.
- b. app1 will continue working by failing over to serverA and serverC. app3 will fail because it is not in a cluster and has no available servers..
- c. The serverA and serverC instances are still up and running, so requests to app1 will failover to either one of them. Similarly, serverD and serverE will handle the failover of requests sent to app3.
- d. app3 will continue working by failing over to serverE and serverD. app1 will fail because it is not in a cluster and has no available servers.

► Solution

Exploring Clustered Applications

In managed domain mode, a cluster is actually a collection of servers in a server group, with each server in the server group representing the nodes of the cluster:



Choose the correct answer to the following questions based on the diagram of clustered servers in a managed domain:

► 1. How many server groups are in this domain? (Choose one.)

- a. One
- b. Two
- c. Three
- d. Four
- e. None

► 2. How many servers are in Server -Group1? (Choose one.)

- a. One
- b. Two
- c. Three
- d. Four
- e. None

► 3. If a distributable application is deployed onto Server -Group1 and Server -Group1 uses the ha profile, does that create a cluster between serverA, serverC, and serverF? (Choose one.)

- a. Yes. When the distributable application is deployed onto the server group, they will automatically create a cluster.
- b. Yes. But the clustering needs to be manually enabled on the server side.
- c. No. The server needs additional configuration on the *-ha profile.
- d. No. The application needs to be deployed onto an individual server, not a server group.

► 4. Suppose app1 is deployed onto Server -Group1 and app3 is deployed onto Server -Group3 (and both applications are distributable). What happens to these applications if node 3 fails? (Choose one.)

- a. Both applications will return errors to users.
- b. app1 will continue working by failing over to serverA and serverC. app3 will fail because it is not in a cluster and has no available servers..
- c. The serverA and serverC instances are still up and running, so requests to app1 will failover to either one of them. Similarly, serverD and serverE will handle the failover of requests sent to app3.
- d. app3 will continue working by failing over to serverE and serverD. app1 will fail because it is not in a cluster and has no available servers.

Configuring Subsystems that Support Clustered Applications

Objectives

After completing this section, students should be able to:

- Configure Infinispan and JGroups to support clusters.
- Start a cluster in a standalone server or a managed domain.

Configuration of the JGroups subsystem

The *JGroups Subsystem* provides all the communication mechanisms for how the servers in a cluster communicate with each other. EAP is preconfigured with two JGroups stacks:

- udp**: the nodes in the cluster use User Datagram Protocol (UDP) multicasting to communicate with each other. This is the default stack.
- tcp**: the nodes in the cluster use Transmission Control Protocol (TCP) to communicate with each other.

Users can use one of these preconfigured stacks, or can define and use a new stack that suits the specific needs of the environment. By default, the UDP protocol is used to communicate between clustered nodes in the default **ee** JGroups channel. The following EAP CLI command adjusts the **ee** JGroups channel to use a **tcp** stack instead of **udp**:

```
/subsystem=jgroups/channel=ee:write-attribute(
    name=stack,value=tcp)
```

By default, the **tcp** stack uses multicast for discovering other members of a cluster. Users can further customize the **tcp** stack by changing the protocol to either **TCPPING** or **TCPGOSSIP**.

- TCPPING**: a protocol that uses a static list to define the cluster members and uses unicast as an alternative to multicast. The following parameters are specific to this protocol:
 - initial_hosts**: a list of the hosts that are available and known to look up for cluster membership.
 - port_range**: the range that the protocol will use to search for hosts based on the initial port. For example, a port range of two on an initial port of **7600** results in the **TCPPING** protocol searching for a viable host on ports **7600** and **7601** to be added to the membership.



References

Further information about the parameters for customizing **TCPPING** can be found at:
Configure TCPPING_stack

https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0/configuration-guide/chapter-21-configuring-high-availability#configure_tcpping_stack

- TCPGOSSIP**: discovers members of a cluster by using an external gossip router.

The instructor will now demonstrate how to configure the JGroups UDP and TCP stacks by using the EAP management console:

Demonstration: Configuring the JGroups stack

1. Open a terminal window from the workstation VM (**Applications → Favorites → Terminal**) and run the following command to create the lab directory, download the required files for the demonstration, and verify that EAP is installed and not currently running:

```
[student@workstation ~]$ demo clustering-jgroups setup
```

2. Often users want to run multiple standalone instances without needing to install EAP multiple times on the same host. The setup script has created two directories to represent the two standalone instances of EAP that we are going to cluster in this demonstration.

Verify that you can see the following three folders under the **/home/student/JB248/labs/jgroups-cluster1** and **/home/student/JB248/labs/jgroups-cluster2** folder:

- **configuration**
- **deployments**
- **lib**

3. You need to open several ports on the workstation VM for this demonstration. Briefly review the firewall configuration in the file **/home/student/JB248/labs/clustering-jgroups/jgroups-firewall-rules.sh**. Execute the script as follows:

```
[student@workstation standalone]$ sudo  
sh \  
/home/student/JB248/labs/clustering-jgroups/jgroups-firewall-rules.sh
```

4. Verify that the ports in the firewall are open by running the following command:

```
[student@workstation standalone]$ firewall-cmd --list-all --zone=public  
public (default, active)  
  interfaces: eth0 eth1  
  sources:  
  services: dhcpcv6-client http ldap mountd mysql nfs rpc-bind ssh  
  ports: 54300/tcp 55200/udp 55300/udp 23364/udp 54200/tcp 7600/tcp 45688/udp  
        7700/tcp  
  ...
```

5. Run the following command to start the first instance of the EAP server using the **standalone.sh** script in the original EAP installation while using the new EAP configuration files from **jgroups-cluster1**.

Note that we are going to cluster the two instances, so you need to start the instances with the **standalone-full-ha.xml** configuration because it has the relevant JGroups subsystems defined in it. We also start each instance with a unique **node** name to avoid warning messages from the clustering subsystem when two instances are run on the same server.

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/jgroups-cluster1/ \
-Djboss.node.name=jgroups-cluster1 \
-c standalone-full-ha.xml
```

Verify that the first instance started without any errors by observing the **/home/student/JB248/labs/jgroups-cluster1/log/server.log** file using the **tail -f** command.



Note

In the **standalone-full-ha.xml** file, there is no console handler defined by default, hence the need to use the **tail** command to view the server logs.

Similarly, start the second instance of the EAP server using the configuration files from **jgroups-cluster2**. Note that to avoid port clashes with the first instance, we start the second instance with a **port-offset** value of 100:

```
[student@workstation standalone]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/jgroups-cluster2/ \
-Djboss.socket.binding.port-offset=100 \
-Djboss.node.name=jgroups-cluster2 \
-c standalone-full-ha.xml
```

Verify that the second instance started without any errors by inspecting the **/home/student/JB248/labs/jgroups-cluster2/log/server.log** file using the **tail -f** command.

6. Observe the log file of **jgroups-cluster1** after the **jgroups-cluster2** instance starts up successfully. You should see the following entries that show that there are now two members in the cluster:

```
2016-05-25 03:08:57,593 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel
server: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]
2016-05-25 03:08:57,596 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel
web: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]
2016-05-25 03:08:57,613 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel
ejb: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]
2016-05-25 03:08:57,614 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel
hibernate: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]
```

**Note**

You may see a few **WARN** messages such as the following in the server logs:

```
2016-05-30 13:13:19,132 WARN
[org.infinispan.topology.ClusterTopologyManagerImpl] (transport-
thread--p13-t2) ISPN000197: Error updating cluster member list:
org.infinispan.util.concurrent.TimeoutException: Replication timeout for
jgroups-cluster2
```

This is a known issue in the EAP 7 GA release (See <https://issues.jboss.org/browse/JBEAP-794>) and can be safely ignored. It will be fixed in a later EAP 7 release.

7. On the **workstation**, navigate to `http://localhost:9990` to access the EAP management console of the **jgroups-cluster1** instance.
Log in with user name **jbossadm** and password **JBoss@RedHat123**.
8. In the EAP management console, the **JGroups** subsystem is configured under the **Configuration → Subsystems → JGroups** section.

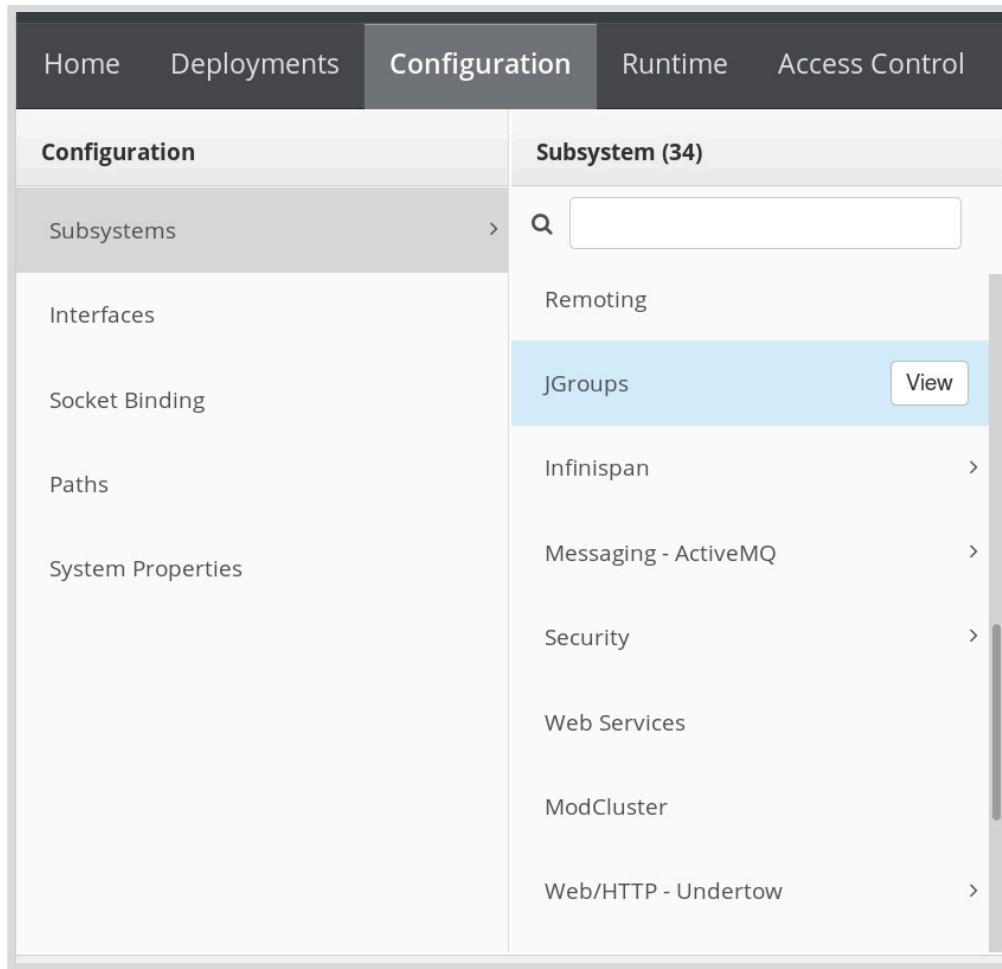


Figure 12.4: JGroups subsystem

Navigate to **Configuration** → **Subsystems** → **JGroups** in the EAP management console and click **View** next to **JGroups** to bring up the JGroups subsystem configuration page.

9. In the JGroups subsystem configuration page, there are two stacks defined by default:
 - **udp:** This is the default JGroups stack which uses UDP multicasting and auto-discovery to form an EAP cluster.
 - **tcp:** JGroups uses the TCP protocol to form an EAP cluster. This is useful in scenarios where UDP traffic is not allowed within the network segment that is running the EAP cluster. In contrast to the UDP stack, where EAP nodes can be added to a cluster automatically, in the TCP stack configuration, the IP addresses of the cluster nodes are managed by the EAP administrator manually.
10. Click **View** next to the **UDP** stack in the JGroups subsystem configuration page and briefly go through the configuration of the UDP stack. Do NOT change any of the settings because it is set up for clustering by default.
11. Navigate back to the Protocol Stack page and click **View** next to the **TCP** stack to view the configuration of the JGroups TCP stack. Briefly go through the configuration of the TCP stack. Do NOT change any of the settings because we are going to define a new TCP stack which is going to configure the cluster with settings specific to our demonstration environment.

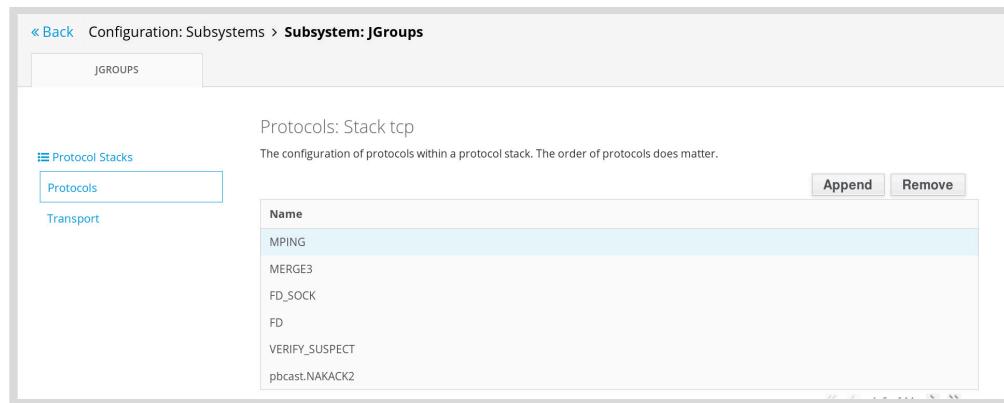


Figure 12.5: JGroups TCP stack

**Note**

The **standalone-full-ha.xml** file already defines a **tcp** stack which uses the TCP protocol for replicating the data between EAP nodes, but it uses the **MPING** protocol for auto discovery of nodes. **MPING** is based on UDP multicasting and may not work in many networking environments where UDP multicast traffic is disabled. Hence, we need to define a new TCP stack which uses a TCP based protocol called **TCPPING**, which is based on node discovery by using a static list of IP addresses and ports of each of the cluster nodes.

12. Test the clustering by using the **UDP** stack that is set up by default in the **standalone-full-ha.xml** file. Use the test application called **cluster.war** that is available in the **/home/student/JB248/labs/clustering-jgroups** folder.
13. To verify that the two EAP instances in the cluster are using the UDP protocol for communication, use the **tcpdump** tool to monitor traffic on the **workstation** VM. The communication happens on multicast address **230.0.0.4** and port **45688** by default.

Open a new terminal window and run the following command:

```
[student@workstation ~]$ sudo tcpdump -i lo udp port 45688 -vvv
```

You should see output like the following:

```
21:30:07.951124 IP (tos 0x8, ttl 2, id 43372, offset 0, flags [DF], proto UDP (17), length 4196)
localhost.55300 > 230.0.0.4.45688: [bad udp cksum 0x7567 -> 0xba48!] UDP, length 4168
21:30:08.163847 IP (tos 0x8, ttl 2, id 43373, offset 0, flags [DF], proto UDP (17), length 4196)
localhost.55200 > 230.0.0.4.45688: [bad udp cksum 0x7567 -> 0x5712!] UDP, length 4168
21:30:09.386034 IP (tos 0x8, ttl 2, id 43374, offset 0, flags [DF], proto UDP (17), length 68)
localhost.55300 > 230.0.0.4.45688: [bad udp cksum 0x6547 -> 0xaf03!] UDP, length 40
```

```
21:30:09.598657 IP (tos 0x8, ttl 2, id 43375, offset 0, flags [DF], proto UDP  
(17), length 68)  
localhost.55200 > 230.0.0.4.45688: [bad udp cksum 0x6547 -> 0x07ff!] UDP, length  
40  
21:30:09.951433 IP (tos 0x8, ttl 2, id 43376, offset 0, flags [DF], proto UDP  
(17), length 4196)
```

To stop capturing **tcpdump** information, press **Ctrl+C** in the terminal window where you started it.

14. Open two terminals and launch the EAP CLI to connect to both instances. Connect to the **jgroups-cluster1** instance using the following command:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:9990  
[standalone@localhost:9990 /]
```

Connect to the **jgroups-cluster2** instance using the following command. Note that the **jgroups-cluster2** instance is running with a port-offset value of **100**:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:10090  
[standalone@localhost:10090 /]
```

15. Deploy the **cluster.war** file on the **jgroups-cluster1** instance:

```
[standalone@localhost:9990 /] deploy \  
/home/student/JB248/labs/clustering-jgroups/cluster.war  
[standalone@localhost:9990 /]
```

16. Similarly, deploy the **cluster.war** file on the **jgroups-cluster2** instance:

```
[standalone@localhost:10090 /] deploy \  
/home/student/JB248/labs/clustering-jgroups/cluster.war  
[standalone@localhost:10090 /]
```

17. In the log files of both instances, verify that the application was successfully deployed. You should see a message similar to the following:

```
2016-05-25 03:41:24,749 INFO  [org.jboss.as.server] (management-handler-thread -  
5) WFLYSRV0010: Deployed "cluster.war" (runtime-name : "cluster.war")
```

18. Navigate to <http://localhost:8080/cluster> to view the test application running on the **jgroups-cluster1** instance. Refresh the page a few times and observe that the counter is incremented by one every time you refresh the page.

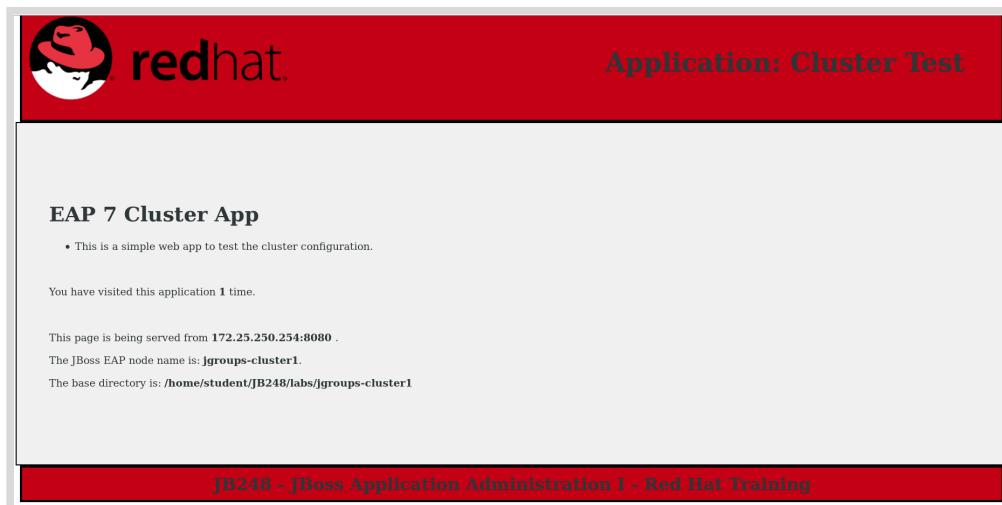


Figure 12.6: The cluster test application.

19. Navigate to <http://localhost:8180/cluster> to view the test application running on the **jgroups-cluster2** instance. Observe that the counter value has not been reset, but incremented by one. Refresh the page a few times and observe that the counter is incremented by one every time you refresh the page.
 20. Shut down the **jgroups-cluster2** instance by pressing **Ctrl+C** on the terminal window where you started the instance. Navigate to <http://localhost:8080/cluster> to view the test application running on the **jgroups-cluster1** instance. Observe that the counter value has not been reset, but reflects the latest value (plus one) as observed before shutting down **jgroups-cluster2**.
- Refresh the page a few times and observe that the counter is incremented by one every time you refresh the page. The EAP clustering component uses the JGroups stack (by default, UDP) to replicate the counter value across all nodes in the cluster.
21. Shut down **the jgroups-cluster1** instance by pressing **Ctrl+C** on the terminal window where you started the instance.
 22. Use the EAP CLI to define a new **TCP** stack configuration. Open the file **/home/student/JB248/labs/clustering-jgroups/new-tcp-stack.cli** in a text editor of your choice and review the commands listed in it. The contents of the file should be as follows:

```
# Add a new TCP stack called "tcpping"
batch
/subsystem="jgroups"/stack="tcpping":add()
/subsystem="jgroups"/stack="tcpping":add-protocol(type="TCPPING")
/subsystem="jgroups"/stack="tcpping"/transport="TRANSPORT":add(socket-
binding="jgroups-tcp", type="TCP")
run-batch

# Customize the protocol settings for tcpping
batch
/subsystem="jgroups"/stack="tcpping"/protocol="TCPPING"/
property="initial_hosts":add(value="localhost[7600],localhost[7700]")
/subsystem="jgroups"/stack="tcpping"/protocol="TCPPING"/
property="port_range":add(value="10")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="MERGE2")
```

```
/subsystem="jgroups"/stack="tcpping":add-protocol(socket-binding="jgroups-tcp-fd", type="FD_SOCK")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="FD")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="VERIFY_SUSPECT")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="BARRIER")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="pbcast.NAKACK")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="UNICAST2")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="pbcast.STABLE")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="pbcast.GMS")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="UFC")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="MFC")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="FRAG2")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="RSVP")
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcpping)
run-batch
:reload
```

The **initial_hosts** property is a comma-separated list of server instances and the ports (in square brackets) that are to be a part of the cluster.

23. Start both EAP instances as outlined in Step 5.
24. Run the following command in a new terminal on the workstation VM to create the new TCP stack on the **jgroups-cluster1** instance:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:9990 \
--file=/home/student/JB248/labs/clustering-jgroups/new-tcp-stack.cli
The batch executed successfully
...
```

25. Similarly, run the following command in a new terminal on the workstation VM to create the new TCP stack on the **jgroups-cluster2** instance:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:10090 \
--file=/home/student/JB248/labs/clustering-jgroups/new-tcp-stack.cli
The batch executed successfully
...
```

26. Verify that the a new TCP stack called **tcpping** is defined in the **/home/student/JB248/labs/jgroups-clusterX/configuration/standalone-full-ha.xml** files of both instances, where 'X' denotes the instance number:

```
<stack name="tcpping">
    <transport type="TCP" socket-binding="jgroups-tcp"/>
    <protocol type="TCPPING">
        <property name="initial_hosts">
            localhost[7600],localhost[7700]
        </property>
        <property name="port_range">
            10
        </property>
```

```
</protocol>
<protocol type="MERGE2"/>
...
</stack>
```

Also verify that the default stack has been set to the new **tcpping** definition:

```
<channels default="ee">
  <channel name="ee" stack="tcpping"/>
</channels>
```

27. Repeat Steps 17-20 to test the new TCP stack. Verify that the test application behaves in a similar manner as it did when you configured the instances with the default UDP stack.
28. Monitor the **tcpdump** command output as outlined in **Step 13** and verify that you do NOT see any UDP traffic on port **45688** because the nodes are using the TCP protocol for communication. Observe the TCP traffic on **localhost** port **7600** when both instances are running using the **tcpdump** command:

```
[student@workstation ~]$ sudo tcpdump -i lo tcp port 7600 -vvv
```

You should see output similar to the following:

```
04:42:11.670370 IP (tos 0x0, ttl 64, id 16573, offset 0, flags [DF], proto TCP
(6), length 52)
  localhost.localdomain.55718 > localhost.localdomain.7600: Flags [.],
  cksun 0xfe28 (incorrect -> 0xfb75), seq 62613, ack 59711, win 32742, options
  [nop,nop,TS val 1811776 ecr 1811776], length 0
04:42:11.718546 IP (tos 0x0, ttl 64, id 65391, offset 0, flags [DF], proto TCP
(6), length 178)
  localhost.localdomain.7600 > localhost.localdomain.55718: Flags [P.], cksun
  0xfea6 (incorrect -> 0xa79b), seq 59711:59837, ack 62613, win 10, options
  [nop,nop,TS val 1811824 ecr 1811776], length 126
...
```

29. Stop the two EAP instances by pressing **Ctrl+C** in the terminal window where you started the instances.

Exit the EAP CLI sessions by typing **quit** or press **Ctrl+C** in the terminal window where you started the CLI sessions.

Exit the tail and **tcpdump** command sessions by pressing **Ctrl+C** in the terminal window where you started the commands.

This concludes the demonstration.

Configuration of the Infinispan subsystem

The Infinispan subsystem provides caching support for JBoss EAP, facilitating the high availability features of clustered servers.

In a clustered environment, similar data is replicated onto each node in the cluster. This data is stored in a cache, and the caching mechanism and features are implemented by a framework

called *Infinispan*. In addition to being able to configure how EAP caches data, Infinispan also provides the facilities to view runtime metrics for cache containers and caches.

A cache is defined within a *cache container*, or a repository for the caches. There are four preconfigured cache containers in the **ha** and **full-ha** profiles:

- **web**: for session replication
- **hibernate**: for entity caching
- **ejb**: for stateful session bean replication
- **server**: for singleton caching

The **web**, **hibernate** and **ejb** caches are used by developers to cache Java components. In clustering, the nodes use a cache in the **cluster** container configured for replicating objects efficiently and effectively over a large cluster of nodes.

There are four different types of caches:

- **Local**: Entries are not distributed to the rest of the cache and are instead stored only on the local node.
- **Invalidation**: Uses a cache store to store entries, pulling from the store when an entry needs it.
- **Replication**: All entries are replicated on each node.
- **Distribution**: Entries are replicated to only some of the nodes.

Accordingly, there are four different pages in the Management Console for defining each type of cache. These pages are in the **Infinispan** section of the **Subsystem** page.



Note

Fine-tuning the default cache is beyond the scope of this course, but it is important to understand that this cache exists, and in some environments users may need to modify the settings so that they are appropriate for the application or environment.

The following XML excerpt displays the default Infinispan configuration:

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
    <cache-container name="server" aliases="singleton cluster" default-
cache="default" module="org.wildfly.clustering.server">
        <transport lock-timeout="60000"/>
        <replicated-cache name="default" mode="SYNC">
            <transaction mode="BATCH"/>
        </replicated-cache>
    </cache-container>
    <cache-container name="web" default-cache="dist"
module="org.wildfly.clustering.web.infinispan">
        <transport lock-timeout="60000"/>
        <distributed-cache name="dist" mode="ASYNC" l1-lifespan="0" owners="2">
            <locking isolation="REPEATABLE_READ"/>
            <transaction mode="BATCH"/>
            <file-store/>
        </distributed-cache>
    </cache-container>
</subsystem>
```

```
</cache-container>
<cache-container name="ejb" aliases="sfsb" default-cache="dist"
module="org.wildfly.clustering.ejb.infinispan">
  <transport lock-timeout="60000"/>
  <distributed-cache name="dist" mode="ASYNC" l1-lifespan="0" owners="2">
    <locking isolation="REPEATABLE_READ"/>
    <transaction mode="BATCH"/>
    <file-store/>
  </distributed-cache>
</cache-container>
<cache-container name="hibernate" default-cache="local-query"
module="org.hibernate.infinispan">
  <transport lock-timeout="60000"/>
  <local-cache name="local-query">
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </local-cache>
  <invalidation-cache name="entity" mode="SYNC">
    <transaction mode="NON_XA"/>
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </invalidation-cache>
  <replicated-cache name="timestamps" mode="ASYNC"/>
</cache-container>
</subsystem>
```

Notice that the subsystem defines the four default cache containers: web, hibernate, ejb, and server. Also notice that each cache container specifies the **default-cache**. For example, the **hibernate** cache container uses the **local-query** cache, which maps to the **local-cache**.

Use the following process to configure a new cache with the EAP CLI:

1. Create a cache container:

```
/subsystem=infinispan/cache-container=<container-name>:add
```

2. Add a replicated cache:

```
/subsystem=infinispan/cache-container=<container-name>/replicated-cache=<cache-name>:add(mode=<mode>)
```

3. Set the default cache:

```
/subsystem=infinispan/cache-container=<container-name>:write-
attribute(name=default-cache,value=<cache-name>)
```

Starting a cluster

To start a cluster either in standalone or a managed domain, the servers can use the **ha** or **full-ha** profile to take advantage of the already configured cluster settings.

The standalone or managed domain should be started as usual with an additional parameter, the **jboss.node.name** parameter as well as providing the bind address. For example in a standalone server:

```
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/opt/jboss-eap-7.0/standalone/ \
-Djboss.node.name=jgroups-cluster1 \
-Djboss.bind.address=172.25.250.254 \
-c standalone-full-ha.xml
```

When a node does join a cluster, the following log events are printed:

```
2016-05-25 03:08:57,593 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel
server: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]
2016-05-25 03:08:57,596 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel
web: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]
...
```



Note

When starting additional standalone instances on the same node and attempting to cluster them, be sure to configure a port offset to avoid port conflicts:

```
-Djboss.socket.binding.port-offset=100
```

To set up a cluster in a managed domain, users need to ensure that the servers in the server groups can communicate with each other using JGroups. This can be as simple as configuring a server group to use the **ha** or **full-ha** profiles.

A cluster in a managed domain is configured at the server group level. To start a cluster in a managed domain:

1. Change the **profile** of the desired **server-group** to **ha** (or **full-ha**).
2. Make sure the **server-group** is using **ha-sockets** (or **full-ha-sockets**) as its **socket-binding-group**.
3. Make sure each **server** in the **server-group** has a unique name.
4. Deploy a **distributable** application onto the server group.

After the domain controller is configured with an appropriate server group, start the host controllers. Any distributable application deployed onto the server group will cause all of the servers in that group to become clustered.



Note

Be sure that the necessary ports are opened on the firewall, including ports used by TCP and UDP. If running in a managed domain, these ports all need to be opened on each node. Be aware to also open ports that are offset.

► Guided Exercise

Configuring JGroups for a Clustered Web Application

In this lab, you will configure the JGroups UDP and TCP stack to enable clustering of web applications.

Resources	
Files:	<code>/home/student/JB248/labs/jgroups-cluster1</code> <code>/home/student/JB248/labs/jgroups-cluster2</code> <code>/home/student/JB248/labs/config-jgroups</code>
Application URL:	<code>http://172.25.250.254:8080/cluster</code> <code>http://172.25.250.254:8180/cluster</code>
Resources	<code>/home/student/JB248/labs/config-jgroups/cluster.war</code> <code>/home/student/JB248/labs/config-jgroups/new-tcp-stack.cli</code>

Outcomes

You should be able to configure and test a two-node cluster of EAP instances using the default UDP stack as well as a custom TCP stack.

Before You Begin

Run the following command to verify that EAP is installed at `/opt/jboss-eap-7.0`, that no instance of EAP is running, to create the required files, and download the cluster.war application.

```
[student@workstation ~]$ lab config-jgroups setup
```

- 1. You need to open several ports on the workstation VM for this lab. Briefly review the firewall configuration in the file `/home/student/JB248/labs/config-jgroups/jgroups-firewall-rules.sh`. Execute the script as follows:

```
[student@workstation standalone]$ sudo sh \
/home/student/JB248/labs/config-jgroups/jgroups-firewall-rules.sh
```

- 2. Verify that the ports in the firewall are open by running the following command:

```
[student@workstation standalone]$ firewall-cmd --list-all --zone=public
public (default)
  interfaces:
  sources:
  services: dhcpcv6-client http ldap mountd mysql nfs rpc-bind ssh
  54300/tcp 57600/tcp 55200/udp 54200/tcp 55300/udp 23364/udp 8080/tcp 8180/tcp
  45700/udp 7600/tcp 57700/tcp 45688/udp 7700/tcp
  ...
```

► 3. Start the Standalone EAP Servers

In this guided exercise, two standalone instances running the **standalone-full-ha.xml** configuration will be required. The **standalone-full-ha.xml** configuration file defines an interface called **private** in the **<interfaces>** section which is used for intracluster communication.



Note

In a production environment, the **public** and the **private** interfaces will be bound to two physically separated networks with different IP addresses. In this lab, we are binding both interfaces to the IP of the **workstation** VM (172.25.250.254).

3.1. The first standalone server will be started with the following characteristics:

- Use the **/home/student/JB248/labs/jgroups-cluster1** directory as the base directory.
- Use the **standalone-full-ha.xml** configuration file.
- Define the **public** interface IP as 172.25.250.254.
- Define the **private** interface IP as 172.25.250.254.
- Define the node name as **jgroups-cluster1**

To start it, run the following commands from the workstation VM:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
  --server-config=standalone-full-ha.xml \
  -Djboss.server.base.dir=/home/student/JB248/labs/jgroups-cluster1 \
  -Djboss.bind.address=172.25.250.254 \
  -Djboss.bind.address.private=172.25.250.254 \
  -Djboss.node.name=jgroups-cluster1
```

Verify that the first instance started without any errors by inspecting the **/home/student/JB248/labs/jgroups-cluster1/log/server.log** file using the **tail -f** command.



Note

In the **standalone-full-ha.xml** file, there is no console handler defined by default, hence the need to use the **tail** command to view the server logs.

3.2. The second standalone server will be started with the following characteristics:

- Use the **/home/student/JB248/labs/jgroups-cluster2** directory as the base directory.
- Use the **standalone-full-ha.xml** configuration file.
- Define the **public** interface IP as 172.25.250.254.
- Define the **private** interface IP as 172.25.250.254.
- To avoid port conflicts, define the port-offset with a value of **100**.
- Define the node name as **jgroups-cluster2**

To start the server, run the following commands from the workstation VM:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh \  
  --server-config=standalone-full-ha.xml \  
-Djboss.server.base.dir=/home/student/JB248/labs/jgroups-cluster2 \  
-Djboss.bind.address=172.25.250.254 \  
-Djboss.bind.address.private=172.25.250.254 \  
-Djboss.socket.binding.port-offset=100 \  
-Djboss.node.name=jgroups-cluster2
```

Verify that the second instance started without any errors by inspecting the **/home/student/JB248/labs/jgroups-cluster2/log/server.log** file using the **tail -f** command.

3.3. Observe the log file of **jgroups-cluster1** after the **jgroups-cluster2** instance starts successfully. You should see the following entries that show that there are now two members in the cluster:

```
2016-05-25 03:08:57,593 INFO  
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]  
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel  
server: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]  
2016-05-25 03:08:57,596 INFO  
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]  
(thread-1,ee,jgroups-cluster1) ISPN000094: Received new cluster view for channel  
web: [jgroups-cluster1|1] (2) [jgroups-cluster1, jgroups-cluster2]  
...
```

**Note**

You may see a few **WARN** messages such as the following in the server logs:

```
2016-05-30 13:13:19,132 WARN
[org.infinispan.topology.ClusterTopologyManagerImpl] (transport-
thread--p13-t2) ISPN000197: Error updating cluster member list:
org.infinispan.util.concurrent.TimeoutException: Replication timeout for
jgroups-cluster2
```

This is a known issue in the EAP 7 GA release (See <https://issues.jboss.org/browse/JBEAP-794>) and can be safely ignored. It will be fixed in a later EAP 7 release.

- ▶ 4. Test the clustering by using the **UDP** stack that is set up by default in the **standalone-full-ha.xml** file. Use the test application **cluster.war** that is available in the **/home/student/JB248/labs/config-jgroups** folder.
- ▶ 5. Verify that the two EAP instances in the cluster are using the UDP protocol for communication with the **tcpdump** tool to monitor traffic on the workstation VM. The communication happens on multi-cast address **230.0.0.4** and port **45688** by default.
Open a new terminal window and run the following command:

```
[student@workstation ~]$ sudo tcpdump -i eth0 udp port 45688 -vvv
```

You should see output similar to the following:

```
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
13:15:31.727165 IP (tos 0x8, ttl 2, id 37698, offset 0, flags [DF], proto UDP
(17), length 133)
    workstation.lab.example.com.55200 > 230.0.0.4.45688: [bad udp cksum 0x8d9f ->
0x7397!] UDP, length 105
13:15:32.532451 IP (tos 0x8, ttl 2, id 37699, offset 0, flags [DF], proto UDP
(17), length 68)
    workstation.lab.example.com.55200 > 230.0.0.4.45688: [bad udp cksum 0x8d5e ->
0x91d3!] UDP, length 40
13:15:32.533185 IP (tos 0x8, ttl 2, id 37700, offset 0, flags [DF], proto UDP
(17), length 68)
    workstation.lab.example.com.55300 > 230.0.0.4.45688: [bad udp cksum 0x8d5e ->
0xdcf2!] UDP, length 40
13:15:32.644452 IP (tos 0x8, ttl 2, id 37701, offset 0, flags [DF], proto UDP
(17), length 113)
    workstation.lab.example.com.55200 > 230.0.0.4.45688: [bad udp cksum 0x8d8b ->
0xdd43!] UDP, length 85
```

- ▶ 6. Deploy the **cluster.war** application

The cluster application should be deployed onto the two standalone servers of the cluster. The application should be deployed twice, one for each node.

- 6.1. Open a new terminal window and run the CLI tool connecting to the first instance of the cluster with the following commands:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh -c --controller=127.0.0.1:9990
```

- 6.2. Deploy the cluster application:

```
[standalone@127.0.0.1:9990 /] deploy \  
/home/student/JB248/labs/config-jgroups/cluster.war
```

- 6.3. Connect to the second instance of the cluster:

```
[standalone@127.0.0.1:9990 /] connect 127.0.0.1:10090
```

Deploy the cluster application:

```
[standalone@127.0.0.1:10090 /] deploy \  
/home/student/JB248/labs/config-jgroups/cluster.war
```

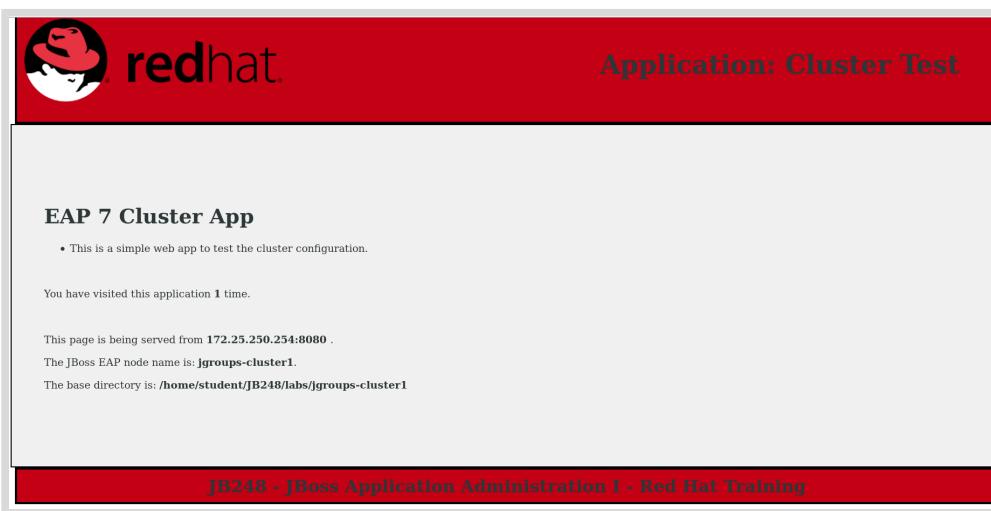
- 6.4. In the log files of both instances, verify that the application was successfully deployed. You should see a message similar to the following:

```
2016-05-25 03:41:24,749 INFO [org.jboss.as.server] (management-handler-thread -  
5) WFLYSRV0010: Deployed "cluster.war" (runtime-name : "cluster.war")
```

► 7. Test the cluster using the UDP stack

By default, EAP is set up for clustering using the UDP stack. In this mode, EAP nodes can automatically join and leave the cluster without any manual configuration, and the nodes discover each other using UDP multicasting.

- 7.1. Navigate to <http://172.25.250.254:8080/cluster> to view the test application running on the **jgroups-cluster1** instance. Refresh the page a few times and observe that the counter is incremented by one every time you refresh the page.



Navigate to `http://172.25.250.254:8180/cluster` to view the test application running on the **jgroups-cluster2** instance. Observe that the counter value has not been reset, but incremented by one. Refresh the page a few times and observe that the counter is incremented by one every time you refresh the page.

- 7.2. Shut down the **jgroups-cluster2** instance by pressing **Ctrl+C** on the terminal window where you started the instance. Navigate to `http://172.25.250.254:8080/cluster` to view the test application running on the **jgroups-cluster1** instance.
Observe that the counter value has not been reset, but reflects the latest value (plus one) as observed before shutting down **jgroups-cluster2**.
Refresh the page a few times and observe that the counter is incremented by one every time you refresh the page. The EAP clustering component uses the JGroups stack (by default, UDP) to replicate the counter value across all nodes in the cluster.
- 7.3. Shut down the **jgroups-cluster1** instance by pressing **Ctrl+C** on the terminal window where you started the instance.

► 8. Test the cluster using the TCP stack

In many data center networks, UDP multi-casting is disabled and you have to use TCP for clustering EAP nodes. In contrast to the UDP stack, where EAP nodes can be added to a cluster automatically, in the TCP stack configuration, the IP addresses of the cluster nodes are managed by the EAP Administrator manually.

- 8.1. Use the EAP CLI to define a new **TCP** stack configuration. Open the file `/home/student/JB248/labs/config-jgroups/new-tcp-stack.cli` in a text editor and review the commands. The contents of the file should be as follows:

```
# Add a new TCP stack called "tcpping"
batch
/subsystem="jgroups"/stack="tcpping":add()
/subsystem="jgroups"/stack="tcpping":add-protocol(type="TCPPING")
/subsystem="jgroups"/stack="tcpping"/transport="TRANSPORT":add(socket-
binding="jgroups-tcp",type="TCP")
run-batch

# Customize the protocol settings for tcpping
batch
/subsystem="jgroups"/stack="tcpping"/protocol="TCPPING"/
property="initial_hosts":add(value="node1[port1],node2[port2]")
/subsystem="jgroups"/stack="tcpping"/protocol="TCPPING"/
property="port_range":add(value="10")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="MERGE2")
/subsystem="jgroups"/stack="tcpping":add-protocol(socket-binding="jgroups-tcp-
fd",type="FD_SOCK")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="FD")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="VERIFY_SUSPECT")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="BARRIER")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="pbcast.NAKACK")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="UNICAST2")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="pbcast.STABLE")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="pbcast.GMS")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="UFC")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="MFC")
```

```
/subsystem="jgroups"/stack="tcpping":add-protocol(type="FRAG2")
/subsystem="jgroups"/stack="tcpping":add-protocol(type="RSVP")
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcpping)
run-batch
:reload
```

The **initial_hosts** property is a comma-separated list of server instances and the ports (in square brackets) that are to be a part of the cluster.

- 8.2. Edit the **initial_hosts** property and replace the values with the IP addresses and ports of the two EAP instances as follows (recall that the **jgroups-cluster2** instance is running with a **port-offset** value of **100**):

```
..."initial_hosts":add(value="172.25.250.254[7600],172.25.250.254[7700]")
```

- 8.3. Start both EAP instances again as outlined in Steps 3.1 and 3.2.
- 8.4. Run the following command in a new terminal on the **workstation** VM to create the new TCP stack on the **jgroups-cluster1** instance:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect \
--controller=localhost:9990 \
--file=/home/student/JB248/labs/config-jgroups/new-tcp-stack.cli
The batch executed successfully
...
```

- 8.5. Similarly, run the following command in a new terminal on the **workstation** VM to create the new TCP stack on the **jgroups-cluster2** instance:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect \
--controller=localhost:10990 \
--file=/home/student/JB248/labs/config-jgroups/new-tcp-stack.cli
The batch executed successfully
...
```

- 8.6. Verify that the a new TCP stack called **tcpping** is defined in the **/home/student/JB248/labs/jgroups-clusterX/configuration/standalone-full-ha.xml** files of both instances, where 'X' denotes the instance number:

```
<stack name="tcpping">
  <transport type="TCP" socket-binding="jgroups-tcp"/>
  <protocol type="TCPPING">
    <property name="initial_hosts">
      172.25.250.254[7600],172.25.250.254[7700]
    </property>
    <property name="port_range">
      10
    </property>
  </protocol>
```

```
<protocol type="MERGE2"/>
...
</stack>
```

Also verify that the default stack has been set to the new **tcpping** definition:

```
<channels default="ee">
  <channel name="ee" stack="tcpping"/>
</channels>
```

- 8.7. Repeat Steps 7.1 and 7.2 to test the new TCP stack. Verify that the test application behaves in a similar manner as when you configured the instances with the default UDP stack.
- 8.8. Observe the **tcpdump** command output as outlined in **Step 5** and verify that you do NOT see any UDP traffic on port **45688** because the nodes are using the TCP protocol for communication. Observe the TCP traffic on **localhost** port **7600** when both instances are running using the **tcpdump** command:

```
[student@workstation ~]$ sudo tcpdump -i lo tcp port 7600 -vvv
```

You should see output similar to the following:

```
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 65535 bytes
13:42:27.894727 IP (tos 0x0, ttl 64, id 56200, offset 0, flags [DF], proto TCP
(6), length 4225)
  workstation.lab.example.com.7600 > workstation.lab.example.com.54083:
  Flags [P.], cksum 0x5ea4 (incorrect -> 0xe452), seq 2980730036:2980734209, ack
  3719238263, win 10, options [nop,nop,TS val 34228000 ecr 34227223], length 4173
13:42:27.894782 IP (tos 0x0, ttl 64, id 27899, offset 0, flags [DF], proto TCP
(6), length 52)
  workstation.lab.example.com.54083 > workstation.lab.example.com.7600: Flags
[.], cksum 0x4e57 (incorrect -> 0x1e8b), seq 1, ack 4173, win 32742, options
[nop,nop,TS val 34228000 ecr 34228000], length 0
```

- 9. Stop the two EAP instances by pressing **Ctrl+C** on the terminal window where you started the instances.
Exit the EAP CLI sessions by typing **quit** or press **Ctrl+C** on the terminal window where you started the CLI sessions.
Exit the tail and **tcpdump** command sessions by pressing **Ctrl+C** on the terminal window where you started the commands.

This concludes the guided exercise.

Configuring Load Balancer

Objectives

After completing this section, students should be able to:

- Configure Undertow as a load balancer.

Load balancing JEE applications

Configuring Infinispan and JGroups to replicate HTTP sessions is only half of the EAP 7 web clustering equation. The other half is setting up a network load balancer to direct HTTP requests to EAP server instances that are cluster members.

Other EAP 7 clustered services, for example JMS clients and remote EJBs, do NOT require network load balancers. They use client software that is capable of handling load balancing and failover. Their client software also handles discovering existing cluster members and learning about other cluster topology changes.

Unfortunately the HTTP protocol does not provide the required load balancing and failover features. Adding this intelligence to the client would mean creating a new web browser, which is impractical. The solution is built over a standard HTTP protocol feature: proxy support.

A web proxy is a web server that receives HTTP requests and forwards them to another web server. It can be used to provide security, caching, and other features. For most web clients the proxy hides the real web servers that provide application content.

Having a front-end web server (or web proxy) between users and JEE application server is a common pattern for a number of reasons:

- Only the front-end web server needs to be accessible from the Internet, and the application servers do not need to be exposed to direct attacks.
- The front-end web server can serve static HTML pages, images, CSS, JavaScript, and other files, lifting some work from the application server.
- SSL termination and processing can be done by the front-end web server, so application servers have more CPU cycles available to handle application logic.
- A front-end web server may hide different kinds of application servers and present them as a coherent set of URLs, for example to make a PHP application and a Java EE application look as being part of the same web site.

Using a web proxy to perform load balancing among a number of other back-end web servers allows for efficient implementations because the web proxy understands the HTTP protocol and can implement things such as session affinity in an optimized way.

Session affinity, also known as sticky sessions, is a web proxy mechanism that sends all requests from the same user to the same back-end web server. Not having session affinity means each request from a user can go to a different back-end web server.

Generic network load balancers (working on the OSI model layers 3 or 4) can be used for a Java EE application server cluster, but they probably will not be as efficient as a web proxy (which works

on the OSI model layer 7). Some networking hardware embeds a web proxy and can be configured to work as layer 7 load balancers as well as software-based load balancers.

The reason a layer 7 load balancer is preferred over a layer 3 or 4 load balancer for a web application is that Java EE Web Containers usually employ an HTTP cookie to store a unique session ID. Only load balancers that understand HTTP cookies can balance user sessions with maximum granularity and keep session affinity at the same time. This is true for most web application runtimes, not just Java EE.

To understand the need and advantages of session affinity, consider the impact on stateful and stateless applications, and whether they are clustered or non-clustered:

- Stateless web applications should work correctly without session affinity because there is no user data to be preserved in an HTTP session object. Any back-end server can process any request and return the expected results.
- Having a clustered web container makes no difference for stateless web applications because there is no user session data to be replicated. They are usually deployed as non-clustered web applications to a non-clustered back-end server farm under a load balancer for increased scalability.
- Stateful web applications can work in a non-clustered environment only if the load balancer provides session affinity, because only one web back-end server has a specific user session data. Having a load balancer over a non-clustered web applications provides scalability advantages even if not providing transparent failover for the user: if a back-end server fails, all user sessions from the failed server are lost, but users can start again on a different server.
- Stateful web applications in a clustered environment have scalability advantages and provide transparent failover for all users. They should also work correctly without session affinity because all back-end servers have access to all user session data, so any back-end server would be able to answer any request and provide the expected results.

The previous discussion shows session affinity is an optional feature for many applications but it does not consider performance. All kinds of web applications usually perform faster and require less hardware resources with session affinity enabled because of data locality.

Even if an application works as intended without session affinity, a sequence of requests made by the same user is usually around the same set of data. This data is cached by many layers, from the back-end web server CPU to a possibly shared database instance. Having the same back-end web server perform the whole sequence maximizes utilization of these caches.

Undertow load balancer architecture and mod_cluster

Undertow is a generic and full-featured web server, capable of replacing traditional web server software such as Apache Httpd and Microsoft IIS for most use cases. As such it includes a web proxy component that can do load balancing.

Undertow is also a web server built specifically to support Java EE developers and administrators and provides most features from Java Web Containers such as Apache Tomcat. Among them, two are of special relevance to EAP clustering:

- **AJP** protocol support: The AJP protocol is a binary replacement for the text-based HTTP protocol. It also employs long-lived persistent connections, while HTTP connections are either single-request or short-lived (when using the HTTP 1.1 Keep-Alive feature). AJP was designed to lower the overhead imposed by a front-end web server on users accessing a Java Web Container. The idea is simple: a web browser uses HTTP to connect to the web proxy, and the web proxy uses AJP to connect to the back-end application servers.

- **mod_cluster** protocol support: The mod_cluster protocol allows a web proxy to dynamically discover back-end web servers and the applications made available by each, allowing a true dynamic environment. It also employs an additional HTTP connection to relay load metrics from each back-end application server to the web proxy so it can make better load-balancing decisions.

Traditional web load balancer software requires **static configuration**: each back-end web server's connection details have to be manually configured on the load balancer, and it keeps trying to connect to failed back-end web servers, generating more network overhead. Adding more back-end servers means reconfiguring and potentially restarting the load balancer. The following figure illustrates a traditional web proxy acting as a load balancer:

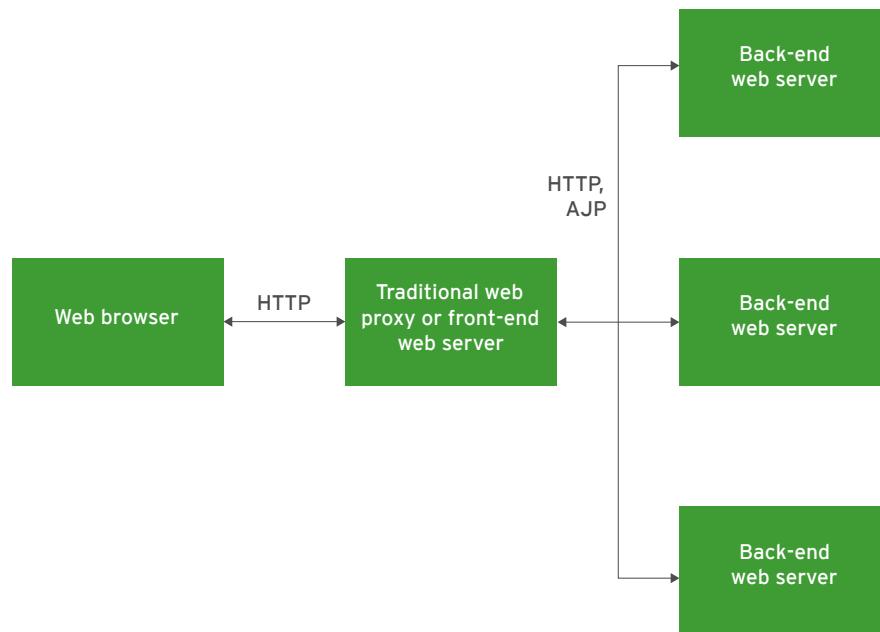


Figure 12.8: Traditional web proxy as a static load balancer.

For this discussion, a Java EE application server is considered to be just a special case of a back-end web server.

mod_cluster builds the back-end web server list dynamically and learns about new cluster members, new deployed applications, or failed cluster members without needing manual configuration.

The **mod_cluster** protocol requires a client component, that should be implemented by the load balancer, and a server part, that is implemented by the EAP 7 **modcluster** subsystem. The following figure illustrates a mod_cluster enhanced web proxy acting as a load balancer:

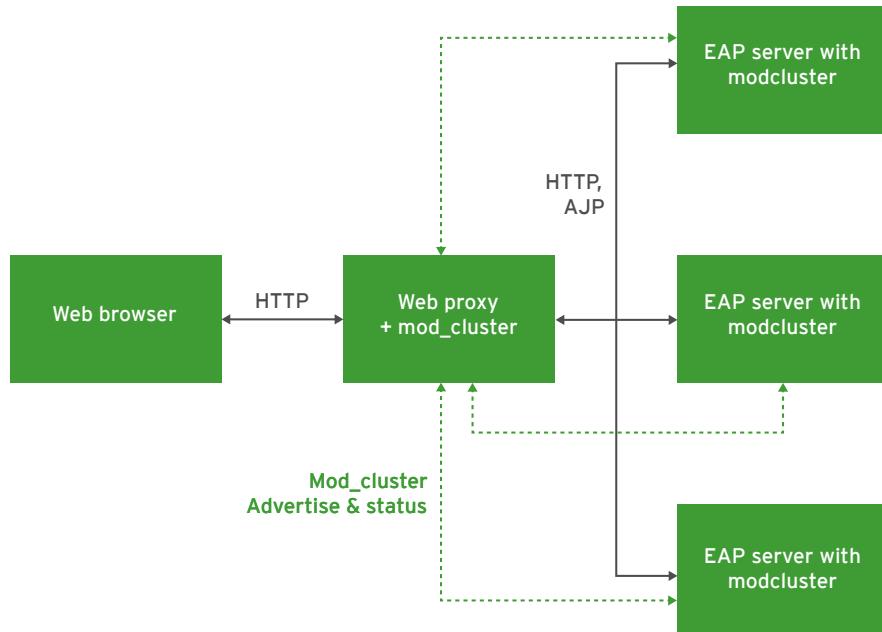


Figure 12.9: Web proxy enhanced with mod_cluster as a dynamic load balancer.

In the previous figure, the **web proxy + mod_cluster** box could be either an EAP 7 server instance with Undertow configured to enable **mod_cluster** or a native web server with a mod_cluster plug-in. The **EAP server with modcluster** box is a back-end web server.

A **mod_cluster** enhanced web proxy such as Undertow sends advertisement messages to all back-end web servers that are listening to the configured multicast address and port. Back-end web servers reply by sending the load balancer their connection parameters and deployed application context paths.

The architecture is fault-tolerant: there can be multiple **mod_cluster** clients on the same network; that is, multiple web proxies acting as load balancers. Back-end web servers receive the advertisement messages from all web proxies and replies to them. The load balancer is NOT a single point of failure.

For networks where multicast traffic is not allowed, advertising is disabled on the **mod_cluster** client. Each back-end web server is then manually configured with a list of web proxies. All of the proxies in the web server's list is sent connection parameters and updates about which applications are deployed and undeployed. Even without multicast a **mod_cluster** load balancer requires no static configuration.

Undertow can be configured to act as either a static (without **mod_cluster**) or dynamic (with **mod_cluster**) load balancer. In either case it is usually configured as a dedicated EAP server instance, where no applications are deployed. This dedicated server instance is NOT a cluster member. A server group consisting only of multiple dedicated load balancer EAP instances can be used to prevent having a single point of failure.

Configuring Undertow as a dynamic load balancer

Configuring Undertow as a dynamic load balancer involves the following high-level steps:

- Add a **mod_cluster** filter to the default Undertow server.
- Configure the advertisement settings in both the **undertow** and the **modcluster** subsystems.

- Either configure multicast parameters on both subsystems, or disable advertise in the **undertow** subsystem and configure a proxy list on the **modcluster** subsystem.

These steps are detailed below, using EAP CLI commands as examples.

Add the mod_cluster Filter to Undertow

The default **undertow** subsystem configuration does NOT include a **mod_cluster** filter, even in the **ha** and **full-ha** clustered EAP profiles. This filter has to be created and configured to use the correct multicast parameters by referring to a **socket-binding**.

To add the **mod_cluster** filter and configure it to default EAP settings, use the following command:

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:add(\n    management-socket-binding=http, advertise-socket-binding=modcluster)
```

The two attributes required by a **mod_cluster** filter are:

- management-socket-binding**: informs Undertow where to receive connection information and load balance metrics from back-end web servers. It should point to the **socket-binding** where EAP receives HTTP requests, which is **http** by default.
- advertise-socket-binding** informs Undertow where to send advertisement messages, that is, the multicast address and UDP port, by referring to a **socket-binding** name.

After creating the filter, it should be enabled in the desired Undertow (virtual) hosts:

```
/profile=ha/subsystem=undertow/server=default-server/host=default-host/filter-\nref=lb:add
```

Notice **lb** is the name assigned to the **mod_cluster** filter defined in the previous command.

Configure Advertise Using Multicast

Socket binding groups **ha-sockets** and **full-ha-sockets** already define the **modcluster** socket binding, which uses the multicast address **224.0.1.105** and port **23364**. The **undertow** subsystem uses this socket binding to know where to send advertise messages. The **modcluster** subsystem uses the same socket binding to know where to listen for advertise messages.

It is recommended to change the multicast address to prevent undesired EAP instances to try to become load balancers for the clustered EAP server instances.

It is also recommended to configure an advertise key shared by the **mod_cluster** client and server. To configure the advertise key on the application server instances, that is, on the cluster members:

```
/profile=ha/subsystem=modcluster/mod_cluster-config=configuration:\n    write-attribute(name=advertise-security-key,value=secret)
```

To configure the key on the load balancer server instance:

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:\n    write-attribute(name=security-key,value=secret)
```

Notice that the previous commands affect different EAP server instances: the first one, on the **modcluster** subsystem, affects EAP instances that are members of a cluster. The second one, on the **undertow** subsystem, affects the EAP instance that acts as the load balancer.

Disable Advertise and Configure a Proxy List

An Undertow dynamic load balancer can be configured to NOT use multicast by disabling advertise on the **mod_cluster** filter. This is done by setting the **advertise-frequency** attribute to zero and the **advertise-socket-binding** to null.

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:\\
write-attribute(name=advertise-frequency,value=0)
```

```
/profile=ha/subsystem=undertow/configuration=filter/mod_cluster=lb:\\
write-attribute(name=advertise-socket-binding,value=null)
```

In the previous example commands, **lb** is the name that was assigned to the **mod_cluster** filter.

Advertise also has to be disabled in cluster members **modcluster** subsystem so they are not listening for advertisement messages anymore:

```
/profile=ha/subsystem=modcluster/mod_cluster-config=configuration:\\
write-attribute(name=advertise,value=false)
```

Cluster members now require a proxy list to know which **mod_cluster** load balancer they should send connection parameters and application status to. Each load-balancer instance has to be configured as an outbound socket binding. Assuming a single load balancer instance:

```
/socket-binding-group=ha-sockets/remote-destination-outbound-socket-binding=lb:\\
add(host=10.1.2.3, port=8080)
```

The port in the outbound socket binding is the HTTP port of the load balancer EAP server instance.

These socket bindings are then used to configure the **proxies** list on the **modcluster** subsystem:

```
/profile=ha/subsystem=modcluster/mod_cluster-config=configuration:\\
write-attribute(name=proxies,value=[lb])
```

In the previous example, **lb** is the name that was assigned to the outbound socket binding.

Configuring Undertow as a static load balancer

Configuring Undertow as a static load balancer involves the following high-level steps:

- Add outbound socket bindings pointing to each cluster member.
- Add a **reverse-proxy** handler to the default Undertow server.
- Add each cluster member to the proxy handler.

These steps are detailed below with EAP CLI command examples.

Each cluster member IP address and AJP port has to be configured as an outbound socket binding. Assuming there are two cluster members:

```
/socket-binding-group=ha-sockets/remote-destination-outbound-socket-binding=\
cluster-member1:add(host=10.1.2.3, port=8009)
```

```
/socket-binding-group=ha-sockets/remote-destination-outbound-socket-binding=\
cluster-member2:add(host=10.1.2.13, port=8009)
```

The **reverse-proxy** handler is created in the **undertow** subsystem:

```
/profile=ha/subsystem=undertow/configuration=handler/reverse-proxy=lb:add
```

Each cluster member is added as a **route** to the **reverse-proxy** handler by using a **host** child:

```
/profile=ha/subsystem=undertow/configuration=handler/reverse-proxy=lb/
host=member1:\\
add(outbound-socket-binding=cluster-member1, scheme=ajp, instance-id=hosta:server1, \
path=/app)
```

```
/profile=ha/subsystem=undertow/configuration=handler/reverse-proxy=lb/
host=member2:\\
add(outbound-socket-binding=cluster-member2, scheme=ajp, instance-id=hostb:server2, \
path=/app)
```

In the previous commands, each route **instance-id** attribute has to match the **jboss.server.node** system property for the referred node. This system property usually takes the form **host_name:server_name**. The **path** attribute value has to match the clustered application context path.

Enable the **reverse-proxy** handler on the desired Undertow (virtual) hosts:

```
/profile=ha/subsystem=undertow/server=default-server/host=default-host/\
location=\app:add(handler=lb)
```

In the previous example command, it was necessary to escape the forward slash (/) in **/app** by using a back slash (\) because the forward slash is part of the CLI syntax.

Notice that this example only configures the load balancer for a single clustered application, whose context path is **/app**. If more applications from the same cluster are to be supported by the same load balancer, add each one as a new host child with a different path attribute.

How Undertow load balancers implement session affinity

Java EE Web Containers usually employ the HTTP cookie named **JSESSIONID** to store a unique session id as mandated by the Servlet API specification. This cookie value is used as an index to retrieve user session data stored in memory.

Undertow uses the session id cookie to provide an efficient session affinity implementation by appending to its value the **jboss.server.node** system property and then matching this extra data to a route **instance-id** attribute. Explaining in more detail:

- The EAP 7 Servlet container appends the **jboss.server.node** system property to the HTTP session id cookie when sending a response to the web browser.
- The session cookie is saved in the web browser memory, and sent back to the web server with the next HTTP request.
- The load balancer intercepting the request extracts the value appended to the cookie sent with the request and forwards the request to the back-end web server whose route **instance-id** attribute matches the value.



Note

Be aware that EAP server and host names containing symbols can break the load balancer's ability to extract the value appended to the session cookie, and this will break session affinity. If this happens, set the **jboss.server.node** system property on each node to a safe value, using only letters and digits.

The dynamic load balancer also uses an **instance-id** attribute in the same way a static load balancer does. What mod_cluster really does is create load balancing configurations at runtime. The **modcluster** subsystem in the cluster member servers provides all necessary information (including the **jboss.server.node** system property value) and the **mod_cluster** filter creates a hidden **reverse_proxy** handler and configures its routes.

Using Apache Httpd and other web servers as load balancers

Earlier EAP releases had no Undertow subsystem, and the embedded web server, provided by the JBoss Web subsystem and based on Apache Tomcat, provided no proxy capabilities.

Red Hat supported a number of native web server plug-ins as load balancers, for many OSes and web servers. Some of them are still supported by Red Hat through the **Red Hat JBoss Core Services** product, which is included as part of the EAP 7 subscription.

JBoss Core Services provides the following load balancers, based on Apache Httpd native modules for Linux, Windows and Solaris OSes:

- **mod_jk**: this was the first module that implemented a static load balancer based on the AJP protocol and was developed by the Tomcat community. It also works with Microsoft IIS, but this particular configuration is NOT supported anymore as part of the EAP subscription.
- **mod_proxy**: this is the Apache Httpd native proxy support, and it can work as a static load balancer using either HTTP or AJP.
- **mod_cluster**: this is the **mod_cluster** client for Apache Httpd. It works alongside Apache Httpd mod_proxy to provide a dynamic load balancer using either HTTP or AJP.

Configuration details for the load balancers follow the same concepts already presented for Undertow. Details and configuration syntax for each one are presented in the **EAP 7 Configuration Guide**.



References

For more information about Undertow and ModCluster:

EAP 7 Configuration Guide, see Chapter "Configuring High Availability" section "Configuring JBoss EAP as a Front-end Load Balancer"

<https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0/configuration-guide/configuration-guide>

Undertow web server project

<http://undertow.io>

Undertow community documentation

<http://undertow.io/undertow-docs/undertow-docs-1.3.0/index.html>

For more information about native Apache Httpd based load balancers:

Red Hat JBoss Core Services documentation

<https://access.redhat.com/documentation/en/red-hat-jboss-core-services/>

mod_cluster Apache Httpd module community page

<http://mod-cluster.jboss.org/>

► Guided Exercise

Configuring Load Balancing

In this lab, you will load balance HTTP requests among a predefined set of servers using Undertow.

Resources	
Files:	/home/student/JB248/labs/standalone /home/student/JB248/labs/config-lb
Application URLs:	http://172.25.250.254:8080/cluster http://172.25.250.254:8180/cluster http://172.25.250.254:8280/cluster
Resources	/home/student/JB248/labs/config-lb/ cluster.war

Outcomes

You should be able to configure a load balancer that will be responsible for balancing the requests between the available hosts in the cluster.

Before You Begin

Run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, that the previous guided exercise has been completed, to create the required files, and to download the cluster.war application.

```
[student@workstation ~]$ lab config-lb setup
```

► 1. Start the Standalone EAP Servers

In this guided exercise, three instances of standalone running the **standalone-ha.xml** configuration will be required. The first one will be the load balancer and the others will be part of a cluster.

- 1.1. The first standalone server will be used as a load balancer with the following characteristics:

- Use the **/home/student/JB248/labs/standalone-instance** folder as the base directory.
- Use the **standalone-ha.xml** configuration file.
- Defines the public interface IP as 172.25.250.254.

To configure it, run the following commands from the workstation VM:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh --server-config=standalone-ha.xml \  
-Djboss.server.base.dir=/home/student/JB248/labs/standalone-instance \  
-Djboss.bind.address=172.25.250.254
```

**Note**

The **ha** configuration file is not required to create a load balancer. It is used here to avoid problems with other labs that are using the **default** configuration file.

- 1.2. The second standalone server will be part of the cluster and it should have the following characteristics:
 - Use the **/home/student/JB248/labs/config-lb/server1** folder as the base directory.
 - Use the **standalone-ha.xml** configuration file.
 - Defines the public interface IP as **172.25.250.254**.
 - To avoid port conflicts, use the value of **100** as a port offset.
 - Defines the node name as **server1**

To configure it, run the following commands from the workstation VM:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh --server-config=standalone-ha.xml \  
-Djboss.server.base.dir=/home/student/JB248/labs/config-lb/server1 \  
-Djboss.bind.address=172.25.250.254 -Djboss.socket.binding.port-offset=100 \  
-Djboss.node.name=server1
```

**Note**

The node name is required to use the sticky session feature.

- 1.3. The third standalone server will be part of the cluster and it should have the following characteristics:
 - Use the **/home/student/JB248/labs/config-lb/server2** folder as the base directory.
 - Use the **standalone-ha.xml** configuration file.
 - Defines the public interface IP as **172.25.250.254**.
 - To avoid port conflicts, use the value of **200** as a port-offset.
 - Defines the node name as **server2**.

To configure it, run the following commands from the workstation VM:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./standalone.sh --server-config=standalone-ha.xml \  
-Djboss.server.base.dir=/home/student/JB248/labs/config-lb/server2 \  
-Djboss.bind.address=172.25.250.254 -Djboss.socket.binding.port-offset=200 \  
-Djboss.node.name=server2
```

► 2. Deploy the cluster.war application

The cluster application should be deployed onto the two Standalone servers of the cluster. The application should be deployed twice, one for each host.

- 2.1. Open a new terminal window and run the CLI tool connecting to the first instance of the cluster with the following commands:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ ./jboss-cli.sh -c --controller=127.0.0.1:10090
```

- 2.2. Deploy the cluster application:

```
[standalone@127.0.0.1:10090 /] deploy \  
/home/student/JB248/labs/config-lb/cluster.war
```

- 2.3. Connect to the second instance of the cluster:

```
[standalone@127.0.0.1:10090 /] connect 127.0.0.1:10190
```

Deploy the cluster application:

```
[standalone@127.0.0.1:10190 /] deploy \  
/home/student/JB248/labs/config-lb/cluster.war
```

► 3. Test the cluster

Open a web browser and point it to <http://172.25.250.254:8180/cluster>. You should see the cluster application. Notice that you have some information about the instance that replied to the request like the base directory.

Refresh the page some times to increase the number of visits and then point the browser to <http://172.25.250.254:8280/cluster>. Notice that the total of visits was increased by one, even though you are running the application from a different server, which means the application is clustered.

► 4. Configure the Load Balancer

Although the cluster is configured, each server must be accessed individually. Configure a new load balancer using first EAP instance (the one without a port offset).

- 4.1. Return to the CLI tool, and connect to the load balancer instance:

```
[standalone@127.0.0.1:10190 /] connect 127.0.0.1:9990
```

- 4.2. To configure a new load balancer, the Undertow subsystem must support a reverse proxy that will load balance the request between two servers. Create a reverse proxy named **cluster-handler**:

```
[standalone@127.0.0.1:9990 /] /subsystem=undertow/configuration=\
handler/reverse-proxy=cluster-handler:add
```

- 4.3. Create a new socket binding that will be used by the AJP protocol running on the server named **server1** (172.25.250.254:8109). Name it **remote-server1**:

```
[standalone@127.0.0.1:9990 /] /socket-binding-group=standard-sockets\
/remote-destination-outbound-socket-binding=remote-server1\
:add(host=172.25.250.254, port=8109)
```



Note

The standalone server named server1 was started with a port offset of **100**, hence the AJP port will be **8109**.

- 4.4. Create in Undertow a reference to the socket binding created previously to the **server1** and bind it with the scheme AJP and it should access the cluster context path:

```
[standalone@127.0.0.1:9990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=cluster-handler/host=server1\
:add(outbound-socket-binding=remote-server1, scheme=ajp, \
instance-id=server1, path=/cluster)
```



Note

To enable the sticky session, the **instance-id** attribute should have the same value specified by the **jboss.node.name** System Property.

- 4.5. Create a new socket binding that will be used by the AJP protocol running on the server named **server2** (172.25.250.254:8209). Name it **remote-server2**:

```
[standalone@127.0.0.1:9990 /] /socket-binding-group=standard-sockets\
/remote-destination-outbound-socket-binding=remote-server2\
:add(host=172.25.250.254, port=8209)
```

- 4.6. Create in Undertow a reference to the socket binding created to the **server2** and bind it with the AJP scheme and access the cluster context path:

```
[standalone@127.0.0.1:9990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=cluster-handler/host=server2\
:add(outbound-socket-binding=remote-server2, scheme=ajp, \
instance-id=server2, path=/cluster)
```

- 4.7. Create a new reverse proxy location named **/cluster** referring to the **cluster-handler** handler:

```
[standalone@127.0.0.1:9990 /] /subsystem=undertow/server=default-server/\
host=default-host\
/location=/cluster:add(handler=cluster-handler)
```

► 5. Test the load balancer

- 5.1. Open a web browser and point it to `http://172.25.250.254:8080/cluster`. You should see the cluster application. Refresh the browser sometimes and notice that each request is served by the same server. Determine which EAP is handling your current request looking at the **The JBoss EAP node name is** label in the application.
- 5.2. Stop the host that is handling the requests by pressing **Ctrl+C** in the appropriate terminal window.
- 5.3. Return to the web browser and refresh the page. The load balancer passes your request to the other server without losing the current visits value.

► 6. Clean Up

- 6.1. Exit the EAP CLI:

```
[standalone@localhost:9990 /] exit
```

- 6.2. Stop the instances of EAP by pressing **Ctrl+C** in the terminal window that is running EAP.

This concludes the guided exercise.

Deploying HA Singleton Applications

Objectives

After completing this section, students should be able to:

- Configure and deploy a high availability application.

The need for cluster singletons

A **singleton** is a popular software development **design pattern** where there is a single shared instance of an object for the whole application. The usual implementation strategies for the singleton design pattern create one instance for each cluster member and this may have unintended consequences for an application, for example data consistency issues.

Sometimes the application logic requires that only one instance be running for the whole cluster. A developer could extract this instance to run as a service that is deployed into a single non-clustered EAP instance and is accessed by a clustered application, but then the service would become a single point of failure.

Many developers believe a `@Singleton` EJB works correctly as a clustered singleton but this is NOT true by the EJB specification. It states that each JVM in a distributed application server gets one instance. All ways to implement a singleton using only standard JVM or JEE features also creates one instance per JVM, that is, one instance for each EAP server in a cluster.

When an application requires a singleton to work as a cluster-wide singleton, and still have high availability, developers and EAP administrators have to work together to extract the singleton parts from the clustered application and adapt it to use one of the following proprietary EAP 7 approaches:

- Singleton deployments (similar to the feature from EAP 5)
- Singleton services (enhanced version of the EAP 6 feature)

Both approaches allow an application to work as an active-passive HA application inside an EAP cluster. They differ on the configuration details and changes required to application packaging and source code.

The singleton subsystem

Both singleton deployments and singleton services are managed by the **singleton** subsystem. It employs an Infinispan cache to register all known singleton deployments and singleton services and which cluster member runs each application. The EAP server instance that runs a singleton application is called the **master** server for that application.

The **singleton** subsystem can be configured with different election policies that define the master. Two kinds of election policy are provided by EAP 7:

- simple**: the first node to join the cluster runs the singleton application.
- random**: a random node is selected to run the singleton application.

An election policy can optionally specify a preferred server which, when available, will be the master for ALL singleton applications under that policy. The policy refers to the node name as specified by the **jboss.node.name** system property.

The default EAP 7 configuration files provide a **simple** election policy named **default** with no preferred server. This policy is shown by the following listing:

```
...
<subsystem xmlns="urn:jboss:domain:singleton:1.0">
    <singleton-policies default="default">
        <singleton-policy name="default" cache-container="server">
            <simple-election-policy/>
        </singleton-policy>
    </singleton-policies>
</subsystem>
...
```

The following commands shows how to create a new election policy named **custom** of type **random**, which prefers the server named **servera1**:

```
/subsystem=singleton singleton-policy=custom/election-policy=random:add()
```

```
/subsystem=singleton singleton-policy=custom/election-policy=random:list-add(\n    name=name-preferences, value=servera1)
```

If the master server fails, the **singleton** subsystem runs a new election for all singleton applications that were using the failed server as their master. These applications are then restarted in new master servers. Each singleton application has to provide their own means to recover or recreate in-memory data lost in the failed master server.

A potential issue with a singleton application is when there is a network partition, also known as the **split-brain** scenario: Two sets of servers from the same cluster cannot connect to each other, but servers from one set have no issue connecting to servers from the same set. Each set of servers think all servers from the other set failed and continue to work as a surviving cluster.

A split-brain scenario has two (or more) independent clusters where there should be a single cluster. This can easily lead to data consistency issues. To prevent that, an election policy can specify a quorum; that is, the minimum required number of cluster members. If a quorum is not reached, all remaining cluster members are shut down.

To configure a quorum of three servers, use the following command:

```
/subsystem=singleton singleton-policy=custom:write-attribute(name=quorum, value=3)
```

The quorum should be at least **N/2+1** where **N** is the anticipated total number of cluster members.

Singleton deployments

Singleton deployments are similar to **HASingletonDeployer** from EAP 5 and earlier. There was no similar feature in EAP 6. It is a way to mark a deployment as a cluster-wide singleton without the need to use proprietary EAP 7 APIs.

An application package is considered to be a singleton deployment if it contains the proprietary deployment descriptor **/META-INF/singleton-deployment.xml**. This file refers to the election policy to be used for the singleton application, and the following example refers to the **custom** policy from the previous example:

```
<singleton-deployment xmlns="urn:jboss:singleton-deployment:1.0" policy="custom"/>
```

Although this approach does not require an application to use EAP 7 proprietary APIs, it still requires the original clustered application to access the extracted singleton application, for example, using remote EJB calls or JMS.

Singleton services

Singleton services are implemented the same way as internal EAP services, that is, their source code uses WildFly Core APIs. Although more intrusive than singleton deployments, singleton services provide the following advantages in very specific use cases:

- The election policy can be defined by the application, without the need to configure the **singleton** subsystem.
- It allows EAP modules to start cluster-wide singleton services similar to the old JBossMQ from EAP 4. That is, it allows an EAP module to work as an active-passive clustered service.

Teaching students how to program a singleton service is outside the scope of this book.



References

For more information about HA Singletons, see the upstream documentation:

Wildfly 10 HA Singleton Features

<https://docs.jboss.org/author/display/WFLY10/HA+Singleton+Features>

For information about singleton EJBs and clustered environments:

JSR-345: EJB 3.2 specification

<https://jcp.org/aboutJava/communityprocess/final/jsr345/index.html>

For information about how to develop singleton services:

EAP 7 Developing EJB Applications

<https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0/developing-ejb-applications/developing-ejb-applications>

► Quiz

Deploying an HA Singleton

Choose the correct answer to the following questions:

► 1. Which file makes a JEE deployment into an EAP 7 singleton application? (Choose one.)

- a. /META-INF/cluster-singleton.xml
- b. /META-INF/singleton-deployment.xml
- c. /META-INF/singleton.xml
- d. /META-INF/singleton-app.xml
- e. /META-INF/jboss-deployment.xml

► 2. Which of the following can be configured as part of a clustered singleton election policy? (Choose three.)

- a. Pick a random server as the master.
- b. Pick a specific server to be the master.
- c. Pick the newest running server as the master.
- d. Pick the oldest running server as the master.
- e. Pick the server with higher available CPU as the master.

► 3. Which of these use cases can be implemented by using a singleton deployment? (Choose two.)

- a. Running an application only in the cluster member with a bigger and free JVM heap.
- b. Running an application in a single cluster member to avoid database deadlock issues.
- c. Running an application only in the cluster member with a specific hardware accelerator card.
- d. Running an application only in the cluster member with lower network latency to an external back-end server.
- e. Running an application in a single cluster member to prevent multiple concurrent writes to the same shared file.

► 4. A cluster is expected to have between 7 and 9 active members. Which would be the minimum partition size for a singleton master election policy in this cluster? (Choose one.)

- a. Three
- b. Four
- c. Five
- d. Six

► Solution

Deploying an HA Singleton

Choose the correct answer to the following questions:

- ▶ 1. Which file makes a JEE deployment into an EAP 7 singleton application? (Choose one.)
 - a. /META-INF/cluster-singleton.xml
 - b. /META-INF/singleton-deployment.xml
 - c. /META-INF/singleton.xml
 - d. /META-INF/singleton-app.xml
 - e. /META-INF/jboss-deployment.xml

- ▶ 2. Which of the following can be configured as part of a clustered singleton election policy? (Choose three.)
 - a. Pick a random server as the master.
 - b. Pick a specific server to be the master.
 - c. Pick the newest running server as the master.
 - d. Pick the oldest running server as the master.
 - e. Pick the server with higher available CPU as the master.

- ▶ 3. Which of these use cases can be implemented by using a singleton deployment? (Choose two.)
 - a. Running an application only in the cluster member with a bigger and free JVM heap.
 - b. Running an application in a single cluster member to avoid database deadlock issues.
 - c. Running an application only in the cluster member with a specific hardware accelerator card.
 - d. Running an application only in the cluster member with lower network latency to an external back-end server.
 - e. Running an application in a single cluster member to prevent multiple concurrent writes to the same shared file.

- ▶ 4. A cluster is expected to have between 7 and 9 active members. Which would be the minimum partition size for a singleton master election policy in this cluster? (Choose one.)
 - a. Three
 - b. Four
 - c. Five
 - d. Six

▶ Lab

Deploying Clustered Applications

In this lab, you will configure a cluster of EAP servers in a managed domain and deploy a cluster-aware application.

Resources	
Files	<code>/opt/domain</code> <code>/opt/standalone</code> <code>/tmp/cluster.war</code> <code>/tmp/new-tcp-stack.cli</code>
Application URL	<code>http://172.25.250.254:9080/cluster</code>

Outcome

You should be able to configure a cluster of EAP server instances and deploy a cluster-aware application to test load balancing and failover.

Before You Begin

Use the following command to download the relevant lab files, create the required lab folders, and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab clustering-lab-final setup
```

You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively, and the domain controller on the **workstation**. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and configure a cluster of two nodes (**servera.1** and **serverb.1**). You will also run a **mod_cluster** load balancer on a standalone server instance, which runs on the **workstation** VM.

1. Open several ports on the **workstation**, **servera** and **serverb** VMs for this lab. Briefly review the firewall configuration of the **workstation** VM in the file **/tmp/jgroups-firewall-rules-workstation.sh**.

Execute the following script on the **workstation** VM:

```
[student@workstation ~]$ sudo sh /tmp/jgroups-firewall-rules-workstation.sh
```

Open a new terminal on the **servera** VM. Briefly review the firewall configuration of the **servera** VM in the file **/tmp/jgroups-firewall-rules-servera.sh**.

Execute the following script on the **servera** VM:

```
[student@servera ~]$ sudo sh /tmp/jgroups-firewall-rules-servera.sh
```

Open a new terminal on the **serverb** VM. Briefly review the firewall configuration of the **serverb** VM in the file **/tmp/jgroups-firewall-rules-serverb.sh**.

Execute the following script on the **serverb** VM:

```
[student@serverb ~]$ sudo sh /tmp/jgroups-firewall-rules-serverb.sh
```

- Verify that the ports in the firewall are open by running the following command on **ALL** the three VMs:

```
[student@workstation ~]$ firewall-cmd --list-all --zone=public
```

- Start the domain controller on the **workstation** VM. Because the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

- Start the load-balancer instance on the **workstation** VM. The setup script has created the load balancer configuration in the **/opt/standalone** folder on **workstation**. Use **/opt/standalone** as the value of the **jboss.server.base.dir** argument that you pass to the **standalone.sh** start-up script. Also note that you are going to start the load balancer with the **standalone-ha.xml** and a port offset of **1000** to avoid port clashes with the domain controller running on the workstation. Because the load balancer is going to be the entry point for the application, it must be configured to listen on the public IP of the workstation (172.25.250.254). (Hint: Pass the **-c standalone-ha.xml -Djboss.socket.binding.port-offset=1000 -Djboss.bind.address=172.25.250.254** argument to **standalone.sh**.)

Note that the **/opt/standalone** directory is owned by the **jboss** user, so you must start the load balancer using **sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh ...**

5. You have been instructed to configure a cluster of two EAP server instances with the following specifications:
 - You should use the **mod_cluster** based dynamic load balancer and the load balancer should load balance requests among the servers in **Group1** (**servera.1** and **serverb.1**).
 - The **cluster.war** application must be deployed on the **Group1** server group.
 - TCP based clustering should be configured for the cluster (for cluster communication between EAP server instances).
 - The default UDP based auto-discovery of the front-end load-balancer from the back-end EAP server instances must be disabled. The EAP server instances should be configured with a static list of **proxies** (load balancers).
 - Failure of one server instance must be handled transparently and the cluster test application must continue functioning without disruption (high availability and failover).

5.1. Configure a TCP based cluster of EAP servers in the managed domain

Use the EAP CLI to define a new **TCP** stack configuration. Open the file **/tmp/new-tcp-stack.cli** on the **workstation** VM and review the commands listed in it. Note that this should be edited as the **jboss** user.

Edit the **initial_hosts** property and replace the values with the host name and ports of the two EAP server instances that should be part of the cluster.

- 5.2. Execute the EAP CLI script file on the domain controller.
- 5.3. Launch the EAP CLI and connect to the domain controller to configure the servers in the managed domain.
- 5.4. Configure the mod_cluster subsystem. By default, EAP is set up for advertising its status to load balancers using UDP multicasting. Disable advertising in the **mod_cluster** subsystem for the **full-ha** profile.
- 5.5. Because you have disabled advertising, you need to configure the back-end EAP nodes with a list of **proxies** (load balancers). Configure the EAP back-end nodes to communicate with the load balancer running on the **workstation** VM. Ensure you add an outbound socket binding that points to the load balancer IP address and port (172.25.250.254:9080).
- 5.6. Changing the default stack requires that you reload the managed domain configuration. Reload the domain controller using the EAP CLI.

6. Configure the load balancer

- 6.1. Launch the EAP CLI and connect to the load balancer instance on **workstation**.
- 6.2. Configure the **modcluster** subsystem to act as a front-end load balancer. Set a unique password for the **advertise-security-key**.
- 6.3. Configure the mod_cluster filter. Advertise the load balancer using the **modcluster** socket-binding and use the HTTP protocol for the management socket binding. Ensure that the **security-key** attribute matches the **advertise-security-key** you configured in the previous step.
- 6.4. As a final step, bind the **modcluster** filter to the undertow **default-server**.

- 6.5. Reload the load balancer configuration.
7. The two host controllers on **servera** and **serverb** connect to the domain controller and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.
 - 7.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
 - 7.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
 - 7.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
 - 7.4. Observe the load balancer terminal window and verify that the load balancer registers the two server instances (nodes) from Group1 that you are using in this lab.
8. **Deploy and test the cluster test application.**
 - 8.1. Using the EAP CLI, stop the servers in the server group **Group2** because they are not used in this lab.
 - 8.2. Deploy the cluster test application to the server group **Group1**. It is available at **/tmp/cluster.war** on the **workstation** VM.
 - 8.3. Observe the load balancer terminal window and verify that both server instances **servera.1** and **serverb.1** have been registered on the load balancer and they are ready to serve the **/cluster** context.
 - 8.4. Navigate to the load balancer at **http://172.25.250.254:9080/cluster** using a browser on the workstation VM. You should see the cluster application. Refresh the browser multiple times and notice that each request is served by the same server (due to session stickiness). Determine which server instance is handling your current request by looking at the **The JBoss EAP node name is** label in the application.
 - 8.5. Stop the host that is actively handling the requests by pressing **Ctrl+C** in the appropriate terminal window to shut down the entire host controller.
 - 8.6. Observe the load balancer terminal window and verify that the server you killed and the **/cluster** context for that server instance have been unregistered from the load balancer.

- 8.7. Return to the web browser and refresh the page. The load balancer fails over your request to the other remaining server without losing the current visits value.

9. Clean Up and Grading

- 9.1. Press **Ctrl+C** in the terminal window where you started the host controllers and the domain controller to stop the managed domain. (Alternatively, you can shut down the domain controller using the JBoss EAP CLI command `/host=master:shutdown()`.)
- 9.2. Press **Ctrl+C** to shut down the load balancer instance in the terminal window where you started it.
- 9.3. Press **Ctrl+C** to exit the EAP CLI if you used the CLI in the lab. (Alternatively, you can exit the CLI by typing **exit**.)
- 9.4. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab clustering-lab-final grade
```

This concludes the lab.

► Solution

Deploying Clustered Applications

In this lab, you will configure a cluster of EAP servers in a managed domain and deploy a cluster-aware application.

Resources	
Files	<code>/opt/domain</code> <code>/opt/standalone</code> <code>/tmp/cluster.war</code> <code>/tmp/new-tcp-stack.cli</code>
Application URL	<code>http://172.25.250.254:9080/cluster</code>

Outcome

You should be able to configure a cluster of EAP server instances and deploy a cluster-aware application to test load balancing and failover.

Before You Begin

Use the following command to download the relevant lab files, create the required lab folders, and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab clustering-lab-final setup
```

You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively, and the domain controller on the **workstation**. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and configure a cluster of two nodes (**servera.1** and **serverb.1**). You will also run a **mod_cluster** load balancer on a standalone server instance, which runs on the **workstation** VM.

1. Open several ports on the **workstation**, **servera** and **serverb** VMs for this lab. Briefly review the firewall configuration of the **workstation** VM in the file **/tmp/jgroups-firewall-rules-workstation.sh**.

Execute the following script on the **workstation** VM:

```
[student@workstation ~]$ sudo sh /tmp/jgroups-firewall-rules-workstation.sh
```

Open a new terminal on the **servera** VM. Briefly review the firewall configuration of the **servera** VM in the file **/tmp/jgroups-firewall-rules-servera.sh**.

Execute the following script on the **servera** VM:

```
[student@servera ~]$ sudo sh /tmp/jgroups-firewall-rules-servera.sh
```

Open a new terminal on the **serverb** VM. Briefly review the firewall configuration of the **serverb** VM in the file **/tmp/jgroups-firewall-rules-serverb.sh**.

Execute the following script on the **serverb** VM:

```
[student@serverb ~]$ sudo sh /tmp/jgroups-firewall-rules-serverb.sh
```

- Verify that the ports in the firewall are open by running the following command on **ALL** the three VMs:

```
[student@workstation ~]$ firewall-cmd --list-all --zone=public
```

- Start the domain controller on the **workstation** VM. Because the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ --host-config=host-master.xml
```

- Start the load-balancer instance on the **workstation** VM. The setup script has created the load balancer configuration in the **/opt/standalone** folder on **workstation**. Use **/opt/standalone** as the value of the **jboss.server.base.dir** argument that you pass to the **standalone.sh** start-up script. Also note that you are going to start the load balancer with the **standalone-ha.xml** and a port offset of **1000** to avoid port clashes with the domain controller running on the workstation. Because the load balancer is going to be the entry point for the application, it must be configured to listen on the public IP of the workstation (172.25.250.254). (Hint: Pass the **-c standalone-ha.xml -Djboss.socket.binding.port-offset=1000 -Djboss.bind.address=172.25.250.254** argument to **standalone.sh**.)

Note that the **/opt/standalone** directory is owned by the **jboss** user, so you must start the load balancer using **sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh ...**

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/standalone.sh \
-Djboss.server.base.dir=/opt/standalone/ -Djboss.bind.address=172.25.250.254 \
-Djboss.socket.binding.port-offset=1000 -c standalone-ha.xml
```

5. You have been instructed to configure a cluster of two EAP server instances with the following specifications:
- You should use the **mod_cluster** based dynamic load balancer and the load balancer should load balance requests among the servers in **Group1** (**servera.1** and **serverb.1**).
 - The **cluster.war** application must be deployed on the **Group1** server group.
 - TCP based clustering should be configured for the cluster (for cluster communication between EAP server instances).
 - The default UDP based auto-discovery of the front-end load-balancer from the back-end EAP server instances must be disabled. The EAP server instances should be configured with a static list of **proxies** (load balancers).
 - Failure of one server instance must be handled transparently and the cluster test application must continue functioning without disruption (high availability and failover).

5.1. Configure a TCP based cluster of EAP servers in the managed domain

Use the EAP CLI to define a new **TCP** stack configuration. Open the file **/tmp/new-tcp-stack.cli** on the **workstation** VM and review the commands listed in it. Note that this should be edited as the **jboss** user.

Edit the **initial_hosts** property and replace the values with the host name and ports of the two EAP server instances that should be part of the cluster.

```
... "initial_hosts":add(value="servera[7600],serverb[7600]")
```

5.2. Execute the EAP CLI script file on the domain controller.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin  
[student@workstation bin]$ sudo -u jboss \  
/opt/jboss-eap-7.0/bin/jboss-cli.sh \  
--connect --controller=172.25.250.254:9990 \  
--file=/tmp/new-tcp-stack.cli  
The batch executed successfully  
...
```

5.3. Launch the EAP CLI and connect to the domain controller to configure the servers in the managed domain.

In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss \  
/opt/jboss-eap-7.0/bin/jboss-cli.sh \  
--connect --controller=172.25.250.254:9990
```

5.4. Configure the mod_cluster subsystem. By default, EAP is set up for advertising its status to load balancers using UDP multicasting. Disable advertising in the **mod_cluster** subsystem for the **full-ha** profile.

```
[domain@172.25.250.254:9990 /]/profile=full-ha/subsystem=modcluster\  
/mod-cluster-config=configuration\  
:write-attribute(name=advertise,value=false)
```

- 5.5. Because you have disabled advertising, you need to configure the back-end EAP nodes with a list of **proxies** (load balancers). Configure the EAP back-end nodes to communicate with the load balancer running on the **workstation** VM. Ensure you add an outbound socket binding that points to the load balancer IP address and port (172.25.250.254:9080).

```
[domain@172.25.250.254:9990 /]/socket-binding-group=full-ha-sockets\  
/remote-destination-outbound-socket-binding=lb:\  
add(host=172.25.250.254,port=9080)
```

Next, add the proxies to the mod_cluster configuration:

```
[domain@172.25.250.254:9990 /]/profile=full-ha/subsystem=modcluster\  
/mod-cluster-config=\  
configuration:list-add(name=proxies,value=lb)
```

- 5.6. Changing the default stack requires that you reload the managed domain configuration. Reload the domain controller using the EAP CLI.

```
[domain@172.25.250.254:9990 /] reload --host=master
```

6. Configure the load balancer

- 6.1. Launch the EAP CLI and connect to the load balancer instance on **workstation**.

In a new terminal window on the workstation, start the EAP CLI and connect to the load balancer as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss \  
/opt/jboss-eap-7.0/bin/jboss-cli.sh \  
--connect --controller=localhost:10990
```

- 6.2. Configure the **modcluster** subsystem to act as a front-end load balancer. Set a unique password for the **advertise-security-key**.

```
[standalone@localhost:10990 /] /subsystem=modcluster/mod-cluster-config=\  
configuration:write-attribute\  
(name=advertise-security-key, value=redhat)
```

- 6.3. Configure the mod_cluster filter. Advertise the load balancer using the **modcluster** socket-binding and use the HTTP protocol for the management socket binding. Ensure that the **security-key** attribute matches the **advertise-security-key** you configured in the previous step.

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=\
filter/mod-cluster=modcluster:add\
(management-socket-binding=http, advertise-socket-binding=modcluster, \
security-key=redhat)
```

- 6.4. As a final step, bind the **modcluster** filter to the undertow **default-server**.

```
[standalone@localhost:10990 /] /subsystem=undertow/server=\
default-server/host=default-host/filter-ref=modcluster:add
```

- 6.5. Reload the load balancer configuration.

```
[standalone@localhost:10990 /] :reload
```

7. The two host controllers on **servera** and **serverb** connect to the domain controller and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

- 7.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Open a new terminal window on the **servera** VM and run the following command:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 7.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** start-up script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

Open a new terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 7.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
- 7.4. Observe the load balancer terminal window and verify that the load balancer registers the two server instances (nodes) from Group1 that you are using in this lab.
The load balancer registers the two nodes **servera.1** and **serverb.1**

```
Registering node servera:servera.1, connection: ajp://172.25.250.10:8009/?#
Registering node serverb:serverb.1, connection: ajp://172.25.250.11:8009/?#
```

8. Deploy and test the cluster test application.

- 8.1. Using the EAP CLI, stop the servers in the server group **Group2** because they are not used in this lab.

```
[domain@172.25.250.254:9990 /] /server-group=Group2:stop-servers(blocking=true)
```

- 8.2. Deploy the cluster test application to the server group **Group1**. It is available at **/tmp/cluster.war** on the **workstation** VM.

```
[domain@172.25.250.254:9990 /] deploy /tmp/cluster.war \
--server-groups=Group1
```

- 8.3. Observe the load balancer terminal window and verify that both server instances **servera.1** and **serverb.1** have been registered on the load balancer and they are ready to serve the **/cluster** context.

```
Registering context /cluster, for node servera:servera.1
Registering context /cluster, for node serverb:serverb.1
```

- 8.4. Navigate to the load balancer at **http://172.25.250.254:9080/cluster** using a browser on the workstation VM. You should see the cluster application. Refresh the browser multiple times and notice that each request is served by the same server (due to session stickiness). Determine which server instance is handling your current request by looking at the **The JBoss EAP node name is** label in the application.
- 8.5. Stop the host that is actively handling the requests by pressing **Ctrl+C** in the appropriate terminal window to shut down the entire host controller.
- 8.6. Observe the load balancer terminal window and verify that the server you killed and the **/cluster** context for that server instance have been unregistered from the load balancer.

```
Unregistering context /cluster, from node servera:servera.1
Removing node servera:servera.1
```

- 8.7. Return to the web browser and refresh the page. The load balancer fails over your request to the other remaining server without losing the current visits value.

9. Clean Up and Grading

- 9.1. Press **Ctrl+C** in the terminal window where you started the host controllers and the domain controller to stop the managed domain. (Alternatively, you can shut down the domain controller using the JBoss EAP CLI command `/host=master:shutdown()`).
- 9.2. Press **Ctrl+C** to shut down the load balancer instance in the terminal window where you started it.
- 9.3. Press **Ctrl+C** to exit the EAP CLI if you used the CLI in the lab. (Alternatively, you can exit the CLI by typing **exit**.)
- 9.4. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab clustering-lab-final grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- A clustered application has the following benefits:
 - High availability
 - Scalability
 - Failover
 - Fault tolerance
- Clustering is available in EAP by using the default configuration in either the **ha** or **full-ha** sever configuration files.
- Clustered servers can run either as standalone servers or in a managed domain.
- Clustering is accomplished by the **Infinispan** and **JGroups** subsystems.
- Making an application distributable requires including a **<distributable>** tag in the **jboss-web.xml**.
- The JGroups subsystem provides communication between clustered servers using one of the preconfigured stacks, **UDP** and **TCP**.
- The Infinispan subsystem provides caching support for maintaining high availability.
- A **cache container** is a repository for caches. There are four default cache containers:
 - **web**
 - **hibernate**
 - **ejb**
 - **server**
- A network load balancer directs HTTP requests to EAP server instances that are cluster members.
- Session affinity is a mechanism that sends all requests coming from the same user to the same web server.
- **mod_cluster** creates a dynamic list of cluster members without needing manual configuration, but can also be configured to use a static list.
- The Undertow server can be used as a load balancer that utilizes **mod_cluster** as either a static or dynamic load balancer.
- Undertow uses the session ID cookie to provide sticky sessions.
- Alternative load balancers to Undertow are available to use and are supported by Red Hat.

- A singleton is a design pattern where a single instance of an object is shared by the entire application and can be configured to be clustered. The singleton deployment and services are configured in the singleton subsystem.

Chapter 13

Configuring the Batch Subsystem

Overview

Goal Configure the batch subsystem to run batch jobs written with JBeret.

Objectives

- Describe the functionality of batch jobs (JSR 352) and the JBeret implementation supported by JBoss EAP.
- Configure the batch subsystem to support batch job execution.

Sections

- Exploring Batch Jobs (and Quiz)
- Configuring the Batch Subsystem (and Guided Exercise)

Lab

- Configuring the Batch Subsystem

Exploring Batch Jobs

Objectives

After completing this section, students should be able to:

- Describe the functionality of batch jobs and the JBeret implementation.

Batch jobs

EAP 7 supports the new Java specification, JSR 352: **Batch applications**. Batch applications are applications that can be used to customize and create batch jobs. A batch job refers to a series of tasks that are executed periodically without needing interactive input. Batch jobs can be particularly useful for running resource-intensive tasks during low server usage times. For example, an organization's point-of-sale system could generate a daily inventory report using a batch job that runs only when all of the stores are closed for the day, preventing server resources used to process orders from being utilized on report generation.

Batch applications are comprised of a series of **steps** that can be executed sequentially or concurrently using separate threads. Each step is either **chunk-oriented** or **task-oriented**. A chunk-oriented step refers to a task used to process information from a source and is usually longer running. Because of its generally long running nature, it is possible to configure checkpoints to restart interrupted executions without losing progress. A task-oriented step is comparatively short running, such as removing old files in a file system or sending an email, as opposed to large data processing and transformation that is typical in chunk-oriented steps.

A chunk-oriented step has three important parts: the reader, the processor, and the writer. The reader is responsible for reading the data. The processor executes a process or action on the data, such as filtering or transforming the data. The writer is then responsible for updating or writing the data change.

Batch jobs are defined using the **Job Specification Language (JSL)**, which is an XML-based language that is part of the JSR-352 standard. Each job file is created with a unique name and placed in the **META-INF/batch-jobs** directory. The following is an example of a job XML definition file:

```
<job id="sampleJob"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
  <step id="sampleStep" >
    <chunk item-count="3">
      <reader ref="reader"/>
      <processor ref="processor"/>
      <writer ref="writer"/>
    </chunk>
  </step>
</job>
```

Based on this example, a single job named **sampleJob** has one step, **sampleStep**, containing a single chunk. Within the chunk is a reader, processor, and writer that needs to be implemented in

the batch application code. Implementing and developing batch applications is outside the scope of this course.

Steps that are independent of each other can be executed concurrently to improve throughput and reduce batch run times. This can be achieved in the job specification by the use of **splits** and **flows**.

A **flow** is a set of steps that runs sequentially. The steps in a flow cannot refer to steps outside the flow. The flow moves to the next element when its last step executes successfully.

A **split** is a set of flows that run concurrently. Each flow runs in a separate thread. The split moves to the next element when all its flows execute successfully.

The batch subsystem also provides the concept of **partitions** partitions that enable the processing of large data sets in parallel. This is usually used in chunk-oriented steps. For example, if your batch job needs to process an input file with a thousand (1000) records, you can separate the processing into two partitions, the first processing records 1-500 and the second processing records from 501-1000. Each partition can be assigned a number of threads from the batch subsystem's thread pool and the batch runtime will process the records in parallel.

The Batch subsystem

Batch jobs can be configured using the batch subsystem. In EAP 7, the batch subsystem is based on the **JBeret** open source implementation of JSR 352. Some important features provided by JBeret are:

- Components to read and write common file formats (CSV, XML, JSON)
- Components to access external resources such as databases and message queues
- Annotations to inject custom scopes (**@JobScoped**, **@StepScoped**, **@PartitionScoped**)
- Support to store statistics in multiple places such as databases (both relational and document-oriented), in-memory and Infinispan



Note

For EAP, only the database and in-memory statistics storage are available.

The following represents the default batch subsystem settings:

Job repository

The job repository is used by JBeret to configure where all logging information about jobs is stored, such as the batch execution state, when it was started, and when it was finished. Due to the batch nature, any failure or server crash must be stored and may be queried to identify problems. JBeret is able to restart any batch that was interrupted. EAP provides an in-memory configuration by default, but if a server crashes, the statistics will be lost. Alternatively, JBeret supports database persistence.

Thread pool

The **max-threads** element defines how many threads can be used by the batch subsystem to process jobs. Note that two threads from the thread pool are reserved for the partition jobs.

► Quiz

Exploring Batch Jobs

Choose the correct answer to the following questions:

► 1. Which step type is for long running processes? (Choose one.)

- a. Chunk-oriented step
- b. Task-oriented step

► 2. Which of the following comprise the basis for chunk-oriented steps? (Choose three.)

- a. Reader
- b. Processor
- c. Analyzer
- d. Writer
- e. Editor
- f. Action

► 3. Which of the following is a good candidate for batching? (Choose one.)

- a. An organization needs to create a PDF file containing a report of all the items sold today.
- b. A user wants to upload new images from their vacation to a blog.
- c. An organization wants to send an email to all administrators every time an error is logged.
- d. An organization needs to update the inventory immediately after every purchase.

► 4. The EAP 7 batching subsystem is based on which implementation? (Choose one.)

- a. JSR 352
- b. Spring Batch
- c. JBeret
- d. Easy Batch
- e. None of the above.

► Solution

Exploring Batch Jobs

Choose the correct answer to the following questions:

► 1. **Which step type is for long running processes? (Choose one.)**

- a. Chunk-oriented step
- b. Task-oriented step

► 2. **Which of the following comprise the basis for chunk-oriented steps? (Choose three.)**

- a. Reader
- b. Processor
- c. Analyzer
- d. Writer
- e. Editor
- f. Action

► 3. **Which of the following is a good candidate for batching? (Choose one.)**

- a. An organization needs to create a PDF file containing a report of all the items sold today.
- b. A user wants to upload new images from their vacation to a blog.
- c. An organization wants to send an email to all administrators every time an error is logged.
- d. An organization needs to update the inventory immediately after every purchase.

► 4. **The EAP 7 batching subsystem is based on which implementation? (Choose one.)**

- a. JSR 352
- b. Spring Batch
- c. JBeret
- d. Easy Batch
- e. None of the above.

Configuring the Batch Subsystem

Objectives

After completing this section, students should be able to:

- Configure the batch subsystem to support batch job execution.

Batch subsystem configuration

Batch jobs are defined using an XML file that is packaged with the application's JAR or WAR file inside the **META-INF/batch-jobs** directory. Each job has a unique name associated with it. The jobs defined in these XML files can be launched and controlled by the application itself using an API that is exposed by the batch subsystem and also by using the EAP CLI.

```
<subsystem xmlns="urn:jboss:domain:batch-jberet:1.0">
    <default-job-repository name="in-memory"/>①
    <default-thread-pool name="batch"/>②
    <job-repository name="in-memory">③
        <in-memory/>④
    </job-repository>
    <thread-pool name="batch">
        <max-threads count="10"/>⑤
    </thread-pool>
</subsystem>
```

- The **default-job-repository** element is used by JBeret to refer to the default **job-repository** configuration defined within the subsystem. This configuration permits an administrator to configure multiple job repositories in a single place and activate them as needed.
- The **default-thread-pool** element refers to which thread pool configuration must be used by JBeret to run each step. Similar to the **default-job-repository**, this approach can be used by an administrator to configure multiple thread pools that can be activated as needed.
- The **job-repository** element is used by JBeret to configure where information about job execution is stored, such as the batch execution state, when it was started, and when it was finished. Since a batch job can have many steps, any failures during step execution must be stored and may be queried to identify problems. JBeret can restart any batch job that was interrupted or failed due to invalid data.
- The **in-memory** element configures JBeret to store the batch logging information in EAP's memory. Alternatively, it is also supports a **jdbc-job-repository** element to store the batch logs in a database, which configured as a data source in EAP.
- The **max-threads** element defines how many threads can be used by the batch subsystem to process threads. Note that two threads from the thread pool are reserved for the partition jobs. The value of **max-threads** should be greater than three.

The runtime environment configuration in the batch subsystem consists of **job repositories** and **thread pools**. A **job-repository** stores the execution details of batch jobs and can be one of two types:

- **in-memory:** An **in-memory** job repository stores the details of the batch jobs that were executed on the server in RAM. The data is lost if the server is shut down or restarted. This is the default job repository. EAP 7 comes with a preconfigured **in-memory** job repository called **in-memory**.
- **JDBC:** A **JDBC** job repository stores the details of the batch jobs that were executed on the server in a database. The data is persistent across server restarts and shut down. A **JDBC** job repository is configured within the **<jdbc-job-repository>** section of the batch subsystem. It references a **datasource** attribute that points to a valid EAP 7 data source. The data source must be configured in the **datasources** subsystem of EAP.

The batch subsystem can be configured with **thread-pools**, which maintain a pool of threads that can be reused by batch jobs. The **max-threads** attribute of a thread pool defines the maximum number of threads that can be used by batch jobs while the jobs are executing. The **keepalive-time** attribute configures the duration for which the thread can remain idle when there are no jobs to be executed. EAP defines a thread pool with 10 **max-threads** and a 30-second **keepalive-time** by default.

You can use the EAP CLI or the EAP management console to configure job repositories, thread pools, and the batch job runtime environment. You can use the EAP CLI to start, stop, and restart batch jobs, and to view the results of job executions.

The batch subsystem can be configured using the management console. For a standalone server, navigate to **Configuration** → **Subsystem** → **Batch** and click **View** to view the batch subsystem configuration page.

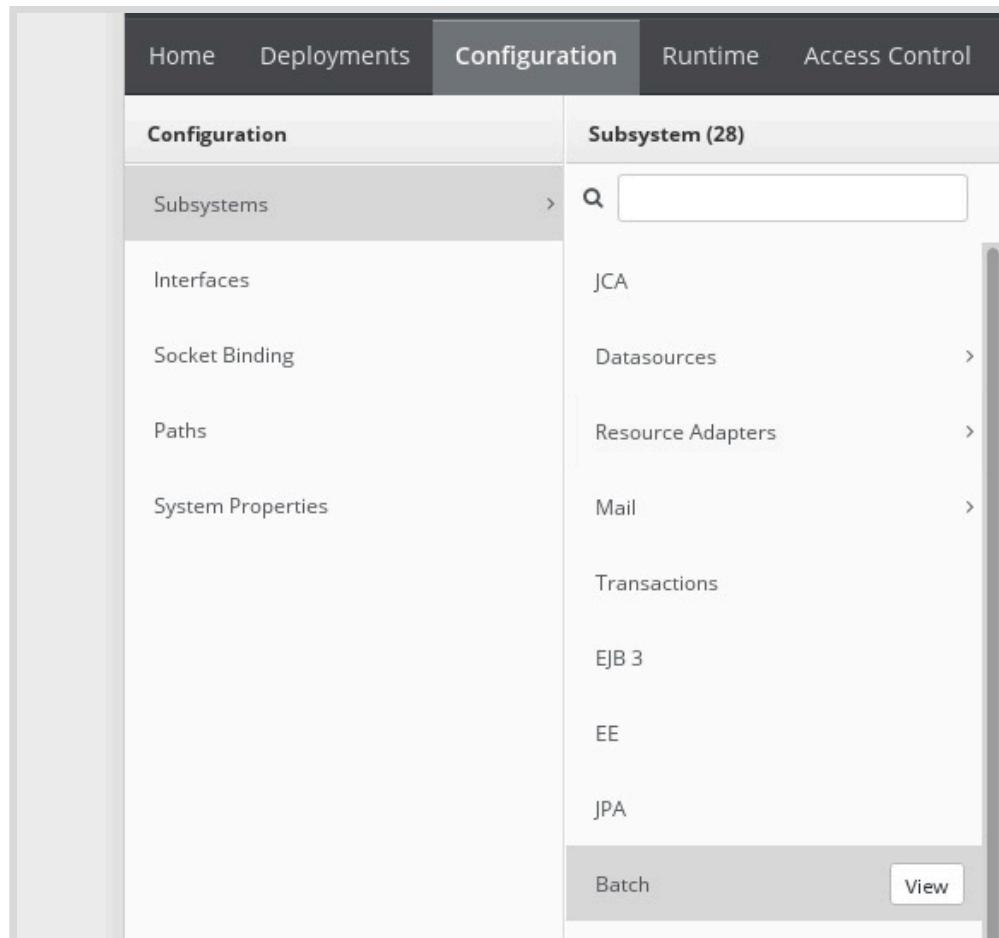


Figure 13.1: Batch subsystem for a standalone server

For an EAP managed domain, navigate to **Configuration** → **Profiles** → <profile-name> → **Subsystem** → **Batch** and click **View** to view the batch subsystem configuration page.

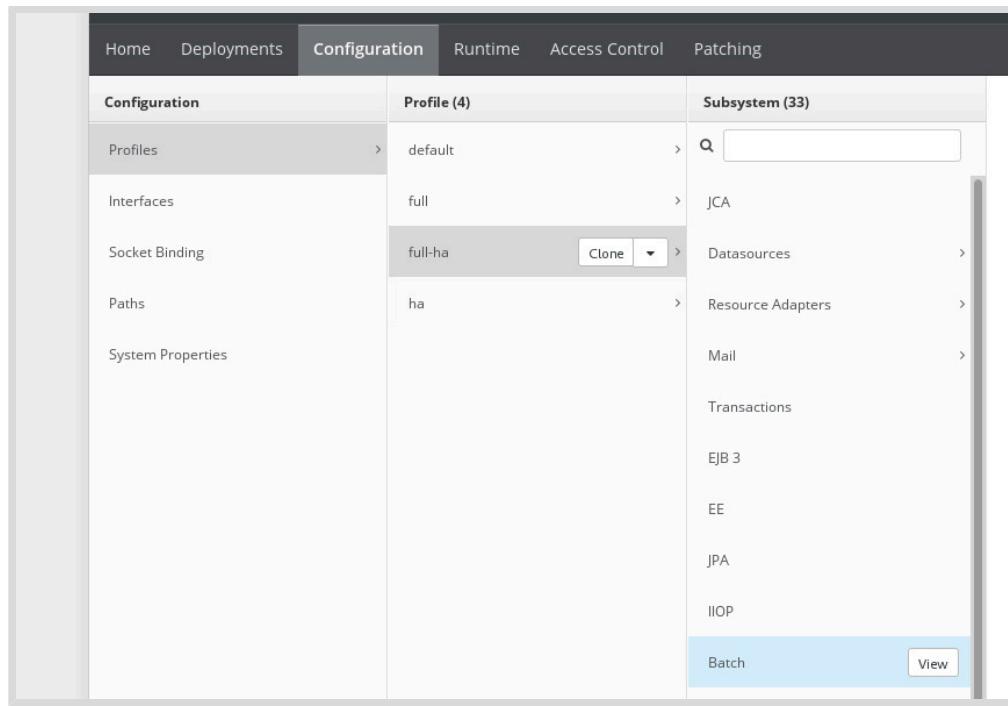


Figure 13.2: Batch subsystem for a managed domain

The default thread pool and job repository for batch jobs, job repositories, thread pools and thread factories can be configured using the links in the left sidebar of the batch subsystem configuration page (for both standalone server and a managed domain).

Batch Subsystem	
Defaults	The configuration of the batch subsystem.
In Memory	Edit
JDBC	Default job repository: in-memory
Thread Factories	Default thread pool: batch
Thread Pools	Restart jobs on resume: true

Figure 13.3: Batch subsystem configuration

The batch subsystem can also be configured using the EAP CLI. The EAP CLI has the added benefit of allowing jobs to be started, stopped and restarted by an EAP administrator. You can also view job run statistics using the EAP CLI.

To create an **in-memory** job repository called **custom-mem-repo** on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\  
in-memory-job-repository=custom-mem-repo:add()
```

To start a batch job called **job1**, which is defined in an application called **test-batch-app.war** on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/subsystem=\  
batch-jberet:start-job(job-xml-name=job1)
```

To stop a batch job on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/subsystem=\  
batch-jberet:stop-job(job-xml-name=job1)
```

To restart a batch job on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/subsystem=\  
batch-jberet:restart-job(job-xml-name=job1)
```

To view statistics about the **job1** execution on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war\  
subsystem=batch-jberet/job=job1:read-resource(recursive=true,include-runtime=true)
```

To create a **thread-pool** called **custom-pool** with a **max-threads** value of 15 and a **keepalive-time** value of 10 seconds that will be used by the batch subsystem for running jobs, run the following command :

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\  
thread-pool=custom-pool:add(max-threads=15,keepalive-time={unit=SECONDS,time=10})
```

To view the **thread-pool** statistics, run the following command :

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\  
thread-pool=custom-pool:read-resource(include-runtime=true,recursive=true)
```

To change the default **thread-pool** that will be used by the batch subsystem for running batch jobs, run the following command :

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\  
:write-attribute(name=default-thread-pool,value=custom-pool)
```

To change the default job repository that will be used by the batch subsystem for storing the details of batch job execution, run the following command :

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\  
:write-attribute(name=default-job-repository,value=custom-mem-repo)
```

For an EAP managed domain consisting of multiple hosts and servers, the job repository and thread pool configuration is similar to the standalone server. The CLI commands for configuring the batch subsystem will be in the **/profile=<profile_name>/subsystem=batch-jberet** namespace.

The one major difference between executing jobs and viewing job statistics in a managed domain compared to a standalone server, is the fact that you have to provide the server name where you want the jobs to be executed. This is because there could be multiple hosts and servers in a managed domain and the batch application WAR file is deployed at the server group level and could be running on multiple servers. To avoid duplication in job execution, provide the server name where the job has to be started, stopped or restarted. For example, to start a job on **server-one** running on **host1**:

```
[domain@172.25.250.254:9990 /] /host=host1/server=server-one\  
/deployment=test-batch-app.war/subsystem=\  
batch-jberet:start-job(job-xml-name=job1)
```

Similarly, to view job statistics in a managed domain:

```
[domain@172.25.250.254:9990 /] /host=host1/server=server-one\  
/deployment=test-batch-app.war/subsystem=\  
batch-jberet:job=job1:\  
read-resource(include-runtime=true, recursive=true)
```

Configuring the batch subsystem for persistent logs

By default, EAP is configured with an **in-memory** job repository that stores details of job execution in RAM. To ensure details of job runs are not lost after a server restart, you can configure persistent storage of job statistics by configuring a **jdbc-job-repository**, which stores the details in a database. Red Hat recommends that you create a separate database to store batch job statistics and you need to create an EAP datasource that points to this database.

This datasource is then referred to in the configuration of the **jdbc-job-repository**. When a new **jdbc-job-repository** is configured, the tables are automatically created after the server configuration is reloaded.

To create a **JDBC** job repository called **custom-jdbc-repo** which references a datasource called **custom-ds** on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet/\  
jdbc-job-repository=custom-jdbc-repo:add(data-source=custom-ds)
```

After you set the default job repository to the newly created **custom-jdbc-repo** and the server configuration is reloaded, you should see the following four tables created in the database:

```
+-----+
| Tables_in_bkjobs      |
+-----+
| JOB_EXECUTION          |
| JOB_INSTANCE           |
| PARTITION_EXECUTION   |
| STEP_EXECUTION          |
+-----+
```

After you execute a few batch jobs, the statistics of the job runs should be persisted in the database and you can check the data in the tables using **SELECT** queries. For example:

```
mysql> select JOBINSTANCEID,STARTTIME,ENDTIME,BATCHSTATUS from JOB_EXECUTION;
+-----+-----+-----+-----+
| JOBINSTANCEID | STARTTIME          | ENDTIME          | BATCHSTATUS |
+-----+-----+-----+-----+
| 1 | 2016-06-09 07:17:02 | 2016-06-09 07:17:04 | COMPLETED    |
| 2 | 2016-06-09 07:24:28 | 2016-06-09 07:24:30 | COMPLETED    |
| 3 | 2016-06-09 07:33:05 | 2016-06-09 07:33:07 | COMPLETED    |
| 4 | 2016-06-09 07:33:15 | 2016-06-09 07:33:18 | COMPLETED    |
+-----+-----+-----+-----+
```

The **JOBINSTANCEID** is a unique identifier for a job execution. **STARTTIME** and **ENDTIME** denotes when the job was started and completed respectively. **BATCHSTATUS** indicates if the job execution was completed successfully or not.

► Guided Exercise

Configuring the Batch Subsystem

In this lab, you will configure the Batch subsystem to support batch job execution.

Resources	
Files:	/home/student/JB248/labs/batch /home/student/JB248/labs/config-batch
Application URL:	N/A
Resources	/home/student/JB248/labs/config-batch/ batch-job.war

Outcomes

You should be able to configure the batch subsystem and execute batch jobs.

Before You Begin

Run the following command to verify that EAP is installed at **/opt/jboss-eap-7.0**, that no instance of EAP is running, that the previous guided exercise has been completed, to create the required files, and to download the batch-job.war application.

```
[student@workstation ~]$ lab config-batch
setup
```

► 1. Start the Standalone EAP Server instance

In this guided exercise, a standalone instance of EAP with the base directory at **/home/student/JB248/labs/batch** on the workstation VM will be used.

- To start it, run the following commands from the workstation VM:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB248/labs/batch
```

- Verify that the instance started without any errors by observing the console where you started the instance.

► 2. Deploy the batch-job.war application.

The **batch-job** application defines a batch job named **myjob** with three steps:

- Read: Simulates reading data from a source location.
- Process: Simulates processing the data that was read from the source location.

3. Write: Simulates writing the data after processing to a destination location.

The application is available in the `/home/student/JB248/labs/config-batch/` folder. Extract the `batch-job.war` file using the command:

```
[student@workstation ~]$ unzip \
/home/student/JB248/labs/config-batch/batch-job.war
```

The job is defined in the `WEB-INF/classes/META-INF/batch-jobs/myjob.xml` file inside the `batch-job.war` file. Each step is described as an XML tag called `step` in the XML file. The `batchlet` refers to a bean name defined using CDI.

Job execution will be triggered by using the EAP CLI.

- 2.1. Open a new terminal window and run the EAP CLI tool to connect to the standalone instance with the following commands:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect
```

- 2.2. Deploy the `batch-job.war` application:

```
[standalone@127.0.0.1:9990 /] deploy \
/home/student/JB248/labs/config-batch/batch-job.war
```

- 2.3. In the console, verify that the application was successfully deployed. You should see a message similar to the following:

```
12:16:27,321 INFO [org.jboss.as.server] (management-handler-thread - 5)
WFLYSRV0010: Deployed "batch-job.war" (runtime-name : "batch-job.war")
```

▶ 3. Start the batch job.

By default, EAP 7 is set up with an **in-memory** job repository and uses a **thread-pool** with 10 threads. An **in-memory** job repository stores the metadata about jobs in RAM and this information is lost when the server is shutdown or re-started.

- 3.1. The `batch-job` application defines a job named "`myjob`". Start this job using the EAP CLI:

```
[standalone@127.0.0.1:9990 /] /deployment=batch-job.war/subsystem=\
batch-jberet:start-job(job-xml-name=myjob)
```

Repeat the above command multiple times and run the job 2-3 times and observe the console output.

- 3.2. You should see the following output on the console from the job run:

```
[stdout] (Batch Thread - 1) ReadBatchlet: Reading Data...
[stdout] (Batch Thread - 1) ProcessBatchlet: Processing Data...
[stdout] (Batch Thread - 1) WriteBatchlet: Writing Data...

[stdout] (Batch Thread - 2) ReadBatchlet: Reading Data...
[stdout] (Batch Thread - 2) ProcessBatchlet: Processing Data...
[stdout] (Batch Thread - 2) WriteBatchlet: Writing Data...
```

▶ 4. View statistics about the job execution.

The statistics about each job that was run is stored in the default **in-memory** repository and EAP manages the execution of each batch job using a thread pool of ten threads. View the job statistics using the EAP CLI.

- 4.1. Run the following command to view the statistics:

```
[standalone@127.0.0.1:9990 /] /deployment=batch-job.war/\
subsystem=batch-jberet:read-resource(recursive=true,include-runtime=true)
```

- 4.2. The output appears as follows:

```
"myjob" => {
    "instance-count" => 3,
    "running-executions" => 0,
    "execution" => {
        "4" => {
            "batch-status" => "COMPLETED",
            "create-time" => "2016-06-08T11:01:08.358+0530",
            "end-time" => "2016-06-08T11:01:08.363+0530",
            "exit-status" => "COMPLETED",
            "instance-id" => 4L,
            "last-updated-time" => "2016-06-08T11:01:08.363+0530",
            "start-time" => "2016-06-08T11:01:08.359+0530"
        },
        ...
    }
}
```

- 4.3. View the **thread-pool** statistics of the batch subsystem using the following command (simulate multiple batch executions by running the command to start the job from Step 3.1 multiple times in rapid succession):

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet/\
thread-pool=batch:read-resource(include-runtime=true,recursive=true)
```

- 4.4. The output appears as follows:

```
{
    "outcome" => "success",
    "result" => {
        "active-count" => 0,
        "completed-task-count" => 14L,
        "current-thread-count" => 10,
        "keepalive-time" => {
```

```
        "time" => 30L,
        "unit" => "SECONDS"
    },
    "largest-thread-count" => 10,
    "max-threads" => 10,
    "name" => "batch",
    "queue-size" => 0,
    "rejected-count" => 0,
    "task-count" => 14L,
    "thread-factory" => undefined
}
}
```

The **active-count** attribute indicates the number of jobs that are currently in running state.

The **completed-task-count** attribute indicates the number of jobs that ran successfully.

The **current-thread-count** attribute indicates the number of threads that are currently active.

- ▶ 5. Stop the EAP instance by pressing **Ctrl+C** on the terminal window where you started the instance.

Exit the EAP CLI sessions by typing **quit** or press **Ctrl+C** on the terminal window where you started the CLI sessions.

This concludes the guided exercise.

▶ Lab

Configuring the Batch Subsystem

In this lab, you will configure and run a batch job in a managed domain.

Resources	
Files	/opt/domain /tmp/bookstore-job.war
Application URL	http://172.25.250.10:8080/bookstore

Outcome

You should be able to configure and run a batch job on the servers in the managed domain. You will also persist and view the job execution statistics in a MySQL database.

Before You Begin

Use the following command to download the relevant lab files, create the required lab folders, verify that the **bookstore** application is deployed, and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab batching-lab-final setup
```

You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively, and the domain controller on the workstation. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and deploy the **bookstore-job** application, which defines a batch job that counts the number of outstanding orders in the **bookstore** database. You will start the batch job using the EAP CLI and configure the batch subsystem to store the details of batch job runs in a separate MySQL database called **bkjobs**.

- Start the domain controller on the **workstation** VM. Because the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)
Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
- The two host controllers on **servera** and **serverb** connect to the domain controller and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.

- 2.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).
Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
- 2.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).
Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**
- 2.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
3. Launch the EAP CLI and connect to the domain controller to configure the servers in the managed domain.
4. **Verify that the bookstore application is deployed to server group Group1.**
 - 4.1. Stop the servers in the server group **Group2** because they are not used in this lab.
 - 4.2. Verify that **bookstore.war** is deployed to the server group **Group1**.
 - 4.3. If the bookstore application is not deployed, deploy the **bookstore.war** to the server group **Group1**. It is available at **/tmp/bookstore.war** on the **workstation** VM.
 - 4.4. Observe the console window on both **servera** and **serverb** and verify that both server instances **servera.1** and **serverb.1** have successfully deployed the **bookstore** application.
5. **Deploy the bookstore-job application.**
 - 5.1. Deploy the **bookstore-job.war** to the server group **Group1**. It is available at **/tmp/bookstore-job.war** on the **workstation** VM.
 - 5.2. Observe the console window on both **servera** and **serverb** VMs where you started the host controllers and verify that both server instances **servera.1** and **serverb.1** have successfully deployed the **bookstore-job** application.
6. **Place orders using the bookstore application.**
 - 6.1. Navigate to **http://172.25.250.10:8080/bookstore** to access the bookstore application. Click the **SIGN IN** link at the upper right corner and log in with the user name **jdoe** and password **redhat**.

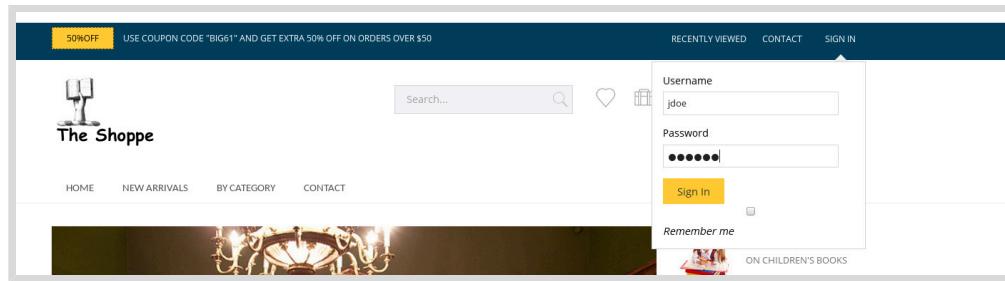


Figure 13.4: Bookstore login

- 6.2. Browse through the different categories of books available in the store and add a few books to the shopping bag by clicking **BUY** on the individual book description page.
- 6.3. After you have added a few books to your shopping bag, click **Place Order** on the **Shopping Bag** page.

Figure 13.5: Bookstore Shopping Bag

- 6.4. On the **Billing Info** page, enter the address details on the form and select the **Ship To Same Address** check box in the **Shipping Info** column. Click **NEXT** to continue.
- 6.5. On the **Payment Info** page, select a **Payment Type**, enter a **Credit Card Number**, the **Expiration Date**, and the **Cardholder Name**, and then click **REVIEW** to continue.
- 6.6. On the **Review** page, review the information you entered and click **SEND** to place the order. If your order was accepted, you should see the following message:



Figure 13.6: Bookstore order success message

7. Configure the batch subsystem

- 7.1. Create a thread pool for the batch jobs called **bk-jobs-pool** with a **max-threads** value of 20 and a **keepalive-time** value of 20 seconds.
- 7.2. Create a new JDBC job repository to store the job run statistics. The setup script has created a datasource called **bkjob-ds** which points to a MySQL database called **bkjobs**. Create a new **jdbc-job-repository** called **bookstore-job-repository** and reference the **bkjob-ds** datasource.
- 7.3. Connect to the **bkjobs** database using the MySQL client and verify that tables to store job run data have been created by the previous CLI command (for the MySQL client, use `username=bookstore, password=redhat`).
- 7.4. Because you have not run any jobs, there should be no data in these tables (Use SQL **select** commands to verify).
- 7.5. By default, the batch subsystem uses an **in-memory** job repository and references a thread pool called **batch**. Change the default job repository to the newly defined **bookstore-job-repository** and the default thread pool to the **bk-jobs-pool**.
- 7.6. Changing the default thread pool and job repository will require a reload of the host controllers. Reload hosts **servera** and **serverb**.

8. Start the batch job defined in the bookstore-job application.

- 8.1. Start the bookstore batch job using the EAP CLI. You need to specify the host and server on which you want the batch job to run. Run the batch job on **servera.1**. The name of the job defined in the **bookstore-job** application is **myjob**.
- 8.2. Observe the console window of **servera**. You should see the following output which indicates that there is one outstanding order in the bookstore database:

```
INFO [com.redhat.training.batch.job.ReportBatchelet] (Batch Thread - 2)
Processing job...
...
INFO [com.redhat.training.batch.job.ReportBatchelet] (Batch Thread - 2) There are
1 Orders in the DB
```

- 8.3. Repeat Steps 6.2-6.6 and place a few more orders in the bookstore and run the batch job again and verify that the correct order count is displayed as the output of the batch job.

9. View statistics about the bookstore job

- 9.1. Use the EAP CLI to view statistics about the batch jobs.
- 9.2. Connect to the **bkjobs** database using the MySQL client as outlined in Step 7.3 and verify that the the output of the select queries from Step 7.4 shows the statistics of the jobs that you ran. Run a few more jobs and check the output of the SELECT queries matches the data displayed from the EAP CLI commands.

10. Clean Up and Grading

- 10.1. Press **Ctrl+C** in the terminal window where you started the host controllers and the domain controller to stop the managed domain. (Alternatively, you can shut down the domain controller using the JBoss EAP CLI command `/host=master:shutdown()`).

- 10.2. Press **Ctrl+C** to exit the EAP CLI if you used the CLI in the lab. (Alternatively, you can exit the CLI by typing **exit**.)
- 10.3. Exit the MySQL client session by typing **quit** or **\q**.
- 10.4. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab  
batching-lab-final grade
```

This concludes the lab.

► Solution

Configuring the Batch Subsystem

In this lab, you will configure and run a batch job in a managed domain.

Resources	
Files	/opt/domain /tmp/bookstore-job.war
Application URL	http://172.25.250.10:8080/bookstore

Outcome

You should be able to configure and run a batch job on the servers in the managed domain. You will also persist and view the job execution statistics in a MySQL database.

Before You Begin

Use the following command to download the relevant lab files, create the required lab folders, verify that the **bookstore** application is deployed, and ensure that the managed domain has been set up correctly:

```
[student@workstation ~]$ lab batching-lab-final setup
```

You can use either the EAP 7 management console or the JBoss EAP CLI to achieve your objectives, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has set up a managed domain with two host controllers running on **servera** and **serverb** VMs respectively, and the domain controller on the workstation. The domain and host configuration files are stored in the **/opt/domain** folder on all three machines. You will start the managed domain and deploy the **bookstore-job** application, which defines a batch job that counts the number of outstanding orders in the **bookstore** database. You will start the batch job using the EAP CLI and configure the batch subsystem to store the details of batch job runs in a separate MySQL database called **bkjobs**.

- Start the domain controller on the **workstation** VM. Because the domain controller configuration files are kept in the **/opt/domain** folder on **workstation**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the domain controller is named **host-master.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-master.xml** argument to **domain.sh**.)
Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the domain controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-master.xml
```

2. The two host controllers on **servera** and **serverb** connect to the domain controller and fetch the latest domain configuration. Start the two host controllers on **servera** and **serverb**.
- 2.1. Start the host controller on **servera**. Because the host controller configuration files are kept in the **/opt/domain** folder on **servera**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.

Open a new terminal window on the **servera** VM and run the following command:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 2.2. Start the host controller on **serverb**. Because the host controller configuration files are kept in the **/opt/domain** folder on **serverb**, use **/opt/domain** as the value of the **jboss.domain.base.dir** argument that you pass to the **domain.sh** startup script. Also note that the host file for the host controller is named **host-slave.xml** and is found in the **/opt/domain/configuration** folder (Hint: Pass the **--host-config=host-slave.xml** argument to **domain.sh**).

Note that the **/opt/domain** directory is owned by the **jboss** user, so you must start the host controller using **sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh ...**.

Open a new terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
-Djboss.domain.master.address=172.25.250.254 \
--host-config=host-slave.xml
```

- 2.3. Verify that both host controllers connect to the domain controller and form a managed domain. Look at the console window where you started the domain controller and verify that both **servera** and **serverb** are registered as slaves to the domain controller.
3. Launch the EAP CLI and connect to the domain controller to configure the servers in the managed domain.

In a new terminal window on the workstation, start the EAP CLI and connect to the domain controller as the **jboss** user:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254:9990
```

4. Verify that the bookstore application is deployed to server group Group1.

- 4.1. Stop the servers in the server group **Group2** because they are not used in this lab.

```
[domain@172.25.250.254:9990 /] /server-group=Group2\
:stop-servers(blocking=true)
```

- 4.2. Verify that **bookstore.war** is deployed to the server group **Group1**.

```
[domain@172.25.250.254:9990 /] deployment-info --server-group=Group1
```

NAME	RUNTIME-NAME	STATE
bookstore.war	bookstore.war	enabled
cluster.war	cluster.war	enabled

- 4.3. If the bookstore application is not deployed, deploy the **bookstore.war** to the server group **Group1**. It is available at **/tmp/bookstore.war** on the **workstation** VM.

```
[domain@172.25.250.254:9990 /] deploy /tmp/bookstore.war \
--server-groups=Group1
```

- 4.4. Observe the console window on both **servera** and **servera** and verify that both server instances **servera.1** and **serverb.1** have successfully deployed the **bookstore** application.

5. Deploy the bookstore-job application.

- 5.1. Deploy the **bookstore-job.war** to the server group **Group1**. It is available at **/tmp/bookstore-job.war** on the **workstation** VM.

```
[domain@172.25.250.254:9990 /] deploy /tmp/bookstore-job.war \
--server-groups=Group1
```

- 5.2. Observe the console window on both **servera** and **serverb** VMs where you started the host controllers and verify that both server instances **servera.1** and **serverb.1** have successfully deployed the **bookstore-job** application.

6. Place orders using the bookstore application.

- 6.1. Navigate to **http://172.25.250.10:8080/bookstore** to access the bookstore application. Click the **SIGN IN** link at the upper right corner and log in with the user name **jdoe** and password **redhat**.

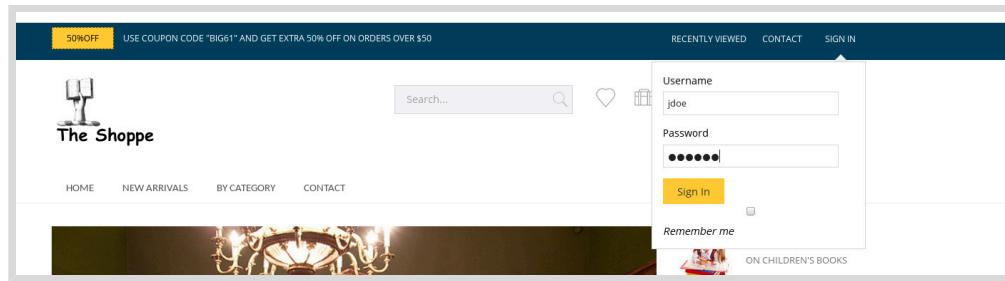


Figure Error.1: Bookstore login

- 6.2. Browse through the different categories of books available in the store and add a few books to the shopping bag by clicking **BUY** on the individual book description page.
- 6.3. After you have added a few books to your shopping bag, click **Place Order** on the **Shopping Bag** page.

Figure Error.2: Bookstore Shopping Bag

- 6.4. On the **Billing Info** page, enter the address details on the form and select the **Ship To Same Address** check box in the **Shipping Info** column. Click **NEXT** to continue.
- 6.5. On the **Payment Info** page, select a **Payment Type**, enter a **Credit Card Number**, the **Expiration Date**, and the **Cardholder Name**, and then click **REVIEW** to continue.
- 6.6. On the **Review** page, review the information you entered and click **SEND** to place the order. If your order was accepted, you should see the following message:

Thank you for your order. We have sent a confirmation email to you.

[CONTINUE SHOPPING](#)

Figure Error.3: Bookstore order success message

7. Configure the batch subsystem

- 7.1. Create a thread pool for the batch jobs called **bk-jobs-pool** with a **max-threads** value of 20 and a **keepalive-time** value of 20 seconds.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=\nbatch-jberet/thread-pool=bk-jobs-pool:\nadd(max-threads=20,keepalive-time={unit=SECONDS,time=20})
```

- 7.2. Create a new JDBC job repository to store the job run statistics. The setup script has created a datasource called **bkjob-ds** which points to a MySQL database called **bkjobs**. Create a new **jdbc-job-repository** called **bookstore-job-repository** and reference the **bkjob-ds** datasource.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=\nbatch-jberet/jdbc-job-repository=\nbookstore-job-repository:add(data-source="bkjob-ds")
```

- 7.3. Connect to the **bkjobs** database using the MySQL client and verify that tables to store job run data have been created by the previous CLI command (for the MySQL client, use username=bookstore, password=redhat).

```
[student@workstation tmp]$ mysql -u bookstore -p\nEnter password: *****\n\nMariaDB [(none)]> use bkjobs;\nDatabase changed\n\nMariaDB [(bkjobs)]> show tables;\n\n+-----+\n| Tables_in_bkjobs |\n+-----+\n| JOB_EXECUTION   |\n| JOB_INSTANCE    |\n| PARTITION_EXECUTION |\n| STEP_EXECUTION   |\n+-----+\n4 rows in set (0.00 sec)
```

- 7.4. Because you have not run any jobs, there should be no data in these tables (Use SQL **select** commands to verify).

```
MariaDB [(bkjobs)]> select * from JOB_EXECUTION;\nEmpty set (0.00 sec)\n\nMariaDB [(bkjobs)]> select * from JOB_INSTANCE;\nEmpty set (0.00 sec)
```

- 7.5. By default, the batch subsystem uses an **in-memory** job repository and references a thread pool called **batch**. Change the default job repository to the newly defined **bookstore-job-repository** and the default thread pool to the **bk-jobs-pool**.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=\nbatch-jberet:write-attribute\n(name=default-job-repository,value=bookstore-job-repository)\n\n[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=\nbatch-jberet:write-attribute\n(name=default-thread-pool,value=BK-Jobs-Pool)
```

- 7.6. Changing the default thread pool and job repository will require a reload of the host controllers. Reload hosts **servera** and **serverb**.

```
[domain@172.25.250.254:9990 /] reload --host=servera\n[domain@172.25.250.254:9990 /] reload --host=serverb
```

8. Start the batch job defined in the bookstore-job application.

- 8.1. Start the bookstore batch job using the EAP CLI. You need to specify the host and server on which you want the batch job to run. Run the batch job on **servera.1**. The name of the job defined in the **bookstore-job** application is **myjob**.

```
[domain@172.25.250.254:9990 /] /host=servera/server=servera.1\\n/deployment=bookstore-job.war/subsystem=\\nbatch-jberet:start-job(job-xml-name=myjob)\\n{\\n    "outcome" => "success",\\n    "result" => 3L\\n}
```

- 8.2. Observe the console window of **servera**. You should see the following output which indicates that there is one outstanding order in the bookstore database:

```
INFO [com.redhat.training.batch.job.ReportBatchelet] (Batch Thread - 2)\nProcessing job...\n...\nINFO [com.redhat.training.batch.job.ReportBatchelet] (Batch Thread - 2) There are\n1 Orders in the DB
```

- 8.3. Repeat Steps 6.2–6.6 and place a few more orders in the bookstore and run the batch job again and verify that the correct order count is displayed as the output of the batch job.

9. View statistics about the bookstore job

- 9.1. Use the EAP CLI to view statistics about the batch jobs.

```
[domain@172.25.250.254:9990 /] /host=servera/server=servera.1\\n/deployment=bookstore-job.war/subsystem=\\nbatch-jberet/job=myjob:\\n
```

```

read-resource(include-runtime=true, recursive=true)
{
    "outcome" => "success",
    "result" => {
        "instance-count" => 3,
        "running-executions" => 0,
        "execution" => {
            "3" => {
                "batch-status" => "COMPLETED",
                "create-time" => "2016-06-09T05:36:07.399-0400",
                "end-time" => "2016-06-09T05:36:09.419-0400",
                "exit-status" => "COMPLETED",
                "instance-id" => 3L,
                "last-updated-time" => "2016-06-09T05:36:09.419-0400",
                "start-time" => "2016-06-09T05:36:07.400-0400"
            ...
        }
    }
}

```

- 9.2. Connect to the **bkjobs** database using the MySQL client as outlined in Step 7.3 and verify that the output of the select queries from Step 7.4 shows the statistics of the jobs that you ran. Run a few more jobs and check the output of the SELECT queries matches the data displayed from the EAP CLI commands.

10. Clean Up and Grading

- 10.1. Press **Ctrl+C** in the terminal window where you started the host controllers and the domain controller to stop the managed domain. (Alternatively, you can shut down the domain controller using the JBoss EAP CLI command **/host=master:shutdown()**.)
- 10.2. Press **Ctrl+C** to exit the EAP CLI if you used the CLI in the lab. (Alternatively, you can exit the CLI by typing **exit**.)
- 10.3. Exit the MySQL client session by typing **quit** or **\q**.
- 10.4. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab
batching-lab-final grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- EAP 7 has introduced a **batch** subsystem and an API that can be used to execute batch jobs that do not require interactive input.
- A batch job is defined in an XML file in the application's JAR or WAR file under the **META-INF/batch-jobs** folder. A batch job is made up of a number of discrete **steps**.
- The EAP batch subsystem configuration consists of **job repositories** and **thread pools**.
- A thread pool consists of a pool of server threads that can be reused to execute multiple batch jobs.
- A job repository stores the execution statistics of batch jobs. There are two main types of job repositories:
 - **in-memory**: stores the details of job executions in memory (RAM) and the details are lost when the server is restarted or shut down.
 - **JDBC**: stores the details of job executions persistently in a database. The details are available even if the server is restarted or shut down.
- EAP comes preconfigured with an **in-memory** job repository and a thread pool with 10 threads.
- The EAP management console can be used to configure job repositories and thread pools in the batch subsystem.
- The EAP CLI can be used to configure job repositories and thread pools in the batch subsystem, as well as to start, stop, restart, and view statistics of batch jobs.
- In an EAP managed domain, you need to provide the **server** on which the job has to be executed when starting the job using the EAP CLI.

Chapter 14

Discovering what is new in EAP 7

Overview

Goal Describe the new features in JBoss EAP 7 and strategies for migrating JBoss EAP 6 applications to JBoss EAP 7.

- Objectives**
- Describe the new features in JBoss EAP 7.
 - Describe strategies for migrating JBoss EAP 6 applications to JBoss EAP 7.

- Sections**
- Exploring the New Features in JBoss EAP 7 (and Quiz)
 - Migrating JBoss EAP 6 Applications to JBoss EAP 7 (and Quiz)

- Lab**
- None

Exploring the New Features in JBoss EAP 7

Objectives

After completing this section, students should be able to:

- Describe the new features in JBoss EAP 7.

Java EE 7 specification

EAP 7 is compliant with the Java EE 7 specification implementing both the full and web profile standards. The following table lists the new and updated Java specification requests for the JEE 7 version. It also lists the subsystem in EAP 7 that is responsible for the technology.

Technology	JSR	Description	Subsystem
Batch 1.0	JSR 352	Provides batch processing.	batch-jberet
JSON-P 1.0	JSR 353	The Java API for JSON processing.	jaxrs
Concurrency Utilities	JSR 236	Provides a simple, standardized API for using concurrency from application components without compromising container integrity.	EE
WebSocket 1.1	JSR 356	Defines a Java API for the WebSocket protocol.	undertow
JMS 2.0	JSR 343	The Java Message Service API is responsible for accessing enterprise messaging systems from Java programs.	messaging-activemq
JPA 2.1	JSR 338	The Java Persistence API is responsible for the persistence management.	jpa
JCA 1.7	JSR 322	Defines a standard architecture for connecting to Enterprise Information Systems.	jca
JAX-RS 2.0	JSR 339	API for RESTful web services in the Java Platform.	jaxrs
JAX-WS 2.2	JSR 224	The JAX-WS 2.0 specification is the next generation web services API replacing JAX-RPC 1.0.	webservices
Servlet 3.1	JSR 340	Servlets receive and respond to requests from web clients.	undertow
JSF 2.2	JSR 344	Technology for building server-side user interfaces.	jsf

Technology	JSR	Description	Subsystem
JSP 2.3	JSR 245	Enables to create dynamic web content.	undertow
EL 3.0	JSR 341	Technology responsible for evaluation of expressions in web pages.	undertow
CDI 1.2	JSR 330	Contexts and Dependency Injection for Java EE.	weld
JTA 1.2	JSR 907	Specifies high-level interfaces between a transaction manager and the parties involved in a distributed transaction system.	transactions
Common Annotations 1.1	JSR 250	Annotations for common semantic concepts in the Java SE and Java EE platforms.	annotations
EJB 3.2	JSR 345	Allows the development of component-based applications.	ejb3
Bean Validation 1.1	JSR 349	Standardizes constraint definition, declaration and validation for the Java platform.	bean-validation

Management Console enhancements

The management console added support to some enhancements from the previous release of EAP, such as:

- Support for multiple languages, including English, Brazilian Portuguese, and German.
- Easier navigation, and enhanced support for large-scale domain configurations.
- Log visualization in the web console.
- Management model to visualize and customize the a CLI command using the a web console.

Other new features of EAP

EAP 7 introduced some useful new features. Principal among these are:

- Admin-only mode of CLI:** Now it is possible to embed an EAP server instance inside the CLI process and work offline.
- Graceful shutdown:** Now it is possible to shut down the server only when all requests are handled.
- Configuration change notifications:** Notifications are a useful mechanism to observe management changes on EAP servers. An administrator is informed of changes outside of his own actions
- Port reduction:** The opened ports was reduced to two in EAP 7. Port 9990 is used for administration and port 8080 is used for applications.
- Performance enhancements:** EAP 7 improved its performance with new features like the undertow subsystem, optimized cluster, JCA distributed work manager and the undertow load balancer.

- **Batch processing:** Now it is possible to create batch jobs in EAP 7 using the batch subsystem.
- **Security enhancements:** New resources are available related to security such as single sign-on using PicketLink and Keycloak, the Elytron for container and Java EE security.

Deprecated specifications and features

In EAP 7, some features were deprecated and may be removed in the future. This means that no enhancements will be made to these features.

All PicketLink modules, including Federation, were deprecated in JBoss EAP 7. The Resteasy Jettison Provider was also deprecated.

The following features are now unsupported:

- **JAX-RPC:** JAX-WS offers a more accurate and complete solution.
- **Java EE application deployment:** JSR 88 had limited adoption.
- **JBoss Web Services:**
 - Bean Validation 1.1 interceptors and features
 - JASPI authentication
- **Messaging:**
 - AMQP, Stomp, REST, MQTT, and OpenWire protocol
 - Netty over HTTP and Netty Servlet transport
 - OIO (Old Java IO) connectors/acceptors type
 - Vert.X, AeroGear, Spring, and Jolokia integration
 - Dynamic queue creation
 - Chain cluster
 - Using ActiveMQ Artemis Management using JMX
 - Use database as shared JDBC store
 - Scaling down in cluster
 - Colocated HA topology configured using http-connector/http-acceptor or using replication-colocated/shared-store-colocated
- **Management console:**
 - All flush operations for connection pools
 - Red Hat Access integration was dropped in JBoss EAP 7
- **Resteasy 3:**
 - jose-jwt
 - resteasy-crypto
 - resteasy-yaml-provider

- **Command Line Interface (CLI):**

- CLI preferences in **.jbossclirc** file
- Simplify working with complex attributes
- CLI tab-completion for attribute name path syntax
- Connection controller alias in **jboss-cli.xml**
- RBAC-based tab completion for the CLI commands

- **Clustering:**

- Cross-site replication
- Declarative channels, channel forks, fork protocol stacks and custom JGroups protocols in the **jgroups** subsystem
- Public API for JGroups channel creation
- Runtime management metrics for JGroups channels
- Ability to configure thread pools per protocol stack in the **jgroups** subsystem
- Ability to configure thread pools per cache container in the **infinispan** subsystem

- **Transactions:**

- Compensable transactions
- REST transactions
- **Add user:** Enable or disable users using **add-user** utility
- **Hibernate:** Use generics in Hibernate native API

- **PicketLink:**

- PicketLink IDM
- PicketLink IDM subsystem
- STS Client Pooling feature of PicketLink Federation
- PicketLink JEE (CDI Security)

- **JBoss Web**

- **Natives:**

- Support was dropped for mod_cluster and mod_jk connectors used with Apache HTTP server from RHEL RPM channels
- Support was dropped for mod_cluster and mod_jk connectors used with Apache HTTP server from the HP-UX Web Server Suites
- OpenSSL was dropped in JBoss EAP 7

- **Undertow:** WebDAV functionality is not provided in JBoss EAP 7. In JBoss EAP 7, to add the WebDAV functionality, it is required to implement a servlet, which implements the WebDAV functionality.
- **ORB**

► Quiz

What's New in JBoss EAP 7

Choose the correct answer to the following questions:

► 1. Which JSRs are handled by the undertow subsystem? (Choose two.)

- a. JSR 356 - WebSocket 1.1.
- b. JSR 353 - JSON-P 1.0.
- c. JSR 340 - Java Servlet 3.1.
- d. JSR 224 - JAX-WS.

► 2. Which new features are available in EAP 7? (Choose four.)

- a. Offline mode of CLI
- b. Domain mode
- c. Port Reduction
- d. JAX-RS
- e. Graceful shutdown
- f. Undertow load balancer

► 3. Which of the following features were deprecated in EAP 7? (Choose two.)

- a. Modcluster
- b. Remote EJB calls
- c. All PicketLink modules
- d. Bean validation 2.0
- e. The Resteasy Jettison Provider

► Solution

What's New in JBoss EAP 7

Choose the correct answer to the following questions:

► 1. Which JSRs are handled by the undertow subsystem? (Choose two.)

- a. JSR 356 - WebSocket 1.1.
- b. JSR 353 - JSON-P 1.0.
- c. JSR 340 - Java Servlet 3.1.
- d. JSR 224 - JAX-WS.

► 2. Which new features are available in EAP 7? (Choose four.)

- a. Offline mode of CLI
- b. Domain mode
- c. Port Reduction
- d. JAX-RS
- e. Graceful shutdown
- f. Undertow load balancer

► 3. Which of the following features were deprecated in EAP 7? (Choose two.)

- a. Modcluster
- b. Remote EJB calls
- c. All PicketLink modules
- d. Bean validation 2.0
- e. The Resteasy Jettison Provider

Migrating JBoss EAP 6 Applications to JBoss EAP 7

Objectives

After completing this section, students should be able to:

- Describe strategies for migrating JBoss EAP 6 applications to JBoss EAP 7.

Migrating applications from EAP 6 to EAP 7

In EAP 7, an effort was made to maintain backward compatibility for applications deployed on EAP 6. However, if the application uses features that are deprecated, you will need to migrate the application to be compatible with EAP 7.

Major features that were available in earlier EAP 7 releases that have been deprecated in EAP 7 are:

- *EJB 2 Container Managed Persistence (CMP)*: EJB 2 style Entity Beans (CMP) are no longer supported. Use JPA for a more flexible and performant API.
- *JAX-RPC*: Migrate to JAX-WS or JAX-RS for a more secure and flexible API.
- *JSR-88*: It is a specification for a standard application deployment API, which was not widely adopted. Use the EAP CLI or the EAP management console for application deployment.
- *Generic JMS Resource Adapter*: The ability to configure a generic JMS resource adapter to connect to a third party JMS provider is no longer supported in JBoss EAP 7. Check with the JMS provider to see if they have their own resource adapter that can be used with JBoss EAP.



References

Unsupported Features

https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/version-7.0/700-release-notes/#release_notes_unsupported_and_deprecated_functionality

Additionally, EAP 7 has a number of new features and changes that may impact application deployment. The major changes are:

- *Undertow*: Undertow has replaced JBoss Web as the web server in JBoss EAP 7. This means that the legacy **web** subsystem configuration must be migrated to the new EAP 7 **undertow** subsystem configuration:
 - The **urn:jboss:domain:web:2.2** subsystem configuration namespace in the server configuration file has been replaced by the **urn:jboss:domain:undertow:3.1** namespace.
 - The **org.jboss.as.web** extension module, located in **JBOSS_HOME/modules/system/layers/base/**, has been replaced with the **org.wildfly.extension.undertow** extension module.

- Earlier versions of EAP supported **global valves**, which are custom classes inserted into the request processing pipeline to make changes to the request or perform additional processing (for example, single sign-on). Valves must be migrated to use undertow handlers instead. Undertow includes a number of built-in handlers that provide common functionality. It also provides the ability to create custom handlers, which can be used to replace custom valve functionality.
- **JGroups**: There are several changes to the **jgroups** subsystem:
 - In earlier EAP versions, JGroups used the **public** interface defined in the `<interfaces>` section of the server configuration file. In EAP 7, JGroups now uses the new **private** interface that is defined in the `<interfaces>` section of the server configuration file (in the **ha** and **full-ha** profiles).
 - JGroups provides group communication support for HA services in the form of JGroups channels. EAP 7 introduces `<channel>` elements to the **jgroups** subsystem in the server configuration file. You can add, remove, or change the configuration of JGroups channels using the EAP CLI.
- **Messaging**: In EAP 7, ActiveMQ Artemis replaces HornetQ as the messaging support provider. The subsystem configuration has been changed:
 - The `urn:jboss:domain:messaging:3.0` subsystem configuration namespace in the server configuration file has been replaced by the `urn:jboss:domain:messaging-activemq:1.0` namespace.
 - The `org.jboss.as.messaging` extension module, located in `JBoss_HOME/modules/system/layers/base/`, has been replaced with the `org.wildfly.extension.messaging-activemq` extension module.
 - In earlier versions of EAP, JMS destination queues were configured in the `<jms-destinations>` element under the `<hornetq-server>` element in the **messaging** subsystem. In EAP 7, the JMS destination queue is configured in the default `<server>` element of the **messaging-activemq** subsystem.
- **Ports Reduction**: Earlier versions of EAP used a number of different ports for protocol-specific communication. By utilizing HTTP upgrade, EAP 7 has moved nearly all of its protocols to be multiplexed over two HTTP ports: a management port (9990), and an application port (8080).
- **EJB Clients**: In EAP 7, the default connector has changed from **remote** to **http-remoting** and the default remote connection port has changed from **4447** to **8080**. The JNDI provider URL for the default configuration has changed from `remote://server:4447` to `http-remoting://server:8080`.
- **Picketbox and Picketlink Federation**: In EAP 7, the Picketbox and Picketlink Federation APIs have been deprecated and will be removed in future EAP releases. They have been replaced by the Elytron project. See <http://lists.jboss.org/pipermail/wildfly-dev/2014-June/002244.html> for more details).

Tools to assist migration to EAP 7

Red Hat provides several tools to assist migrating applications deployed on earlier EAP versions to EAP 7.

The EAP CLI provides a **migrate** operation to migrate the **jacorb**, **messaging** and **web** subsystems from EAP 6. Note that the **migrate** operation does not fully automate the migration to EAP 7 and a few subsystems that were deprecated in EAP 7 have to be removed manually.

Before performing the actual migration, you can run the **describe-migration** operation to view the list of proposed changes that the migration operation will make to the configuration. For example, to view the changes for the **messaging** subsystem:

```
[standalone@localhost:9999 /] /subsystem=messaging:describe-migration
```

After understanding what changes are going to be made, execute the actual migration:

```
[standalone@localhost:9999 /] /subsystem=messaging:migrate
```

Similarly, migrate the **jacorb** and **web** subsystems:

```
[standalone@localhost:9999 /] /subsystem=jacorb:migrate  
[standalone@localhost:9999 /] /subsystem=web:migrate
```



Note

The migration operation sometimes displays a lot of warning messages. Consult the EAP 7 migration guide for the detailed list of warning messages and how to fix them.

The **Windup** utility, which is part of the **JBoss Migration Toolkit**, is an extensible and customizable rule-based tool that helps simplify migrations. It scans binary and source code files of applications and provides a detailed report of the changes required to migrate the application. It also provides an estimate of the complexity of the changes that are required and hints and tips on which files need to be changed.

You can use Windup to analyze the code, libraries, and XML deployment descriptors of your EAP 6 applications before you migrate them to JBoss EAP 7. The Windup tool reports on XML configuration files and specific application code and parameters that need to be replaced by an alternative configuration when migrating to JBoss EAP 7.



References

JBoss Migration Toolkit

<https://access.redhat.com/documentation/en/red-hat-jboss-migration-toolkit/>

Red Hat is currently developing the JBoss server migration tool, which will become the preferred tool to update your configuration to include the new features and settings in JBoss EAP 7 while keeping your existing configuration.

The JBoss server migration tool reads your existing JBoss EAP 6 server configuration file and adds configurations for the new subsystems, updates existing subsystem configurations with new features, and removes obsolete subsystem configurations. See <https://docs.jboss.org/author/display/CMT001/JBoss+Server+Migration+Tool+ User+Guide> for more details. A prerelease version is available for testing.



Note

The prerelease version of the JBoss Server Migration Tool is not yet supported officially by Red Hat.



References

EAP 7 migration guide

<https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/version-7.0/migration-guide>

► Quiz

Migrating Applications to JBoss EAP 7

Choose the correct answer to the following questions:

- ▶ 1. Which of the following can be used for request preprocessing in EAP 7 instead of valves? (Choose one.)
 - a. Undertow load balancer
 - b. Undertow valves
 - c. Undertow handlers
 - d. There is no replacement in EAP 7
- ▶ 2. Which of the following sentences is true for Undertow? (Choose one.)
 - a. It is an improved version of JBoss Web.
 - b. Valves must be migrated from previous releases of EAP to handlers.
 - c. The module of Undertow is located at **org.jboss.as.web**.
 - d. No migration is needed from EAP 6 JBoss Web.
- ▶ 3. Which of the following statements about the JGroups subsystem in EAP 7 are true? (Choose two.)
 - a. JGroups uses the **private** interface defined in the EAP configuration file.
 - b. JGroups uses the **public** interface defined in the EAP configuration file.
 - c. JGroups has been deprecated in EAP 7.
 - d. JGroups does not support TCP-based clustering. This feature has been deprecated.
 - e. JGroups supports both UDP- and TCP-based clustering.
- ▶ 4. The EAP 7 messaging subsystem is based on which implementation? (Choose one.)
 - a. HornetQ
 - b. ActiveMQ Artemis
 - c. JBoss Messaging
 - d. JBossMQ
 - e. None of the above

► Solution

Migrating Applications to JBoss EAP 7

Choose the correct answer to the following questions:

► 1. Which of the following can be used for request preprocessing in EAP 7 instead of valves? (Choose one.)

- a. Undertow load balancer
- b. Undertow valves
- c. Undertow handlers
- d. There is no replacement in EAP 7

► 2. Which of the following sentences is true for Undertow? (Choose one.)

- a. It is an improved version of JBoss Web.
- b. Valves must be migrated from previous releases of EAP to handlers.
- c. The module of Undertow is located at `org.jboss.as.web`.
- d. No migration is needed from EAP 6 JBoss Web.

► 3. Which of the following statements about the JGroups subsystem in EAP 7 are true?

(Choose two.)

- a. JGroups uses the `private` interface defined in the EAP configuration file.
- b. JGroups uses the `public` interface defined in the EAP configuration file.
- c. JGroups has been deprecated in EAP 7.
- d. JGroups does not support TCP-based clustering. This feature has been deprecated.
- e. JGroups supports both UDP- and TCP-based clustering.

► 4. The EAP 7 messaging subsystem is based on which implementation? (Choose one.)

- a. HornetQ
- b. ActiveMQ Artemis
- c. JBoss Messaging
- d. JBossMQ
- e. None of the above

Summary

In this chapter, you learned:

- EAP 7 implements Java EE 7 full and web profiles.
- The management console was greatly improved to simplify management of large deployment scenarios.
- EAP 7 decreased the number of ports and improved performance.
- Deprecated APIs are still available in EAP 7.
- A migration tool is provided to support EAP 7 server configuration migration.
- Windup is an application to migrate JavaEE applications developed for multiple platforms (including EAP 6).

Chapter 15

Comprehensive Review of Red Hat JBoss Application Administration I

Overview

Goal

Demonstrate how to configure and manage JBoss EAP standalone server and a managed domain.

Objectives

- Review the key tasks that will be practiced in the comprehensive review.

Sections

- Red Hat JBoss Application Administration I Comprehensive Review

Lab

- Comprehensive Review of Red Hat JBoss Application Administration I

Red Hat JBoss Application Administration I Comprehensive Review

Objectives

After completing this section, students should be able to:

- Practice skills learned in the Red Hat JBoss Application Administration course.

Overview

The following points will be covered by this comprehensive review:

- Install EAP with the quiet installer (with response file).
- Deploy a managed domain with two host controllers, and on each controller, two server instances running.
- Configure all servers running a full-ha profile.
- Run an Undertow load balancer instance in a standalone server.
- Deploy a persistent, default data source (replace H2 in-memory data source with a MySQL data source) and JDBC driver deployed as a module.
- Create a management user account.
- Configure the logging system with a size rotating file handler (to an NFS share so that all server logs are available there and clearly identified by the server).
- Lower the default minimum memory for the server JVMs to 256 MB, increase the maximum to 512 MB.
- Implement a security domain backed by a database login module where the data is stored in the configured MySQL data source .
- Configure an HTTPS listener including a self-signed certificate.
- Secure local CLI access.
- Replace the default welcome application.
- Password Vault configured and used for all passwords.
- Configure connection factory for the InVM connector.
- Configure messaging options configured: redelivery options, dead letter queue, and expiry queue.
- Implement Artemis message cluster with journals and I/O optimized and group discovery options.
- Deploy a messaging queue and an MDB deployed to test.
- Implement an HA-singleton with application deployed.
- Deploy a clustered web application on a TCP-based stack.

- Deploy the bookstore application.

► Lab

Comprehensive Review of Red Hat JBoss Application Administration I

In this lab, you will practice key skills that were presented in this course.

Resources	
Files	/home/student/JB248/install /home/student/JB248/labs/review-final
Application URL	http://localhost:8081 https://172.25.250.254:8443

Outcomes

You should be able to:

- Install EAP 7 on **workstation**, **servera**, and **serverb** VMs.
- Run a standalone server on the **workstation** VM.
- Configure a managed domain in **servera**, **serverb**, and **workstation** VMs.
- Configure all the main subsystems to support a load balanced application running on the managed domain.

Before You Begin

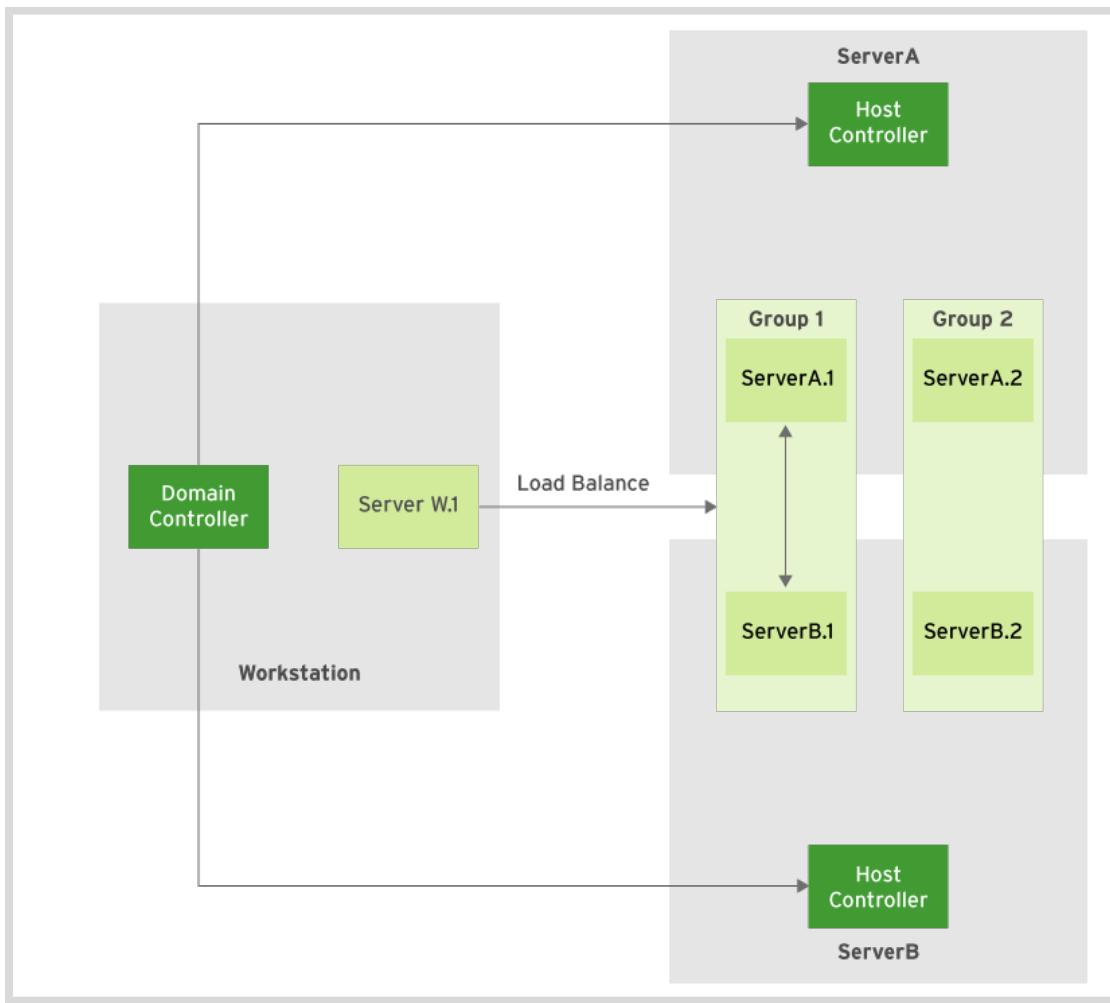
Reset the **workstation**, **servera**, and **serverb** VMs to remove all existing laboratories. Follow the *Orientation to the Classroom Environment* chapter to reset your environment accordingly and run the following command to download the relevant lab directory:

```
[student@workstation ~]$ lab review-final setup
```

1. Install EAP and configure the directory owners on each VM host.

A high-availability EAP environment will be installed on three different hosts (**workstation**, **servera**, and **serverb** VMs) to support a successful e-commerce application. It is a bookstore application using a web application that connects to a MySQL database which stores all production information and orders.

The following diagram shows the server architecture that will be created during this lab. In this environment you will install, configure, and deploy the enterprise application.



The environment is based on three virtual machines running Red Hat Enterprise Linux (RHEL 7) with a minimal set of tools installed, including Java 8:

- **workstation** VM (IP: 172.25.250.254): It is the only VM with a graphical interface installed. It will host the MySQL database and EAP running a standalone server and a

domain controller. The standalone server will be responsible for load balancing requests for the bookstore application.

- **servera** VM (IP: 172.25.250.10): It is a text-based VM responsible for running multiple EAP instances managed by the domain controller installed at the workstation.
- **serverb** VM (IP: 172.25.250.11): Similarly to the **servera** VM, it will run multiple EAP instances.

In this step, EAP will be installed on each host from the classroom environment, using recommended approaches to run EAP as a user with limited access to the OS environment. To access each VM a user is provided with the following credentials:

- *login*: **student**
- *password*: **student**

This user has **sudo** privileges and it will be used to install and launch EAP.

EAP must be installed with an answer file to automate the installation process on each host in a uniform way. Each host controller must use the same credentials as mentioned later:

- *username*: **eapadmin**
- *password*: **34p@R3dH4t123**

1. Create a system user named **eap** on workstation VM to start and stop EAP instances.
2. Create a system user named **eap** on **servera** VM to start and stop EAP instances.

```
[student@workstation ~]$ ssh root@servera useradd -r eap
```

3. Create a system user named **eap** on **serverb** VM to start and stop EAP instances.
4. EAP should be installed at **/opt/eap-7.0.0** on each host. Because the **eap** user does not have access to the **/opt** directory, the administrator must set it up as **root** and then change the owner of the **eap-7.0.0** folder to the **eap** user. In order to simplify the installation process, an answer file was downloaded on **workstation**, **servera**, and **serverb** VMs during the installation process at **/home/student/JB248/labs/review-lab**, but some fields must be customized to support the requirements of this lab. The EAP installer and a template answer file are available at **/home/student/JB248/labs/review-final** on each of the hosts.
5. Log in on each host and install EAP as **student/student** and customize the answer files **/home/student/JB248/labs/review-final/install.xml** and **/home/student/JB248/labs/review-final/install.xml.variables**. The **install.xml** file will be used to configure the location where EAP should be installed and the login used to access the management console, whereas the latter will be responsible for defining the password for the login defined at **install.xml**. Use the following as administrator credentials for each server:
 - *login*: **eapadmin**
 - *password*: **34p@R3dH4t123**

**Note**

Because the password is complicated, a file called **credentials.txt** is available in **/tmp** directory for copy and paste purposes.

Additionally, EAP must be installed in the **/opt/eap-7.0.0/** directory. Install EAP as the **root** user because the **/opt** directory belongs by default to the **root** user.

- 1.6. Run the EAP installer using the answer files updated in the previous step. The installer is available at **/home/student/JB248/installers**.
- 1.7. Install EAP on **servera** VM.
- 1.8. Check if EAP was installed on **servera** VM.
- 1.9. Install EAP on **serverb** VM.
- 1.10. Check if EAP was installed on **serverb** VM.
- 1.11. Update the owner of the directory where EAP was installed on each VM to the **eap** user in order to run and access all files from the EAP installation directory.

2. Configure the standalone server to run in a different directory.

A standalone server will run on the **workstation** as a load balancer. It will be responsible for forwarding requests to the servers running on a managed domain whose domain controller will also run on the workstation. In this step the standalone server will be customized to avoid port conflicts with the domain controller.

To create a standalone server without touching the EAP installation, a custom directory will be used to store all configuration files used by this review lab. In this step you will create a directory *only* on the **workstation** VM with the set of directories required to start an EAP standalone server instance. Follow the directives listed below to configure the standalone server:

- *base directory: /opt/standalone*
- *directory owner: eap*

Change the standalone server to run on a set of ports different from the default to avoid port conflicts. Because a managed domain will also run on the same VM and some ports are identical to the ones used by a standalone server, the standalone server configuration file must be updated. To make sure the changes are persisted, change the standalone configuration file to run on a port offset value of 1000.

Recall that all network changes require a full reload of the standalone server to take effect.

- 2.1. Create a directory called **/opt/standalone**.
- 2.2. Copy the standalone directory from the EAP installation to the **/opt/standalone**.
- 2.3. Change the owner to the **eap** user.
- 2.4. Start the EAP standalone server using the recently created directory.

**Note**

The command used to start the standalone server will be used throughout this review. To minimize typing it many times, it is strongly suggested to create a text file with the command that can be copied and pasted.

- 2.5. Open a browser window and access `http://localhost:9990` to check if the EAP management console was started. The credentials to access it are:
 - *username:* **eapadmin**
 - *password:* **34p@R3dH4t123**

**Note**

Because the password is complicated, a file called **credentials.txt** is available at the `/tmp` directory for copy and paste purposes.

- 2.6. In this step the CLI will be used to update network ports. Start it as the **eap** user and connect to the standalone server.
- 2.7. Update the port offset from the standalone server to 1000. This change will require a server reload, and reconnect CLI because the connection will be stale.
- 2.8. After running the command, disconnect from the server because the administration port was changed.
Test if the configuration was successfully set by accessing `http://localhost:9080` from the web browser. A welcome page should be displayed.
- 2.9. Update the public interface to make it available for external access. The load balancer must be visible from an outside environment, via the **eth0** interface. However, the default configuration makes it visible only to the loopback consumption.
Reload the server configuration to apply the changes.
Disconnect from the CLI to refresh the configuration.
Test if the configuration was successfully set by accessing `http://172.25.250.254:9080` from the web browser.
Reopen the CLI session with the standalone server by running the following command from the same terminal window where CLI was exited.

3. Configure the domain and host controllers in a custom directory location.

The standalone server was created to become a load balancer. However, a managed domain will be responsible for deploying JavaEE applications on each application server.

This architecture can be easily expanded creating new host controllers and binding them to the domain controller. Any update or customization can be executed by the management console or the CLI, without a large set of commands.

In this step, the domain controller will be configured and started on the **workstation** VM. Likewise, a host controller will be installed on **servera** and **serverb**. They will use a different base directory from the installation process to keep the install directory pristine.

To keep consistency, the base directory, where all the directories and files are stored in a controller, will be stored in the same directory owned by the **eap** user. All changes will be made in the configuration file to minimize the need to change parameters during the start-up process and to simplify the way a controller should be started.

- *Base directory: /opt/domain*
 - *Base directory owner: eap*
- 3.1. Copy as root the base directory from the EAP installed in step 1 to the directory created in the previous step for each host VM. Also change the owner of the directory again because the owner was inherited by the **cp** command.
- Copy the base directory (located at **/opt/eap-7.0.0/domain**) used by the domain controller in the **workstation** VM to **/opt/domain** directory.
- Change the owner of the **/opt/domain** directory to the **eap** user on the **workstation** VM.
- 3.2. Copy the base directory (located at **/opt/eap-7.0.0/domain**) used by the domain controller in the **servera** VM to **/opt/domain** directory.
- Change the owner of the **/opt/domain** directory to the **eap** user on the **servera** VM.
- 3.3. Copy the base directory (located at **/opt/eap-7.0.0/domain**) used by the domain controller on the **serverb** VM to **/opt/domain** directory.
- Change the owner of the **/opt/domain** directory to the **eap** user on the **serverb** VM.
- 3.4. Test the controller on each VM.
- In this step, we will check if the domain controller can be started individually on each VM for verification purposes. To start it as a domain controller, run the **domain.sh** script from the **JBOSS_HOME/bin** directory and point to the **/opt/domain** directory as the base directory. Recall that the owner of the base directory is the **eap** user and it should be started using the **sudo -u** command. Additionally, a configuration file called **host-master.xml**, used to start it as a domain controller, must be passed to start EAP without any conflicts.
- Start the EAP as a managed domain from the workstation VM. The **eap** user should be used to start the process and the base directory should point to the **/opt/domain** directory.
- 3.5. Start the EAP as a managed domain from the **servera** VM. The **eap** user should be used to start the process and the base directory should point to the **/opt/domain** directory.
- 3.6. Start EAP as a managed domain from the **serverb** VM. The **eap** user should be used to start the process and the base directory should point to the **/opt/domain** directory.
- 3.7. Stop each controller by pressing **Ctrl+C** on each terminal window where EAP was started.
4. Configure network permissions for the domain and host controller from the managed domain.
- Some low level configuration should be done on the domain controller to support external connections from the host controllers. The management interface from the domain controller by default is visible only from the loopback interface. Host controllers, on the other hand, will need to connect to the server using the ethernet interface instead. In this step the network configuration will be updated using the CLI and some firewall commands. The cluster password used by the **full-ha** profile should be updated to use the same password as the **eapadmin** user to allow authentication among all the messaging services running on EAP and to exchange their data.

- 4.1. To change the network interface configuration, start the EAP domain controller on the **workstation** VM to configure it using CLI. It should use the **/opt/domain/configuration/host-master.xml** using as the base directory the **/opt/domain** directory. Recall that EAP should be running using the **ejap** user.
Start the domain controller of the managed domain from the **workstation** VM as the **ejap** user.

**Note**

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted in the workstation VM.

- 4.2. Start the CLI installed in the **JBOSS_HOME/bin** directory from the workstation VM and connect to domain controller running locally. Recall that EAP is running with the **ejap** user, and to modify any configuration the CLI process must also be started as the **ejap** user.
Start the CLI from the workstation VM as the **ejap** user.
Connect to the domain controller started at the **localhost**.
- 4.3. Update the management interface from the domain controller to use the workstation IP address using the CLI. Recall that the domain controller name is master and the IP address from the workstation is 172.25.250.254.
- 4.4. Update the password used by the messaging subsystem to allow authentication among all the ActiveMQ instances running on the domain controller in a clustered environment. Recall that the profile that supports a clustered environment and with messaging subsystem support is the full-ha profile, and the password is **34p@R3dH4t123**.

**Note**

Because the password is complicated, a file named **credentials.txt** is available at the **/tmp** directory for copy and paste purposes.

- 4.5. The server configuration must be reloaded to activate the IP address changes.
- 4.6. Configure firewall rules to allow access from host controllers to the domain controller. By default the TCP ports 9990 and 9999 should be opened for public access in the workstation VM. Reload the rules immediately after setting these rules. A script is provided to disable these rules and it is available at **/tmp/firewall-mgmt.sh** in the workstation VM.

```
[student@workstation ~]$ sudo /tmp/firewall-mgmt.sh
```

- 4.7. Unfortunately the CLI connection to the domain controller will be stale right after the reload because the domain controller will now listen to the IP address 172.25.250.254. Connect again to the domain controller with the CLI.
5. Connect to the host controller running on the **serverA** VM using the domain controller. The EAP installation on the **serverA** VM will be a host controller in a managed domain. It should point to the domain controller started on the **workstation** VM. To achieve this goal, the host controller from **serverA** VM have the following characteristics:
 - Host controller name: **serverA**

- Domain controller IP address: **172.25.250.254** (the IP address of the **workstation** VM)

All the servers managed by the host controller must be accessible externally and the public interface (where all servers will listen to requests) must point to the network interface, not the loopback interface. Therefore, the public interface on the host controller must be configured as follows:

- *interface name: **public***
- *IP address: **172.25.250.10***

To prevent unknown hosts from connecting to the domain controller, create a management user. Use the following parameters when creating a the management user:

- *login: **jbossmgmt***
- *password: **JBoss@RedHat123***
- *Is this new user going to be used for one AS process to connect to another AS process?: **yes***

The credentials should be added to the **/opt/domain/configuration** directory and the **add-user.sh** script should be executed with the **eap** user.

Configure firewall rules to allow external access to the servers running on the host controller. It should allow external access for TCP port 8080 and 8230 (the two existing servers that will be removed during a later step).

Finally, remove the server associated with this server, because new ones should be created and they will be associated with different server groups that will be updated.

- 5.1. Create the management user with the **add-user.sh** script from the workstation VM as the **eap** user to update the **mgmt-users.properties** file.
Copy the resulting XML content to the file: **/home/student/secret.xml** because it will be used by the host controllers to connect.
- 5.2. Open an SSH session as the **student** user from the **workstation** VM to the **serverA** VM and edit the **host-slave.xml** file from the base directory to use the XML tag copied from the previous step to update the existing **secret** tag. Remember to edit the file as the **eap** user.
Update the **remote** tag from the same XML file to add the username attribute with the login **jbossmgmt** used to create the management user.
- 5.3. From the **serverA** VM edit the file **/opt/domain/configuration/host-slave.xml** to change the host name to **serverA** and update the IP address pointing to the domain controller address (172.25.250.254).
Update the host name from the **serverA** VM.
- 5.4. Update the IP address from the static-discovery tag from the **serverA** host controller to point to 172.25.250.254.
Change the **static-discovery** tag to point to the domain controller.
- 5.5. Enable external access to the web applications by updating the public interface where EAP will listen to requests. The same configuration file is responsible for enabling this access. Recall that the public IP address is 172.25.250.10.

In the **host-slave.xml** file, change the **interface** tag named public to point to the network interface, not to the 127.0.0.1 address.

- 5.6. Start the EAP host controller from the **serverA** VM using **/opt/domain** as the base directory. Also recall that the owner of the base directory is **eap** and you should use the **host-slave.xml** configuration file that was customized in the previous step. Recall that a script was created previously to start EAP.



Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted. Create it on the **serverA** VM.

- 5.7. Check if the **serverA** host controller was connected to the domain controller by checking the logs from the domain controller terminal window.
- 5.8. Update the firewall rules on the **serverA** VM to allow external machines to access the servers running on the **serverA** host controller.
- 5.9. Test if the servers are available via the public interface. Using a web browser, open the following URLs and check if the EAP welcome pages are displayed:
 - <http://172.25.250.10:8080>
 - <http://172.25.250.10:8230>
- 5.10. In the CLI terminal window, stop the servers running on the host controller **serverA** to remove them.
- 5.11. Remove the server-one instance from **serverA** host controller to avoid conflicts with the other host controller that will be installed.
In order to achieve our goal, use the CLI and remove the servers from the host named **serverA**.
- 5.12. Remove the server-two instance from **serverA** host controller to avoid conflicts with the other host controller that will be installed.

6. Connect the host controller running on the **serverB** VM to the domain controller.

The EAP installation on the **serverB** VM will be used as a second host controller of a managed domain. It should point to the domain controller that will be started on the workstation VM. Similarly to **serverA** host controller, the **serverB** VM should follow some characteristics:

- *Host controller name: **serverB***
- *Domain controller IP address: **172.25.250.254** (the IP address of the **workstation** VM)*

All the servers managed by the host controller must be accessible externally and the public interface (where all servers will listen to requests) must point to the network interface, not the loopback interface. Therefore, the public interface of the host controller must be configured as follows:

- *interface name: **public***
- *IP address: **172.25.250.11***

To prevent unknown hosts from connecting to the domain controller, a management user was created in the previous step and it should be associated in a similar fashion as the **serverA** host controller.

Finally, remove the server associated with this server, because new ones should be created and they will be associated with different server groups that will be updated.

- 6.1. Open an SSH session as the student user from the **workstation** VM to the **serverB** VM and edit the **host-slave.xml** file from the base directory to use the XML tag copied from the previous step to update the existing **secret** tag. Remember to edit the file as the **eap** user.

Update the **remote** tag from the same XML file to add the user name attribute with the login **jbossmgmt** used to create the management user.

- 6.2. From the **serverB** VM edit the file **/opt/domain/configuration/host-slave.xml** to change the host name to **serverB** and update the IP address pointing to the domain controller address (172.25.250.254).

Change the **static-discovery** tag to point to the domain controller.

- 6.3. Enable external access to the web applications by updating the public interface where EAP listens to requests. The same configuration file is responsible for enabling this access. Recall that the **serverB** public IP address is 172.25.250.11.

In the **host-slave.xml** file, change the **interface** tag named **public** to point to the network interface, not to the 127.0.0.1 address.

- 6.4. Start EAP host controller on **serverB** VM using **/opt/domain** as the base directory. Also recall that the owner of the base directory is **eap** and you should use the **host-slave.xml** configuration file that was customized in the previous step.



Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted from the **serverB** VM.

- 6.5. Check if the **serverB** host controller was connected to the domain controller by checking the logs from the domain controller terminal window.

- 6.6. Update the firewall rules from the **serverB** VM to allow external machines to access the servers running on the **serverB** host controller. The script **/tmp/firewall-http.sh** with the commands to drop the rules is available in the **/tmp** directory:

- 6.7. Test if the servers are available via the public interface. Using a web browser, open the following URLs and check if the EAP welcome pages are displayed:

- <http://172.25.250.11:8080>
- <http://172.25.250.11:8230>

- 6.8. In the CLI terminal window, stop the servers running on the host controller **serverB** so that you can remove them.

- 6.9. Remove the **server-one** instance.

- 6.10. Remove the **server-two** instance.

7. Customize the server groups to support a high availability environment.

The default server groups provided by EAP are focused on testing. In an actual production environment that must address high availability concerns, and that should avoid downtime of a critical application, two server groups should be created that share the same profile.

In this step, the existing server groups will be removed:

- main-server-group
- other-server-group

Two new server groups will be created focused on an environment that should support a rolling upgrades. Thus, the following configuration must be used by the server groups:

First server group characteristics:

- *Server group name:* **Group1**
- *Profile:* **full-ha**
- *Socket binding configuration:* **full-ha-sockets**

Second server group characteristics:

- *Server group name:* **Group2**
- *Profile:* **full-ha**
- *Socket binding configuration:* **full-ha-sockets**

7.1. Remove server group **main-server-group** from the domain controller.

An "**outcome**" => "**success**" is expected.

7.2. Remove server group **other-server-group** from the domain controller.

An "**outcome**" => "**success**" is expected.

7.3. Create a new server group called **Group1** on the domain controller.

An "**outcome**" => "**success**" is expected.

7.4. Create a new server group called **Group2** on the domain controller.

An "**outcome**" => "**success**" is expected.

7.5. Reload the configuration from the **serverA** host controller to update the servers.

7.6. Reload the configuration from the **serverB** host controller to update the servers.

7.7. Reload the configuration from the domain controller to update the servers.

7.8. Exit the CLI.

7.9. Reconnect to the managed domain server.

8. Create servers on the **serverA** host controller.

In this step, two new servers will be created on the **serverA** host controller. Create two servers associated with the server groups created in the previous step. They will run on the serverA VM using the following definitions. Recall that all new servers are not started automatically:

First server:

- *Name:* **serverA.1**

- Server group: **Group1**
- Socket binding port offset: None (Leave at default value of 0)
- Auto start?: **true**

Second server:

- Name: **serverA.2**
 - Server group: **Group2**
 - Socket binding port offset: **150**
 - Auto start?: **true**
- 8.1. Configure the **serverA.1** on **serverA**.
 - 8.2. Configure the **serverA.2** on **serverA**.
 - 8.3. Start **serverA.1**.
 - 8.4. Start **serverA.2**.
 - 8.5. Test if the servers are running and if they are accessible with a web browser from the workstation VM. Use the following addresses to check if the welcome page is displayed.
 - <http://172.25.250.10:8080>
 - <http://172.25.250.10:8230>

9. Create servers on the **serverB** host controller.

In this step, two new servers will be created on the **serverB** host controller. Create two servers associated with the server groups created in the previous step. They will run on the **serverB** VM using the following definitions. Recall that all new servers are not started automatically:

First server:

- Name: **serverB.1**
- Server group: **Group1**
- Socket binding port offset: None (Leave at default value of 0)
- Auto start?: **true**

Second server:

- Name: **serverB.2**
 - Server group: **Group2**
 - Socket binding port offset: **150**
 - Auto start?: **true**
- 9.1. Configure the **serverB.1** on **serverB**.
 - 9.2. Configure the **serverB.2** on **serverB**.

- 9.3. Start **serverB.1**.
- 9.4. Test if the servers are running and if they are accessible with a web browser from the **workstation** VM. Use the following addresses to check if the welcome page is displayed.
 - <http://172.25.250.11:8080>
 - <http://172.25.250.11:8230>
10. Install the MySQL JDBC driver as a module, install the driver in the subsystem, and create the data sources.

In the managed domain, two databases instances from MySQL will be used by two applications:

- *Bookstore*: The bookstore application will store all the orders and products information from a e-commerce application.
- *Example*: The example application will be used to check user names and passwords stored at the MySQL database instance called **bksecurity**.

Both database instances are managed by the workstation VM as a service. This step will create and configure both data sources to connect to these databases and make them available for those applications.

Details about each data source will be provided later.

A JDBC driver is a JAR file responsible for accessing a database by a Java application. RHEL7 provides a JDBC driver for MySQL from a yum repository. It is available on all VMs at the **/usr/share/java** directory under the name of **mysql-connector-java.jar**. In this step, the JDBC driver should be installed as a module to allow a reusable approach to connect to a database throughout the host controllers.

Any JDBC driver requires that some dependencies (already provided by EAP) should be available, and they should be defined as dependencies:

- **javax.api**
- **java.transaction.api**

Furthermore, the driver must have a unique name. Usually, the driver vendor ID is used, and for MySQL it is the reverse address name **com.mysql**.

To enable these modules, they should be installed using a disconnected CLI session on each host in the managed domain. Recall that the module will be installed in the same directory where EAP is installed and the command should be executed as the **eap** user.

- 10.1. Start a disconnected CLI session on the **workstation** VM.
- 10.2. Add the module to the domain controller.

Determine if the module was installed by inspecting the directory **/opt/eap-7.0.0/modules/com/mysql/main** directory.
- 10.3. Exit the CLI window.
- 10.4. Open a CLI disconnected session on the **servera** VM.
- 10.5. Add a module to the **servera** VM.

Evaluate if the module was installed by inspecting the directory **/opt/eap-7.0.0/modules/com/mysql/main** directory.

- 10.6. Exit the CLI window.
- 10.7. Start a disconnected CLI session on the **serverb** VM.
- 10.8. Add a module to the **serverb** VM.

Determine if the module was installed by inspecting the directory **/opt/eap-7.0.0/modules/com/mysql/main** directory.
- 10.9. Exit the CLI window:
- 10.10. Create a reference in the domain controller to the module recently installed. The driver will be used by the profile called **full-ha** that is used by all the server groups created so far. It must be added to the datasource subsystem and it should be named mysql. Recall that the module was named **com.mysql**. Use the CLI open and connected to the domain controller to configure the module.
- 10.11. Configure the datasource to access the bookstore application database. Because this is a configuration used by a profile and the only one used is the full-ha, create it using the following parameters. A file called **/tmp/bookstore.cli** is provided to minimize typing:
 - *Datasource type: Non-XA*
 - *JDBC provider: MySQL datasource*
 - *JNDI name: java:jboss/datasources/bookstore*
 - *Datasource name: bookstore*
 - *Connection URL: jdbc:mysql://172.25.250.254:3306/bookstore*
 - *User name: bookstore*
 - *Password: redhat*

To configure the data source, use the CLI with the **--file** parameter to load the **/tmp/bookstore.cli**.

To create the data source, run the following command from a workstation terminal window:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \
--file="/tmp/bookstore.cli"
```

Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted from the workstation VM.

- 10.12. Test if the datasource is working by accessing the management console using a web browser. Access the management console by navigating <http://172.25.250.254:9990>. Use the user name **eapadmin** and the password **34p@R3dH4t123**. At the top of the page, click **Configuration** → **Profiles** → **full-ha** → **Subsystems** → **Datasources** → **Non-XA** → **bookstore**. Click **View** and in the overview page click the **Connection** and then **Test Connection**.

10.13. Configure the data source to access the example application database. Because this is a configuration used by a profile and the only one used is the full-ha, create it using the following parameters:

- *Datasource type: Non-XA*
- *JDBC provider: MySQL datasource*
- *JNDI name: java:jboss/datasources/bksecurity-ds*
- *Datasource name: bksecurity*
- *Connection URL: jdbc:mysql://172.25.250.254:3306/bksecurity*
- *User name: bkadmin*
- *Password: redhat*

A CLI script is provided at **/tmp/bksecurity.cli** but some fields must be populated.

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \
--file="/tmp/bksecurity.cli"
```

Remember to reuse the command from a previous step.

Test if the datasource is working by accessing the management console using a web browser. Access the management console by navigating <http://172.25.250.254:9990>.

Use the user name **eapadmin** and the password **34p@R3dH4t123**. At the top of the page, click **Configuration** → **Profiles** → **full-ha** → **Subsystems** → **Datasources** → **Non-XA** → **bksecurity**. Click **View** and in the overview page click the Connection and then **Test Connection**.

11. Configure the logging subsystem to store the logs in an NFS share.

Each server will store logging information in an NFS share that is mounted in the **servera** and **serverb** VMs. The NFS mountpoint is configured at **/var/log/jboss** and they are all pointing to the **/var/log/jboss** directory from the **workstation** VM.

Recall that in order to point to a directory different from the base directory, a path must be created and it must point to the NFS share.

To minimize the amount of data generated by logging, a size rotating file handler must be used. It should have a maximum 1 MB size and a maximum of five files should be created. The debug level should be set for this handler and the handler should be named **BOOKSTORE_LOG_HANDLER**. Some extra details about the logging will be provided later in this step.

A category should be created for the **com.redhat.training** package, which will be responsible for capturing logs and storing them using the previously mentioned handler.

- 11.1. Create a new path called **custom.log.dir** pointing to the NFS mount point **/var/log/jboss**.
- 11.2. Configure a handler called **BOOKSTORE_LOG_HANDLER** that will use the custom.log.dir path created in the previous step. It should use the following specifications:
 - Log file name: **bookstore.log**

- It should be enabled.
 - All the logs should appended to bookstore.log, not deleted.
 - All the contents stored in the file buffer should be auto-flushed.
 - Each log file should have a maximum size of 1 MB.
 - The maximum number of files for the log should be five. Any extra files needed should overwrite the oldest file created.
 - All logs less critical than DEBUG should be written to the file.
- 11.3. Create a category that will capture only log messages from the bookstore application. Determine if the **/var/log/jboss** directory has directories representing each server and inside each of them, there is a **bookstore.log** file.
12. Customize the messaging subsystem to create queues and customize an **in-vm ConnectionFactory**.
The messaging subsystem must be updated to support a message queue with a dead letter queue if some error raises. In addition to that, an expiry queue should be associated with the same queue if the message is not consumed after a certain amount of time. It should be able to receive messages from an application called **messaging-client.war** that should be deployed for testing purposes. Finally, all the messages will be consumed by a message-driven bean running on an application called **messaging-mdb.jar**.
The following characteristics must be used to create each element described previously and they should be configured in the full-ha profile:
- A new connection factory published at `java:/jms/CustomCF`. It should use an **in-vm** connector.
 - A message queue named `TestQueue` that should be bound to the `java:/jms/queue/JB248TestQueue` JNDI name. Take care because the queue address used to bind a queue is different from the queue address passed as a parameter to create a queue.
 - A message queue named `TestDLQ` that should be bound to the `java:/jms/JB248TestDLQ` JNDI name. It will be the dead letter queue that will store all the messages that caused any error (just one) while stored at the `TestQueue`. Take care because the queue address used to bind a queue with a DLQ is different from the queue address passed as a parameter to create a queue.
 - A message queue named `TestExpiry` that should be bound to the `java:/jms/JB248TestExpiry` JNDI name. It will be the expiry queue that will store all the messages that were stored at the `TestQueue` for a five second delay. Take care because the queue address used to bind a queue with an expiry queue is different from the queue address passed as a parameter to create a queue.

Because there are some firewall rules as well as some OS-specific configurations that should be updated, a script will be provided to customize them. It was downloaded by the setup process and is available at **/home/student/JB248/labs/review-lab/artemis-firewalld-rules.sh** on each VM. They must be executed as root.

Lastly, check if the messaging servers can identify each other by running the following command:

```
[domain@172.25.250.254:9990 /] /host=serverA/server=serverA.1/\  
subsystem=messaging-activemq/server=default/\  
cluster-connection=my-cluster:read-attribute(name=topology)
```

- 12.1. Deploy the message queues.
 - 12.2. Identify the queue address from the TestQueue that was published in a server instance. By default the queue address is identical among servers running on the same server group and it will be used to configure the dead letter queue and the expiry queue.
The resulting output is the queue address used to create an address setting that will be used to bound the queue with a dead letter queue and an expiry queue.
 - 12.3. Identify the queue address for the TestDLQ published in a server instance. By default the queue address is identical among servers running on the same server group and it will be used to bind the TestDLQ with TestQueue as a dead letter queue.
The resulting output is the queue address used to create an address setting that will be used to identify the dead letter queue.
 - 12.4. Identify the queue address for the TestExpiry published in a server instance. By default the queue address is identical among servers running on the same server group and it will be used to bind the TestExpiry with TestQueue as an expiry queue.
The resulting output is the queue address used to create an address setting that will be used to identify the expiry queue.
 - 12.5. To configure the dead letter queue and the expiry queue to capture messages from the TestQueue, you should bind them.
 - 12.6. Create a **ConnectionFactory** that will use an **in-vm** connector under the full-ha profile. It should be available for JNDI name lookup at **java:/jms/CustomCF**.
 - 12.7. Run the **artemis-firewalld-rules.sh** to drop firewall rules from each VM.
13. Enforce security concerns by encrypting passwords, disabling local automatic authentication, and creating a security domain.
Any attempt to connect to the domain controller or to the standalone server using a CLI will not request a password if the session is opened locally to the workstation VM. However, this is not a secure behavior and it should be changed to avoid malicious access. In this step, the user access in either the domain controller and the standalone server should require a user name (**eapadmin**) and a password. (**34p@R3dH4t123**).
 - 13.1. Recall that the standalone server management interface is available via loopback interface only and the administration port is 10990 to avoid port collision with the domain controller. Remove from the management realm (which holds the credentials for management purposes) the local authentication. The management realm is under the core-services level, under the security realm service.
Reload the server and connect again to the standalone server using the CLI. This time credentials will be requested.
 - 13.2. Use the opened CLI session to remove the local authentication from the domain controller. The management realm (which holds the credentials for management purposes) is responsible for the local authentication. The management realm is under the core-services level, under the security realm service.

Reload the host named master and connect the CLI again to the domain controller. This time credentials will be requested.

- 13.3. Create a vault where all the passwords will be stored in an encrypted way. The vault will be represented by a keystore generated by the keytool utility from the Java Development Kit (JDK). Each new password added to the keystore is managed by the vault.sh tool provided by EAP 7 and it is available at the **JBOSS_HOME/bin** directory. The file will be saved in the **/opt/domain** directory and it should be saved as **key.store**. To create the keystore a script is provided; run the following command:

```
[student@workstation ~]$ sudo /tmp/keystore-vault.sh
```

- 13.4. Create one parameter that should hold the password from the bookstore data source. The following parameters should be used to run the **vault.sh** script and they must be executed as the **eap** user.

- Keystore file name: **/opt/domain/key.store**
- Keystore password: **34p@R3dH4t123**
- alias: **Vault**
- vault block **bookstore-vb**
- Attribute: **bookstore-pw**
- Value: **redhat**
- Encrypt directory: **/opt/domain/vault**
- iteration: **95**
- salt: **ABCD1234**

Copy the resulting XML from the execution to a file called **/tmp/vault.xml** that will be used to update the **host-slave.xml** files on each host controller. Additionally, take note of the content right below the **Configuration should be done as follows**:. This string will be used to refer to the password from the bookstore app.

- 13.5. Create one parameter to hold the password from the **bksecurity-ds** datasource. The following parameters should be used to run the **vault.sh** script.

- Keystore file name: **/opt/domain/key.store**
- Keystore password: **34p@R3dH4t123**
- alias: **Vault**
- vault block **realm-vb**
- Attribute: **realm-pw**
- Value: **redhat**
- Encrypt directory: **/opt/domain/vault**
- iteration: **95**

- salt: **ABCD1234**

13.6. Transfer the **key.store**, the **vault.xml**, and the **vault** file to other servers by using **scp** to copy these files it to the other servers. Because the **eap** user is a **nologin** user, transfer them as **root**.

Change the owner of these files to **eap**.



Note

Even though an NFS share directory is available, it will not be used. Each time a new password is added, the copy should be executed to avoid NFS share unavailability issues.

13.7. Stop all host controllers and the domain controller to update the **host-slave.xml** configuration file to include the **vault.xml** file created in step 13.4 between the **<extensions>** and **<management>** entries. This configuration will allow EAP to open the keystore file and access the passwords. Press **Ctrl+C** on all the windows running either a host controller or a domain controller. Also hit **Ctrl+C** on the CLI window too.

Using your preferred text editor, open the **vault.xml** file and add the XML configuration to the **host-slave.xml** configuration file as described by the **vault.sh** output.

13.8. Update the password from the datasources created in the full-ha profile by substituting the plain text password to the vault reference mentioned in the **vault.sh** output. It should substitute to something like: **\${VAULT::XXXXX::YYYYY}** .

13.9. Restart the host controllers and the domain controller. To prepare the servers to run using a cluster, the **-Djboss.bind.address.private** must be passed with the IP addresses from **servera** and **serverb** VMs.

13.10. With a web browser, open the management console from the workstation VM. The URL for the management console is **172.25.250.254:9990**. The credentials to access it are **eapadmin/34p@R3dH4t123**. Similarly to step 11.4, test the connection. Navigate to the data source subsystem from the full-ha profile and locate the bksecurity and bookstore data sources and look for the **Test Connection** button to check if the connection was successful.

13.11. Create a security realm that uses the **bksecurity-ds** data source. It is used by a custom application that will require authentication. The password is encrypted in the users table from the bksecurity and it should use the **SHA-256** hash algorithm and a **base64** hash encoding. In the same table there is a column named user name which contains the user name. The roles can be obtained from the roles table. This table has a user name column too that should be used to identify the roles of a user name. A file named **/tmp/bksecurity-domain.cli** is provided to avoid type errors.

It should have the following parameters and it must be deployed on the full-ha profile:

- Security domain name: **bksecurity**
- Login module configuration flag: **required**
- Login module configuration code: **Database**
- Login module configuration options:
- **dsJndiName: java:jboss/datasources/bksecurity-ds**

- *principalsQuery*: **SELECT password FROM users WHERE username = ?**
- *rolesQuery*: **SELECT role, "ROLES" FROM roles WHERE username = ?**
- *hashAlgorithm*: **sha-256**
- *hashEncoding*: **base64**

To start the CLI, use the following command from a terminal window in the workstation VM.

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \
--file="/tmp/tcp-stack.cli"
```

13.12. Deploy the application called **/tmp/example.war** that was downloaded during the setup process. Use the admin / admin credentials to access the application page.

14. Update the amount of memory allocated for all the server groups.

Change the amount of memory allocated to the heap for each server to 256 MB minimum and 512 MB maximum. To change memory settings, update the memory configuration for each server group, instead of changing individually. Additionally, create a default JVM configuration for the server group.

Reload the servers to activate the changes.

15. Customize the root application on the servers.

By default, a welcome page is provided however a custom web application should be used instead. To minimize the risk of environment differences, the default web module must be updated. The file is available at the **/tmp/welcome.war** and it should be initially deployed to all server groups and it must update the default web module.

Remove the default web module.

Update the welcome web module to access the recently updated version of the **welcome.war** using the CLI.

Navigate to the following addresses to check if the welcome page was updated:

- **http://172.25.250.10:8080**
- **http://172.25.250.10:8230**
- **http://172.25.250.11:8080**
- **http://172.25.250.11:8230**

A welcome page with a Red Hat logo should be presented.

16. Enable encrypted communication with the standalone server.

All the requests in this architecture will be sent to the standalone server. To guarantee those requests with sensitive information are not being transmitted in plain text, a certificate must be installed. For this lab, a self-signed certificate will be created, but in a real production environment a signed certificate from a trusted source should be created. To create a self-signed certificate, run the **/tmp/self-signed.sh** script from a terminal window on the workstation VM.

Configure the realm that will refer to the keystore where the certificate is stored. EAP 7 requires that the keystore should be referred by a realm. Add a realm named **HTTPSRealm** to the security-realm from the standalone server.

Link the **HTTPSRealm** to the keystore file with the certificate. It should provide the keystore password (**34p7-k3yst0r3**) and the alias where the certificate is bound to (**appserver**) by using a CLI window. Also recall to reload the standalone server.

Add a new HTTPS listener using the configured **HTTPSRealm** realm.

Test if the server can be accessed using HTTPS. Open a browser and access the standalone server that has a port offset of 1000 to avoid port conflicts with the domain controller. An alert from the browser is expected and it can be safely ignored because we are using a self-signed certificate, not a certificate signed by a trusted source.

17. Configure the cluster to use the TCP stack instead of the default UDP stack.

EAP is configured by default using a UDP stack to create a cluster. Because UDP supports multicast, new servers can be easily identified and added to a cluster. Unfortunately, the network does not support UDP and EAP should use a TCP stack, identifying which servers should be connected by a unicast connection.

To achieve this goal, the JGroups subsystem must be customized to use a protocol called TCPPING that mimics an MPING behavior looking for all servers running on a network. Because TCP cannot multicast messages, the TCPPING must be configured to look for each server that is part of the cluster:

172.25.250.10 port 7600
172.25.250.10 port 7750
172.25.250.11 port 7600
172.25.250.11 port 7750

Also the stack attribute must be updated to support the TCPPING stack. To minimize the risks from updating the stack a new stack should be created called tcpping and it should be the default stack.

To allow communication among the servers, the firewall rules should be customized on each server VM. Recall that the **serverX.2** have a port offset of 150. A script is provided to drop these firewall rules. Run the **/tmp/firewall-cluster.sh** on **servera** and **serverb** VMs.

```
[student@servera ~]$ sudo /tmp/firewall-cluster.sh
```

```
[student@serverb ~]$ sudo /tmp/firewall-cluster.sh
```

Reload the domain controller and all host controllers to make the TCPPING stack valid. From the CLI on the domain controller, run:

18. Create a static load balancer using the standalone server.

From the original architecture, the standalone server will load balance all the requests from an external environment. To make this work, the undertow subsystem must be customized

to support a reverse proxy that will be responsible for load balancing the request among multiple servers. Create a reverse proxy called bookstore-handler in the standalone server.

Configure a socket that will connect to **serverA.1** (172.25.250.10) using the AJP protocol running on TCP port 8009. Name it **remote-serverA.1**.

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverA.1**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverA.1**:

Configure a socket that will connect to the **serverA.2** (172.25.250.10) using the AJP protocol running on TCP port 8159. Name it **remote-serverA.2**.

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverA.2**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverA.2**:

Configure a socket that will connect to serverB.1 (172.25.250.11) using the AJP protocol running on TCP port 8009. Name it **remote-serverB.1**.

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverB.1**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverB.1**:

Configure a socket that will connect to serverB.2 (172.25.250.11) using the AJP protocol running on TCP port 8159. Name it **remote-serverB.2**.

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverB.2**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverB.2**:

Deploy the **bookstore.war** application on all group servers and test it.

19. Run the grading script to check if all the tasks were accomplished by opening a new terminal window from the workstation VM and running the following command:

```
[student@workstation ~]$ lab review-final grade
```

► Solution

Comprehensive Review of Red Hat JBoss Application Administration I

In this lab, you will practice key skills that were presented in this course.

Resources	
Files	/home/student/JB248/install /home/student/JB248/labs/review-final
Application URL	http://localhost:8081 https://172.25.250.254:8443

Outcomes

You should be able to:

- Install EAP 7 on **workstation**, **servera**, and **serverb** VMs.
- Run a standalone server on the **workstation** VM.
- Configure a managed domain in **servera**, **serverb**, and **workstation** VMs.
- Configure all the main subsystems to support a load balanced application running on the managed domain.

Before You Begin

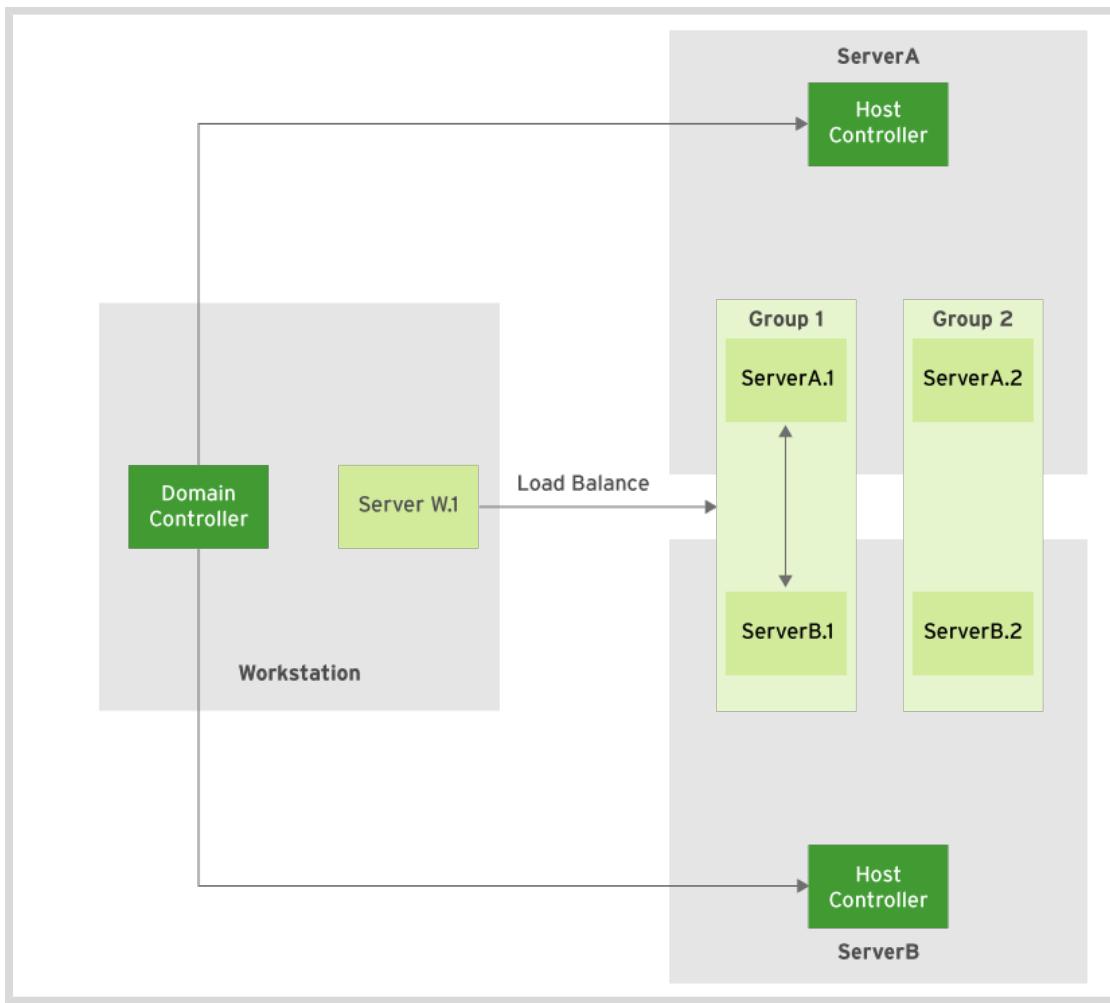
Reset the **workstation**, **servera**, and **serverb** VMs to remove all existing laboratories. Follow the *Orientation to the Classroom Environment* chapter to reset your environment accordingly and run the following command to download the relevant lab directory:

```
[student@workstation ~]$ lab review-final setup
```

1. Install EAP and configure the directory owners on each VM host.

A high-availability EAP environment will be installed on three different hosts (**workstation**, **servera**, and **serverb** VMs) to support a successful e-commerce application. It is a bookstore application using a web application that connects to a MySQL database which stores all production information and orders.

The following diagram shows the server architecture that will be created during this lab. In this environment you will install, configure, and deploy the enterprise application.



The environment is based on three virtual machines running Red Hat Enterprise Linux (RHEL 7) with a minimal set of tools installed, including Java 8:

- **workstation** VM (IP: 172.25.250.254): It is the only VM with a graphical interface installed. It will host the MySQL database and EAP running a standalone server and a

domain controller. The standalone server will be responsible for load balancing requests for the bookstore application.

- **servera** VM (IP: 172.25.250.10): It is a text-based VM responsible for running multiple EAP instances managed by the domain controller installed at the workstation.
- **serverb** VM (IP: 172.25.250.11): Similarly to the **servera** VM, it will run multiple EAP instances.

In this step, EAP will be installed on each host from the classroom environment, using recommended approaches to run EAP as a user with limited access to the OS environment. To access each VM a user is provided with the following credentials:

- *login: student*
- *password: student*

This user has **sudo** privileges and it will be used to install and launch EAP.

EAP must be installed with an answer file to automate the installation process on each host in a uniform way. Each host controller must use the same credentials as mentioned later:

- *username: eapadmin*
- *password: 34p@R3dH4t123*

- 1.1. Create a system user named **eap** on workstation VM to start and stop EAP instances.

On the **workstation** VM, open a terminal window and run the following command to create the user:

```
[student@workstation ~]$ sudo useradd -r eap
```

- 1.2. Create a system user named **eap** on **servera** VM to start and stop EAP instances.

Run the following command to create the **eap** user in the **servera** VM:

```
[student@workstation ~]$ ssh root@servera useradd -r eap
```

- 1.3. Create a system user named **eap** on **serverb** VM to start and stop EAP instances.

Run the following command to create the **eap** user in the **serverb** VM:

```
[student@workstation ~]$ ssh root@serverb useradd -r eap
```

- 1.4. EAP should be installed at **/opt/eap-7.0.0** on each host. Because the **eap** user does not have access to the **/opt** directory, the administrator must set it up as **root** and then change the owner of the **eap-7.0.0** folder to the **eap** user. In order to simplify the installation process, an answer file was downloaded on **workstation**, **servera**, and **serverb** VMs during the installation process at **/home/student/JB248/labs/review-lab**, but some fields must be customized to support the requirements of this lab. The EAP installer and a template answer file are available at **/home/student/JB248/labs/review-final** on each of the hosts.

- 1.5. Log in on each host and install EAP as **student/student** and customize the answer files **/home/student/JB248/labs/review-final/install.xml** and **/home/student/JB248/labs/review-final/install.xml.variables**. The **install.xml** file will be used to configure the location where EAP should be

installed and the login used to access the management console, whereas the latter will be responsible for defining the password for the login defined at **install.xml**.

Use the following as administrator credentials for each server:

- *login: eapadmin*
- *password: 34p@R3dH4t123*



Note

Because the password is complicated, a file called **credentials.txt** is available in **/tmp** directory for copy and paste purposes.

Change the **install.xml** file to:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
<productName>EAP</productName>
<productVersion>7.0.0</productVersion>
<com.izforge.izpack.panels.HTMLLicencePanel id="HTMLLicencePanel"/>
<com.izforge.izpack.panels.TargetPanel id="DirectoryPanel">
<installpath>/opt/eap-7.0.0</installpath>
</com.izforge.izpack.panels.TargetPanel>
<com.izforge.izpack.panels.TreePacksPanel id="TreePacksPanel">
<pack index="0" name="Red Hat JBoss Enterprise Application Platform" selected="true"/>
<pack index="1" name="AppClient" selected="true"/>
<pack index="2" name="Bin" selected="true"/>
<pack index="3" name="XMLs and XSDs" selected="true"/>
<pack index="4" name="Domain" selected="true"/>
<pack index="5" name="Domain Scripts" selected="true"/>
<pack index="6" name="Modules" selected="true"/>
<pack index="7" name="Standalone" selected="true"/>
<pack index="8" name="Standalone Scripts" selected="true"/>
<pack index="9" name="Welcome Content" selected="true"/>
<pack index="10" name="Quickstarts" selected="false"/>
<pack index="11" name="Icons" selected="false"/>
</com.izforge.izpack.panels.TreePacksPanel>
<com.izforge.izpack.panels.UserInputPanel id="CreateUserPanel">
<userInput>
<entry key="adminUser" value="eapadmin"/>
<entry autoPrompt="true" key="adminPassword"/>
</userInput>
</com.izforge.izpack.panels.UserInputPanel>
<com.izforge.izpack.panels.SummaryPanel id="SummaryPanel"/>
<com.izforge.izpack.panels.InstallPanel id="InstallPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="postinstall">
<userInput>
<entry key="postinstallServer" value="false"/>
</userInput>
</com.izforge.izpack.panels.UserInputPanel>
<com.izforge.izpack.panels.UserInputPanel id="vaultsecurity"/>
<com.izforge.izpack.panels.UserInputPanel id="sslsecurity"/>
<com.izforge.izpack.panels.UserInputPanel id="ldapsecurity"/>
```

```

<com.izforge.izpack.panels.UserInputPanel id="ldapsecurity2"/>
<com.izforge.izpack.panels.UserInputPanel id="infinispan"/>
<com.izforge.izpack.panels.UserInputPanel id="Security Domain Panel"/>
<com.izforge.izpack.panels.UserInputPanel id="jsssecuritydomain"/>
<com.izforge.izpack.panels.UserInputPanel id="QuickStartsPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="MavenRepoCheckPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketBindingPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketStandalonePanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketHaStandalonePanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketFullStandalonePanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketFullHaStandalonePanel"/>
<com.izforge.izpack.panels.UserInputPanel id="HostDomainPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketDomainPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketHaDomainPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketFullDomainPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="SocketFullHaDomainPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="ServerLaunchPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="LoggingOptionsPanel"/>
<com.izforge.izpack.panels.UserInputPanel id="JDBC Setup Panel"/>
<com.izforge.izpack.panels.UserInputPanel id="Datasource Configuration Panel"/>
<com.izforge.izpack.panels.ProcessPanel id="ProcessPanel"/>
<com.izforge.izpack.panels.ShortcutPanel id="ShortcutPanel"/>
<com.izforge.izpack.panels.FinishPanel id="FinishPanel"/>
</AutomatedInstallation>

```

Also update the **install.xml.variables** to:

```
adminPassword=34p@R3dH4t123
```

Additionally, EAP must be installed in the **/opt/eap-7.0.0/** directory. Install EAP as the **root** user because the **/opt** directory belongs by default to the **root** user.

- Run the EAP installer using the answer files updated in the previous step. The installer is available at **/home/student/JB248/installers**.

To run the EAP installer, run the following command on the workstation:

```
[student@workstation ~]$ sudo java -jar \
/home/student/JB248/installers/jboss-eap-7.0.0-installer.jar \
/home/student/JB248/labs/review-final/install.xml \
-variablefile /home/student/JB248/labs/review-final/install.xml.variables
```

Check if EAP was installed on the workstation by running the following command:

```
[student@workstation ~]$ ls -la /opt/eap-7.0.0
```

You should see an output similar to the following:

```
drwxrwxr-x. 3 root root 4096 Apr 25 14:19 appclient
drwxrwxr-x. 3 root root 4096 Apr 25 14:19 bin
drwxrwxr-x. 4 root root 4096 Apr 25 14:19 docs
drwxrwxr-x. 5 root root 4096 Apr 25 14:19 domain
drwxrwxr-x. 2 root root 4096 Apr 25 14:20 installation
drwxrwxr-x. 2 root root 4096 Apr 25 14:19 .installation
-rw-rw-r--. 1 root root 419 Nov 25 22:23 JBossEULA.txt
```

```
-rw-rw-r-- 1 root root 366430 Nov 25 22:23 jboss-modules.jar  
-rw-rw-r-- 1 root root 26530 Nov 25 22:23 LICENSE.txt  
drwxrwxr-x 3 root root 4096 Apr 25 14:19 modules  
drwxrwxr-x 6 root root 4096 Apr 25 14:20 standalone  
drwxrwxr-x 2 root root 4096 Apr 25 14:20 uninstaller  
-rw-rw-r-- 1 root root 68 Nov 25 22:23 version.txt  
drwxrwxr-x 4 root root 4096 Apr 25 14:20 welcome-content
```

1.7. Install EAP on **servera** VM.

To install the EAP installer on the **servera**, run the following command:

```
[student@workstation ~]$ ssh root@servera java -jar \  
/home/student/JB248/install/jboss-eap-7.0.0-installer.jar \  
/home/student/JB248/labs/review-final/install.xml \  
-variablefile /home/student/JB248/labs/review-final/install.xml.variables
```

1.8. Check if EAP was installed on **servera** VM.

```
[student@workstation ~]$ ssh root@servera ls -la /opt/eap-7.0.0
```

1.9. Install EAP on **serverb** VM.

```
[student@workstation ~]$ ssh root@serverb java -jar \  
/home/student/JB248/install/jboss-eap-7.0.0-installer.jar \  
/home/student/JB248/labs/review-final/install.xml \  
-variablefile /home/student/JB248/labs/review-final/install.xml.variables
```

1.10. Check if EAP was installed on **serverb** VM.

```
[student@workstation ~]$ ssh root@serverb ls -la /opt/eap-7.0.0
```

1.11. Update the owner of the directory where EAP was installed on each VM to the **eap** user in order to run and access all files from the EAP installation directory.

```
[student@workstation ~]$ sudo chown eap:eap -R /opt/eap-7.0.0  
[student@workstation ~]$ ssh root@servera chown eap:eap -R /opt/eap-7.0.0  
[student@workstation ~]$ ssh root@serverb chown eap:eap -R /opt/eap-7.0.0
```

Check if the owner was changed by running the following command:

```
[student@workstation ~]$ ls -ld /opt/eap-7.0.0
```

You should see output similar to the following:

```
drwxrwxr-x. 12 eap eap 4096 Apr 25 14:21 /opt/eap-7.0.0/
```

2. Configure the standalone server to run in a different directory.

A standalone server will run on the **workstation** as a load balancer. It will be responsible for forwarding requests to the servers running on a managed domain whose domain controller will also run on the workstation. In this step the standalone server will be customized to avoid port conflicts with the domain controller.

To create a standalone server without touching the EAP installation, a custom directory will be used to store all configuration files used by this review lab. In this step you will create a directory **only on the workstation VM** with the set of directories required to start an EAP standalone server instance. Follow the directives listed below to configure the standalone server:

- *base directory: /opt/standalone*
- *directory owner: eap*

Change the standalone server to run on a set of ports different from the default to avoid port conflicts. Because a managed domain will also run on the same VM and some ports are identical to the ones used by a standalone server, the standalone server configuration file must be updated. To make sure the changes are persisted, change the standalone configuration file to run on a port offset value of 1000.

Recall that all network changes require a full reload of the standalone server to take effect.

2.1. Create a directory called **/opt/standalone**.

```
[student@workstation ~]$ sudo mkdir /opt/standalone
```

2.2. Copy the standalone directory from the EAP installation to the **/opt/standalone**.

```
[student@workstation ~]$ sudo cp -R /opt/eap-7.0.0/standalone /opt
```

2.3. Change the owner to the **eap** user.

```
[student@workstation ~]$ sudo chown -R eap:eap /opt/standalone
```

2.4. Start the EAP standalone server using the recently created directory.

```
[student@workstation bin]$ sudo -u eap /opt/eap-7.0.0/bin/standalone.sh \
-Djboss.server.base.dir=/opt/standalone/
```



Note

The command used to start the standalone server will be used throughout this review. To minimize typing it many times, it is strongly suggested to create a text file with the command that can be copied and pasted.

2.5. Open a browser window and access `http://localhost:9990` to check if the EAP management console was started. The credentials to access it are:

- *username: eapadmin*
- *password: 34p@R3dH4t123*



Note

Because the password is complicated, a file called **credentials.txt** is available at the `/tmp` directory for copy and paste purposes.

- 2.6. In this step the CLI will be used to update network ports. Start it as the **eap** user and connect to the standalone server.

```
[student@workstation bin]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh \
--connect --controller=localhost:9990
```

- 2.7. Update the port offset from the standalone server to 1000. This change will require a server reload, and reconnect CLI because the connection will be stale.

From the CLI, run the following commands:

```
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets:\nwrite-attribute(name=port-offset,value=1000)
```

The expected output is:

```
{
  "outcome" => "success",
  "response-headers" => {
    "operation-requires-reload" => true,
    "process-state" => "reload-required"
  }
}
```

To update the port offset value, reload the server:

```
[standalone@localhost:9990 /] :reload
```

The expected output is:

```
{
  "outcome" => "success",
  "result" => undefined
}
```

- 2.8. After running the command, disconnect from the server because the administration port was changed.

Run the following command from the CLI:

```
[standalone@localhost:9990 /] exit
```

Test if the configuration was successfully set by accessing <http://localhost:9080> from the web browser. A welcome page should be displayed.

Connect to EAP using the CLI using the following command:

```
[student@workstation bin]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh \
--connect --controller=localhost:10990
```

- 2.9. Update the public interface to make it available for external access. The load balancer must be visible from an outside environment, via the **eth0** interface. However, the default configuration makes it visible only to the loopback consumption.

From the CLI, run the following commands:

```
[standalone@localhost:10990 /] /interface=public:\nwrite-attribute(name=inet-address,value=172.25.250.254)
```

The expected output is:

```
{\n    "outcome" => "success",\n    "response-headers" => {\n        "operation-requires-reload" => true,\n        "process-state" => "reload-required"\n    }\n}
```

Reload the server configuration to apply the changes.

Run the following command to reload the server and update the address:

```
[standalone@localhost:10990 /] :reload
```

The expected output is:

```
{\n    "outcome" => "success",\n    "result" => undefined\n}
```

Disconnect from the CLI to refresh the configuration.

Because the administration port was changed, you need to disconnect from the server and then reconnect. Run the following command to disconnect:

```
[standalone@localhost:10990 /] exit
```

Test if the configuration was successfully set by accessing <http://172.25.250.254:9080> from the web browser.

Reopen the CLI session with the standalone server by running the following command from the same terminal window where CLI was exited.

```
[student@workstation bin]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh \\n--connect --controller=localhost:10990
```

3. Configure the domain and host controllers in a custom directory location.

The standalone server was created to become a load balancer. However, a managed domain will be responsible for deploying JavaEE applications on each application server.

This architecture can be easily expanded creating new host controllers and binding them to the domain controller. Any update or customization can be executed by the management console or the CLI, without a large set of commands.

In this step, the domain controller will be configured and started on the **workstation** VM. Likewise, a host controller will be installed on **servera** and **serverb**. They will use a different base directory from the installation process to keep the install directory pristine.

To keep consistency, the base directory, where all the directories and files are stored in a controller, will be stored in the same directory owned by the **eap** user. All changes will be

made in the configuration file to minimize the need to change parameters during the start-up process and to simplify the way a controller should be started.

- *Base directory: /opt/domain*
 - *Base directory owner: eap*
- 3.1. Copy as root the base directory from the EAP installed in step 1 to the directory created in the previous step for each host VM. Also change the owner of the directory again because the owner was inherited by the **cp** command.
- Copy the base directory (located at **/opt/eap-7.0.0/domain**) used by the domain controller in the **workstation** VM to **/opt/domain** directory.

On the **workstation** VM, run the following commands as the **student** user.

```
[student@workstation ~]$ sudo cp -R /opt/eap-7.0.0/domain /opt/
```

Change the owner of the **/opt/domain** directory to the **eap** user on the **workstation** VM.

On the **workstation** VM, run the following commands as the **student** user.

```
[student@workstation ~]$ sudo chown -R eap:eap /opt/domain
```

- 3.2. Copy the base directory (located at **/opt/eap-7.0.0/domain**) used by the domain controller in the **servera** VM to **/opt/domain** directory.

On the **servera** VM, run the following commands as the **student** user.

```
[student@servera ~]$ sudo cp -R /opt/eap-7.0.0/domain/ /opt/
```

Change the owner of the **/opt/domain** directory to the **eap** user on the **servera** VM.

On the **servera** VM, run the following commands as the **student** user.

```
[student@servera ~]$ sudo chown -R eap:eap /opt/domain
```

- 3.3. Copy the base directory (located at **/opt/eap-7.0.0/domain**) used by the domain controller on the **serverb** VM to **/opt/domain** directory.

On the **serverb** VM, run the following commands as the **student** user.

```
[student@serverb ~]$ sudo cp -R /opt/eap-7.0.0/domain/ /opt/
```

Change the owner of the **/opt/domain** directory to the **eap** user on the **serverb** VM.

On the **serverb** VM, run the following commands as the **student** user.

```
[student@serverb ~]$ sudo chown -R eap:eap /opt/domain
```

- 3.4. Test the controller on each VM.

In this step, we will check if the domain controller can be started individually on each VM for verification purposes. To start it as a domain controller, run the **domain.sh** script from the **JBOSS_HOME/bin** directory and point to the **/opt/domain** directory as the base directory. Recall that the owner of the base directory is the **eap** user and it

should be started using the **sudo -u** command. Additionally, a configuration file called **host-master.xml**, used to start it as a domain controller, must be passed to start EAP without any conflicts.

Start the EAP as a managed domain from the workstation VM. The **eax** user should be used to start the process and the base directory should point to the **/opt/domain** directory.

Run the following command to start a domain controller on the **workstation** VM:

```
[student@workstation ~]$ sudo -u eax /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain --host-config=host-master.xml
```

Output similar to the following is expected:

```
[Host Controller] 19:29:47,270 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0025: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) (Host
Controller) started in 4717ms - Started 51 of 51 services (14 services are lazy,
passive or on-demand)
```

- 3.5. Start the EAP as a managed domain from the **servera** VM. The **eax** user should be used to start the process and the base directory should point to the **/opt/domain** directory.

Run the following command to start a domain controller on the **servera** VM:

```
[student@servera ~]$ sudo -u eax /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain --host-config=host-master.xml
```

Output similar to the following is expected:

```
[Host Controller] 19:29:47,270 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0025: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) (Host
Controller) started in 4717ms - Started 51 of 51 services (14 services are lazy,
passive or on-demand)
```

- 3.6. Start EAP as a managed domain from the **serverb** VM. The **eax** user should be used to start the process and the base directory should point to the **/opt/domain** directory.

Run the following command to start a domain controller on the **serverb** VM:

```
[student@serverb ~]$ sudo -u eax /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain --host-config=host-master.xml
```

Output similar to the following is expected:

```
[Host Controller] 19:29:47,270 INFO [org.jboss.as] (Controller Boot Thread)
WFLYSRV0025: JBoss EAP 7.0.0.GA (WildFly Core 2.1.2.Final-redhat-1) (Host
Controller) started in 4717ms - Started 51 of 51 services (14 services are lazy,
passive or on-demand)
```

- 3.7. Stop each controller by pressing **Ctrl+C** on each terminal window where EAP was started.

4. Configure network permissions for the domain and host controller from the managed domain.

Some low level configuration should be done on the domain controller to support external connections from the host controllers. The management interface from the domain controller by default is visible only from the loopback interface. Host controllers, on the other hand, will need to connect to the server using the ethernet interface instead. In this step the network configuration will be updated using the CLI and some firewall commands. The cluster password used by the **full-ha** profile should be updated to use the same password as the **eapadmin** user to allow authentication among all the messaging services running on EAP and to exchange their data.

- 4.1. To change the network interface configuration, start the EAP domain controller on the **workstation** VM to configure it using CLI. It should use the **/opt/domain/configuration/host-master.xml** using as the base directory the **/opt/domain** directory. Recall that EAP should be running using the **eap** user.

Start the domain controller of the managed domain from the **workstation** VM as the **eap** user.

Run the following command from the **workstation**:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain --host-config=host-master.xml
```



Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted in the workstation VM.

- 4.2. Start the CLI installed in the **JBOSS_HOME/bin** directory from the workstation VM and connect to domain controller running locally. Recall that EAP is running with the **eap** user, and to modify any configuration the CLI process must also be started as the **eap** user.

Start the CLI from the workstation VM as the **eap** user.

Run the following command from a new terminal window from the workstation:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh
```

Connect to the domain controller started at the **localhost**.

A prompt should be displayed. Connect it to the domain controller using the following command:

```
[disconnected /] connect localhost:9990
```

- 4.3. Update the management interface from the domain controller to use the workstation IP address using the CLI. Recall that the domain controller name is master and the IP address from the workstation is 172.25.250.254.

Run from the CLI:

```
[domain@localhost:9990 /] /host=master/interface=management:\ \
write-attribute(name=inet-address,value=172.25.250.254)
```

The expected output is:

```
{  
    "outcome" => "success",  
    "result" => undefined,  
    "server-groups" => undefined,  
    "response-headers" => {"process-state" => "reload-required"}  
}
```

- 4.4. Update the password used by the messaging subsystem to allow authentication among all the ActiveMQ instances running on the domain controller in a clustered environment. Recall that the profile that supports a clustered environment and with messaging subsystem support is the full-ha profile, and the password is **34p@R3dH4t123**.

Run the following command:

```
[domain@localhost:9990 /] /profile=full-ha/subsystem=messaging-activemq/\  
server=default:write-attribute(name=cluster-password,value=34p@R3dH4t123)
```



Note

Because the password is complicated, a file named **credentials.txt** is available at the **/tmp** directory for copy and paste purposes.

The expected output is:

```
{  
    "outcome" => "success",  
    "result" => undefined,  
    "server-groups" => undefined,  
    "response-headers" => {"process-state" => "reload-required"}  
}
```

- 4.5. The server configuration must be reloaded to activate the IP address changes.

Run the following command to reload the configuration files:

```
[domain@localhost:9990 /] reload --host=master
```



Note

A message similar to the following should appear:

```
[disconnected /] Failed to establish connection in 6047ms
```

This is because the management interface was updated and EAP cannot be reached by the CLI using the loopback interface.

- 4.6. Configure firewall rules to allow access from host controllers to the domain controller. By default the TCP ports 9990 and 9999 should be opened for public access in the workstation VM. Reload the rules immediately after setting these rules. A script is

provided to disable these rules and it is available at `/tmp/firewall-mgmt.sh` in the workstation VM.

```
[student@workstation ~]$ sudo /tmp/firewall-mgmt.sh
```

- 4.7. Unfortunately the CLI connection to the domain controller will be stale right after the reload because the domain controller will now listen to the IP address 172.25.250.254. Connect again to the domain controller with the CLI.

Run the following commands:

```
[disconnect /] connect 172.25.250.254:9990  
[domain@172.25.250.254:9990 /]
```

5. Connect to the host controller running on the **serverA** VM using the domain controller.

The EAP installation on the **serverA** VM will be a host controller in a managed domain. It should point to the domain controller started on the **workstation** VM. To achieve this goal, the host controller from **serverA** VM have the following characteristics:

- *Host controller name:* **serverA**
- *Domain controller IP address:* **172.25.250.254** (the IP address of the **workstation** VM)

All the servers managed by the host controller must be accessible externally and the public interface (where all servers will listen to requests) must point to the network interface, not the loopback interface. Therefore, the public interface on the host controller must be configured as follows:

- *interface name:* **public**
- *IP address:* **172.25.250.10**

To prevent unknown hosts from connecting to the domain controller, create a management user. Use the following parameters when creating a the management user:

- *login:* **jbossmgmt**
- *password:* **JBoss@RedHat123**
- *Is this new user going to be used for one AS process to connect to another AS process?:* **yes**

The credentials should be added to the `/opt/domain/configuration` directory and the `add-user.sh` script should be executed with the `eap` user.

Configure firewall rules to allow external access to the servers running on the host controller. It should allow external access for TCP port 8080 and 8230 (the two existing servers that will be removed during a later step).

Finally, remove the server associated with this server, because new ones should be created and they will be associated with different server groups that will be updated.

- 5.1. Create the management user with the `add-user.sh` script from the workstation VM as the `eap` user to update the `mgmt-users.properties` file.

From the workstation VM, open a new terminal window, and run the following command:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/add-user.sh \
-dc /opt/domain/configuration
```

Copy the resulting XML content to the file: **/home/student/secret.xml** because it will be used by the host controllers to connect.

- 5.2. Open an SSH session as the **student** user from the **workstation** VM to the **serverA** VM and edit the **host-slave.xml** file from the base directory to use the XML tag copied from the previous step to update the existing **secret** tag. Remember to edit the file as the **eap** user.

Update the **remote** tag from the same XML file to add the username attribute with the login **jbossmgmt** used to create the management user.

```
...
<domain-controller>
    <remote username="jbossmgmt" security-realm="ManagementRealm">
        <discovery-options>
...

```

- 5.3. From the **serverA** VM edit the file **/opt/domain/configuration/host-slave.xml** to change the host name to **serverA** and update the IP address pointing to the domain controller address (172.25.250.254).

Update the host name from the **serverA** VM.

In the beginning of the file, update the host name as follows:

```
<host name="serverA" xmlns="urn:jboss:domain:4.1">
...

```

- 5.4. Update the IP address from the static-discovery tag from the **serverA** host controller to point to 172.25.250.254.

Change the **static-discovery** tag to point to the domain controller.

```
...
<static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}" host="172.25.250.254" port="${jboss.domain.master.port:9999}"/>
...
```

- 5.5. Enable external access to the web applications by updating the public interface where EAP will listen to requests. The same configuration file is responsible for enabling this access. Recall that the public IP address is 172.25.250.10.

In the **host-slave.xml** file, change the **interface** tag named **public** to point to the network interface, not to the 127.0.0.1 address.

```
...
<interface name="public">
    <inet-address value="172.25.250.10"/>
</interface>
...

```

- 5.6. Start the EAP host controller from the **serverA** VM using **/opt/domain** as the base directory. Also recall that the owner of the base directory is **eap** and you should use the **host-slave.xml** configuration file that was customized in the previous step. Recall that a script was created previously to start EAP.

From a terminal window in the **serverA** VM, run the following command:

```
[student@servera ~]$ sudo -u eap /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain --host-config=host-slave.xml
```



Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted. Create it on the **serverA** VM.

- 5.7. Check if the **serverA** host controller was connected to the domain controller by checking the logs from the domain controller terminal window.
- 5.8. Update the firewall rules on the **serverA** VM to allow external machines to access the servers running on the **serverA** host controller.

```
[student@servera ~]$ sudo /tmp/firewall-http.sh
```

- 5.9. Test if the servers are available via the public interface. Using a web browser, open the following URLs and check if the EAP welcome pages are displayed:
 - <http://172.25.250.10:8080>
 - <http://172.25.250.10:8230>
- 5.10. In the CLI terminal window, stop the servers running on the host controller **serverA** to remove them.

```
[domain@172.25.250.254:9990] :stop-servers
```

- 5.11. Remove the server-one instance from **serverA** host controller to avoid conflicts with the other host controller that will be installed.

In order to achieve our goal, use the CLI and remove the servers from the host named **serverA**.

```
[domain@172.25.250.254:9990] /host=serverA/server-config=server-one:remove
```

- 5.12. Remove the server-two instance from **serverA** host controller to avoid conflicts with the other host controller that will be installed.

In order to achieve our goal, use the CLI and remove the servers from the host named **serverA**.

```
[domain@172.25.250.254:9990] /host=serverA/server-config=server-two:remove
```

6. Connect the host controller running on the **serverB** VM to the domain controller.

The EAP installation on the **serverB** VM will be used as a second host controller of a managed domain. It should point to the domain controller that will be started on the workstation VM. Similarly to **serverA** host controller, the **serverB** VM should follow some characteristics:

- *Host controller name: **serverB***
- *Domain controller IP address: **172.25.250.254** (the IP address of the **workstation** VM)*

All the servers managed by the host controller must be accessible externally and the public interface (where all servers will listen to requests) must point to the network interface, not the loopback interface. Therefore, the public interface of the host controller must be configured as follows:

- *interface name: **public***
- *IP address: **172.25.250.11***

To prevent unknown hosts from connecting to the domain controller, a management user was created in the previous step and it should be associated in a similar fashion as the **serverA** host controller.

Finally, remove the server associated with this server, because new ones should be created and they will be associated with different server groups that will be updated.

- 6.1. Open an SSH session as the student user from the **workstation** VM to the **serverB** VM and edit the **host-slave.xml** file from the base directory to use the XML tag copied from the previous step to update the existing **secret** tag. Remember to edit the file as the **eap** user.

Update the **remote** tag from the same XML file to add the user name attribute with the login **jbossmgmt** used to create the management user.

```
...  
<domain-controller>  
  <remote username="jbossmgmt" security-realm="ManagementRealm">  
    <discovery-options>  
  ...
```

- 6.2. From the **serverB** VM edit the file **/opt/domain/configuration/host-slave.xml** to change the host name to **serverB** and update the IP address pointing to the domain controller address (172.25.250.254).

In the beginning of the file, update the host name as follows:

```
<host name="serverB" xmlns="urn:jboss:domain:4.1">  
  ...
```

Change the **static-discovery** tag to point to the domain controller.

```
...  
<static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"  
  host="172.25.250.254" port="${jboss.domain.master.port:9999}"/>  
...
```

- 6.3. Enable external access to the web applications by updating the public interface where EAP listens to requests. The same configuration file is responsible for enabling this access. Recall that the serverB public IP address is 172.25.250.11.

In the **host-slave.xml** file, change the **interface** tag named public to point to the network interface, not to the 127.0.0.1 address.

```
....  
    <interface name="public">  
        <inet-address value="172.25.250.11"/>  
    </interface>  
....
```

- 6.4. Start EAP host controller on **serverB** VM using **/opt/domain** as the base directory. Also recall that the owner of the base directory is **ejap** and you should use the **host-slave.xml** configuration file that was customized in the previous step.

From a terminal window in the **serverA** VM, run the following command:

```
[student@serverb ~]$ sudo -u ejap /opt/eap-7.0.0/bin/domain.sh \  
-Djboss.domain.base.dir=/opt/domain --host-config=host-slave.xml
```



Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted from the **serverB** VM.

- 6.5. Check if the **serverB** host controller was connected to the domain controller by checking the logs from the domain controller terminal window.
- 6.6. Update the firewall rules from the serverB VM to allow external machines to access the servers running on the serverB host controller. The script **/tmp/firewall-http.sh** with the commands to drop the rules is available in the **/tmp** directory:

```
[student@serverb ~]$ sudo /tmp/firewall-http.sh
```

- 6.7. Test if the servers are available via the public interface. Using a web browser, open the following URLs and check if the EAP welcome pages are displayed:
 - <http://172.25.250.11:8080>
 - <http://172.25.250.11:8230>
- 6.8. In the CLI terminal window, stop the servers running on the host controller **serverB** so that you can remove them.

```
[domain@172.25.250.254:9990] :stop-servers
```

- 6.9. Remove the **server-one** instance.

To remove the server from the host **serverB**, run the following command:

```
[domain@172.25.250.254:9990] /host=serverB/server-config=server-one:remove
```

6.10. Remove the **server-two** instance.

To remove the server from the host **serverB**, run the following command:

```
[domain@172.25.250.254:9990] /host=serverB/server-config=server-two:remove
```

7. Customize the server groups to support a high availability environment.

The default server groups provided by EAP are focused on testing. In an actual production environment that must address high availability concerns, and that should avoid downtime of a critical application, two server groups should be created that share the same profile.

In this step, the existing server groups will be removed:

- main-server-group
- other-server-group

Two new server groups will be created focused on an environment that should support a rolling upgrades. Thus, the following configuration must be used by the server groups:

First server group characteristics:

- Server group name: **Group1**
- Profile: **full-ha**
- Socket binding configuration: **full-ha-sockets**

Second server group characteristics:

- Server group name: **Group2**
- Profile: **full-ha**
- Socket binding configuration: **full-ha-sockets**

7.1. Remove server group **main-server-group** from the domain controller.

```
[domain@172.25.250.254:9990] /server-group=main-server-group:remove
```

An "**outcome**" => "**success**" is expected.

7.2. Remove server group **other-server-group** from the domain controller.

```
[domain@172.25.250.254:9990] /server-group=other-server-group:remove
```

An "**outcome**" => "**success**" is expected.

7.3. Create a new server group called **Group1** on the domain controller.

To create these server groups, run the following CLI commands:

```
[domain@172.25.250.254:9990] /server-group=Group1:add\
(profile=full-ha,socket-binding-group=full-ha-sockets)
```

An "outcome" => "success" is expected.

- 7.4. Create a new server group called **Group2** on the domain controller.

```
[domain@172.25.250.254:9990] /server-group=Group2:add\  
  (profile=full-ha,socket-binding-group=full-ha-sockets)
```

An "outcome" => "success" is expected.

- 7.5. Reload the configuration from the **serverA** host controller to update the servers.

```
[domain@172.25.250.254:9990] reload --host=serverA
```

- 7.6. Reload the configuration from the **serverB** host controller to update the servers.

```
[domain@172.25.250.254:9990] reload --host=serverB
```

- 7.7. Reload the configuration from the domain controller to update the servers.

```
[domain@172.25.250.254:9990] reload --host=master
```

- 7.8. Exit the CLI.

```
[domain@172.25.250.254:9990] exit
```

- 7.9. Reconnect to the managed domain server.

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh \  
  --connect --controller=172.25.250.254:9990
```

8. Create servers on the **serverA** host controller.

In this step, two new servers will be created on the **serverA** host controller. Create two servers associated with the server groups created in the previous step. They will run on the **serverA** VM using the following definitions. Recall that all new servers are not started automatically:

First server:

- Name: **serverA.1**
- Server group: **Group1**
- Socket binding port offset: None (Leave at default value of 0)
- Auto start?: **true**

Second server:

- Name: **serverA.2**
- Server group: **Group2**
- Socket binding port offset: **150**

- *Auto start?: true*

8.1. Configure the **serverA.1** on **serverA**.

Run the following command from the CLI:

```
[domain@172.25.250.254:9990 /] /host=serverA/server-config=serverA.1:add\
(auto-start=true,group=Group1,socket-binding-port-offset=0)
```

8.2. Configure the **serverA.2** on **serverA**.

Run the following command from the CLI:

```
[domain@172.25.250.254:9990 /] /host=serverA/server-config=serverA.2:add\
(auto-start=true,group=Group2,socket-binding-port-offset=150)
```

8.3. Start **serverA.1**.

```
[domain@172.25.250.254:9990 /] /host=serverA/server-config=serverA.1:start
```

8.4. Start **serverA.2**.

```
[domain@172.25.250.254:9990 /] /host=serverA/server-config=serverA.2:start
```

8.5. Test if the servers are running and if they are accessible with a web browser from the workstation VM. Use the following addresses to check if the welcome page is displayed.

- <http://172.25.250.10:8080>
- <http://172.25.250.10:8230>

9. Create servers on the **serverB** host controller.

In this step, two new servers will be created on the **serverB** host controller. Create two servers associated with the server groups created in the previous step. They will run on the **serverB** VM using the following definitions. Recall that all new servers are not started automatically:

First server:

- *Name: serverB.1*
- *Server group: Group1*
- *Socket binding port offset: None (Leave at default value of 0)*
- *Auto start?: true*

Second server:

- *Name: serverB.2*
- *Server group: Group2*
- *Socket binding port offset: 150*
- *Auto start?: true*

9.1. Configure the **serverB.1** on **serverB**.

Run the following commands in the CLI:

```
[domain@172.25.250.254:9990 /] /host=serverB/server-config=serverB.1:add\
(auto-start=true,group=Group1,socket-binding-port-offset=0)
```

9.2. Configure the **serverB.2** on **serverB**.

Run the following commands in the CLI:

```
[domain@172.25.250.254:9990 /] /host=serverB/server-config=serverB.2:add\
(auto-start=true,group=Group2,socket-binding-port-offset=150)
```

9.3. Start **serverB.1**.

```
[domain@172.25.250.254:9990 /] /host=serverB/server-config=serverB.1:start
[domain@172.25.250.254:9990 /] /host=serverB/server-config=serverB.2:start
```

9.4. Test if the servers are running and if they are accessible with a web browser from the **workstation** VM. Use the following addresses to check if the welcome page is displayed.

- <http://172.25.250.11:8080>
- <http://172.25.250.11:8230>

10. Install the MySQL JDBC driver as a module, install the driver in the subsystem, and create the data sources.

In the managed domain, two databases instances from MySQL will be used by two applications:

- *Bookstore*: The bookstore application will store all the orders and products information from a e-commerce application.
- *Example*: The example application will be used to check user names and passwords stored at the MySQL database instance called **bksecurity**.

Both database instances are managed by the workstation VM as a service. This step will create and configure both data sources to connect to these databases and make them available for those applications.

Details about each data source will be provided later.

A JDBC driver is a JAR file responsible for accessing a database by a Java application. RHEL7 provides a JDBC driver for MySQL from a yum repository. It is available on all VMs at the **/usr/share/java** directory under the name of **mysql-connector-java.jar**. In this step, the JDBC driver should be installed as a module to allow a reusable approach to connect to a database throughout the host controllers.

Any JDBC driver requires that some dependencies (already provided by EAP) should be available, and they should be defined as dependencies:

- **javax.api**
- **java.transaction.api**

Furthermore, the driver must have a unique name. Usually, the driver vendor ID is used, and for MySQL it is the reverse address name **com.mysql**.

To enable these modules, they should be installed using a disconnected CLI session on each host in the managed domain. Recall that the module will be installed in the same directory where EAP is installed and the command should be executed as the **eap** user.

- 10.1. Start a disconnected CLI session on the **workstation** VM.

Open a terminal window on the workstation VM and run the following command:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh
```

- 10.2. Add the module to the domain controller.

In the CLI session, run the following command:

```
[disconnected /] module add --name=com.mysql \
--resources=/usr/share/java/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

Determine if the module was installed by inspecting the directory **/opt/eap-7.0.0/modules/com/mysql/main** directory.

- 10.3. Exit the CLI window.

```
[disconnected /] exit
```

- 10.4. Open a CLI disconnected session on the **servera** VM.

Run the following command:

```
[student@servera ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh
```

- 10.5. Add a module to the **servera** VM.

In the CLI session, run the following command:

```
[disconnected /] module add --name=com.mysql \
--resources=/usr/share/java/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

Evaluate if the module was installed by inspecting the directory **/opt/eap-7.0.0/modules/com/mysql/main** directory.

- 10.6. Exit the CLI window.

```
[disconnected /] exit
```

- 10.7. Start a disconnected CLI session on the **serverb** VM.

Open a terminal window on the **serverb** VM and run the following command:

```
[student@serverb ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh
```

- 10.8. Add a module to the **serverb** VM.

In the CLI session, run the following command:

```
[disconnected /] module add --name=com.mysql \
--resources=/usr/share/java/mysql-connector-java.jar \
--dependencies=javax.api,javax.transaction.api
```

Determine if the module was installed by inspecting the directory **/opt/eap-7.0.0/modules/com/mysql/main** directory.

10.9. Exit the CLI window:

```
[disconnected /] exit
```

10.10. Create a reference in the domain controller to the module recently installed. The driver will be used by the profile called **full-ha** that is used by all the server groups created so far. It must be added to the datasource subsystem and it should be named mysql. Recall that the module was named **com.mysql**. Use the CLI open and connected to the domain controller to configure the module.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=datasources/\
jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql)
```

Make sure the driver was installed by running the following command:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=datasources/\
jdbc-driver=mysql:read-resource
```

The expected output is:

```
{
  "outcome" => "success",
  "result" => {
    "deployment-name" => undefined,
    "driver-class-name" => undefined,
    "driver-datasource-class-name" => undefined,
    "driver-major-version" => undefined,
    "driver-minor-version" => undefined,
    "driver-module-name" => "com.mysql",
    "driver-name" => "mysql",
    "driver-xa-datasource-class-name" => undefined,
    "jdbc-compliant" => undefined,
    "module-slot" => undefined,
    "profile" => undefined,
    "xa-datasource-class" => undefined
  }
}
```

10.11. Configure the datasource to access the bookstore application database. Because this is a configuration used by a profile and the only one used is the full-ha, create it using the following parameters. A file called **/tmp/bookstore.cli** is provided to minimize typing:

- Datasource type: **Non-XA**

- JDBC provider: **MySQL datasource**
- JNDI name: **java:jboss/datasources/bookstore**
- Datasource name: **bookstore**
- Connection URL: **jdbc:mysql://172.25.250.254:3306/bookstore**
- User name: **bookstore**
- Password: **redhat**

The resulting content for the file should be:

```
data-source add --profile=full-ha --name=bookstore \
--driver-name=mysql --jndi-name=java:jboss/datasources/bookstore \
--connection-url=jdbc:mysql://172.25.250.254:3306/bookstore \
--user-name=bookstore --password=redhat
```

To configure the data source, use the CLI with the **--file** parameter to load the **/tmp/bookstore.cli**.

To create the data source, run the following command from a workstation terminal window:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \
--file="/tmp/bookstore.cli"
```



Note

This command will be used many times throughout this lab and to minimize typing, it is strongly suggested to create a text file with the command that can be copied and pasted from the workstation VM.

- 10.12. Test if the datasource is working by accessing the management console using a web browser. Access the management console by navigating <http://172.25.250.254:9990>. Use the user name **eapadmin** and the password **34p@R3dH4t123**. At the top of the page, click **Configuration** → **Profiles** → **full-ha** → **Subsystems** → **Datasources** → **Non-XA** → **bookstore**. Click **View** and in the overview page click the **Connection** and then **Test Connection**.
- 10.13. Configure the data source to access the example application database. Because this is a configuration used by a profile and the only one used is the full-ha, create it using the following parameters:
 - Datasource type: **Non-XA**
 - JDBC provider: **MySQL datasource**
 - JNDI name: **java:jboss/datasources/bksecurity-ds**
 - Datasource name: **bksecurity**
 - Connection URL: **jdbc:mysql://172.25.250.254:3306/bksecurity**

- User name: **bkadmin**
- Password: **redhat**

A CLI script is provided at **/tmp/bksecurity.cli** but some fields must be populated.

The resulting content is:

```
data-source add --profile=full-ha --name=bksecurity --driver-name=mysql \
--jndi-name=java:jboss/datasources/bksecurity-ds \
--connection-url=jdbc:mysql://172.25.250.254:3306/bksecurity \
--user-name=bkadmin --password=redhat
```

To create the datasource, run the following command from a workstation terminal window:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \
--file="/tmp/bksecurity.cli"
```

Remember to reuse the command from a previous step.

Test if the datasource is working by accessing the management console using a web browser. Access the management console by navigating <http://172.25.250.254:9990>.

Use the user name **eapadmin** and the password **34p@R3dH4t123**. At the top of the page, click **Configuration** → **Profiles** → **full-ha** → **Subsystems** → **Datasources** → **Non-XA** → **bksecurity**. Click **View** and in the overview page click the Connection and then **Test Connection**.

11. Configure the logging subsystem to store the logs in an NFS share.

Each server will store logging information in an NFS share that is mounted in the **servera** and **serverb** VMs. The NFS mountpoint is configured at **/var/log/jboss** and they are all pointing to the **/var/log/jboss** directory from the **workstation** VM.

Recall that in order to point to a directory different from the base directory, a path must be created and it must point to the NFS share.

To minimize the amount of data generated by logging, a size rotating file handler must be used. It should have a maximum 1 MB size and a maximum of five files should be created. The debug level should be set for this handler and the handler should be named **BOOKSTORE_LOG_HANDLER**. Some extra details about the logging will be provided later in this step.

A category should be created for the **com.redhat.training** package, which will be responsible for capturing logs and storing them using the previously mentioned handler.

11.1. Create a new path called **custom.log.dir** pointing to the NFS mount point **/var/log/jboss**.

In the CLI, run the following command to configure the path:

```
[domain@172.25.250.254:9990 /] /path=custom.log.dir:\nadd(path=/var/log/jboss/)
```

- 11.2. Configure a handler called **BOOKSTORE_LOG_HANDLER** that will use the custom.log.dir path created in the previous step. It should use the following specifications:

- Log file name: **bookstore.log**
- It should be enabled.
- All the logs should appended to bookstore.log, not deleted.
- All the contents stored in the file buffer should be auto-flushed.
- Each log file should have a maximum size of 1 MB.
- The maximum number of files for the log should be five. Any extra files needed should overwrite the oldest file created.
- All logs less critical than DEBUG should be written to the file.

To configure the handler, use the following command:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging/\
size-rotating-file-handler=BOOKSTORE_LOG_HANDLER:add\
(file={"relative-to"=>"custom.log.dir", \
"path"=>"${jboss.server.name}/bookstore.log"}, \
enabled=true, append=true, autoflush=true, rotate-size=1m, \
max-backup-index=5, level=DEBUG)
```

- 11.3. Create a category that will capture only log messages from the bookstore application.

Finally to create the category, run the following command from the CLI:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging/\
logger=com.redhat.training:add\
(category=com.redhat.training, level=DEBUG, \
handlers=[ "BOOKSTORE_LOG_HANDLER" ])
```

Determine if the **/var/log/jboss** directory has directories representing each server and inside each of them, there is a **bookstore.log** file.

12. Customize the messaging subsystem to create queues and customize an **in-vm ConnectionFactory**.

The messaging subsystem must be updated to support a message queue with a dead letter queue if some error raises. In addition to that, an expiry queue should be associated with the same queue if the message is not consumed after a certain amount of time. It should be able to receive messages from an application called **messaging-client.war** that should be deployed for testing purposes. Finally, all the messages will be consumed by a message-driven bean running on an application called **messaging-mdb.jar**.

The following characteristics must be used to create each element described previously and they should be configured in the full-ha profile:

- A new connection factory published at **java:/jms/CustomCF**. It should use an **in-vm** connector.
- A message queue named **TestQueue** that should be bound to the **java:/jms/queue/JB248TestQueue** JNDI name. Take care because the queue address used to bind a queue is different from the queue address passed as a parameter to create a queue.

- A message queue named TestDLQ that should be bound to the `java:/jms/JB248TestDLQ` JNDI name. It will be the dead letter queue that will store all the messages that caused any error (just one) while stored at the TestQueue. Take care because the queue address used to bind a queue with a DLQ is different from the queue address passed as a parameter to create a queue.
- A message queue named TestExpiry that should be bound to the `java:/jms/JB248TestExpiry` JNDI name. It will be the expiry queue that will store all the messages that were stored at the TestQueue for a five second delay. Take care because the queue address used to bind a queue with an expiry queue is different from the queue address passed as a parameter to create a queue.

Because there are some firewall rules as well as some OS-specific configurations that should be updated, a script will be provided to customize them. It was downloaded by the setup process and is available at **/home/student/JB248/labs/review-lab/artemis-firewalld-rules.sh** on each VM. They must be executed as root.

Lastly, check if the messaging servers can identify each other by running the following command:

```
[domain@172.25.250.254:9990 /] /host=serverA/server=serverA.1/\
subsystem=messaging-activemq/server=default/\
cluster-connection=my-cluster:read-attribute(name=topology)
```

12.1. Deploy the message queues.

Run the following commands in the CLI:

```
[domain@172.25.250.254:9990 /] jms-queue add --profile=full-ha \
--queue-address=TestQueue \
--entries=[java:/jms/queue/JB248TestQueue]
[domain@172.25.250.254:9990 /] jms-queue add --profile=full-ha \
--queue-address=TestExpiry \
--entries=[java:/jms/JB248TestExpiry]
[domain@172.25.250.254:9990 /] jms-queue add --profile=full-ha \
--queue-address=TestDLQ \
--entries=[java:/jms/JB248TestDLQ]
```

12.2. Identify the queue address from the TestQueue that was published in a server instance.

By default the queue address is identical among servers running on the same server group and it will be used to configure the dead letter queue and the expiry queue.

```
[domain@172.25.250.254:9990 /] /host=serverA/server=serverA.1/\
subsystem=messaging-activemq\
/server=default/jms-queue=TestQueue:read-attribute(name=queue-address)
```

The resulting output is the queue address used to create an address setting that will be used to bound the queue with a dead letter queue and an expiry queue.

12.3. Identify the queue address for the TestDLQ published in a server instance.

By default the queue address is identical among servers running on the same server group and it will be used to bind the TestDLQ with TestQueue as a dead letter queue.

```
[domain@172.25.250.254:9990 /] /host=serverA/server=serverA.1/\  
subsystem=messaging-activemq\  
/server=default/jms-queue=TestDLQ:read-attribute(name=queue-address)
```

The resulting output is the queue address used to create an address setting that will be used to identify the dead letter queue.

- 12.4. Identify the queue address for the TestExpiry published in a server instance. By default the queue address is identical among servers running on the same server group and it will be used to bind the TestExpiry with TestQueue as an expiry queue.

```
[domain@172.25.250.254:9990 /] /host=serverA/server=serverA.1/\  
subsystem=messaging-activemq\  
/server=default/jms-queue=TestExpiry:read-attribute(name=queue-address)
```

The resulting output is the queue address used to create an address setting that will be used to identify the expiry queue.

- 12.5. To configure the dead letter queue and the expiry queue to capture messages from the TestQueue, you should bind them.

Run the following command from the CLI:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=messaging-activemq/\  
server=default/address-setting=jms.queue.TestQueue:add\  
(expiry-address=jms.queue.TestExpiry,dead-letter-address=jms.queue.TestDLQ,\  
expiry-delay=5,max-delivery-attempts=1)
```

- 12.6. Create a **ConnectionFactory** that will use an **in-vm** connector under the full-ha profile. It should be available for JNDI name lookup at **java:/jms/CustomCF**.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=messaging-activemq/\  
server=default/pooled-connection-factory=custom:add\  
(connectors=[in-vm], entries=[java:/jms/CustomCF])
```

- 12.7. Run the **artemis-firewalld-rules.sh** to drop firewall rules from each VM.

```
[student@workstation ~]$ sudo /tmp/artemis-firewalld-rules.sh  
[student@workstation ~]$ ssh root@servera /tmp/artemis-firewalld-rules.sh  
[student@workstation ~]$ ssh root@serverb /tmp/artemis-firewalld-rules.sh
```

- 12.8. Deploy the **messaging-mdb.jar** and **messaging-client.war** on the server group **Group1** to test if the deployment was successful.

```
[domain@172.25.250.254:9990 /] deploy \  
/tmp/messaging-mdb.jar --server-groups=Group1  
[domain@172.25.250.254:9990 /] deploy \  
/tmp/messaging-client.war --server-groups=Group1
```

Use a web browser to open the client at <http://172.25.250.10:8080/> messaging-client

Check the output from the terminal window running the host controller **serverA** to determine if the message was consumed.

13. Enforce security concerns by encrypting passwords, disabling local automatic authentication, and creating a security domain.

Any attempt to connect to the domain controller or to the standalone server using a CLI will not request a password if the session is opened locally to the workstation VM. However, this is not a secure behavior and it should be changed to avoid malicious access. In this step, the user access in either the domain controller and the standalone server should require a user name (**eapadmin**) and a password.(**34p@R3dH4t123**).

- 13.1. Recall that the standalone server management interface is available via loopback interface only and the administration port is 10990 to avoid port collision with the domain controller. Remove from the management realm (which holds the credentials for management purposes) the local authentication. The management realm is under the core-services level, under the security realm service.

To update the standalone server behavior, connect to the standalone server using the following command line from the workstation VM:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh \
--connect \
--controller=localhost:10990
```

To enable the standalone server local authentication, run the following command:

```
[standalone@localhost:10990 /] /core-service=management \
/security-realm=ManagementRealm/authentication=local:remove()
```

Reload the server and connect again to the standalone server using the CLI. This time credentials will be requested.

- 13.2. Use the opened CLI session to remove the local authentication from the domain controller. The management realm (which holds the credentials for management purposes) is responsible for the local authentication. The management realm is under the core-services level, under the security realm service.

To enable the standalone server local authentication, run the following command:

```
[domain@172.25.250.254:9990 /]/host=master/core-service=management \
/security-realm=ManagementRealm/authentication=local:remove()
```

Reload the host named master and connect the CLI again to the domain controller. This time credentials will be requested.

- 13.3. Create a vault where all the passwords will be stored in an encrypted way. The vault will be represented by a keystore generated by the keytool utility from the Java Development Kit (JDK). Each new password added to the keystore is managed by the vault.sh tool provided by EAP 7 and it is available at the **JBOSS_HOME/bin** directory. The file will be saved in the **/opt/domain** directory and it should be saved as **key.store**. To create the keystore a script is provided; run the following command:

```
[student@workstation ~]$ sudo /tmp/keystore-vault.sh
```

- 13.4. Create one parameter that should hold the password from the bookstore data source. The following parameters should be used to run the **vault.sh** script and they must be executed as the **eap** user.
- Keystore file name: **/opt/domain/key.store**
 - Keystore password: **34p@R3dH4t123**
 - alias: **Vault**
 - vault block **bookstore-vb**
 - Attribute: **bookstore-pw**
 - Value: **redhat**
 - Encrypt directory: **/opt/domain/vault**
 - iteration: **95**
 - salt: **ABCD1234**

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/vault.sh \
--keystore /opt/domain/key.store \
--keystore-password 34p@R3dH4t123 --alias Vault \
--vault-block bookstore-vb --attribute bookstore-pw \
--sec-attr redhat --enc-dir /opt/domain/vault --iteration 95 --salt ABCD1234
```

Copy the resulting XML from the execution to a file called **/tmp/vault.xml** that will be used to update the **host-slave.xml** files on each host controller. Additionally, take note of the content right below the **Configuration should be done as follows:**. This string will be used to refer to the password from the bookstore app.

- 13.5. Create one parameter to hold the password from the **bksecurity-ds** datasource. The following parameters should be used to run the **vault.sh** script.
- Keystore file name: **/opt/domain/key.store**
 - Keystore password: **34p@R3dH4t123**
 - alias: **Vault**
 - vault block **realm-vb**
 - Attribute: **realm-pw**
 - Value: **redhat**
 - Encrypt directory: **/opt/domain/vault**
 - iteration: **95**
 - salt: **ABCD1234**

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/vault.sh \
--keystore /opt/domain/key.store \
--keystore-password 34p@R3dH4t123 --alias Vault --vault-block realm-vb \
--attribute realm-pw \
--sec-attr redhat --enc-dir /opt/domain/vault --iteration 95 --salt ABCD1234
```

Notice that most of the parameters from the previous command are the same, except for the vault block, attribute, and value. Take note of the content right below the **Configuration should be done as follows:**. This string will be used to refer to the password from the bksecurity app.

- 13.6. Transfer the **key.store**, the **vault.xml**, and the **vault** file to other servers by using **scp** to copy these files it to the other servers. Because the **eap** user is a **nologin** user, transfer them as **root**.

```
[student@workstation ~]$ scp /opt/domain/key.store root@servera:/opt/domain/
[student@workstation ~]$ scp /tmp/vault.xml root@servera:/tmp/
[student@workstation ~]$ scp -r /opt/domain/vault root@servera:/opt/domain/
```

```
[student@workstation ~]$ scp /opt/domain/key.store root@serverb:/opt/domain/
[student@workstation ~]$ scp /tmp/vault.xml root@serverb:/tmp/
[student@workstation ~]$ scp -r /opt/domain/vault root@serverb:/opt/domain/
```

Change the owner of these files to **eap**.

```
[student@servera ~]$ sudo chown eap:eap /opt/domain/*
[student@serverb ~]$ sudo chown eap:eap /opt/domain/*
```



Note

Even though an NFS share directory is available, it will not be used. Each time a new password is added, the copy should be executed to avoid NFS share unavailability issues.

- 13.7. Stop all host controllers and the domain controller to update the **host-slave.xml** configuration file to include the **vault.xml** file created in step 13.4 between the **<extensions>** and **<management>** entries. This configuration will allow EAP to open the keystore file and access the passwords. Press **Ctrl+C** on all the windows running either a host controller or a domain controller. Also hit **Ctrl+C** on the CLI window too. Using your preferred text editor, open the **vault.xml** file and add the XML configuration to the **host-slave.xml** configuration file as described by the **vault.sh** output.

- 13.8. Update the password from the datasources created in the full-ha profile by substituting the plain text password to the vault reference mentioned in the **vault.sh** output. It should substitute to something like: **\${VAULT: :XXXXX: :YYYYY}**.

From the CLI window accessing the domain controller:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=datasources/\
data-source=bookstore:\
write-attribute(name=password,value=${VAULT::XXXXX::YYYYY})
```

- 13.9. Restart the host controllers and the domain controller. To prepare the servers to run using a cluster, the **-Djboss.bind.address.private** must be passed with the IP addresses from **servera** and **serverb** VMs.

Run the following command on the **workstation** VM:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain \
--host-config=host-master.xml
```

Run the following command on the **servera** VM:

```
[student@servera ~]$ sudo -u eap /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain \
-Djboss.bind.address.private=172.25.250.10 \
--host-config=host-slave.xml
```

Run the following command on the **serverb** VM:

```
[student@serverb ~]$ sudo -u eap /opt/eap-7.0.0/bin/domain.sh \
-Djboss.domain.base.dir=/opt/domain \
-Djboss.bind.address.private=172.25.250.11 \
--host-config=host-slave.xml
```

- 13.10. With a web browser, open the management console from the workstation VM. The URL for the management console is **172.25.250.254:9990**. The credentials to access it are **eapadmin/34p@R3dH4t123**. Similarly to step 11.4, test the connection. Navigate to the data source subsystem from the full-ha profile and locate the bksecurity and bookstore data sources and look for the **Test Connection** button to check if the connection was successful.

- 13.11. Create a security realm that uses the **bksecurity-ds** data source. It is used by a custom application that will require authentication. The password is encrypted in the users table from the bksecurity and it should use the **SHA-256** hash algorithm and a **base64** hash encoding. In the same table there is a column named user name which contains the user name. The roles can be obtained from the roles table. This table has a user name column too that should be used to identify the roles of a user name. A file named **/tmp/bksecurity-domain.cli** is provided to avoid type errors.

It should have the following parameters and it must be deployed on the full-ha profile:

- Security domain name: **bksecurity**
- Login module configuration flag: **required**
- Login module configuration code: **Database**
- Login module configuration options:
- **dsJndiName: java:jboss/datasources/bksecurity-ds**

- *principalsQuery*: **SELECT password FROM users WHERE username = ?**
- *rolesQuery*: **SELECT role, "ROLES" FROM roles WHERE username = ?**
- *hashAlgorithm*: **sha-256**
- *hashEncoding*: **base64**

The following output is expected:

```
/profile=full-ha/subsystem=security/security-domain=bksecurity:add()  
/profile=full-ha/subsystem=security/security-domain=bksecurity/  
authentication=classic:add(login-modules=[{"code"=>"Database", "flag"=>"required",  
"module-options"=>[("dsJndiName"=>"java:jboss/datasources/bksecurity-  
ds"), ("principalsQuery"=>"SELECT password FROM users WHERE username = ?"),  
("rolesQuery"=>"SELECT role, 'ROLES' FROM roles WHERE username = ?"),  
("hashAlgorithm"=>"SHA-256"), ("hashEncoding"=>"base64")]])
```

To start the CLI, use the following command from a terminal window in the workstation VM.

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \  
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \  
--file="/tmp/tcp-stack.cli"
```

Check if the output has a success string. Otherwise, double-check the command.

Reload the servers to allow the changes to take place:

```
[domain@172.25.250.254:9990 /] :reload-servers
```

13.12. Deploy the application called **/tmp/example.war** that was downloaded during the setup process. Use the admin / admin credentials to access the application page.

14. Update the amount of memory allocated for all the server groups.

Change the amount of memory allocated to the heap for each server to 256 MB minimum and 512 MB maximum. To change memory settings, update the memory configuration for each server group, instead of changing individually. Additionally, create a default JVM configuration for the server group.

```
[domain@172.25.250.254:9990 /] /server-group=Group1\  
jvm=default:add(heap-size=256m,max-heap-size=512m)  
[domain@172.25.250.254:9990 /] /server-group=Group2\  
jvm=default:add(heap-size=256m,max-heap-size=512m)
```

Reload the servers to activate the changes.

```
[domain@172.25.250.254:9990 /] reload --restart-servers=true --host=serverA  
[domain@172.25.250.254:9990 /] reload --restart-servers=true --host=serverB
```

15. Customize the root application on the servers.

By default, a welcome page is provided however a custom web application should be used instead. To minimize the risk of environment differences, the default web module must be

updated. The file is available at the **/tmp/welcome.war** and it should be initially deployed to all server groups and it must update the default web module.

From the CLI, run the following command to deploy the application:

```
[domain@172.25.250.254:9990 /] deploy \
/tmp/welcome.war --all-server-groups
```

Remove the default web module.

Run the following command in the CLI:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=undertow\
/server=default-server/host=default-host/location=/:remove
```

Update the welcome web module to access the recently updated version of the **welcome.war** using the CLI.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=undertow/\
server=default-server/host=default-host:write-attribute\
(name=default-web-module,value=welcome.war)
```

Navigate to the following addresses to check if the welcome page was updated:

- <http://172.25.250.10:8080>
- <http://172.25.250.10:8230>
- <http://172.25.250.11:8080>
- <http://172.25.250.11:8230>

A welcome page with a Red Hat logo should be presented.

16. Enable encrypted communication with the standalone server.

All the requests in this architecture will be sent to the standalone server. To guarantee those requests with sensitive information are not being transmitted in plain text, a certificate must be installed. For this lab, a self-signed certificate will be created, but in a real production environment a signed certificate from a trusted source should be created. To create a self-

signed certificate, run the `/tmp/self-signed.sh` script from a terminal window on the workstation VM.

```
[student@workstation ~]$ sudo /tmp/self-signed.sh
```

Configure the realm that will refer to the keystore where the certificate is stored. EAP 7 requires that the keystore should be referred by a realm. Add a realm named **HTTPSRealm** to the security-realm from the standalone server.

In the CLI opened to access the standalone server, run the following command:

```
[standalone@localhost:10990 /] /core-service=management/\nsecurity-realm=HTTPSRealm:add()
```

Link the **HTTPSRealm** to the keystore file with the certificate. It should provide the keystore password (**34p7-k3yst0r3**) and the alias where the certificate is bound to (**appserver**) by using a CLI window. Also recall to reload the standalone server.

```
[standalone@localhost:10990 /] /core-service=management/security-realm=HTTPSRealm/\nserver-identity=ssl:add(keystore-path=/opt/standalone/identity.jks,\nkeystore-password=34p7-k3yst0r3, alias=appserver)
```

To make the change persistent, reload the standalone server.

```
[standalone@localhost:10990 /] reload
```

Add a new HTTPS listener using the configured **HTTPSRealm** realm.

```
[standalone@localhost:10990 /] /subsystem=undertow/server=\ndefault-server/https-listener=https:\nadd(socket-binding=https, security-realm=HTTPSRealm)
```

Test if the server can be accessed using HTTPS. Open a browser and access the standalone server that has a port offset of 1000 to avoid port conflicts with the domain controller. An alert from the browser is expected and it can be safely ignored because we are using a self-signed certificate, not a certificate signed by a trusted source.

Open a browser and open the `https://172.25.250.254:9443` address and check if the connection is secured.

17. Configure the cluster to use the TCP stack instead of the default UDP stack.

EAP is configured by default using a UDP stack to create a cluster. Because UDP supports multicast, new servers can be easily identified and added to a cluster. Unfortunately, the network does not support UDP and EAP should use a TCP stack, identifying which servers should be connected by a unicast connection.

To achieve this goal, the JGroups subsystem must be customized to use a protocol called TCPPING that mimics an MPING behavior looking for all servers running on a network.

Because TCP cannot multicast messages, the TCPPING must be configured to look for each server that is part of the cluster:

```
172.25.250.10 port 7600
172.25.250.10 port 7750
172.25.250.11 port 7600
172.25.250.11 port 7750
```

Also the stack attribute must be updated to support the TCPPING stack. To minimize the risks from updating the stack a new stack should be created called tcpping and it should be the default stack.

A prepopulated configuration called **tcp-stack.cli** is available in **/tmp** directory. Update it to use the **TCPPING** protocol and customize the host list to include all the servers running on the managed domain. Additionally, change the stack used by JGroups to use the tcpping stack instead of the UDP stack.

```
batch
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":add()
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":add-
protocol(type="TCPPING")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping"/\
transport="TRANSPORT":add(socket-binding="jgroups-tcp",type="TCP")
run-batch
batch
/profile="full-ha"/subsystem="jgroups"/stack="tcpping"/protocol="TCPPING"/\
property="initial_hosts":add\
(value="172.25.250.10[7600],172.25.250.10[7750],172.25.250.11[7600],172.25.250.11[7750]")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping"/protocol="TCPPING"/\
property="port_range":add(value="10")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="MERGE2")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(socket-binding="jgroups-tcp-fd",type="FD_SOCK")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="FD")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="VERIFY_SUSPECT")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="BARRIER")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="pbcast.NAKACK")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="UNICAST2")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="pbcast.STABLE")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="pbcast.GMS")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="UFC")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="MFC")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
add-protocol(type="FRAG2")
/profile="full-ha"/subsystem="jgroups"/stack="tcpping":\
```

```
add-protocol(type="RSVP")
/profile=full-ha/subsystem=jgroups/channel=ee:\n
write-attribute(name=stack,value=tcping)
run-batch
```

Run the CLI script.

Run the following command from the **workstation**:

```
[student@workstation ~]$ sudo -u eap /opt/eap-7.0.0/bin/jboss-cli.sh --connect \
--user=eapadmin --password=34p@R3dH4t123 --controller=172.25.250.254:9990 \
--file="/tmp/tcp-stack.cli"
```

To allow communication among the servers, the firewall rules should be customized on each server VM. Recall that the **serverX**.**2** have a port offset of 150. A script is provided to drop these firewall rules. Run the **/tmp/firewall-cluster.sh** on **servera** and **serverb** VMs.

```
[student@servera ~]$ sudo /tmp/firewall-cluster.sh
```

```
[student@serverb ~]$ sudo /tmp/firewall-cluster.sh
```

Reload the domain controller and all host controllers to make the TCPPING stack valid. From the CLI on the domain controller, run:

```
[domain@172.25.250.254:9990 /] /host=serverA:reload
[domain@172.25.250.254:9990 /] /host=serverB:reload
[domain@172.25.250.254:9990 /] /host=master:reload
```

18. Create a static load balancer using the standalone server.

From the original architecture, the standalone server will load balance all the requests from an external environment. To make this work, the undertow subsystem must be customized

to support a reverse proxy that will be responsible for load balancing the request among multiple servers. Create a reverse proxy called bookstore-handler in the standalone server.

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=bookstore-handler:add()
```

Configure a socket that will connect to **serverA.1** (172.25.250.10) using the AJP protocol running on TCP port 8009. Name it **remote-serverA.1**.

```
[standalone@localhost:10990 /] /socket-binding-group=standard-sockets\
/remote-destination-outbound-socket-binding=remote-serverA.1/:\
add(host=172.25.250.10, port=8009)
```

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverA.1**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverA.1**:

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=bookstore-handler/host=serverA.1:\
add(outbound-socket-binding=remote-serverA.1, \
scheme=ajp, instance-id=myroute, path=/bookstore)
```

Configure a socket that will connect to the **serverA.2** (172.25.250.10) using the AJP protocol running on TCP port 8159. Name it **remote-serverA.2**.

```
[standalone@localhost:10990 /] /socket-binding-group=standard-sockets\
/remote-destination-outbound-socket-binding=remote-serverA.2/:\
add(host=172.25.250.10, port=8159)
```

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverA.2**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverA.2**:

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=bookstore-handler/host=serverA.2:add\
(outbound-socket-binding=remote-serverA.2, \
scheme=ajp, instance-id=myroute, path=/bookstore)
```

Configure a socket that will connect to serverB.1 (172.25.250.11) using the AJP protocol running on TCP port 8009. Name it **remote-serverB.1**.

```
[standalone@localhost:10990 /] /socket-binding-group=standard-sockets\
/remote-destination-outbound-socket-binding=remote-serverB.1/:\
add(host=172.25.250.11, port=8009)
```

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverB.1**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverB.1**:

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=bookstore-handler/host=serverB.1:\\
add(outbound-socket-binding=remote-serverB.1, \
scheme=ajp, instance-id=myroute, path=/bookstore)
```

Configure a socket that will connect to serverB.2 (172.25.250.11) using the AJP protocol running on TCP port 8159. Name it **remote-serverB.2**.

```
[standalone@localhost:10990 /] /socket-binding-group=standard-sockets\
/remote-destination-outbound-socket-binding=remote-serverB.2:\\
add(host=172.25.250.11, port=8159)
```

Create in Undertow:

- Use a reference to the socket binding created previously for the **serverB.2**
- Bind it to the AJP scheme
- Configure Undertow to access the bookstore application path from **serverB.2**:

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=handler\
/reverse-proxy=bookstore-handler/host=serverB.2:\\
add(outbound-socket-binding=remote-serverB.2, \
scheme=ajp, instance-id=myroute, path=/bookstore)
```

Deploy the **bookstore.war** application on all group servers and test it.

```
[domain@172.25.250.254:9990 /] deploy \
/tmp/bookstore.war --all-server-groups
```

19. Run the grading script to check if all the tasks were accomplished by opening a new terminal window from the workstation VM and running the following command:

```
[student@workstation ~]$ lab review-final grade
```

Summary

In this chapter, you had a chance to practice all the skills learned in the course.