JBOSS EAP 7: Administration

Binnur Kurt, PhD

DEEPCLOUDLABS

binnur.kurt@deepcloudlabs.com

Course Goals

After completing this course, you should be able to:

- Install and run **JBoss** EAP 7 on Linux/Windows
- Configure the application server's main configuration file and some core parts of it, e.g. Thread pool configuration and the Logging subsystem.
- Configure a **JBoss** EAP 7 Domain
- Deploy applications on **JBoss** EAP 7
- Create a cluster
- Load balance web applications
- Configuring JVM
- Monitoring and Performance Tuning

Jakarta Course Content

1. Introduction to Java EE
2. Introduction to JBoss EAP *7*
3. Installing JBoss EAP *7*
4. Configuring the Application Server
5. Configuring Enterprise Services
6. Web Server Configuration
7. Configuring a JBoss EAP *7* Domain
8. Deploying Applications on JBoss EAP *7*
9. Managing the Application Server
10. Clustering
11. Load Balancing Web Applications
12. Monitoring and Tuning JBoss EAP *7*

Course Information

- Duration
  - 4 Days
- Course Hours
  - 9:30-12:30
  - 13:30-16:30

Background

- **1995, B.Sc., ITU Computer Engineering**
- **1997, M.Sc., ITU Computer Engineering**
- **2007, Ph.D., ITU Computer Engineering**
- **1995-2003, RA, *ITU CE***
- **2004-2008, Lecturer, ITU CE**
- **2008-2017, Consultant & Trainer,**
  Omega Training & Consultancy
- **2018-to date, Founder & CEO, DEEPCLOUDLABS**
  https://www.deepcloudlabs.com

Areas of Interest

- Java Desktop and Enterprise Technologies
- Spring Framework: Spring Data, Spring MVC, Spring Boot
- MySQL, MySQL Cluster
- Programming Paradigms: OOP, FP, GP, RP
- Oracle Fusion Middleware Technologies: SOA, BPM, OSB
- Big Data and Machine Learning
- Algorithmic Trading & High-frequency Trading
- Crypto-Currency & Block-chain
- Digital Image & Video Analysis and Processing
- Real-Time Computer Vision Systems

Trainings Offered

Big Data Machine Learning



Programming



Database Messaging



Application Servers



Frameworks



My Certifications







Contact

> **binnur.kurt**@gmail.com

> https://**binkurt.**blogspot.com

> https://www.**deepcloudlabs**.com

> https://www.**deepcloudlabs**.com/**akbank**

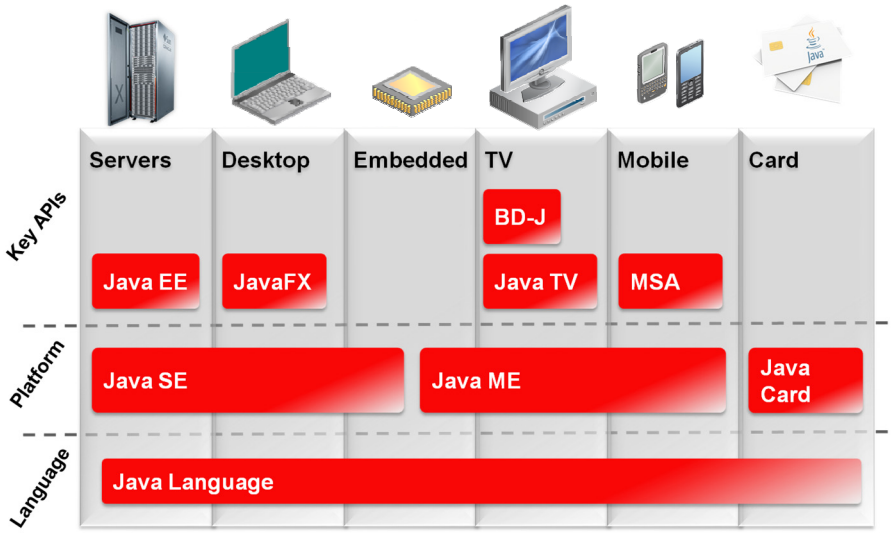
MODULE 1

INTRODUCTION TO JAVA EE 8

MODULE 1

INTRODUCTION TO JAKARTA EE 8

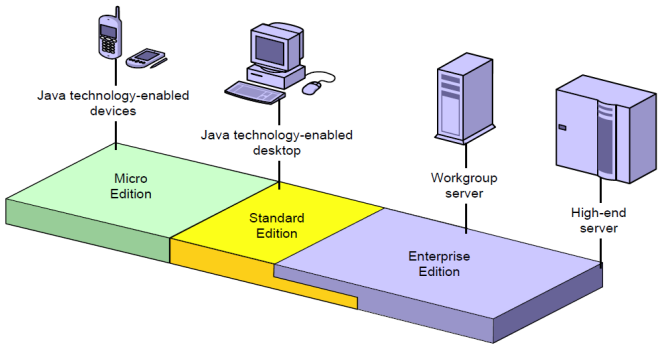
Identifying Java Technology Product Groups



Java™ Technology Platforms

> The Java EE platform is an architecture for implementing enterprise-class applications using Java and Internet technology.

> A primary goal of Java EE technology is to simplify the development of enterprise-class applications through a vendor-neutral, component-based application model.

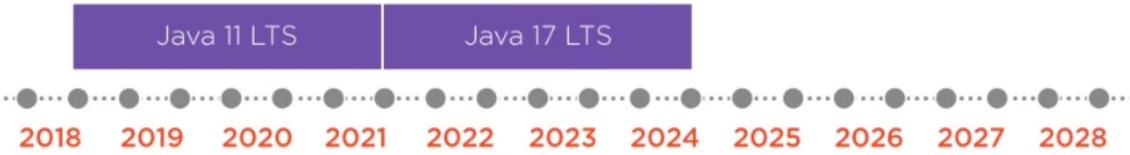


Java SE Platform Versions

Year	Developer Version (JDK)	Platform
1996	1.0	1
1997	1.1	1
1998	**1.2**	**2**
2000	1.3	2
2002	1.4	2
2004	**1.5**	**5**
2006	1.6	6
2011	1.7	7
2014	**1.8**	**8**
2017	**1.9**	**9**
2018	1.10	10

Java SE Platform Versions

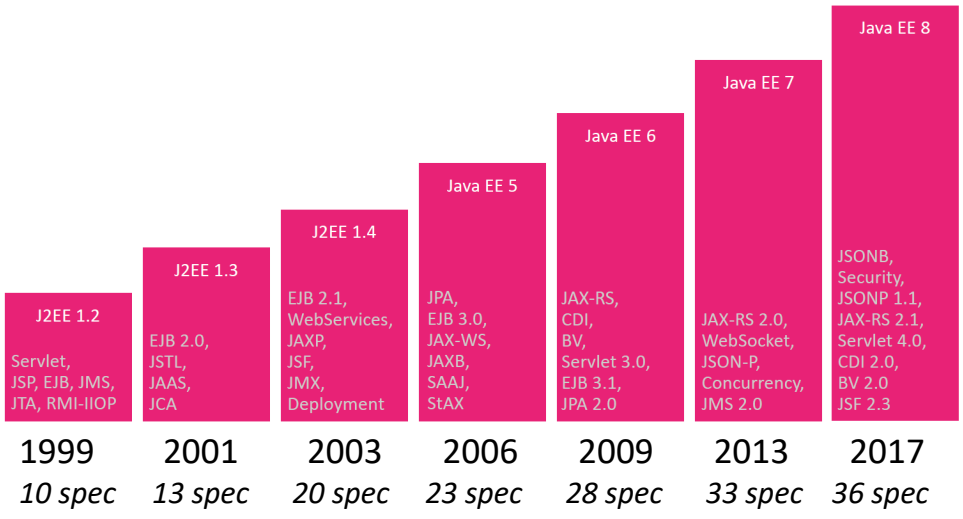
Year	Developer Version (JDK)	Platform
2018	1.11	11
2019	1.12	12
2019	1.13	13
2020	1.14	14
2020	1.15	15
2021	1.16	16
2021	1.17	17



Java EE Past, Present, & Future

> Java EE is based on standards.

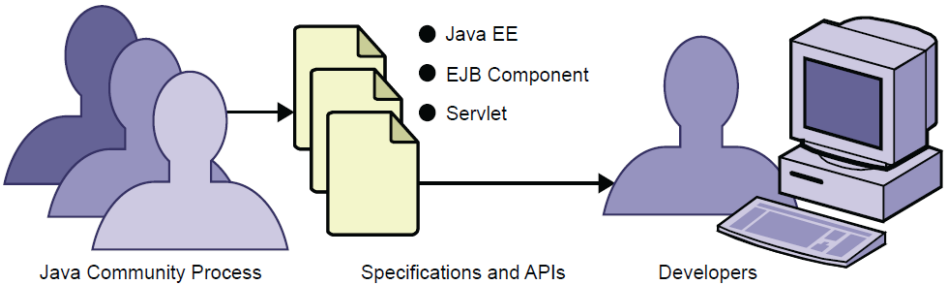
> It is an umbrella specification that bundles together a number of other JSRs.



Java EE and the Java Community Process

> The Java EE specification defines the types of components and the associated APIs that are available to Java EE application component developers.

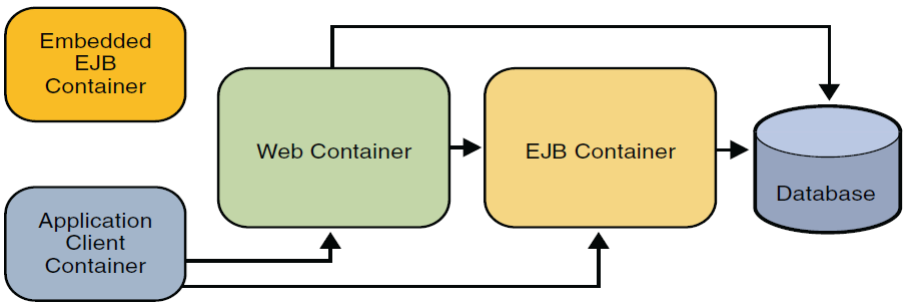
> The Java EE specification also defines the infrastructure requirements for a robust, scalable, and reliable runtime environment for distributed, enterprise applications.



Architecture

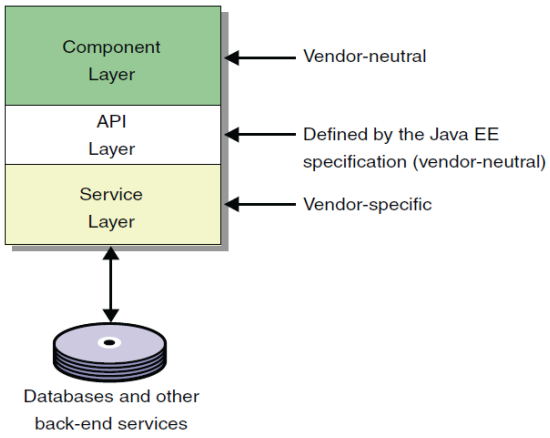
> Java EE is a set of specifications implemented by different containers.

> Containers are Java EE runtime environments that provide certain services to the components they host such as life-cycle management, dependency injection, and so on.

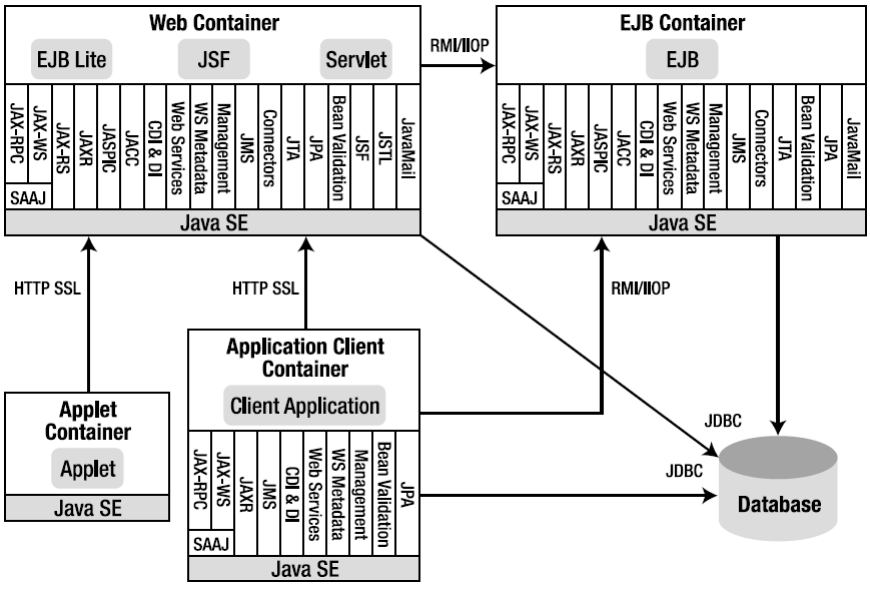


Component, API, and Service Layer

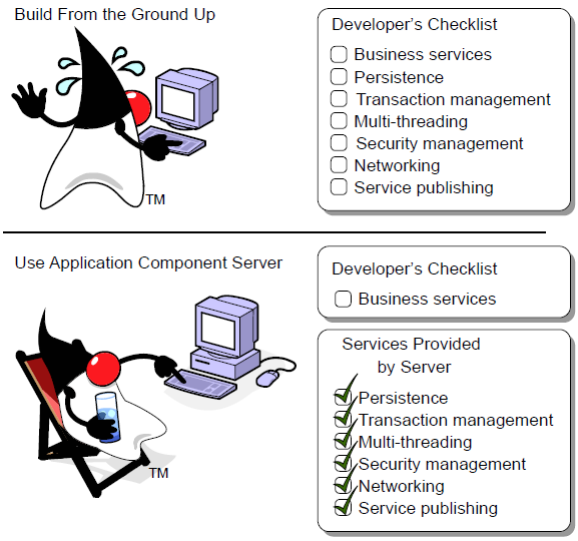
> The Java EE APIs define the contract between the application component developer and the platform provider using the interface mechanism that is defined by the Java programming language.



Java EE Service Infrastructure



Advantages of using Server-Provided Services



Java EE Platform APIs and Services

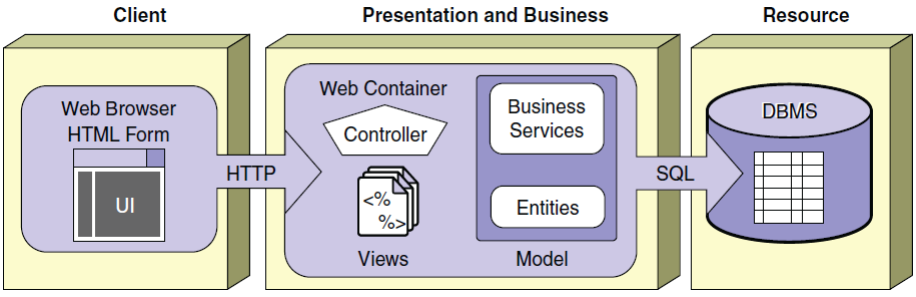
> Java EE service categories:
  - Deployment-based services
  - API-based services
  - Inherent services
  - Vendor-specific functionality

Java EE Application Architecture

- Web-centric architecture
- Combined web and EJB™ component-based architecture, sometimes called EJB component-centric architecture
- Business-to-business (B2B) application architecture
- Web service application architecture

Java EE Web-Centric Architecture

- The introduction of EJB Lite in Java EE 6 allows the use of some EJB technology in web-centric architectures.

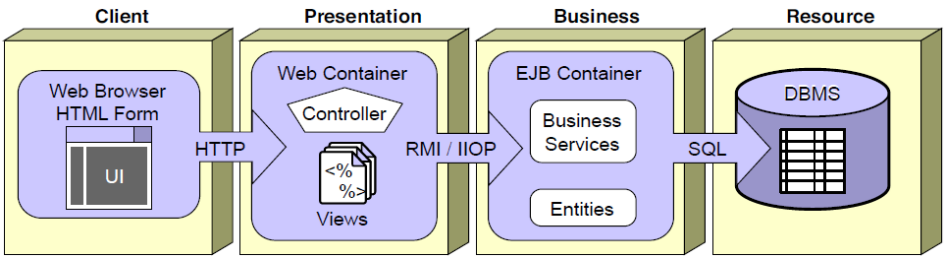


Java EE EJB Component-Centric Architecture

> This configuration uses both the web container and EJB containers of the Java EE server.

> In this model, the business logic and data access components are located in the EJB container.

> The web-tier components process the incoming request and generate the view based on the results of the business process.



Java EE Profiles

> Java EE 6 standardized profiles for application developers that do not need the full Java EE platform but require application portability.

> Developers can choose between:

  – The Java EE 6 **Web Profile** - Similar to Tomcat, for when developers are working primarily on web applications.

  – The Java EE 6 **Full Platform** - Complete Java EE 6 application servers

> Technically “Full Platform” is **not** a profile. The only profile included in the Java EE 6 specification is the web profile.

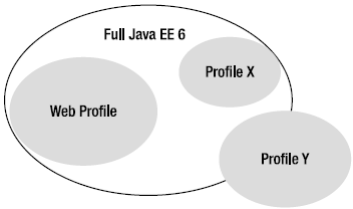
> Additional profiles may be added in future releases.

Java EE Profiles

- Profiles are a major new feature in the Java EE 6 environment.
- Their main goal is to reduce the size of the platform to suit the developer's needs more efficiently.
- No matter the size or complexity of the application you develop today, you will deploy it in an application server that offers you APIs and services for 33 specifications.
- A major criticism leveled against Java EE was that it was too large.
- Profiles are designed precisely to address this issue.

Java EE Profiles

- Java EE 6 defines a single profile called the Web Profile. Its aim is to allow developers to create web applications with the appropriate set of technologies.
- Web Profile 1.0 is specified in a separate JSR and is the first profile of the Java EE 6 platform.
- Others might be created in the future (you could think of a minimal profile or a portal profile).
- The Web Profile will evolve at its own pace, and we might end up with a Web Profile 1.1 or 1.2 before Java EE 7 is released.

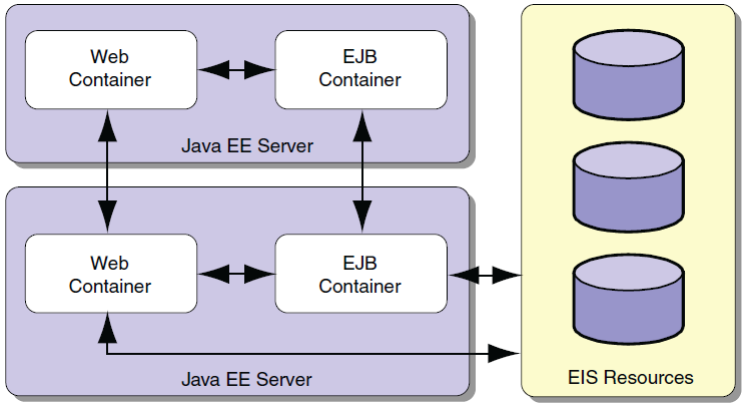


Web Profile 1.0 Specifications

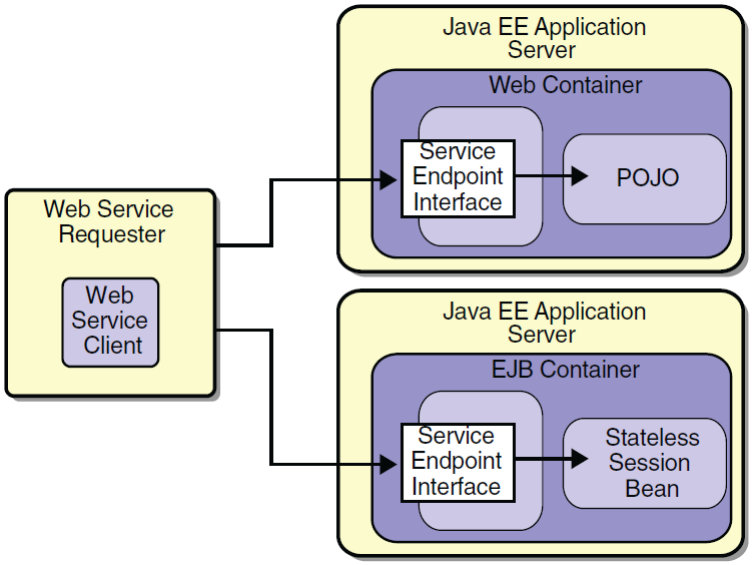
Specification	Version	JSR	URL
JSF	2.0	314	http://jcp.org/en/jsr/detail?id=314
JSP	2.2	245	http://jcp.org/en/jsr/detail?id=245
JSTL	1.2	52	http://jcp.org/en/jsr/detail?id=52
Servlet	3.0	315	http://jcp.org/en/jsr/detail?id=315
Expression Language	2.2	245	http://jcp.org/en/jsr/detail?id=245
EJB Lite	3.1	318	http://jcp.org/en/jsr/detail?id=318
JPA	2.0	317	http://jcp.org/en/jsr/detail?id=317
JTA	1.1	907	http://jcp.org/en/jsr/detail?id=907
Bean Validation	1.0	303	http://jcp.org/en/jsr/detail?id=303
Managed Beans	1.0	316	http://jcp.org/en/jsr/detail?id=316
Interceptors	1.1	318	http://jcp.org/en/jsr/detail?id=318
Contexts and Dependency Injection	1.0	299	http://jcp.org/en/jsr/detail?id=299
Dependency Injection for Java	1.0	330	http://jcp.org/en/jsr/detail?id=330
Debugging Support for Other Languages	1.0	45	http://jcp.org/en/jsr/detail?id=45
Common Annotations	1.1	250	http://jcp.org/en/jsr/detail?id=250

B2B Application Architecture

- The B2B application architecture is an extension of the Java EE technology EJB component-based architecture.
- It involves two EJB servers, one in each business location.
- Each Java EE server hosts a web container and an EJB container.

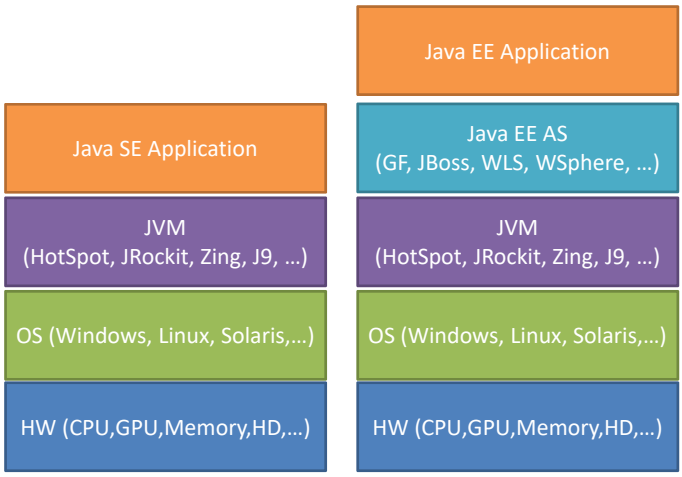


Web Service Application Architecture

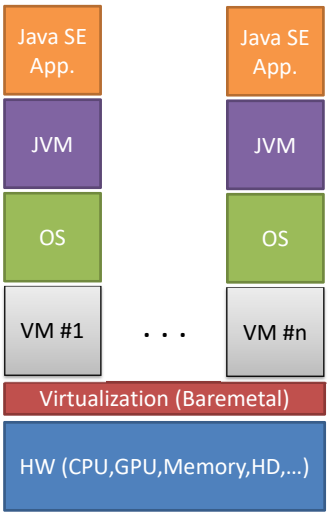


PLATFORM ARCHITECTURE

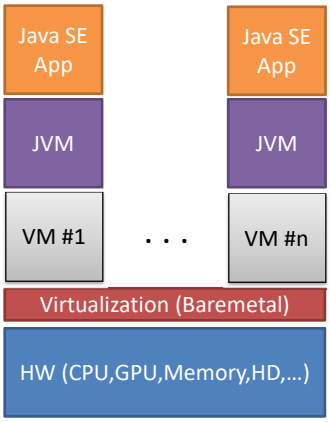
The Platform Architecture 1/4



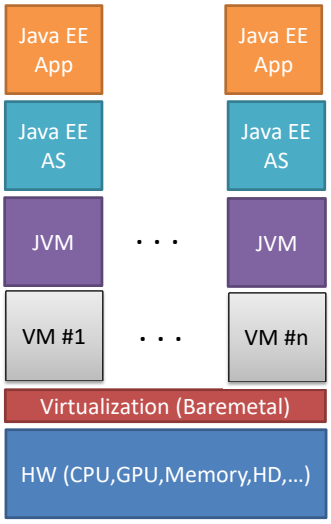
The Platform Architecture 2/4



JRockit VE – Virtual Edition
WebSphere HE – Hypervisor Edition

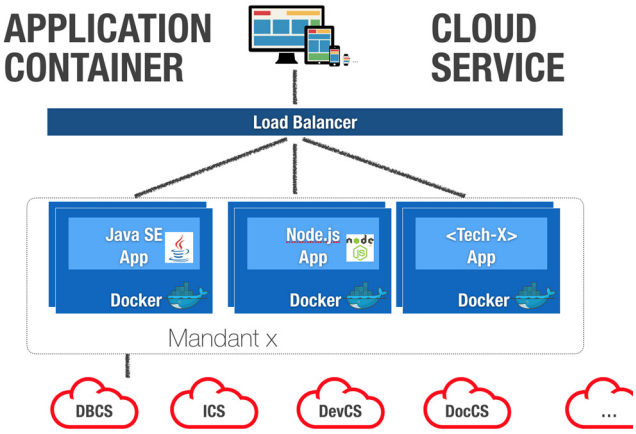


The Platform Architecture 3/4



The Platform Architecture 4/4





JVMs

Different JVMs

- Oracle
  - HotSpot
- Azul Systems
  - Zing
  - Zulu
- IBM
  - J9
  - OpenJ9 (Eclipse)

Different JVMs

- OpenJDK
  - The source repository
  - Distribution
- AdoptOpenJDK
  - Prebuilt OpenJDK binaries maintained by community

OpenJDK Providers and Comparison

Provider	Free Builds From Source	Free Binary Distribution	Extended Updates (LTS)	Commercial Support	Permissive License
AdoptOpenJDK	Yes	Yes	Yes	No	Yes
Amazon Corretto	Yes	Yes	Yes	No	Yes
Azul Zulu	No	Yes	Yes	Yes	Yes
IBM	No	No	Yes	Yes	Yes
OpenJDK	Yes	Yes	Yes	No	Yes
Oracle JDK	No	Yes	No	Yes	No
Oracle OpenJDK	Yes	Yes	No	No	Yes
Red Hat	Yes	Yes	Yes	Yes	Yes
SAPMachine	Yes	Yes	Yes	Yes	Yes

GraalVM





GraalVM





GraalVM





OpenJDK

GraalVM



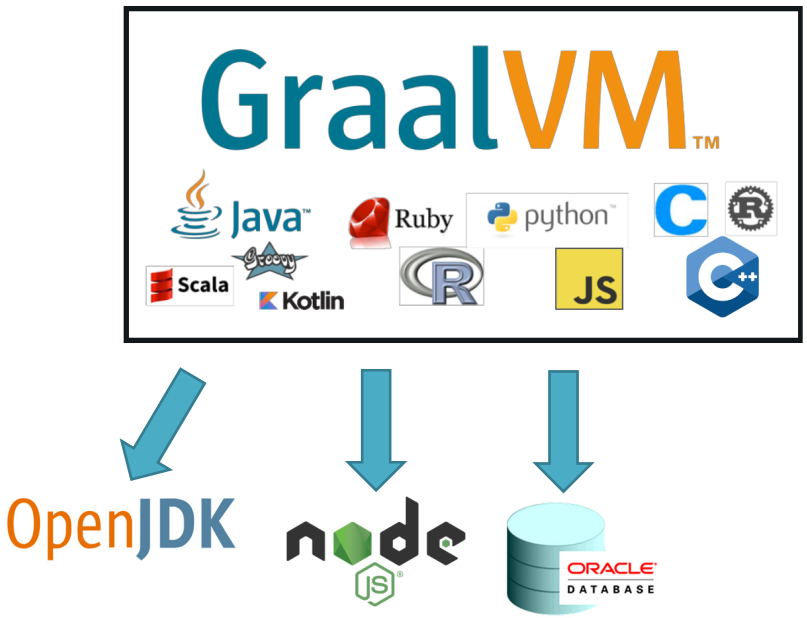


OpenJDK

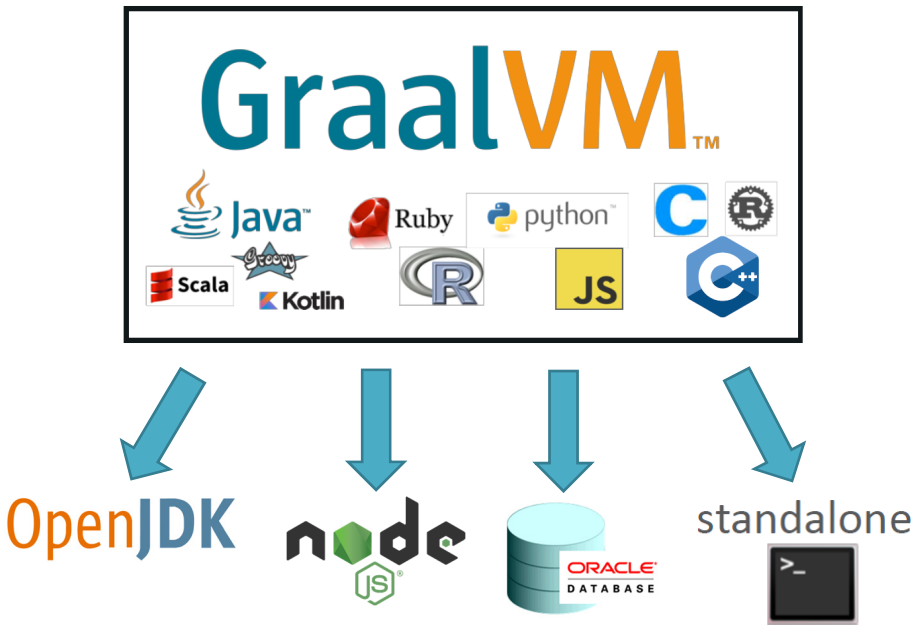


node

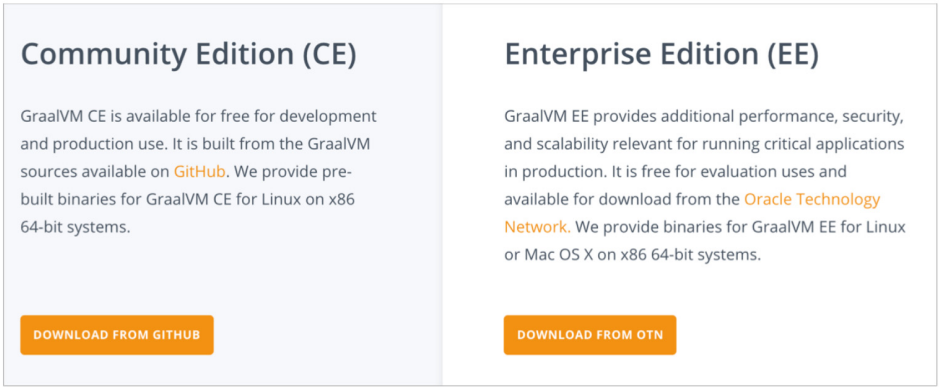
GraalVM



GraalVM



GraalVM



GraalVM



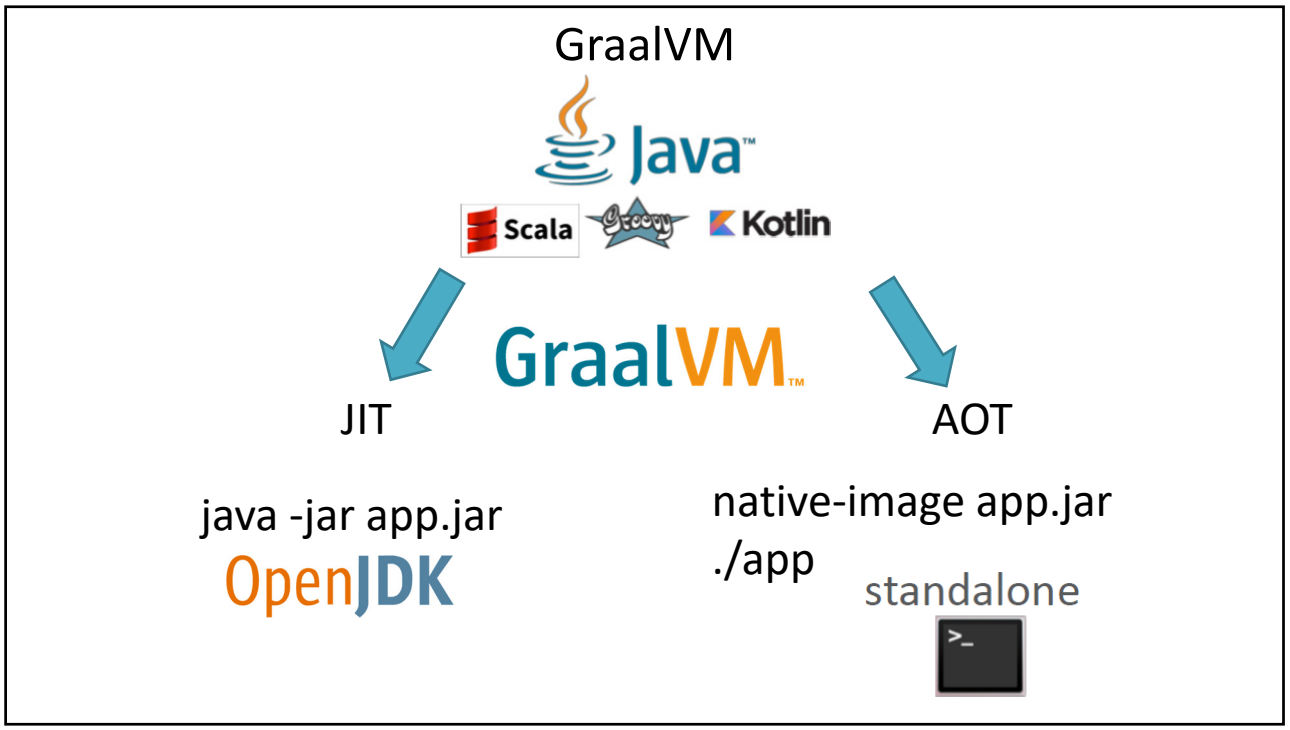


GraalVM

JIT

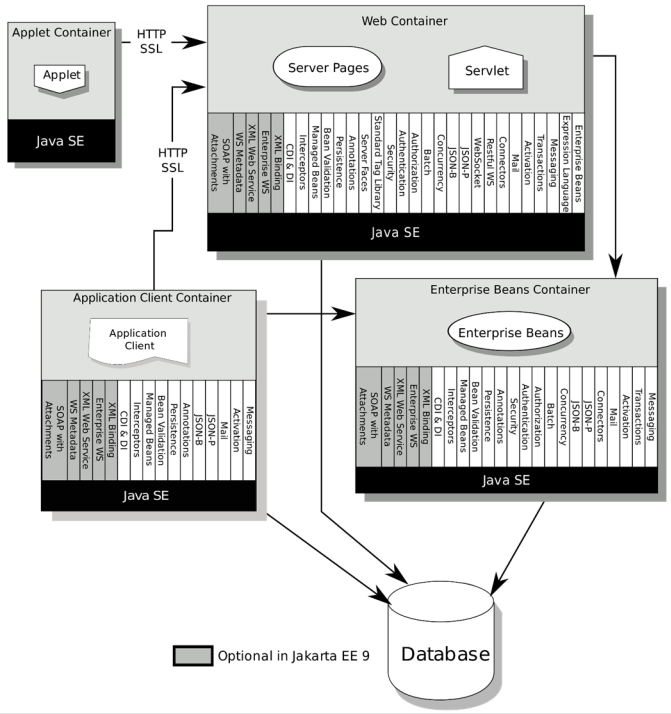
java -jar app.jar

OpenJDK

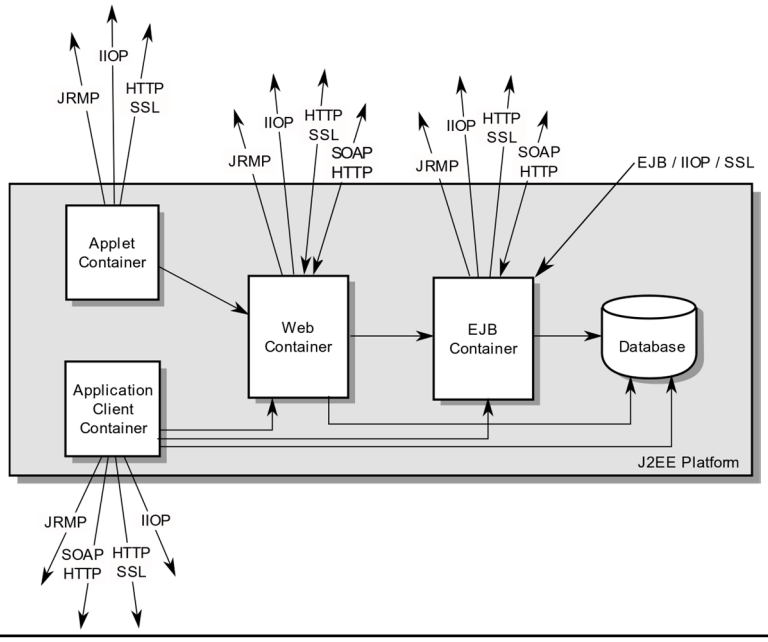


JAVA EE 8: NEW FEATURES

Jakarta EE Service Infrastructure



Jakarta EE Interoperability



Java EE Application Architecture

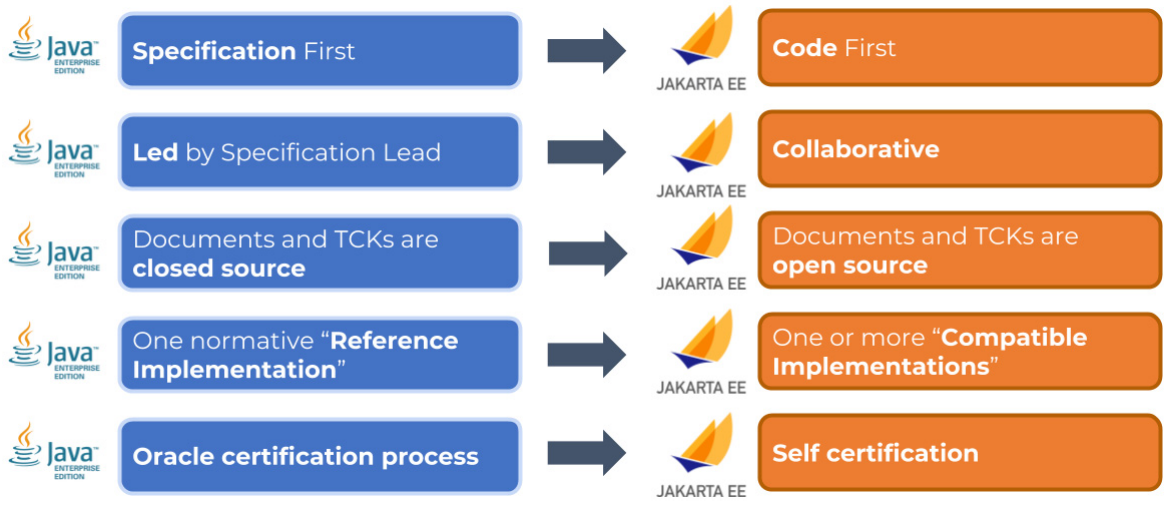
- Web-centric architecture
- Combined web and EJB™ component-based architecture, sometimes called EJB component-centric architecture
- Business-to-business (B2B) application architecture
- Web service application architecture

Discussion Questions

- What version of the Java Enterprise edition platform do you currently use
  - J2EE
  - Java EE 5
  - Java EE 6
  - Java EE 7
  - Java EE 8
  - Jakarta EE 8
  - Jakarta EE 9
- If you are not using Java/Jakarta EE 8/9, do you have plans to adopt or migrate to it?

INTRODUCTION TO JAKARTA EE

Java EE vs Jakarta EE



EE4J Eclipse Project

- Umbrella for all Jakarta EE projects
  - Specification Projects
  - Jakarta <specification name>
  - Implementations
  - Eclipse <project name>
- Contains
  - Descriptions of each project
  - Project downloads
  - Membership
  - Repositories
  - CI Nodes

Source Code

- On GitHub under Eclipse-EE4J organization
- All formerly Oracle led TCKs
- All specification and API projects
- 123 repositories

Governance Code

- Project Management Committee (PMC)
  - Development, Project Management
  - Membership: https://projects.eclipse.org/projects/ee4j/who
- Jakarta EE Steering Committee
  - Top level project
  - Membership: https://jakarta.ee/about/#steering-committee
- Jakarta EE Specification Committee
  - Specification process, oversight, architecture
  - Membership: https://jakarta.ee/about/#specifications-committee
- Jakarta EE Marketing Committee
  - Marketing, Brand, and Outreach
  - Membership: https://jakarta.ee/about/#marketing-brand-committee

Jakarta EE Specification Process (JESP)

- JESP
  - https://jakarta.ee/about/jesp
- Based on EFSP with small adjustments
- Delivery
  - Specification document
  - API
  - Javadoc
  - TCK
  - Compatible Implementation

Jakarta EE 8

> Launched during JakartaOne LiveStream on Sep. 10, 2019
> Compatible with Java EE 8
> Open Specifications
> Open Source TCK License and Process
> Compatible Implementations

Jakarta EE 9

> Jakarta EE 9 Milestone 1
  – June 23rd, 2020

What's available in Milestone 1?

> All Specification Projects
  - https://jakarta.ee/specifications/ (milestone)
    - Specifications (html and pdf)
    - Javadoc
    - APIs (maven)
    - TCK (eclipse download)
    - Compatible Implementation (maven)

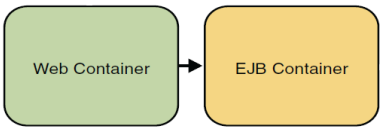
> Platform Project
  - https://jakarta.ee/specifications/platform/9/
  - https://jakarta.ee/specifications/webprofile/9/
  - https://jakarta.ee/specifications/managedbeans/2.0/

MODULE 2

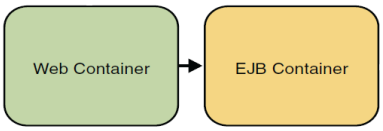
INTRODUCTION TO JBOSS EAP 7.4

JAVA EE APPLICATION SERVERS

Full Application Server



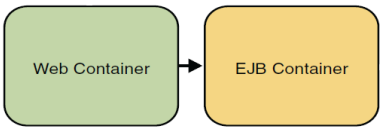
Full Application Server



Reference Implementation since Java EE 5

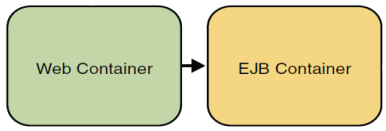


Full Application Server



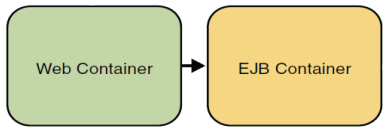


Full Application Server





Full Application Server

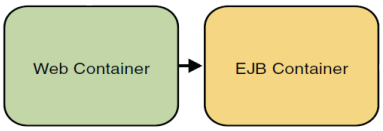






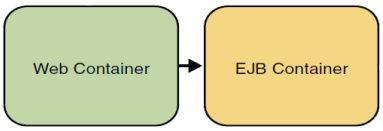


Full Application Server



ORACLE FUSION MIDDLEWARE WEBLOGIC SERVER 12c

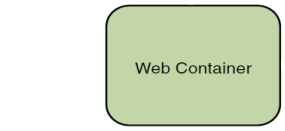
Full Application Server



WAS
Liberty Project

IBM WebSphere

Web Profile Application Server





Deployment Files

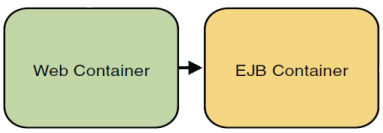
> EAR : Enterprise ARchive file -> Enterprise Application Project -> Full Platform

> **WAR** : Web Archive file -> **Dynamic Project** -> Web Container

> EJB-JAR: Java Archive file -> EJB Project -> EJB Container

> RAR: Resource Archive -> Connector Project -> EJB Container

(JCA : Java Connector API) -> JCA Adapter



Deployment Files

> De facto deployment file since Java EE 6

> **WAR** : Web Archive file -> **Dynamic Project** -> Web Container

Java EE 6 -> **Profile** -> Web Profile -> TomEE (Web Profile Compliant App Server)

WAR deployment

EJB Component -> Business Logic -> EJB Container

Web Container -> EJB Light Component

JBOSS ENTERPRISE APPLICATION PLATFORM

ARCHITECTURE AND FEATURES

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

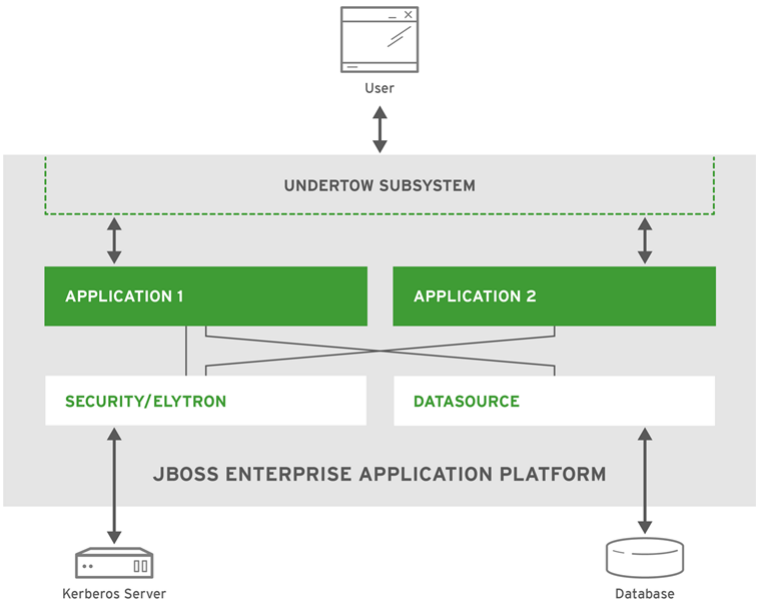
Uses of JBoss EAP 7

- JBoss Enterprise Application Platform (JBoss EAP) 7.4 is compatible with Jakarta EE 8 specifications
  - Web profile specification
  - Full-platform specification
- Each major version of JBoss EAP provides you with a tested, stabilized, and certified product.
- JBoss EAP provides preconfigured options for features such as high-availability clustering, messaging, and distributed caching.
- You can customize JBoss EAP to include only the subsystems required to meet your needs.
  - To improve the startup speed of your JBoss EAP 7.4 instance
  - To isolate any security issue from your application

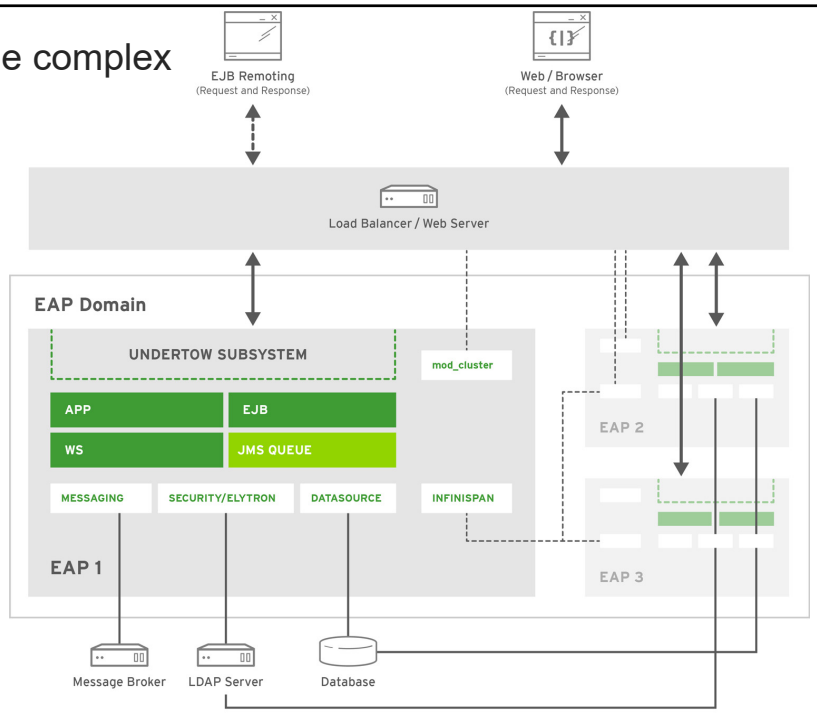
JBoss EAP 7 Features

> Jakarta EE compatible
> Managed Domain
> Management console and management CLI
> Simplified directory layout
> Modular class-loading mechanism
> Streamlined datasource management
> Unified security framework

Exploring the Architecture of JBoss EAP



JBoss EAP can handle complex use cases



High availability (HA) functionality of JBoss EAP

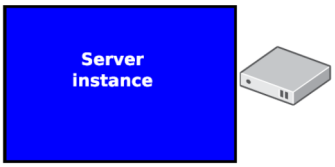
- You can use the JBoss EAP HA functionality to enhance any running applications by providing improved data sharing among multiple running JBoss EAP instances.
- HA in JBoss EAP refers to multiple JBoss EAP instances working together to deliver enhanced applications
  - resistant to fluctuations in data flow, server load, and server failure
- HA incorporates numerous qualities
  - Scalability
  - Load balancing
  - Fault tolerance

Supported operating modes in JBoss EAP

- JBoss EAP has powerful management capabilities for deployed applications.
- These capabilities differ depending on which operating mode is used to start JBoss EAP
- JBoss EAP offers the following operating modes:
  - Standalone server
    - To manage instances individually
  - Managed domain
    - To manage groups of instances from a single control point

Supported operating modes in JBoss EAP

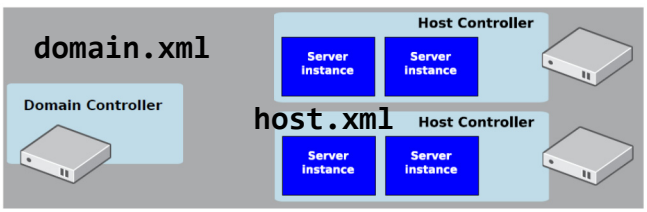
- A JBoss EAP 7 installation can run in two operating modes: serverstandalone and managed domain
- Standalone Server



`standalone.xml`

- Managed Domain

Management Console



QUIZ

USE CASES FOR STANDALONE SERVER VS. MANAGED DOMAIN

Q1

> A developer uses his/her computer to develop Java EE applications with JBoss Developer Studio and test his/her code on the same computer. What would be the mode most appropriate operating system for EAP 7?

(Choose only one option).

a) Standalone server
b) Managed domain
c) Integrated server
d) EAP 7 cannot be integrated into current versions of JBDS

Q2

> You must implement a two-node cluster for failover. What operating modes can be implemented with this requirement? (Choose two options)

a) Standalone server
b) Managed domain
c) Clustered domain
d) Integrated server
e) Standalone server along with external cluster middleware, for example, Red Hat RHEL7 High Availability Plug-in

Q3

> The server topology requires EAP to operate in a cluster of four computers, each with an EAP server instance and each server with identical configurations. Which EAP 7 operating mode would provide the most efficient and simple management? (Choose only one option)

> Standalone server
> Managed domain
> Clustered domain
> Integrated server
> Both the standalone server and the managed domain require manual duplication of configuration between server instances

Q4

> Three Java EE applications are deployed on an EAP server instance. Which EAP 7 operating mode would provide the easiest and lightest installation? (Choose only one option)

a) Standalone server
b) Managed domain
c) Clustered domain
d) Integrated server
e) There is no difference between the standalone server and the managed domain in this stage

Q5

> A single Java EE application is deployed on an EAP server instance. What EAP 7 operating mode can evolve in a cluster with two server instances? (Choose two options)

a) Standalone server
b) Managed domain
c) Clustered domain
d) Integrated server

Q6

> You have 12 EAP servers that must be configured identically. Which EAP 7 operating mode would allow the fastest installation time for this scenario? (Choose only one option)
> Standalone server
> Managed domain
> Clustered domain
> Integrated server
> Both the standalone server and the managed domain would be the same

MODULE 3

INSTALLING JBOSS EAP 7

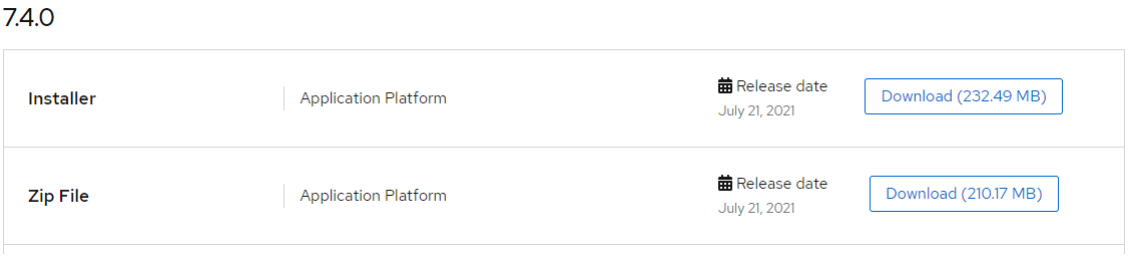
Objectives

> After completing this section, you should be able to do the following:

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

Downloading the installer

> https://developers.redhat.com/products/eap/download

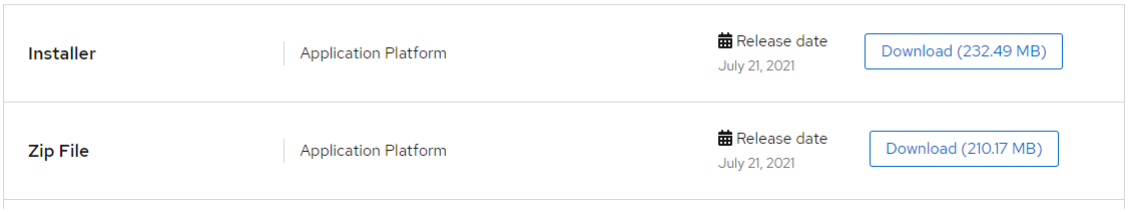


JBoss EAP 7 Installation Methods

> GUI Installer

> ZIP Installer

7.4.0



GUI INSTALLER

GUI Installer

> Installation
  - `java -jar jboss-eap-7.4.0-installer.jar`
  - `java -jar jboss-eap-7.0.0-installer.jar auto.xml`
  - `java -jar jboss-eap-7.4.0-installer.jar -console`
> Uninstallation
  - `java -jar uninstaller.jar`
  - `rm -rf jboss-eap-7.4.0`

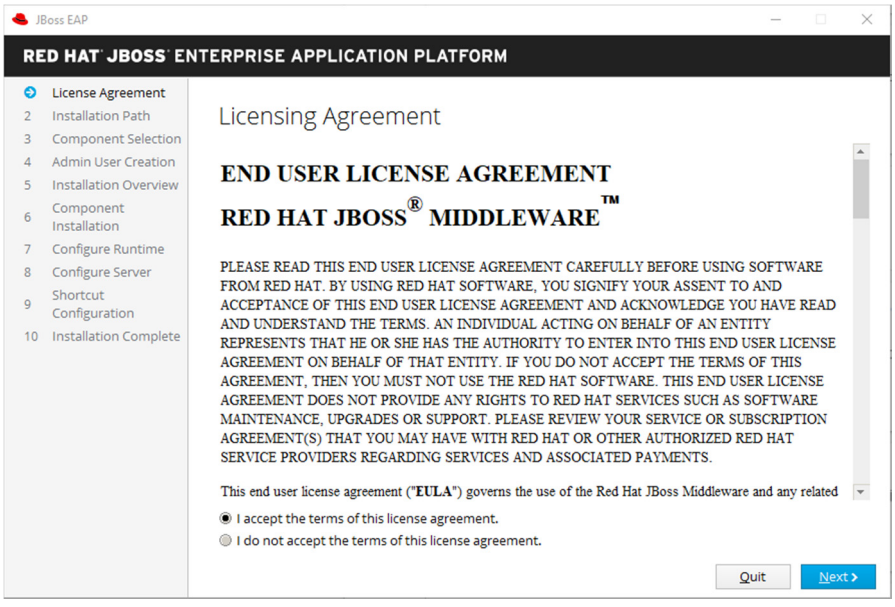
GUI Installer



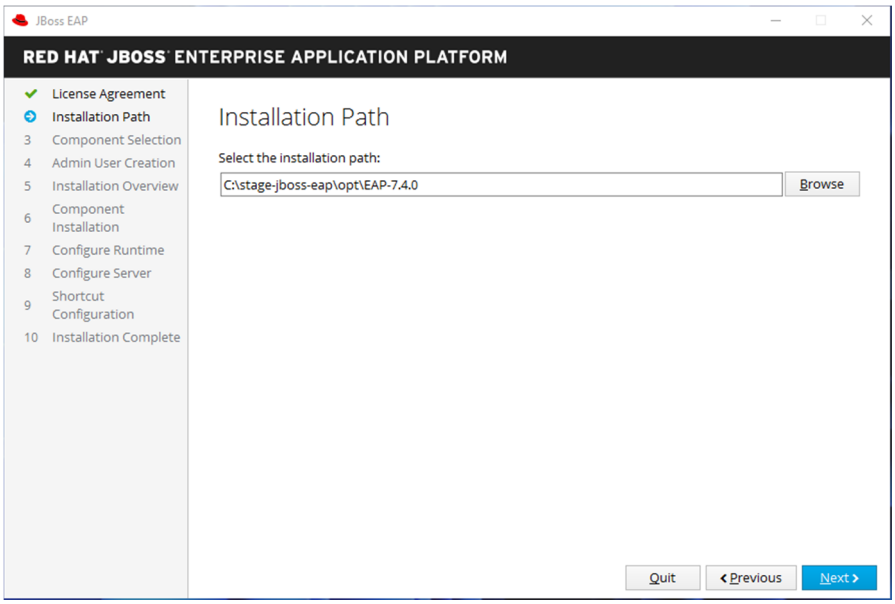
GUI Installer



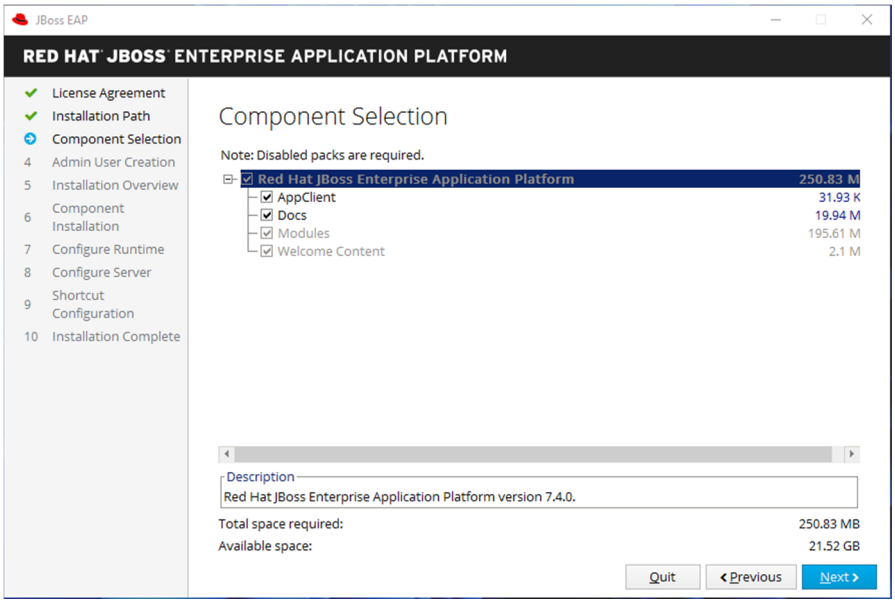
GUI Installer



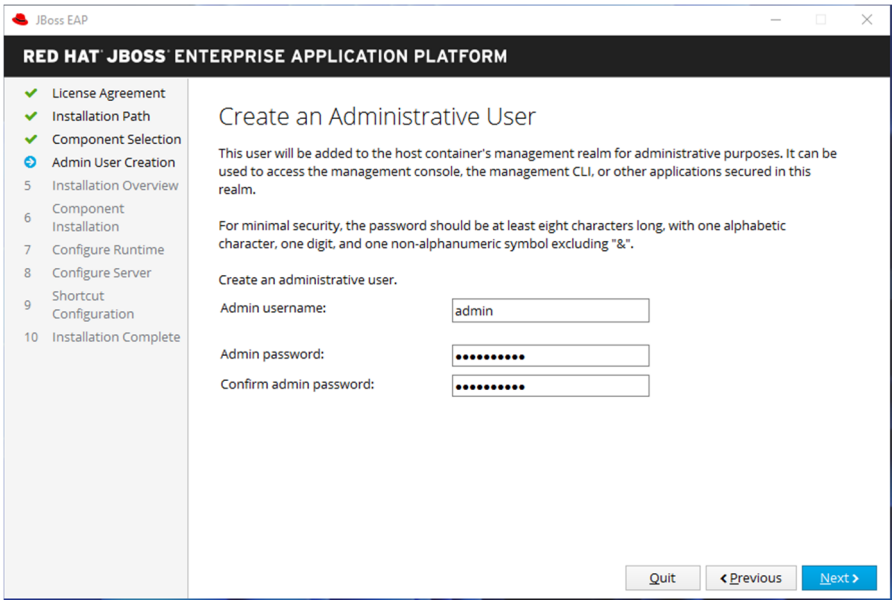
GUI Installer



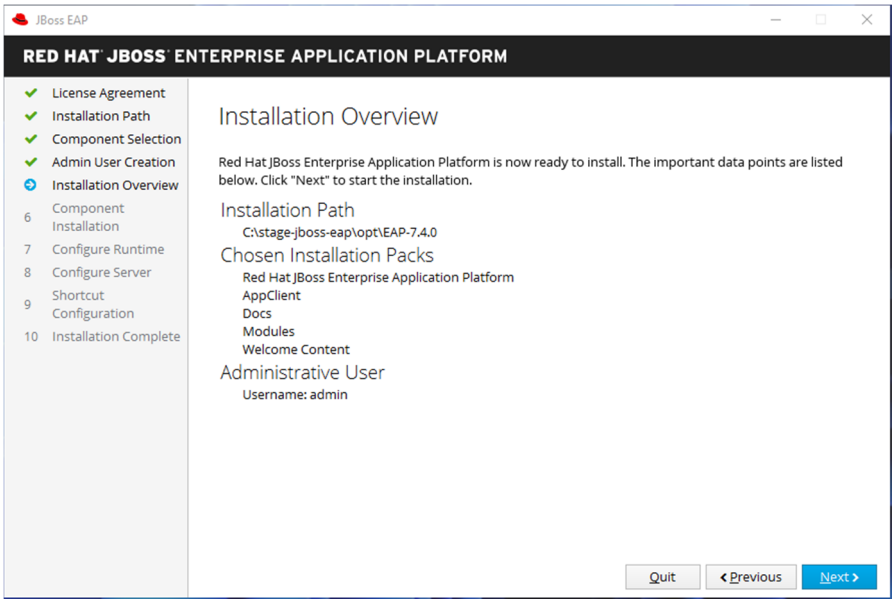
GUI Installer



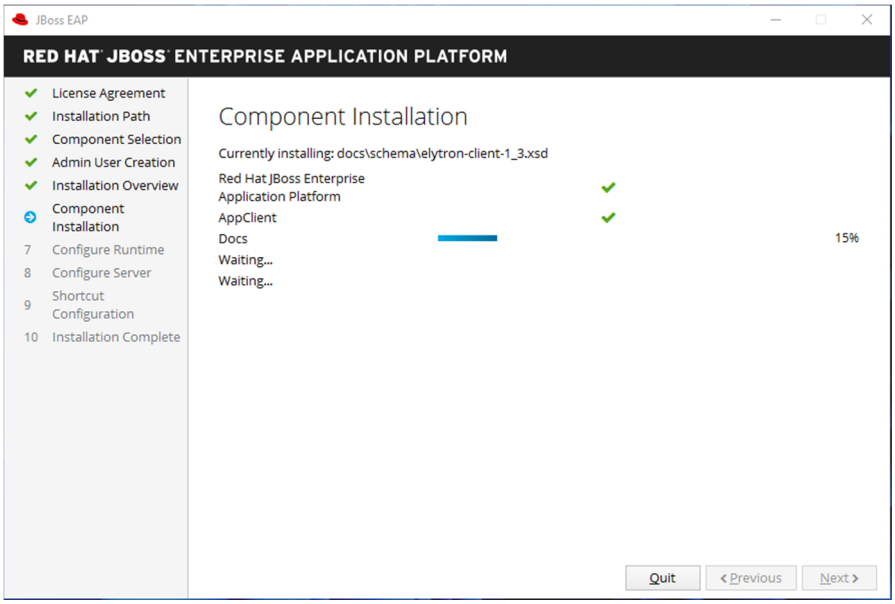
GUI Installer



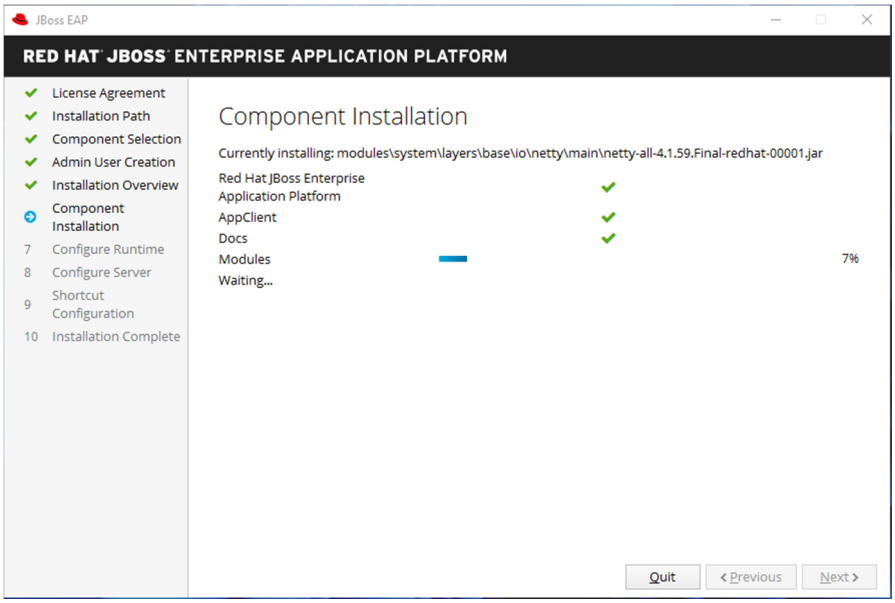
GUI Installer



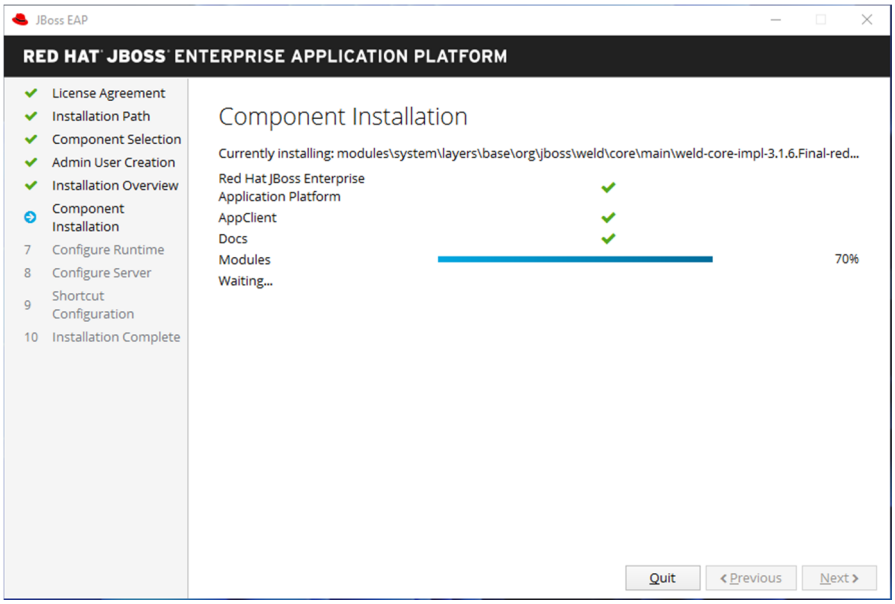
GUI Installer



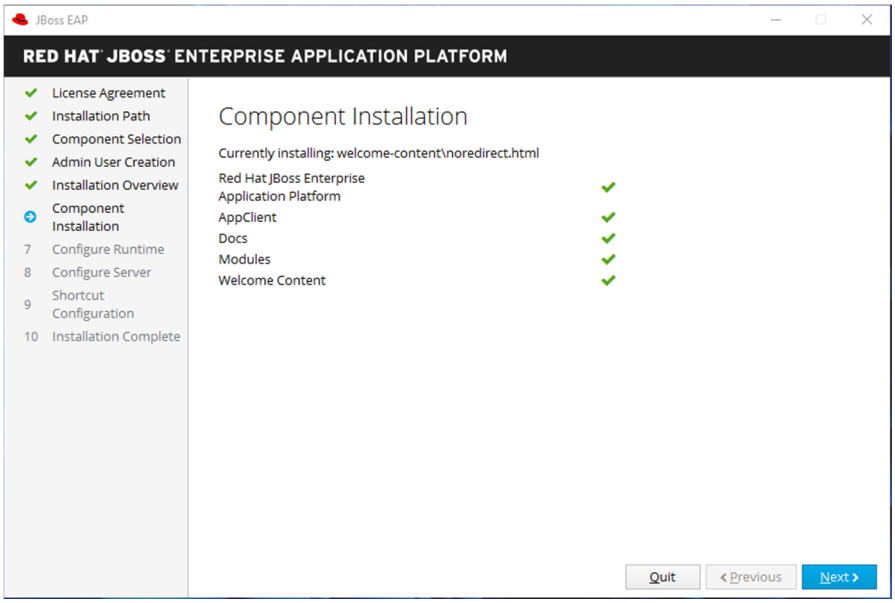
GUI Installer



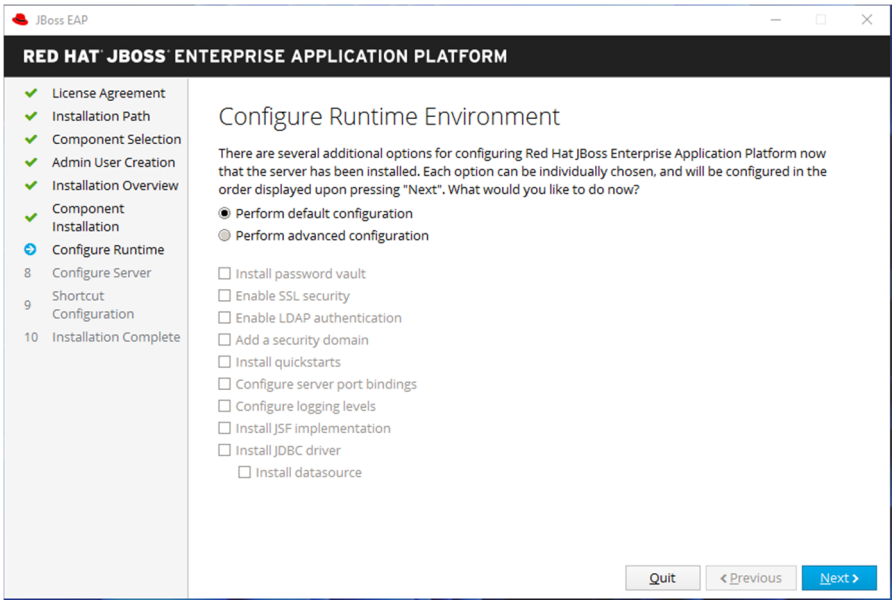
GUI Installer



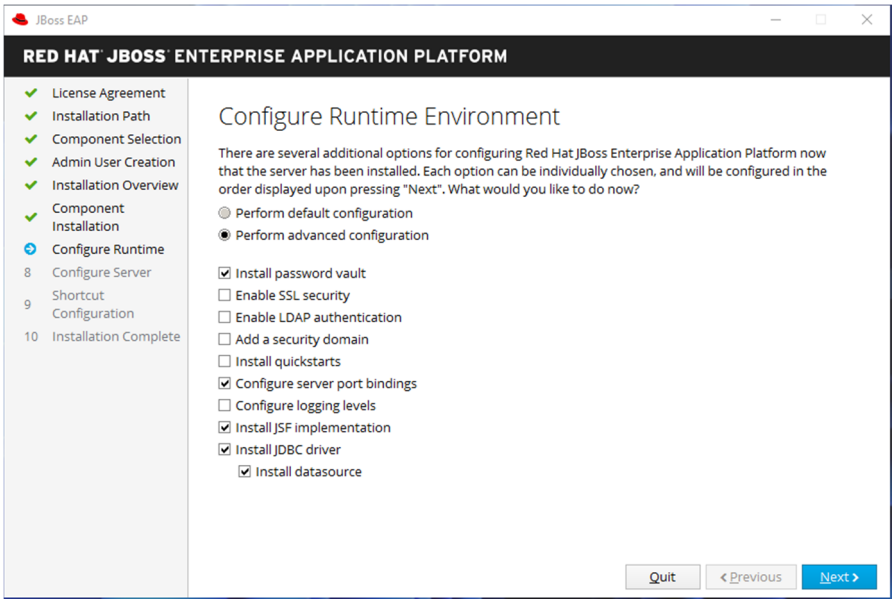
GUI Installer



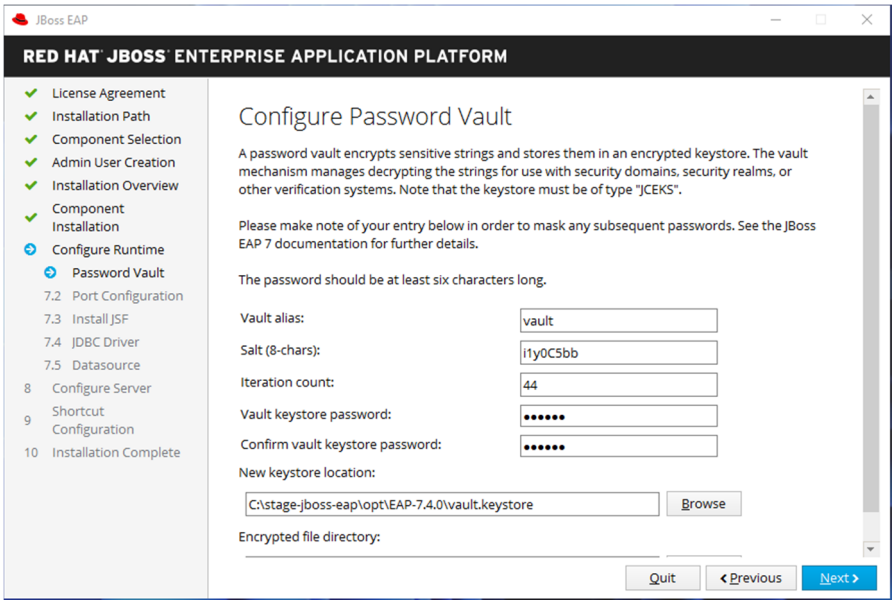
GUI Installer



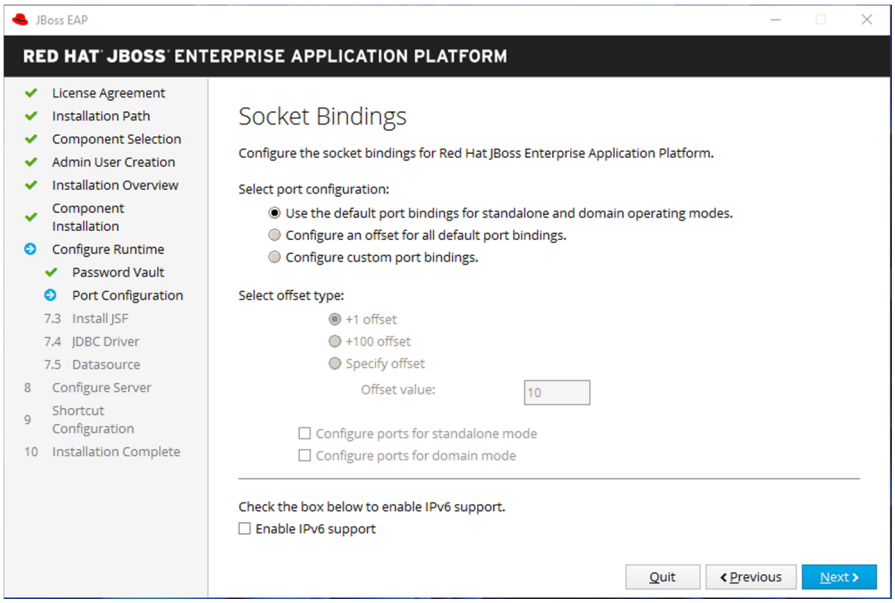
GUI Installer



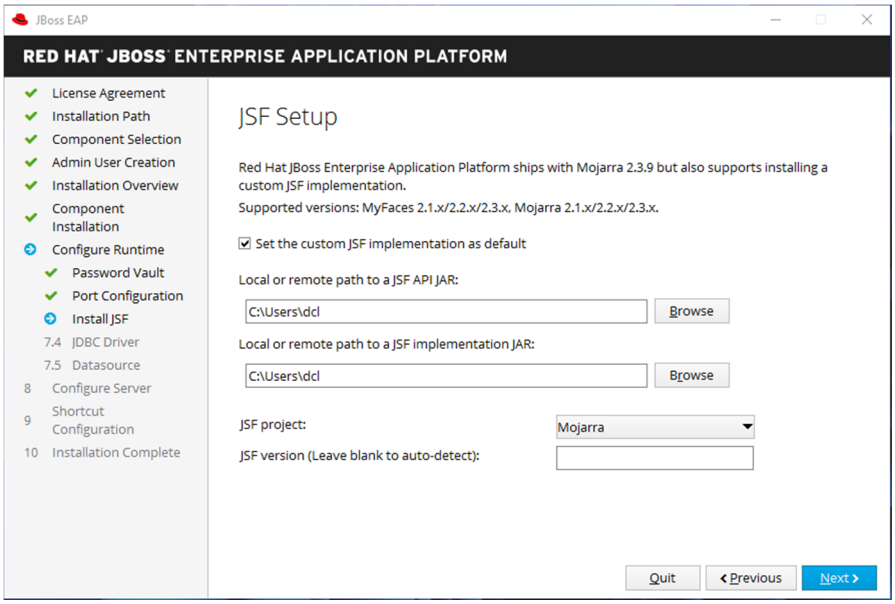
GUI Installer



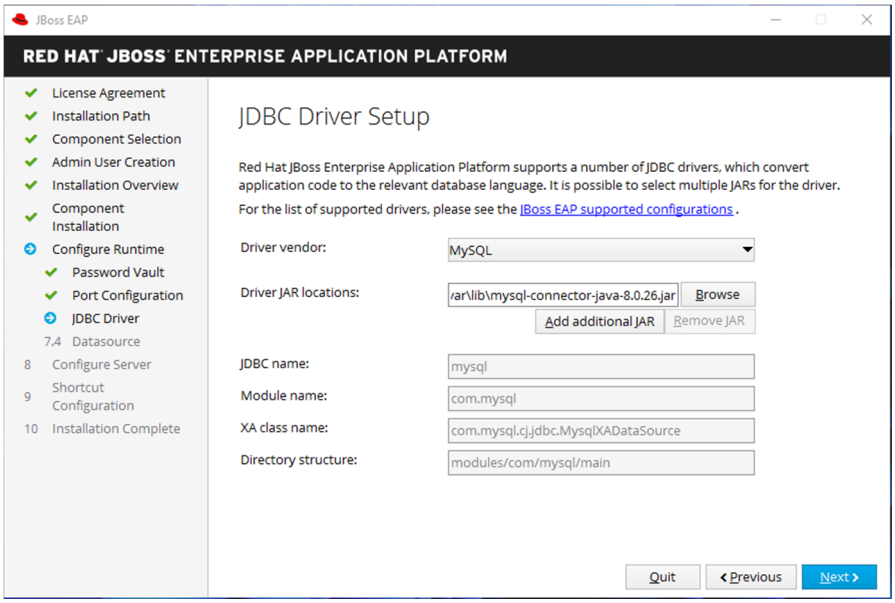
GUI Installer



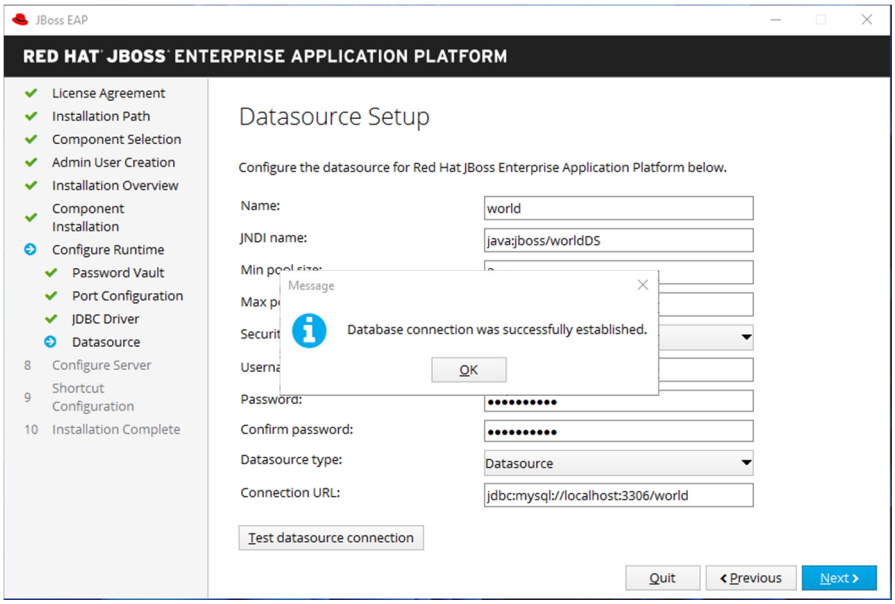
GUI Installer



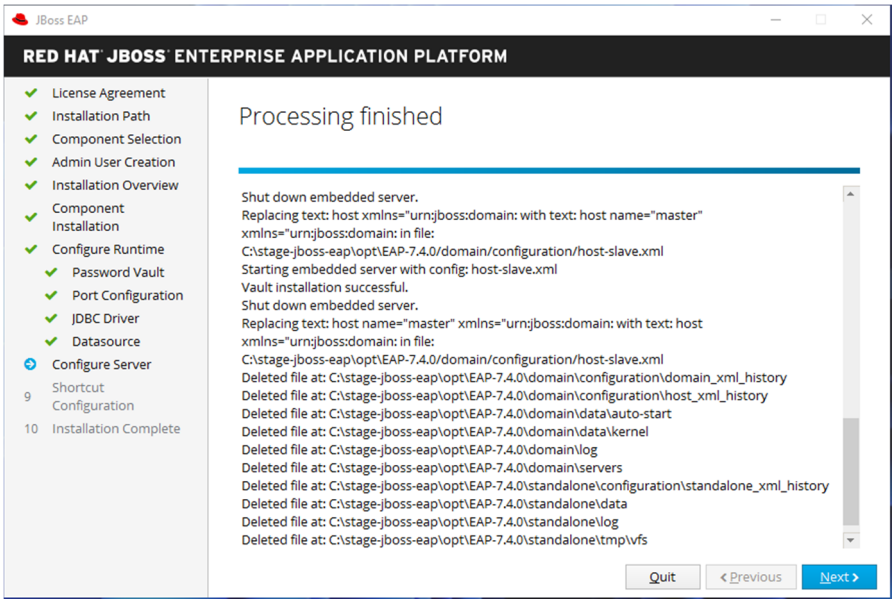
GUI Installer



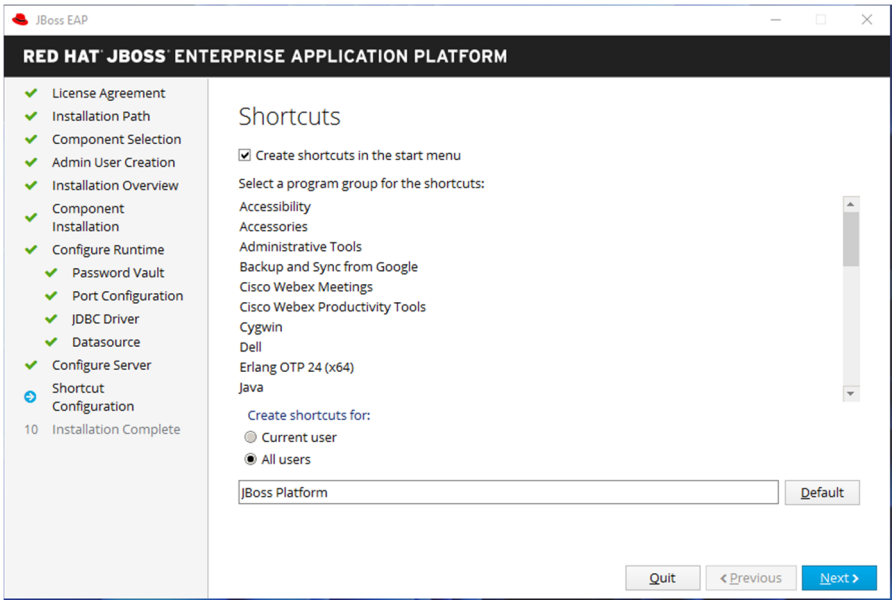
GUI Installer



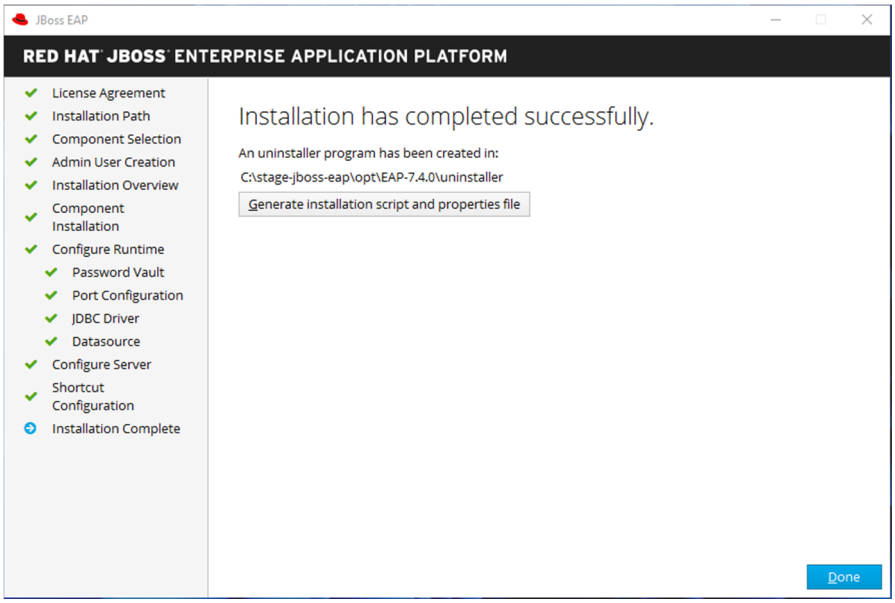
GUI Installer



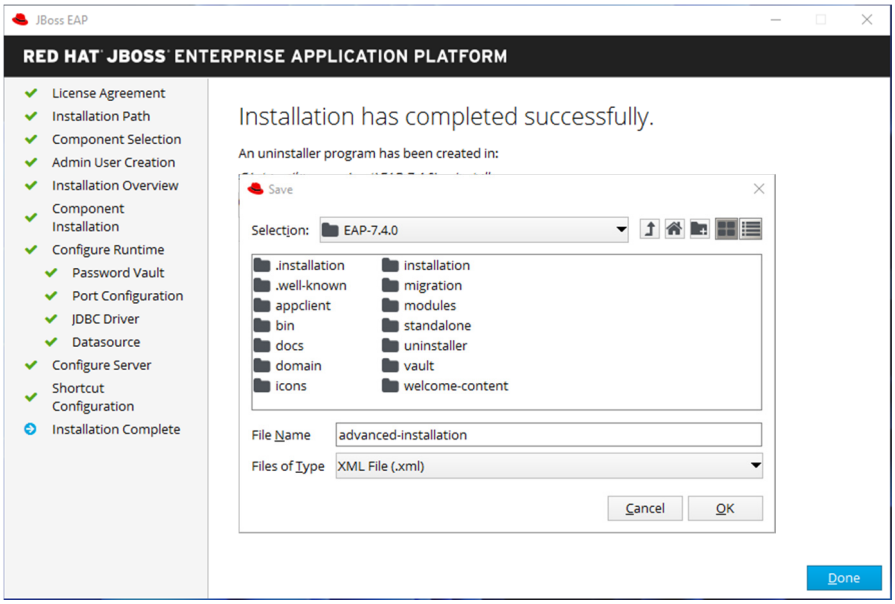
GUI Installer



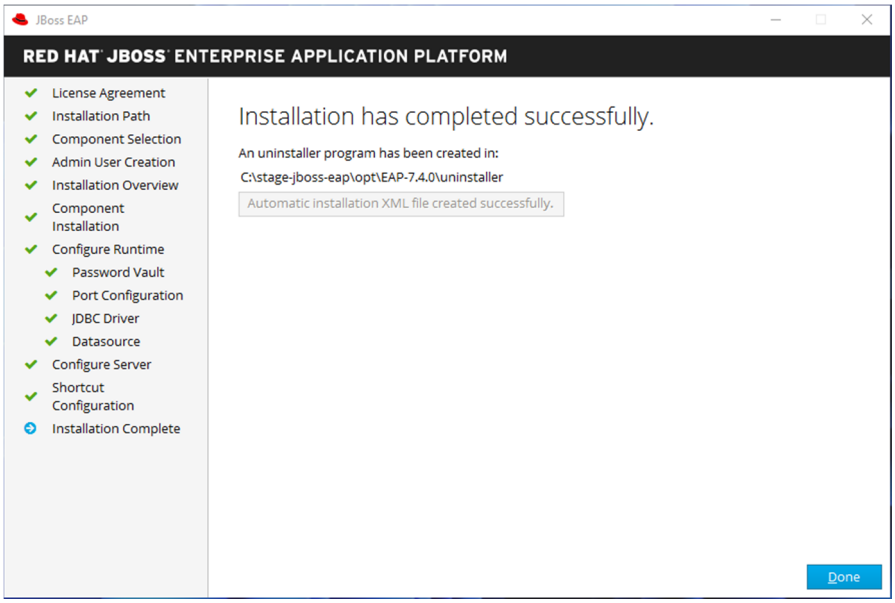
GUI Installer



GUI Installer

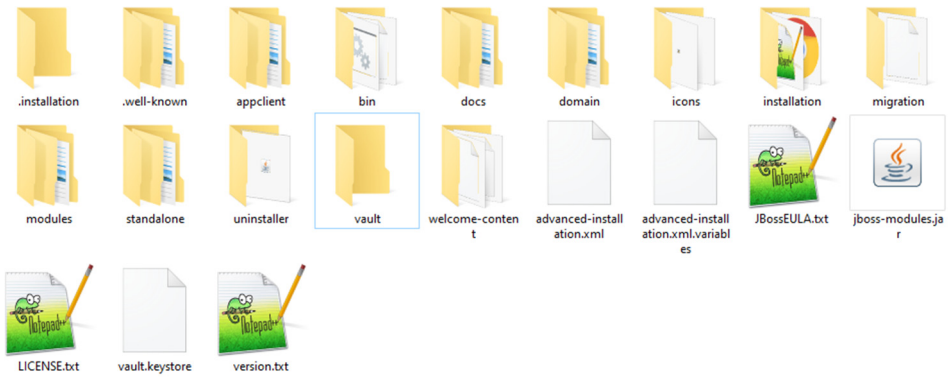


GUI Installer

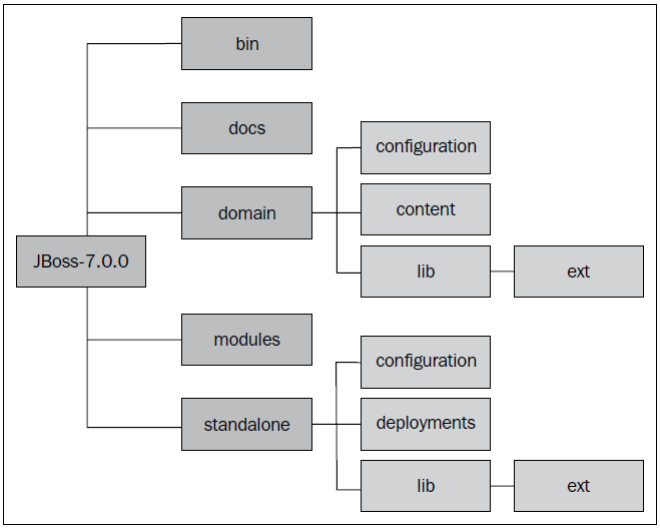


GUI Installer

> EAP 7 installation folders



The Application Server File System



The **bin** folder

This is where you start your application server instances.

> **standalone.bat**

starts a standalone server

> **standalone.conf.bat**

used to customize **JBoss**' bootstrap process

> **domain.bat**

starts a domain of servers.

> **jboss-cli.bat**

starts the new interactive command-line interface.

The **docs** folder

> In spite of its name, this folder does not contain the server documentation

> It contains two subfolders

1. The first one named **licenses** barely contains the licenses information for the application server and its dependencies.
   - You might use the **licenses.xml** file as quick reference for finding out which are the version shipped with the default application server modules or dependencies.
2. The other subfolder named **schema** contains the **.xsd** files which are used by the configuration as schema.

The **domain** folder

- The configuration folder contains the configuration files.
- The main configuration file is **domain.xml**
  - contains all services which are used by the nodes of the domain.
  - configures the socket-binding interfaces for all services.
- The other key file is **host.xml**
  - is used to define the management aspects of domain.
- The last file contained in the configuration folder is **logging.properties** which are used to define the logging format of the bootstrap process.
- The **content** folder is used as a repository to store deployed modules.

The **standalone** folder

- Its structure is quite similar to the **domain** folder with the notable exception of a **deployment** folder

The `welcome-content` folder

> This directory contains the default page which is loaded when you surf on the home of your application server (**`http://localhost:8080`**).

> In terms of web server configuration, this is the Web root context.

The `modules` folder

> Beneath the modules folder, you will find the application server's set of libraries, which are a part of the server distribution.

> The following table summarizes the different approaches used across different server releases:

AS Release	Bootstrap libs	Server libs
4.x	`JBOSS_HOME/server`	`JBOSS_HOME/server/<server>/lib`
5.x – 6.x	`JBOSS_HOME/server`	`JBOSS_HOME/common/lib` and `JBOSS_HOME/server/<server>/lib`
7.x	`JBOSS_HOME/jboss-modules.jar`	`JBOSS_HOME/modules` subfolders

Loading Application Server Modules

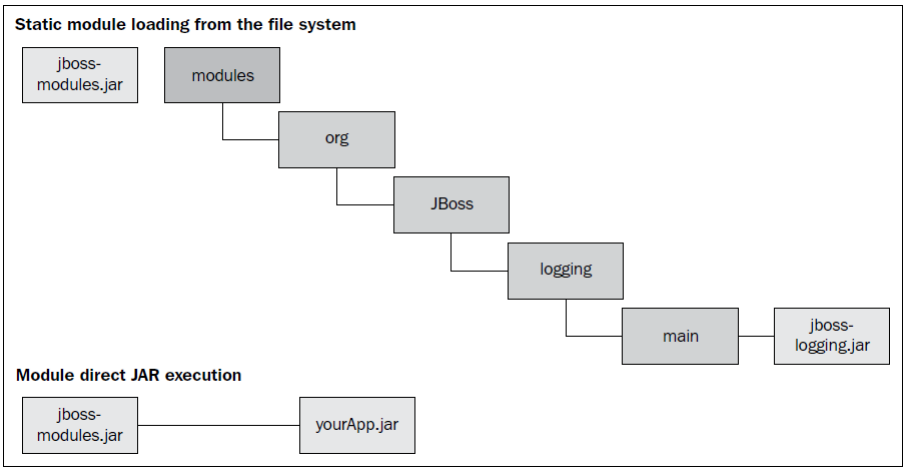
> Basically, each module is a pluggable unit. As depicted by the following picture.

> **JBoss** modules can load libraries using two approaches:

– Using a static file system

– Using direct JAR execution

Loading Application Server Modules



Unattended Automated GUI Installer

> Open the **.variables** file in a text editor and provide a password value for each key

```
adminPassword = password#2
vault.keystorepwd = vaultkeystorepw
ssl.password = user12345
```

> Run the installer using the automatic installation script XML file

```
java -jar jboss-eap-7.4.0-installer.jar
          auto.xml -variablefile auto.xml.variables
```

ZIP Installer

> EAP 7 is distributed as a zip file that can be extracted and run on an operating system with JDK installed.

> Contains all libraries, configuration files, and scripts required to start EAP, without further customization.

> Generally, this is the option chosen by developers interested in using EAP7.

Starting and stopping EAP

- Unix shell scripts and Windows batch scripts for starting EAP 7 are provided as a standalone server or in domain mode:
  - `standalone.sh`/`standalone.bat`
  - `domain.sh`/`domain.bat`
- They are located in the `bin` folder in the EAP installation directory.

Installing JBoss EAP as a service on Microsoft Windows Server

- For a standalone server:

```
service.bat install /loglevel INFO
```

- For a server in a managed domain

```
service.bat install /host EAP_HOST_NAME /loglevel INFO
```

- To start the service

```
net start JBossEAP7
```

- To stop the service

```
net stop JBossEAP7
```

- Removing JBoss EAP Service

```
service.bat uninstall
```

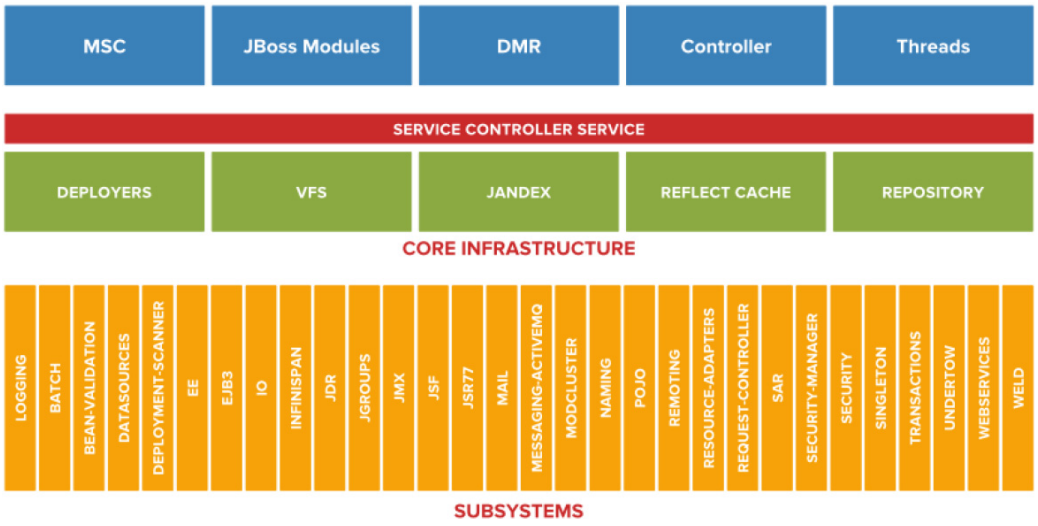
UNDERSTANDING EXTENSIONS, SUBSYSTEMS, AND PROFILES

Objectives

After completing this section, you should be able to do the following:

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

EAP 7 Advanced Architecture



JBoss EAP 7 Key Features

> Fast and Lightweight
> **Supports domain** (multi-node) management
> Multiple consistent management interfaces
> **CLI**, Java API, HTTP API, Console
> Unified, user focused configuration
> No more intermixing of internal wiring and configuration
> **Modular**
> Only APIs, no EAP implementation exposure
> True isolation (users can finally use their own XML parsers)

Key MSC Attributes

- Small, lightweight, and efficient
  - Tiny memory overhead
- Highly concurrent & scalable state machine
  - State transitions are “tasks”
- Services are primarily interface based
  - No reflection or XML required!
- Only two non-error, non-transition states
  - UP & DOWN
  - N-State systems are implemented with N services

Other notable features

- Multiple startup modes
  - Active, Passive, Lazy, On-Demand, Never
- Strong typing
- Listeners
  - Capable of modeling advanced cases
- Child services
- Asynchronous start & stop
- Advanced Cycle Detection and Error Detection
- Immediate & Deferred resolution

Modules and Extensions

- A module that provides functions and capabilities to the application server is called an extension.
- An extension does not simply provide a code, it also provides a management model, which consists of one or more subsystems, which allow the core configure the extension.
- The EAP core only loads and activates a module when it is required
  - For example, a non-clustered EAP server instance does not have any overhead of clusters.

Modules and Extensions

- Most of the EAP functions that are related to implementation and Server configuration and Java EE applications are implemented through extensions.
- A subset of all extensions provided by EAP is sufficient to implement the Java EE web profile.
- Adding a few more extensions, and EAP will meet the Java full profile.
- If the Java EE web profile is too much for an application, you can use a subset smallest of the available extensions.
- EAP 7 even provides some extensions that add capabilities beyond the Java EE standards.
  - An example of this is clustered singletons.

Modules and Extensions

> Extensions are added to an EAP instance by using the <extension> element,that appears at the beginning of the main EAP configuration file (**`standalone.xml`** or **`domain.xml`**):

```
<?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"/>
... remainder of configuration file ...
</server>
```

Modules and Extensions

> Adding an extension to the config files doesn't fully load that module extension or make it active.

> It just makes the extension manageable, which implies that the extension management model is loaded into memory and can be modified using EAP 7 administration tools.

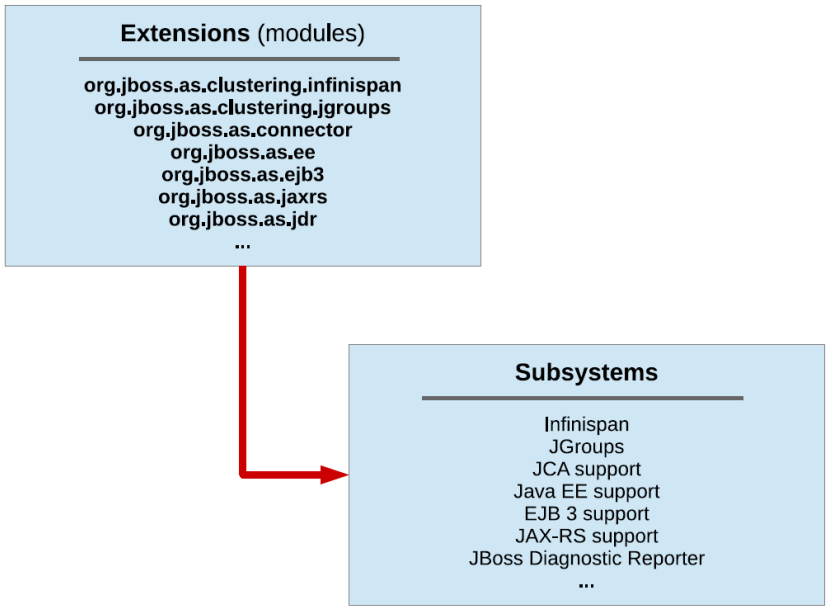
> For example, cluster configurations can be done if related extensions are available, but related modules will only be loaded and activated when an application that requires cluster services is deployed.

> If a clustered application is not deployed, none of the cluster-related extensions will be activated, even if the cluster extension management model has been modified by the administrator.

Extensions and Subsystems

> An extension management model provides one or more subsystems
> Subsystems can be configured and managed using the management API provided by the core, through 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's management model are configured within the <subsystem> element in the EAP configuration file.

Extensions and Subsystems



Subsystems and Profiles

- A subsystem configuration is not maintained by itself in the EAP 7 configuration files.
- A set of subsystem configurations is grouped within a <profile> element.
- The **`standalone.xml`** configuration file for the standalone server instance contains a single, anonymous profile definition.
- The **`domain.xml`** configuration file for the managed domain contains four factory predefined profiles that should meet most use cases for applications deployed to EAP:
  - **`default`**
  - **`ha`**
  - **`full`**
  - **`full-ha`**

Subsystems and Profiles

default profile:

Includes, among others, the following subsystems:

logging
datasources
infinispan
jca
jsf
remoting
security
weld
batch-jberet
ejb3
jaxrs
jpa
mail
webservices
transactions
undertow

full profile:

All subsystems from **default** profile plus:

iiop-openjdk
jsr77
messaging-activemq

ha profile:

All subsystems from **default** profile plus:

jgroups
modcluster
singleton

full-ha profile:

All subsystems from **full** profile plus:

jgroups
modcluster
singleton

QUIZ

Question #1

> What profiles use the messaging-activemq subsystem? (Choose two options)

a) default
b) ha
c) full
d) full-ha

Question #2

> What profiles use the registry subsystem? (Choose only one option).

a) default
b) ha
c) full
d) full-ha
e) All of them

Question #3

> What profile would you use if you had a Java EE application that used message-driven EJBs, that is, the messaging-activemq subsystem, and you were NOT planning to configure a cluster of server instances? (Choose only one option).

a) default
b) ha
c) full
d) full-ha
e) None of them, you would need to create a custom profile

Question #4

> What profile would you use for a WAR file deployed to a group of four clustered server instances? Assume that this WAR contains a simple web application that only requires access to the database. (Choose only one option).

a) default
b) ha
c) full
d) full-ha
e) None of them, you would need to create a custom profile

Question #5

> What 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 option).

a) default
b) ha
c) full
d) full-ha
e) None of them, you would need to create a custom profile

Question #6

> What profile would you use for a WAR file deployed on a single server? Assume that this WAR contains a simple web application that only requires access to the database. (Choose only one option).

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 Systems Administrator should be able to do the following:

> Describe the management options available for JBoss EAP.

JBoss EAP 7 Management Options

> Management CLI
> Management Console
> Management APIs
  – HTTP API
  – Native API

Management CLI

- The management command-line interface (CLI) is a command-line administration tool for JBoss EAP.
- Use the management CLI
  - To start and stop servers
  - To deploy and undeploy applications
  - To configure system settings
  - To perform other administrative tasks.
- Operations can be performed in batch mode
  - Allows multiple tasks to be run as a group.
- Many common terminal commands are available, such as ls, cd, and pwd.
- The management CLI also supports tab completion.

Management CLI

```
c:\stage-jboss-eap\opt\EAP-7.4.0\bin>jboss-cli.bat --connect
[standalone@localhost:9990 /] :product-info
{
    "outcome" => "success",
    "result" => [{"summary" => {
        "host-name" => "DESKTOP-HSPQDSO",
        "instance-identifier" => "4d0b33c5-14c4-4e6e-be99-7ecfededdff9",
        "product-name" => "JBoss EAP",
        "product-version" => "7.4.0.GA",
        "product-community-identifier" => "Product",
        "product-home" => "C:\\stage-jboss-eap\\opt\\EAP-7.4.0",
        "standalone-or-domain-identifier" => "STANDALONE_SERVER",
        "host-operating-system" => "Windows 10",
        .
        .
        .
```

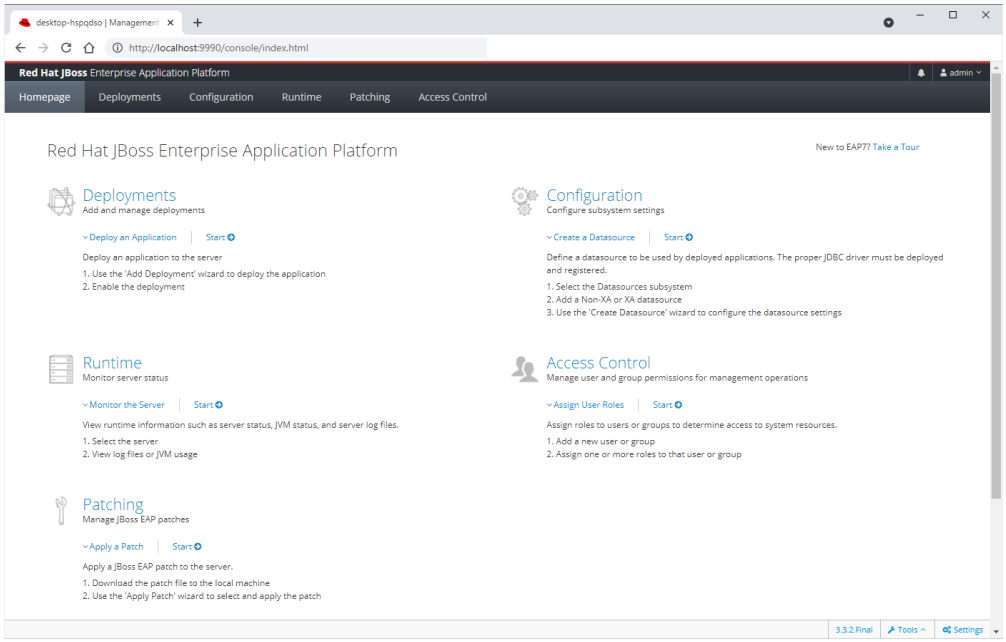
Management CLI

```
          "host-cpu" => {
             "host-cpu-arch" => "amd64",
             "host-core-count" => 8
         },
         "jvm" => {
             "name" => "Java HotSpot(TM) 64-Bit Server VM",
             "java-version" => "1.8",
             "jvm-version" => "1.8.0_281",
             "jvm-vendor" => "Oracle Corporation",
             "java-home" => "C:\\Program Files\\Java\\jre1.8.0_281"
         }
     }}]
}
```

Management Console

- The management console is a web-based administration tool for JBoss EAP.
- Use the management console
  - To start and stop servers
  - To deploy and undeploy applications
  - To tune system settings
  - To make persistent modifications to the server configuration
- The management console also has the ability
  - to perform administrative tasks
  - live notifications when any changes performed by the current user require the server instance to be restarted or reloaded.
- Management console can be accessed through a web browser at **http://localhost:9990/console/index.html**.

Management Console



Administrative user creation

- The EAP installer creates an administrative user during the installation process.
- If additional administrative users are required, or if the installation was NOT carried out using the EAP installer, new users can be created using the **add-user.sh** script.
- The Administration user type is responsible for accessing the administrative tools from EAP.
  - The credentials are be stored in the file `JBOSS_HOME/standalone/configuration/mgmt-users.properties`
  - with an encrypted password
- Java EE applications will use the Application using JAAS user type
  - The credentials are be stored in the file `JBOSS_HOME/standalone/configuration/app-users.properties`

Management API : HTTP API

> HTTP API

- The HTTP API endpoint is the entry point for management clients that rely on the HTTP protocol to integrate with the JBoss EAP management layer.
- The HTTP API is used by the JBoss EAP management console but offers integration capabilities for other clients as well.
- The HTTP API is accessible at

  `http://HOST_NAME:9990/management`
- While you can read, write, or perform other operations using the HTTP POST method, you can perform some read operations using a GET request.

Management API : HTTP API

```
curl --digest
http://localhost:9990/management/subsystem/undertow/server/default-server/http-listener/default
-H "Content-Type: application/json"
-u admin:Secret_123
```

Management API : Native API

- The native API endpoint is the entry point for management clients that rely on the native protocol to integrate with the JBoss EAP management layer.
- The native API is used by the JBoss EAP management CLI but offers integration capabilities for other clients as well.

Management API : Native API

```
// Create the management client
ModelControllerClient client = ModelControllerClient.Factory.create("localhost", 9990);

// Create the operation request
ModelNode op = new ModelNode();

// Set the operation
op.get("operation").set("read-resource");

// Set the address
ModelNode address = op.get("address");
address.add("subsystem", "undertow");
address.add("server", "default-server");
address.add("http-listener", "default");
```

Management API : Native API

```
// Execute the operation and manipulate the result
ModelNode returnVal = client.execute(op);
System.out.println("Outcome: " + returnVal.get("outcome").toString());
System.out.println("Result: " + returnVal.get("result").toString());

// Close the client
client.close();
```

MODULE 4

CONFIGURING JBOSS EAP AS A STANDALONE SERVER

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.

Standalone Server Overview

- Standalone servers are ideal for running a single EAP instance as a single server.
- On 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 functions of this server, such as logs, messaging, and data source management.



Standalone Server Overview

- A standalone server has only one profile defined by default, while a managed domain has four defined by default.
- A profile refers to a set of subsystems and their settings used by the server.
- Running multiple separate EAP server instances can create various complexities, such as requiring administrators to manage port conflicts, having multiple directories with repeating files and keeping the settings synchronized for each server.
- To resolve these issues, EAP can be run in a managed domain that will be discussed in more depth later in this course.
- The standalone EAP server provides the capabilities for clustering and high availability.
- These functions can be configured using several of the other configuration files provided by EAP, in addition to standalone.xml.

Standalone Server Directory Structure

> After installing an EAP distribution, the standalone folder initially contains the following subfolders:
  - `configuration/`
  - `deployments/`
  - `lib/`
> These directories are essential for starting EAP.
  - Without them, EAP could not start!

configuration/

> configuration/ contains the configuration files for a standalone server
  - The standalone_xml_history subfolder
    - maintains a version history of configuration files
  - The properties file **`logging.properties`** for configuring loggers
  - A users file, **`mgmt-users.properties`**, to define login credentials to secure management interfaces
  - A groups file, **`mgmt-groups.properties`**, to define login roles.

deployments/

- **`deployments/`** contains the application deployment files used by Java EE, such as EAR, WAR, and JAR files.
- Bookmark files (discussed later in this section) also appear in this folder.

lib/

- it is intended for common JAR file implementations.

Standalone Server Directory Structure

> After running EAP for the first time, the following standalone subfolders are created:
  - `data/`
    - A location available to subsystems that store content on the file system, such as the message queue or an in-memory database.
  - `log/`
    - The default location for server log files, including **`gc.log.0.current`** and **`server.log`**, which contain the EAP startup logs.
  - `tmp/`
    - For temporary files, such as the shared key mechanism used by the Command Line Interface (CLI), to authenticate users local to the server.

JBoss EAP base directory

> You can separate the configuration and data folders of an EAP instance from the folder in which the EAP application is installed.
> This allows users to keep settings out of the box, making it easier to upgrade and adopt a higher version of EAP.
> It also allows users to install EAP once and use it multiple times on the same computer.

JBoss EAP base directory

- There are two command line properties that allow users to create this separation between the installation folder and the folders used by the running instance:
- **`jboss.home.dir`**
  - This property represents the root directory where EAP is installed.
  - If this property is not defined, **`JBOSS_HOME`** is set as the default.
- **`jboss.server.base.dir`**
  - This property represents the base directory for the server configuration files.
  - If you do not define this property, **`JBOSS_HOME/standalone`** is set as the default.

JBoss EAP base directory

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

`standalone.bat`

```
-Djboss.server.base.dir=file://c:/path/to/base/directory
-Djboss.home.dir=file://c:/path/to/home/directory
```

Using a custom configuration file

- By default, EAP will use `BASE_DIR/configuration/standalone.xml` to start a standalone server.
- However, an alternate configuration file can be used by passing the file name as a parameter to the `standalone.sh` script.
- This approach is useful for customizing the configuration file without touching the default configuration files.
- To use a different config file, the `--server-config` parameter can be used as follows

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

- An alternative parameter (`-c`) can also be used:

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

CONFIGURING JBOSS EAP AS A STANDALONE SERVER

Objectives

After completing this section, you should be able to do the following:

> Describe the standalone.xml configuration file.

> Make configuration changes and updates to EAP as a standalone server

The `standalone.xml` configuration file

> The general structure of standalone.xml looks like this:

```
<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.

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

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

    <management>
     ...management int
    </management>

    <profile>
     ...list of subsys
    </profile>

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

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

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

```
<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 module is stored in

JBOSS_HOME/modules/system/layers/base/

Extensions

Java EE 7
full profile extensions

```
<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>
```

Administrating Interfaces

> The <management> section is used to define the management interfaces

```
<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
    </profile>

    <interfaces>
     ...interface definitic
    </interfaces>

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

    <deployments>
     ...deployed applicatic
    </deployments>
</server>
```

```
<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>
```

Administrating Interfaces

> The management interfaces allow remote clients to connect to the EAP instance to manage it.

> EAP exposes two management interfaces:

– HTTP interface: grants access to the administration console.

– Native interface - Allows administrative tasks to be performed in accordance with a proprietary, binary protocol, which is what the Command Line Interface (CLI) tool uses.

Administrating Interfaces

- **ManagementRealm** is used by management interfaces
- Client applications that need to have remote access to EAP use the `security-realm` tag called **ApplicationRealm**

```
<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>
```

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

Profiles and subsystems

- A profile is a group of subsystems.
- A subsystem is where the extensions for the standalone EAP instance can be configured.
- A subsystem is added to a profile with the following two objectives:
  - If a subsystem appears within a profile, the subsystem will be made available to the server using that profile.
  - The subsystem allows you to configure the extension to meet the specific needs of the user.
- When starting a standalone EAP server, there is only one profile available.

Profiles and subsystems

- The <profile> element has a group of <subsystem> child elements
- Each <subsystem> entry consists of the unique configuration settings for that particular extension.
- Some subsystem definitions do not require any values, such as those for 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>
```

Profiles and subsystems

> Some subsystems have many configuration settings, such as 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>
```

QUIZ

Question #1

> What sections in standalone.xml are required to add and configure a module on the EAP server? (Select two).

a) Profile
b) Extensions
c) Administration interface
d) Socket binding group

Question #2

> Which section in standalone.xml is responsible for describing which users can access the admin console? (Select one)

a) Profile
b) Extensions
c) Administration interface
d) System properties

Question #3

> What management interface is responsible for providing access to the management console? (Select one)

a) Native interface
b) HTTP interface
c) HTTPS interface
d) Unsecured interface

Question #4

> How many profiles can be defined in standalone.xml? (Choose one)

a) One
b) Four
c) As many as necessary
d) Zero

Question #5

> What methods are recommended for configuring EAP? (Select two)

a) Edit standalone.xml directly with a text editor
b) The management console
c) The EAP Command Line Interface
d) Make minor changes to standalone.xml and major changes to the management console.

CONFIGURING INTERFACES AND SOCKET BINDING GROUP

Objectives

After completing this section, students should be able to do the following:

> Configure JBoss EAP network interfaces and socket binding groups

Interfaces

> JBoss EAP references named interfaces throughout the configuration.
> This allows the configuration to reference individual interface declarations with logical names, rather than requiring the full details of the interface at each use.
> This also allows for easier configuration in a managed domain, where network interface details can vary across multiple machines.
> Each server instance can correspond to a logical name group.
> The `standalone.xml`, `domain.xml`, and `host.xml` files all include interface declarations.

Interfaces

```
<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>
```

Interfaces

> By default, JBoss EAP binds these interfaces to 127.0.0.1

> These values can be overridden at runtime by setting the appropriate property.

```
standalone.bat -Djboss.bind.address=192.168.2.108
standalone.bat -bmanagement 192.168.2.108
```

Configuring Interfaces

> If a host has multiple network interface cards (NICs) with multiple IP addresses and only one NIC should be exposed, use the following interface configurations

```
<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 all available IP addresses on the server:

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

Configuring Interfaces

> The following interface uses a criteria that will only bind with an IPv4 address:

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

Configuring Interfaces

- Criteria can be combined to be highly specific in defining an interface.
- For example, the following interface defines a specific subnet that supports multicast

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

Socket binding groups

- A socket binding group is a set of named socket bindings, which allows users to define all the ports required for an EAP instance.
- AS ocket binding group is defined by the <socket-binding-group> element, which consists of a set of <socket-binding> child elements.
- For example, 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>
```

QUIZ

> Based on the settings provided, select the correct answers from 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>
```

Question #1

> Which of the following commands does NOT start a standalone EAP server that exposes the public IP address to 127.0.0.1, depending on the configuration? (Choose 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
```

Question #2

> What port does the EAP management console use, based on the previous configuration with all defaults used? (Choose one)

a) 9990

b) 8080

c) 19990

d) 18080

Question #3

> What interfaces bind to 127.0.0.1 in the above configuration? (Select one)

a) Global
b) Management
c) Public
d) None of the above

Question #4

> What ports can be used with HTTPS binding, based on the above configuration? (Choose one)

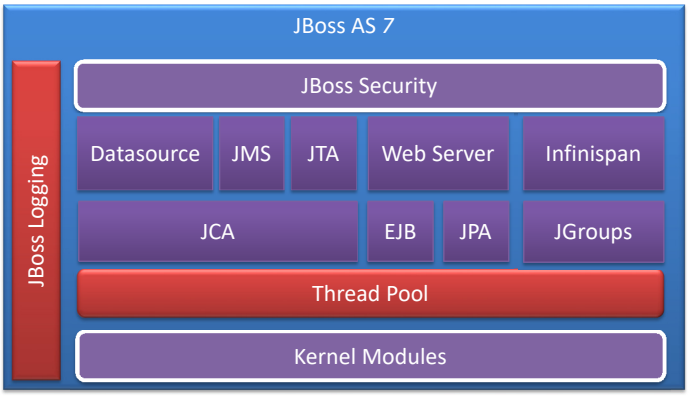
a) 8443
b) 8443
c) 9993, 18443
d) 18443, 19993

Deployments

> The last section of the configuration file contains the deployed application, which has been registered on the application server.
> Each time a new application is deployed or un-deployed, this section is updated to reflect the new application stack.

Configuring core subsystems

> A rough representation of core **JBoss AS 7** subsystems



> We will explore the areas that are highlighted in bold:
  – The **Thread Pool** subsystem
  – The **JBoss Logging** subsystem

Configuring the Thread Pool subsystem

> Thread Pools address two different problems
  - They usually deliver improved performance when executing large numbers of asynchronous tasks, due to reduced per-task invocation overhead
  - They provide a means of bounding and managing the resources, including Threads, consumed when executing a collection of tasks.

Configuring the Thread Pool subsystem

> By appropriately configuring the Thread Pool section, you can effectively tune the specific areas that use that kind of Pool to deliver new tasks.
> The application server Thread Pool configuration can include the following elements:
  1. Thread factory configuration
  2. Bounded Threads configuration
  3. Unbounded Threads configuration
  4. Queueless Thread Pool configuration
  5. Scheduled Thread configuration

Configuring the Thread factory

- A **Thread factory** (implementing **`java.util.concurrent.ThreadFactory`**) is an object that creates new Threads on demand.
- Using Thread factories removes hardwiring of calls to new Thread, enabling applications to use special Thread subclasses, priorities, and so on.
- The Thread factory is not included by default in the server configuration as it relies on defaults, which you will hardly need to modify.
- Nevertheless, we will provide a sample configuration of it for the experienced user who requires complete control of the Thread configuration.

Configuring the Thread factory

Example

```
<thread-factory
      name="MyThreadFactory"
      thread-name-pattern="My Thread %t"
      priority="5"
      group-name="dummy" />
```

Configuring the Thread factory

> The **`thread-name-pattern`** is the template used to create names for Threads.

> The avaliable patterns

Pattern	Output
`%%`	Emits a percent sign
`%g`	Emits the per-factory Thread sequence number
`%f`	Emits the global Thread sequence number
`%i`	Emits the Thread ID
`%G`	Emits the Thread group name

Bounded Thread Pool

> A bounded Thread Pool is the most common kind of Pool used by the application server

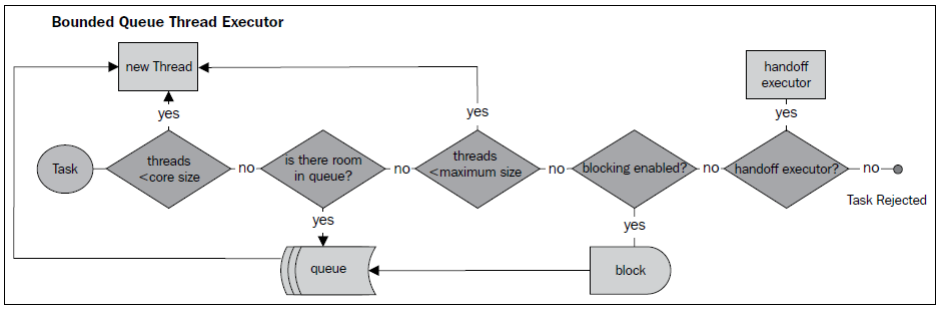
> It helps prevent resource exhaustion by defining a constraint on the Thread Pool's size

> It is also the most complex to use.

Bounded Thread Pool

> Each time a new task is submitted, if the number of running Threads is less than the core size, a new Thread is created.

> Otherwise, if there is room in the queue, the task is queued.



Bounded Thread Pool

> Here's a sample of bounded Thread Pools, which is included in the configuration:

```
<bounded-queue-thread-pool
        name="jca-short-running"
        blocking="true">
  <core-threads count="10" per-cpu="20"/>
  <queue-length count="10" per-cpu="20"/>
  <max-threads count="10" per-cpu="20"/>
  <keepalive-time time="10" unit="seconds"/>
</bounded-queue-thread-pool>
```

Bounded Thread Pool

> A short description of each attribute

Attribute	Description
`name`	Specifies the bean name of the created executor
`allow-core-timeout`	Specifies whether core Threads may time out or not; if `false`, only Threads above the core size will time out
`blocking`	Specifies whether the submitter Thread will block if no space is available in this executor
`core-threads`	Specifies the core Thread Pool size, which is smaller than the maximum Pool size
`max-threads`	Specifies the maximum Thread Pool size
`queue-length`	Specifies the executor queue length
`keepalive-time`	Specifies the amount of time that Threads beyond the core Pool size should be kept running, when idle
`thread-factory`	Specifies the bean name of a specific Thread factory to use to create worker Threads
`handoff-executor`	Specifies an executor to delegate tasks to in the event that a task cannot be accepted

Performance

> **Queue size** and **Pool size** are two samples of performance trade-off

> When using a small Pool with a large queue, you cause to minimize the CPU usage, OS resources, and context-switching overhead.

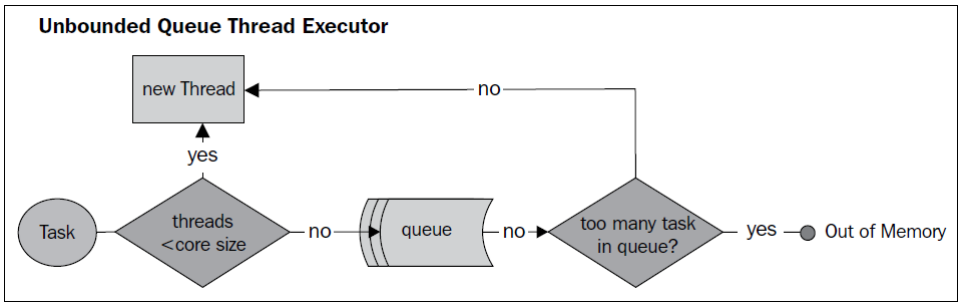
  – It can, however, produce an artificially low throughput.

> If tasks are strongly I/O bound (and thus frequently blocked), a system may be able to schedule time for more Threads than you otherwise allow.

> Use of small queues generally requires larger Pool sizes, which keeps the CPUs busier but may encounter unacceptable scheduling overhead, which also decreases throughput.

Unbounded Thread Pool

> This Thread Pool executor follows a simpler but more risky! approach
  – Always accepts new tasks.



Unbounded Thread Pool

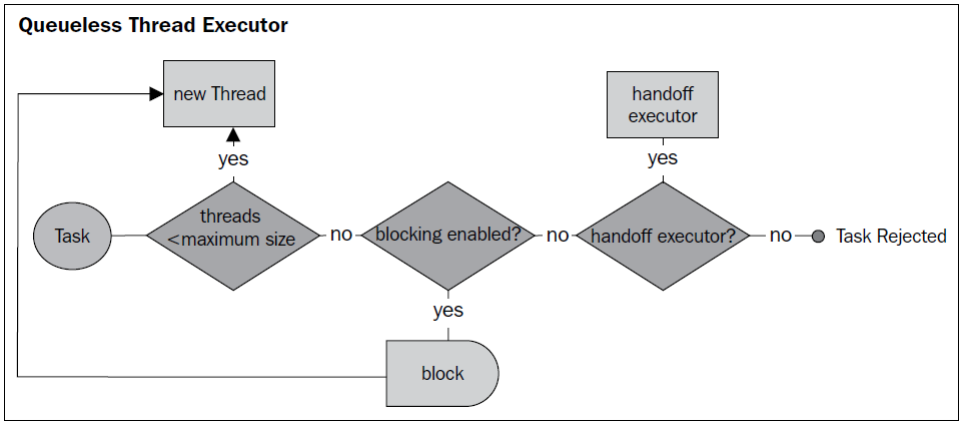
> In practice, the unbounded Thread Pool has a core size and a queue with no upper bound.
> When a task is submitted, if the number of running Threads is less than the core size, a new Thread is created.
> Otherwise, the task is placed in a queue.
> If too many tasks are allowed to be submitted to this type of executor, an out of memory condition may occur.
> Due to its inherent risk, unbounded Thread Pools are not included by default in the server configuration.

```
<unbounded-queue-thread-pool name="unbounded-threads" >
   <max-threads count="10" per-cpu="20"/>
   <keepalive-time time="10" unit="seconds"/>
</unbounded-queue-thread-pool>
```

Queueless Thread Pool

- This is a Thread Pool executor with no queue.
- Basically, this executor short-circuits the same logic of the bounded Thread executor, as it does not attempt to store the task in a queue.
- When a task is submitted, if the number of running Threads is less than the maximum size, a new Thread is created.
- Otherwise, if blocking is enabled, the caller blocks until another Thread completes its task and accepts the new one.
- If blocking is not enabled, the task is assigned to the designated hand-off executor, if one is specified.
- Without any designated hand-off, the task will be rejected.

Queueless Thread Pool

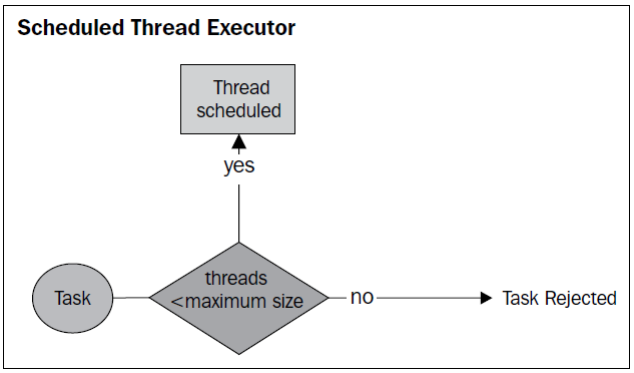



```
<queueless-thread-pool
  name="queueless-thread-pool" blocking="true">
  <max-threads count="10" per-cpu="20"/>
  <keepalive-time time="10" unit="seconds"/>
</queueless-thread-pool>
```

Scheduled Thread Pool

> The server-scheduled Thread Pool is used for activities on the server-side that require running periodically or with delays.

> It maps internally to a **`java.util.concurrent.ScheduledThreadPoolExecutor`** instance



Configuring the application server logging

> Every application needs to trace logging statements.

> At the moment, there are several implementations of logging libraries for Java applications, the most popular ones are:

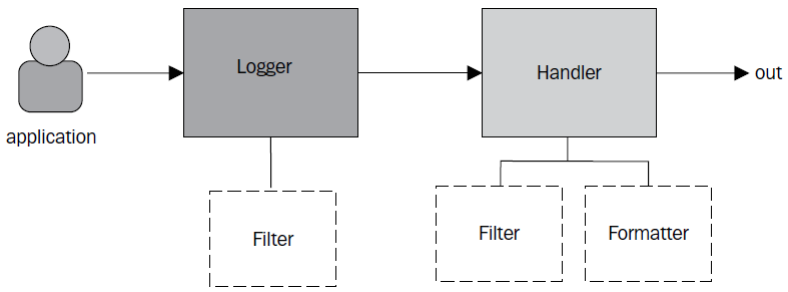
- **Log4j**
  - A flexible open source logging library from Apache.
  - Log4j is widely used in the open source community, and it was the default logging implementation on earlier releases of JBoss AS.
- **J2SE logging libraries** (**JUL**)
  - Provides the logging classes and interfaces as part of the J2SE platform's standard libraries.

Configuring the application server logging

- Log4j and JUL are indeed very similar APIs.
- They differ conceptually only in small details, and in the end do more or less the same thing, except Log4j has more features that you may or may not need.
- JBoss logging framework is substantially based on JUL, which is built around three main concepts: **loggers**, **handlers**, and **formatters**.
- These concepts allow developers to log messages according to their type and priority, to control where messages end up and how they look when they get there.

Configuring the application server logging

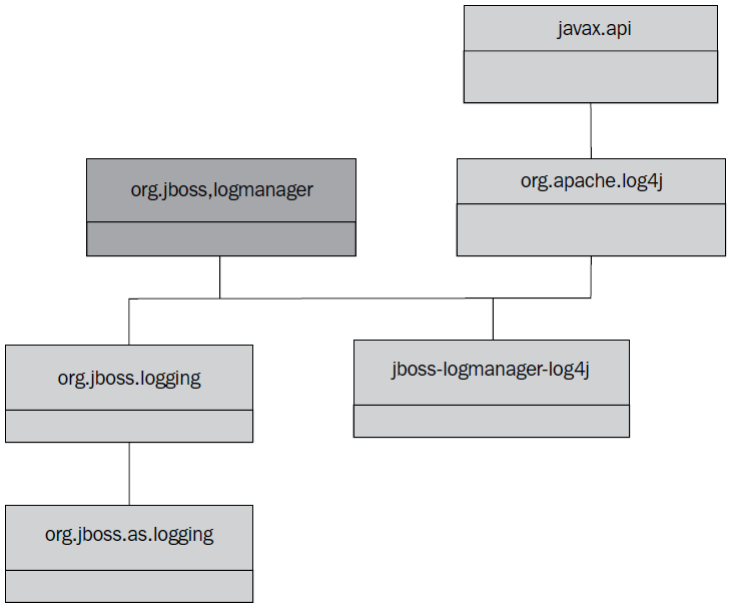
- **Applications** make logging calls on **Logger** objects.
- These **Logger** objects allocate LogRecord objects, which are passed to **Handler** objects for publication.
- Both the **Logger** and the **Handler** may use the **Formatter** to arrange the layout of logs and **Filter** to decide if they are interested in a particular LogRecord.



Choosing your logging implementation

- **JBoss AS**, through its releases, has used different frameworks to handle application server logs.
- In **JBoss AS 5** and earlier, **log4j** was the default logging API used by the application server, and it was defined in the server/<server>/conf/jboss-log4j.xml file.
- Since JBoss AS 6, the logging provider switched to JBoss's own implementation, which is based on the JDK 1.4 logging system.
  - However, it provides several fixes or works around many serious problems in the default JDK implementation.
  - JBoss AS replaces the default JUL LogManager implementation with its own implementation

JBoss AS7 logging subsystem



The console-handler

> The console-handler defines a handler, which simply writes log messages to the console.

```
<console-handler name="CONSOLE"            autoflush="true">
   <level name="INFO"/>
   <formatter>
     <pattern-formatter
        pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
   </formatter>
</console-handler>
```

The periodic-rotating-file-handler

> The 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, which should be in a format understood by **`java.text.SimpleDateFormat`**.

```
<periodic-rotating-file-handler name="FILE"
                                autoflush="true">
  <level name="INFO"/>
  <formatter>
    <pattern-formatter
     pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
  </formatter>
  <file relative-to="jboss.server.log.dir"
        path="server.log"/>
  <suffix value=".yyyy-MM-dd"/>
</periodic-rotating-file-handler>
```

The size-rotating-file-handler

> The size-rotating-file-handler defines a handler that writes to a file, rotating the log after the size of the file grows beyond a certain point.
> It also keeps a fixed number of backups.
> There's no size handler defined in the standard configuration

The size-rotating-file-handler

> we can find out its basic configuration from the **`docs/schema/jboss-as-logging.xsd`** file:

```
<size-rotating-file-handler name="FILESIZE"
                            autoflush="true" >
  <rotate-size value="500k" />
  <level name="INFO"/>
  <formatter>
   <pattern-formatter
     pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
  </formatter>
  <file relative-to="jboss.server.log.dir"
        path="server.log"/>
</size-rotating-file-handler>
```

The asynchronous handler

> The asynchronous handler is a composite handler, which attaches to other handlers to produce asynchronous logging events.
> Behind the scenes, this handler uses a bounded queue to store events.
> Every time a log is emitted, the asynchronous handler appends the log into the queue and returns immediately.

The asynchronous handler

> An example of asynchronous logging for the FILE appender:

```
<async-handler name="ASYNC" >
    <level name="INFO"/>
    <queue-length>1024</queue-length>
    <overflow-action>block
    </overflow-action>
    <sub-handlers>
      <handler-ref name="FILE" />
    </sub-handlers>
</async-handler>
```

When to use asynchronous handler

> The asynchronous handler produces a substantial benefit to applications, which are heavily I/O bound, since an asynchronous logging event might be fired at times where
> Threads are blocked on intensive I/O operations.
> At the opposite end, CPU-bound applications might not benefit at all from asynchronous logging as it will put additional stress on the CPU.

MODULE 5

CONFIGURING ENTERPRISE SERVICES

Objectives

> This module completes the configuration of the application server, by adding a comprehensive description of the Java Enterprise services that can be run on top of the application server.
> Each service itself is a core subsystem, which can be included or removed, depending on the kind of applications you are delivering

Objectives

Upon completion of this module, you should be able to:

> Configure the database connectivity
> Configure the enterprise Java Bean container
> Configure the messaging service
> Configure the transaction service

Configuring database connectivity

> In any application server, you can configure database connectivity by adding **DataSource**s to your server configuration.
> Each **DataSource** contains a pool of database connections that are reserved as soon as the server is started up.

Configuring database connectivity

> Applications acquire a database connection from the **DataSource** by looking it up on the **JNDI** tree and then calling **getConnection**():

```
Connection result = null;
try {
  Context initialContext = new InitialContext();
  DataSource datasource = (DataSource)
        initialContext.lookup("java:/MySqlDS");
  result = datasource.getConnection();
}
catch ( Exception ex ) {
  log("Cannot get connection: " + ex);
}
```

Configuring database connectivity

- Once the connection is established, the application should call `connection.close()` as early as possible, which returns the database connection to the pool for other applications to use.
- Basically, any database configuration requires a two-step procedure:
  1. Installing the JDBC driver.
  2. Adding the **`DataSource`** to your configuration.

Installing the JDBC driver

- As we have seen in the previous module, the procedure for installing a new module requires copying the .jar libraries in the appropriate modules path and adding a module.xml file, which declares the module and its dependencies.

Adding a local datasource

- Once the JDBC driver is installed, you need to configure the **DataSource** in the application server.
- In **JBoss AS 7**, you can configure two kind of **DataSource**s:
  1. Local **DataSource**
  2. XA **DataSource**
- A **local datasource** does not support two phase commit using a **java.sql.Driver**.
- An **xa-datasource** supports two phase commit using a **javax.sql.XADataSource**

Adding a local datasource

```
<datasources>
   <datasource jndi-name="java:/MySqlDS"
               pool-name="MySqlDS_Pool"
               enabled="true"
               jta="true"
               use-java-context="true"
               use-ccm="true">
   <connection-url>
      jdbc:mysql://localhost:3306/MyDB
   </connection-url>
   <driver>mysql</driver>
   <pool />
   <security>
      <user-name>root</user-name>
      <password>root</password>
   </security>
```

Adding a local datasource

```
    <statement/>
    <timeout>
       <idle-timeout-minutes>0
       </idle-timeout-minutes>
       <query-timeout>600</query-timeout>
    </timeout>
    </datasource>
    <drivers>
       <driver name="mysql" module="com.mysql"/>
    </drivers>
</datasources>
```

Note

> Since **JBoss AS 7.1**, it's mandatory that the datasource is bound into the `java:/` or `java:jboss/` JNDI namespace.

Configuring the connection pool

- One key aspect of the datasource configuration is the pool section.
- Strictly speaking, in order to use connection pooling, no configuration is required, because without any configuration **JBoss AS** will choose some default settings.
- However, if you want to customize how pooling is done, such as to control the size of the pools and which types of connections are pooled, you would be better learning about its available attributes.

Configuring the connection pool

- An example of pool configuration

```
<pool>
  <min-pool-size>5</min-pool-size>
  <max-pool-size>10</max-pool-size>
  <prefill>true</prefill>
  <use-strict-min>true</use-strict-min>
</pool>
```

Configuring the connection pool

Attribute	Meaning
min-pool-size	The minimum number of connections in the pool (default 0 - zero)
max-pool-size	The maximum number of connections in the pool (default 20)
prefill	Attempt to pre-fill the connection pool to the minimum number of connections
use-strict-min	Whether idle connections below the **min-pool-size** should be closed

Configuring the statement cache

- For each connection in a connection pool in your system, **JBoss AS** Server is able to create a statement cache.
- When a prepared statement or callable statement is used on a connection, **JBoss AS** caches the statement so that it can be reused.
- In order to activate the statement cache, you have to specify a value of **prepared-statement-cache-size** > 0:

```
<statement>
   <track-statements>true</track-statements>
   <prepared-statement-cache-size>10
   </prepared-statement-cache-size>
   <share-prepared-statements/>
</statement>
```

Adding an xa-datasource

- Adding an **xa-datasource** requires some tweaks in the datasource configuration.
- The connection information is now acquired as **xa-datasource** properties.
- The **xa-datasource** class needs to be specified in the driver section.

Adding an xa-datasource

```
<datasources>
  <xa-datasource jndi-name="java:/XAMySqlDS"
                 pool-name="MySqlDS_Pool"
                 enabled="true"
                 use-java-context="true"
                 use-ccm="true">
    <xa-datasource-property name="URL">
      jdbc:mysql://localhost:3306/MyDB
    </xa-datasource-property>
    <xa-datasource-property name="User">root
    </xa-datasource-property>
    <xa-datasource-property name="Password">root
    </xa-datasource-property>
```

Adding an xa-datasource

```
  <driver>mysql-xa</driver>
  </xa-datasource>
  <drivers>
    <driver name="mysql-xa" module="com.mysql">
      <xa-datasource-class>
 com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
      </xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

Configuring a DS Programmatically

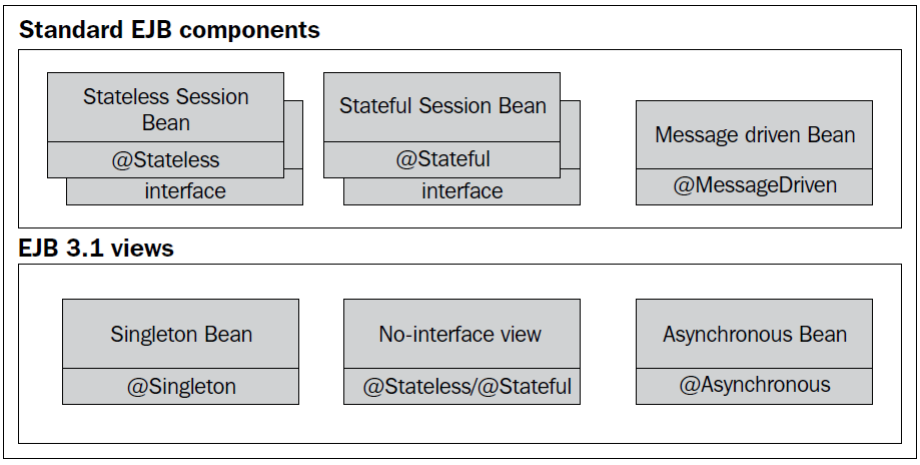
> Java EE 6 supports for configuring a DS programmatically:

```
  @DataSourceDefinition(
    name = "java:/OracleDS",
    className = "oracle.jdbc.OracleDriver",
    portNumber = 1521,
    serverName = "192.168.1.1",
    databaseName = "OracleSID",
    user = "demo",
    password = "fidelio",
    properties = {"createDatabase=create"} )
  @Singleton
  public class DataSourceEJB {
    @Resource(lookup = "java:/OracleDS")
    private DataSource ds;
  }
```

Configuring Enterprise Java Beans

- The **Enterprise Java Bean** (**EJB**) container is a fundamental part of the Java Enterprise architecture.
- The EJB container provides the environment used to host and manage the EJB components deployed in the container.
- The container is responsible for providing a standard set of services, including caching, concurrency, persistence, security, transaction management, and locking services.

Configuring Enterprise Java Beans



Configuring Enterprise Java Beans

With **JBoss AS 7**, you are able to deploy

- **Stateless session beans (SLSB)**
  - SLSB are objects whose instances have no conversational state.
  - All bean instances are equivalent when they are not servicing a client.
- **Stateful session beans (SFSB)**
  - SFSB support conversational services with tightly coupled clients.
  - A stateful session bean accomplishes a task for a particular client.
  - It maintains state for the duration of a client session.
  - After session completion, state is not retained.

Configuring Enterprise Java Beans

With **JBoss AS 7**, you are able to deploy

- **Message-driven beans (MDB)**
  - MDB are a kind of Enterprise Beans that are able to asynchronously process messages sent by any JMS producer.

Configuring Enterprise Java Beans

Besides standard EJB components, AS also supports the new EJB 3.1 variants introduced by Java EE 6

- **Singleton EJB**
  - This is essentially similar to a stateless session bean
  - It uses a single instance to serve the client requests
  - You are guaranteed to use the same instance across invocations.
  - Singletons can use a richer lifecycle set of events and a stricter locking policy to control concurrent access to the instance.
  - Used to hold cached data

Configuring Enterprise Java Beans

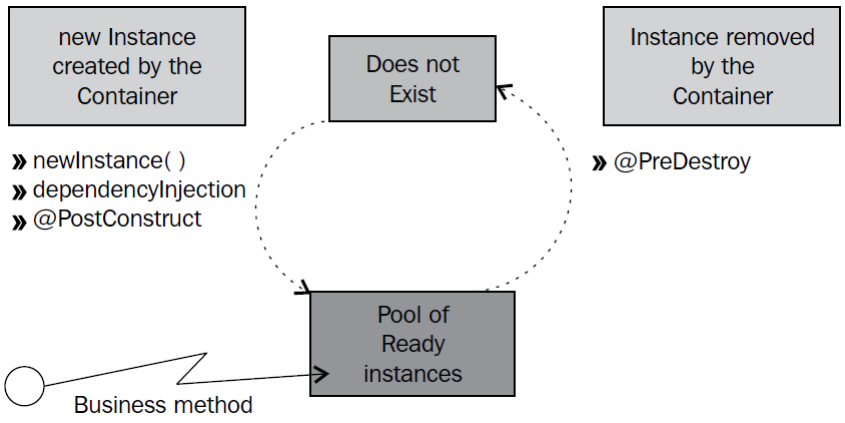
- **No-interface EJB**
  - This is just another view of the standard session bean except that local clients do not require a separate interface
  - All public methods of the bean class are automatically exposed to the caller
- **Asynchronous EJB**
  - These are able to process client requests asynchronously just like MDBs, except that they expose a typed interface and follow a more complex approach for processing client requests

Configuring the EJB components

> We will study the following components:

1. The SLSB configuration
2. The SFSB configuration
3. The MDB configuration
4. The Timer service configuration

Stateless Session Bean Life cycle



Configuring Stateless Session Beans

- No Stateless Session Bean instances exist in JBoss AS at start-up time
- As individual beans are invoked, the EJB container initializes new SLSB instances.
- These instances are maintained in a free pool, which will be used to service EJB method calls.
- The EJB remains active for the duration of the client's method call. After the method completes, the EJB instance is returned to the free pool.
- Because the EJB container unbinds Stateless Session Beans from clients after each method call, the actual bean class instance that a client uses may be different from invocation to invocation.

Configuring Stateless Session Beans

- If all instances of an EJB class are active and the pool's **`max-pool-size`** has been reached, new clients requesting the EJB class will be blocked until an active EJB completes a method call.
- Depending on how you have configured your stateless pool, an acquisition timeout can be triggered, if you are not able to acquire an instance from the pool within a maximum time.
- EJB are part of the ejb3.1.2 subsystems
- In order to configure your pool, you can operate on two parameters:
  - **`max-pool-size`**
  - **`instance-acquisition-timeout`**

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <session-bean>
    <stateless>
       <bean-instance-pool-ref
              pool-name="slsb-strict-max-pool"/>
    </stateless>
     . . .
  </session-bean>
  <pools>
    <bean-instance-pools>
      <strict-max-pool
 name="slsb-strict-max-pool" max-pool-size="25"
        instance-acquisition-timeout="5"
    instance-acquisitiontimeout-unit="MINUTES"/>
    </bean-instance-pools>
  </pools>
  . . .
</subsystem>
```

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <session-bean>
    <stateless>
<bean-instance-pool-ref pool-name="large-pool"/>
    </stateless>
</session-bean>
  <pools>
   <bean-instance-pools>
<strict-max-pool name="large-pool" max-pool-
size="100" instance-acquisition-timeout="5"
    instance-acquisitiontimeout-unit="MINUTES"/>
      <strict-max-pool
 name="slsb-strict-max-pool" max-pool-size="25"
        instance-acquisition-timeout="5"
    instance-acquisitiontimeout-unit="MINUTES"/>
    </bean-instance-pools>
  </pools>
</subsystem>
```

Using CLI to configure the Stateless pool size

> Add a new pool named large-pool to your EJB 3 sub-system:

```
/subsystem=ejb3/strict-max-bean-instance-pool=large-pool:add(max-pool-size=100)
```

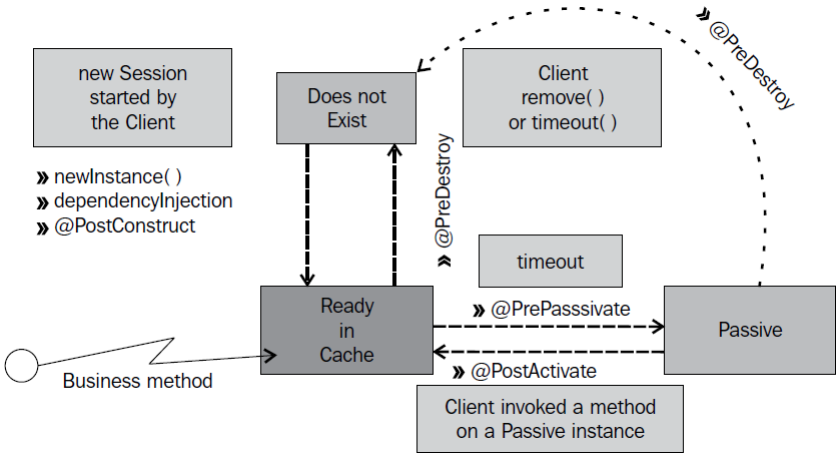
> Set this pool as the default used by the EJB container:

```
/subsystem=ejb3:write-attribute(name=default-slsb-instance-pool,value=large-pool)
```

> Change the pool size property by operating on the **max-pool-size** attribute:

```
/subsystem=ejb3/strict-max-bean-instance-pool=large-pool:write-attribute(name="max-pool-size",value=50)
```

Stateful Session Bean Life cycle



Configuring Stateful session beans

- SFSB are bound to particular client instances: the application server uses a cache to store active EJB instances in memory, so that they are immediately available for client requests.
- The cache contains EJBs that are currently in use by a client and instances that were recently in use.
- Having EJBs in memory is a costly operation, so you should move as soon as possible out of it, either by passivating them or removing them.
- **Passivation** is a process by which the EJB container ensures that idle SFSB instances are freed from the cache, but save their state on disk.

Configuring Stateful session beans

- Removing a bean from the cache, on the other hand, is a process, which can be either triggered by the user or by the EJB container.
- When triggered by the user, it is accomplished by adding a `@javax.ejb.Remove` annotation on a method of your EJBs, which will actually remove the EJB when invoked:

```
@Remove
public void remove() {}
```

Configuring Stateful session beans

> The configuration tweaks available for the stateful session beans include just the access timeout for the stateful instances.

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <session-bean>
  . . .
    <stateful default-access-timeout="5000"/>
  </session-bean>
  . . .
</subsystem>
```

Configuring Stateful session beans

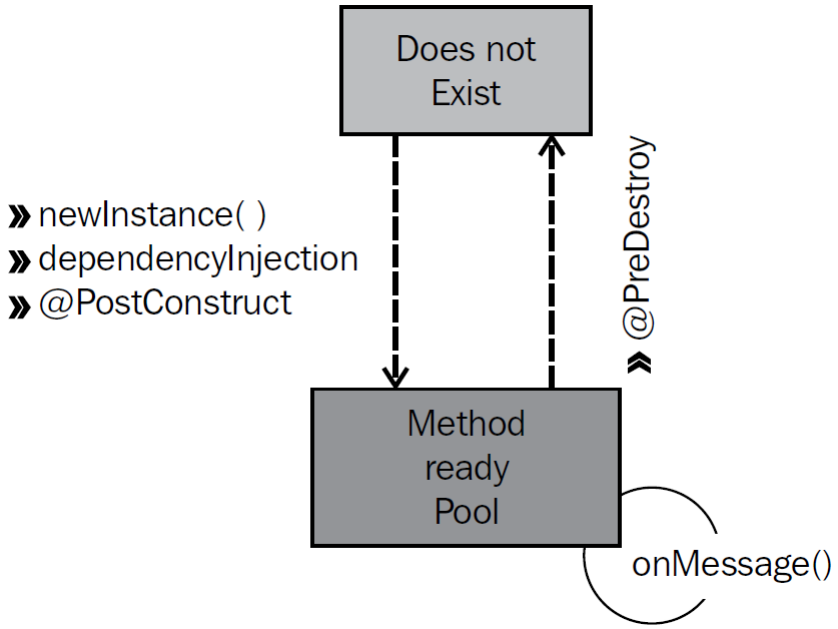
> Jboss 7.1 enables cache definitions

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <session-bean>
    <stateful default-access-timeout="5000"
              cache-ref= "passivating"
              clustered-cache-ref="clustered"/>
  </session-bean>
  <caches>
    <cache name="simple"/>
    <cache name="passivating"
           passivation-store-ref="file"/>
    <cache name="clustered"
           passivation-store-ref="infinispan"/>
  </caches>
```

Configuring Stateful session beans

```
  <passivation-stores>
    <file-passivation-store name="file"
                            max-size="500" />
    <cluster-passivation-store name="infinispan"
                           backing-cache= "sfsb"
                           max-size="500"/>
  </passivation-stores>
  . . .
</subsystem>
```

Message Driven Bean Life cycle



Configuring the message driven beans

- **Message-driven beans** (**MDBs**) are stateless, server-side, transaction-aware components for processing asynchronous JMS messages.
- When a message is received, the EJB container checks to see if any MDB instance is available in the pool.
- If a bean is available in the free pool, JBoss uses that instance.
- After an MDB instance's onMessage() method returns, the request is complete, and the instance it is placed back in the free pool.
- This results in the best response time, as the request is served without waiting for a new instance to be created.

Configuring the message driven beans

- If no bean instances are handy, the container checks if there is room for more MDB in the pool by comparing the MDB's **`MaxSize`** attribute with the pool size.
- If **`MaxSize`** still has not been reached, a new MDB is initialized.
  - The creation sequence is just the same as stateless bean.
  - Failure to create a new instance, on the other hand, will imply that the request will be blocked until an active MDB completes.
  - If the request cannot acquire an instance from the pool within the **`instance-acquisition-timeout`**, an exception is thrown

Configuring the message driven beans

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <mdb>
    <resource-adapter-ref
         resource-adapter-name="hornetq-ra"/>
    <bean-instance-pool-ref
              pool-name="mdb-strict-max-pool"/>
  </mdb>
  <pools>
    <bean-instance-pools>
     <strict-max-pool name="mdb-strict-max-pool"
                      max-pool-size="20"
                instance-acquisition-timeout="5"
   instance-acquisition-timeout-unit="MINUTES"/>
    </bean-instance-pools>
  </pools>
</subsystem>
```

The Timer Service

- Some Java EE applications need to schedule tasks in order to get notified at certain times.
- The EJB timer service is a container service that allows EJBs to be registered for callback invocation.
- EJB notifications may be scheduled to occur according to a *calendar-based schedule*, *at a specific time*, *after a specific elapsed duration*, or *at specific recurring intervals*.
- The container keeps a record of all the timers, and invokes the appropriate bean instance method when a timer has expired
- Timers are intended for long-lived business processes and are by default persistent. This means they survive server shutdowns, and, once the server starts again, the timers are executed as if no shutdown had happened.

Calendar-Based Expression

> The timer service uses a calendar-based syntax that takes its roots from the Unix cron utility

```
@Stateless
public class StatisticsEJB {
    @Schedule(dayOfMonth="1",hour="5",minute="30")
    public void statisticsItemsSold() {
       // ...
     }
    @Schedules({
        @Schedule(hour = "2"),
        @Schedule(hour = "14", dayOfWeek = "Wed")
    })
    public void generateReport() {
    // ...
    }
```

Programmatic Timer Creation

```
@Stateless
public class CustomerEJB {
    @Resource
    TimerService timerService;
    public void createCustomer(Customer customer) {
      em.persist(customer);
      ScheduleExpression birthDay = new ScheduleExpression()
                    .dayOfMonth(customer.getBirthDay())
                    .month(customer.getBirthMonth());
       timerService.createCalendarTimer(birthDay,
                              new TimerConfig(customer, true));
    }
    @Timeout
    public void sendBirthdayEmail(Timer timer) {
        Customer customer = (Customer) timer.getInfo();
        // ...
    }
```

Configuring the Timer service

> You can define the data-store path, that is temporary file system path where planned executions are stored and the amount of threads to reserve to the timer service with the thread-pool-name attribute:

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <timer-service thread-pool-name="default">
    <data-store path="timer-service-data"
          relative-to="jboss.server.data.dir"/>
  </timer-service>
  <thread-pools>
    <thread-pool name="default"
       max-threads="10" keepalive-time="100"/>
  </thread-pools>
</subsystem>
```

Configuring the messaging system

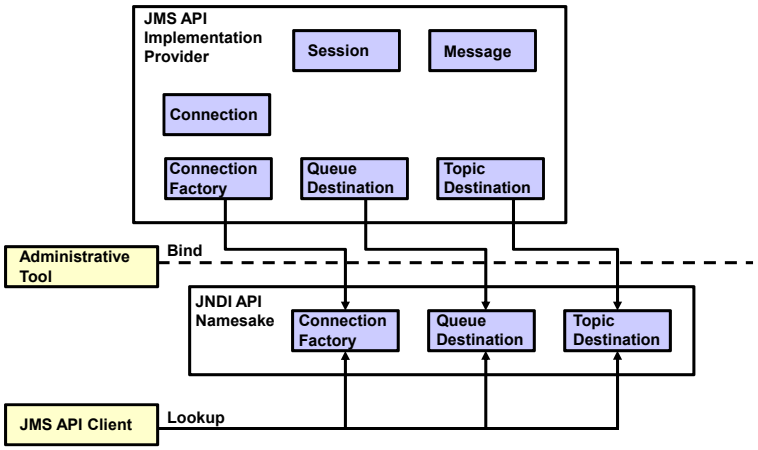
> Message oriented middleware has been an integral part of the application server, since the first releases of the application server.

> Messaging systems allow you to loosely couple heterogeneous systems together, whilst typically providing reliability, transactions, and many other features.

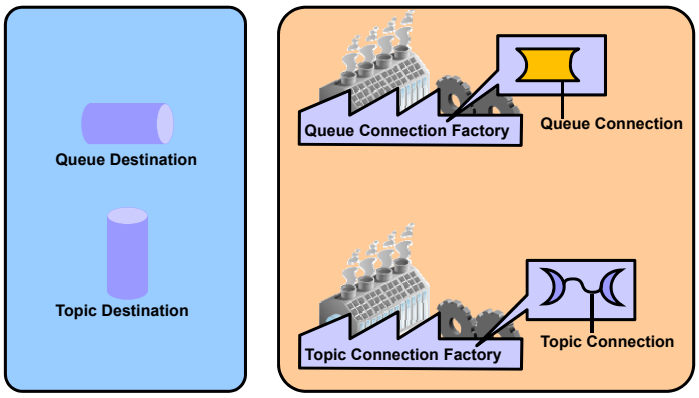
> In JBoss 7.0.2, the messaging subsystem is included in a separate configuration file named **`standalone-preview.xml`**.

> Starting from the release 7.1.0 of the application server, the messaging sub-system will be included in the configuration file named **`standalone-full.xml`**.

Messaging System Participants

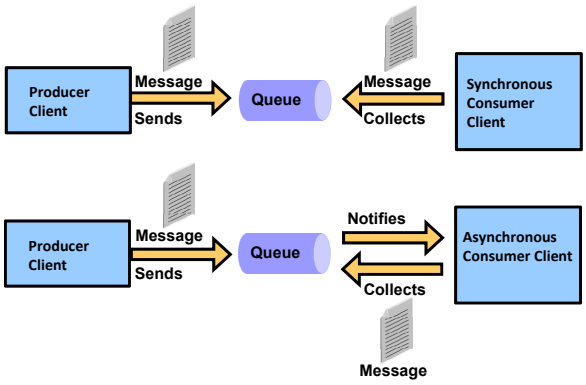


Administered Objects

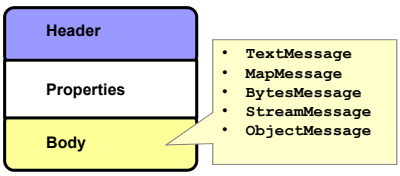


Messaging Clients

> The synchronous client blocks communication until a message arrives.

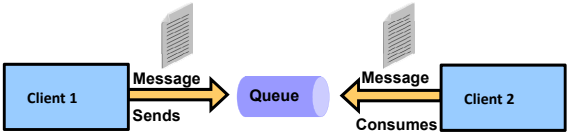


Messages

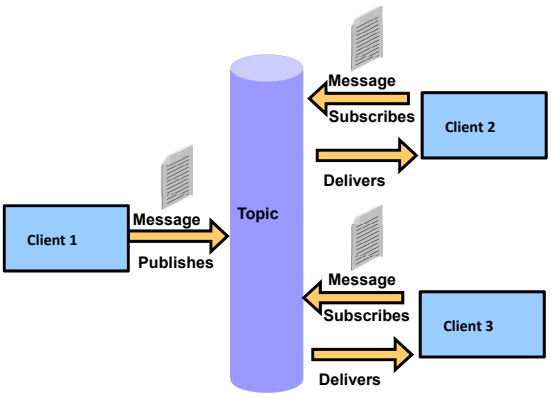


Message Type	Contents of the Message Body
`TextMessage`	A `java.lang.String` object
`MapMessage`	A set of name–value pairs, for example, a hash table
`BytesMessage`	A stream of uninterpreted bytes or binary data
`StreamMessage`	A stream of Java technology primitive values filled and read sequentially
`ObjectMessage`	A serializable Java technology object

Point-to-Point Messaging Architecture



Publish/Subscribe Messaging Architecture



Comparison

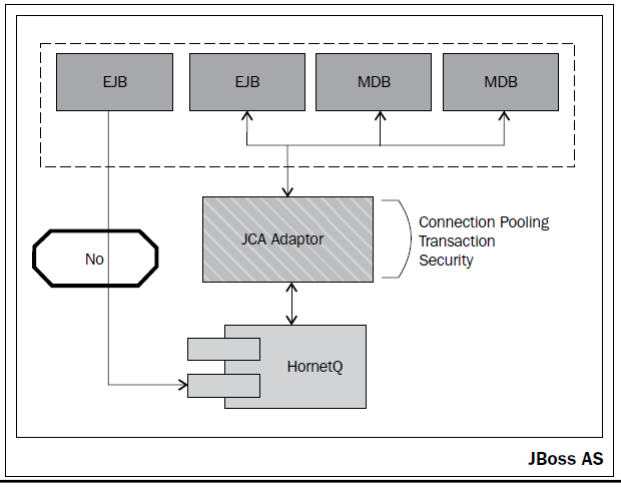
Point-to-point messaging	Publish/Subscribe
Only one consumer gets the message.	Multiple consumers (or none) will receive the message.
The producer does not have to be running at the time the consumer consumes the message, nor does the consumer need to be running at the time the message is sent.	The publisher has to create a message topic for clients to subscribe. The subscriber has to remain continuously active to receive messages, unless it has established a durable subscription. In that case, messages published while the subscriber is not connected will be redistributed whenever it reconnects.
Every message successfully processed is acknowledged by the consumer.	

HornetQ

- JBoss AS has used different JMS implementations across its releases.
- Since release 6.0, the default JMS provider is HornetQ (http://www.jboss.org/hornetq)
- It provides a multi-protocol, embeddable, very high performance, clustered, asynchronous messaging system.
- At its core, HornetQ is designed simply as set of **Plain Old Java Objects** (**POJOs**), with few dependencies on external jars.
- The only one jar dependency is the Netty library which leverages the Java **New Input-Output** (**NIO**) API for building high-performance network applications.

HornetQ

> A key part of the HornetQ integration is the **JCA Adaptor** that handles the communication between the application server and the HornetQ Server



Configuring the transport

> Configuring the transport of JMS Message is a key part of the messaging system tuning.

> Out of the box, **`HornetQ`** currently uses **`Netty`** as its high-performance low-level network library.

> **`Netty`** is a NIO client server framework, which enables quick and easy development of network applications, such as **protocol servers** and **clients**.

> It greatly simplifies and streamlines network programming, such as TCP and UDP socket server.

> One of the most important concepts in **`HornetQ`** transports is the definition of ***acceptors*** and ***connectors***.

Configuring the transport

> An **acceptor** defines which type of connection is accepted by the **HornetQ** server.
> A **connector** defines how to connect to a **HornetQ** server.
  – The connector is used by a **HornetQ** client.

Configuring the transport

> **HornetQ** defines two types of acceptor/connector:
  – **invm**

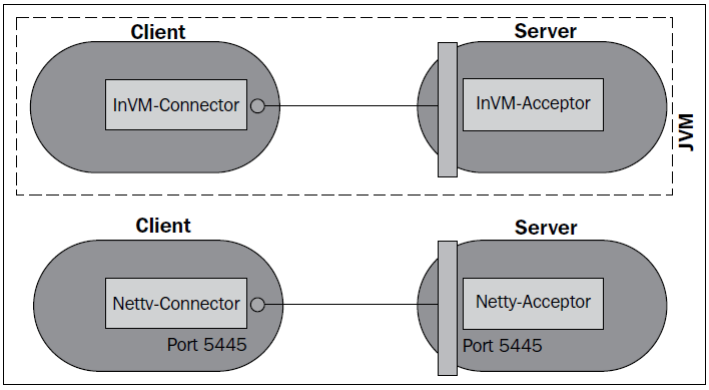
    This type can be used when both **HornetQ** client and server run in the same **Virtual Machine** (**invm for Intra Virtual Machine**)
  – **netty**

    This type must be used when **HornetQ** client and server run in different virtual machines (this connector type uses the netty project to handle the IO)

Configuring the transport

> To communicate, a HornetQ client must use a connector compatible with the server's acceptor.

> A compatible client-server communication requires that it is carried on using the same type of acceptor/connector



Configuring the transport

> JBoss AS 7 comes with a preconfigured acceptor/connector pair:

```
<acceptors>
   <in-vm-acceptor name="in-vm" server-id="0"/>
   <netty-acceptor name="netty" socket-binding="messaging"/>
   <netty-acceptor name="netty-throughput"
                   socket-binding="messaging-throughput">
      <param key="batch-delay" value="50"/>
      <param key="direct-deliver" value="false"/>
   </netty-acceptor>
</acceptors>
<connectors>
   <in-vm-connector name="in-vm" server-id="0"/>
   <netty-connector name="netty" socket-binding="messaging"/>
   <netty-connector name="netty-throughput"
                    socket-binding="messaging-throughput">
       <param key="batch-delay" value="50"/>
   </netty-connector>
</connectors>
```

Configuring the transport

Parameter	Description
`use-nio`	If this is **`true`**, then Java non-blocking NIO will be used. If set to **`false`**, then old blocking Java IO will be used. Default is **`true`**.
`host`	This specifies the host name or IP address to connect to or to listen on. The default value for this property is localhost. Multiple hosts or IP addresses can be specified by separating them with commas.
`port`	This specifies the port to connect to (when configuring a connector) or to listen on (when configuring an acceptor). The default value for this property is **`5445`**.
`tcp-no-delay`	If this is **`true`**, then Nagle's algorithm will be disabled. The default value for this property is **`true`**.

Configuring the transport

Parameter	Description
`tcp-send-buffer-size`	This parameter determines the size of the TCP send buffer in bytes. The default value for this property is **`32768`** bytes.
`tcp-receive-buffer-size`	This parameter determines the size of the TCP receive buffer in bytes. The default value for this property is **`32768`** bytes.
`batch-delay`	This parameter lets you configure **`HornetQ`**, so that messages are batched up writes for a maximum of batch-delay milliseconds, before sending them to the transport. This can increase overall throughput for very small messages.
`direct-deliver`	If this is **`true`**, then Nagle's algorithm will be disabled. The default value for this property is **`true`**.

Configuring connection factories

> The definition of connection-factory instances is included in the default server configuration:

```
<connection-factory name="InVmConnectionFactory">
   <connectors>
      <connector-ref connector-name="in-vm"
                     backup-connector-name="netty"/>
   </connectors>
   <entries>
      <entry name="java:/ConnectionFactory"/>
   </entries>
</connection-factory>
<connection-factory name="RemoteConnectionFactory">
   <connectors>
      <connector-ref connector-name="netty"
                     backup-connector-name="in-vm"/>
   </connectors>
   <entries>
      <entry name="RemoteConnectionFactory"/>
   </entries>
</connection-factory>
```

Configuring JMS destinations

> Along with the definition of connection factories in the JMS subsystem, you can find the JMS destinations (**Queues** and Topics):

```
   </jms-destinations>
      <jms-queue name="testQueue">
         <entry name="queue/test"/>
      </jms-queue>
      <jms-topic name="testTopic">
         <entry name="topic/test"/>
      </jms-topic>
   </jms-destinations>
</hornetq-server>
```

Configuring JMS destinations

> To configure a Message Driven Bean component to consume messages from the **"queue/test" Queue**

```
@MessageDriven(name = "MessageMDBSample",
 activationConfig = {
   @ActivationConfigProperty(propertyName = "destinationType",
                             propertyValue = "javax.jms.Queue"),
   @ActivationConfigProperty(propertyName = "destination",
                             propertyValue = "queue/test"),
   @ActivationConfigProperty(propertyName="acknowledgeMode",
                            propertyValue="Auto-acknowledge") })
 public class SampleMDBean implements MessageListener {
   @Resource
   private MessageDrivenContext context;
 }
```

MODULE 6

JBOSS EAP: WEB SERVER CONFIGURATION

Purpose

- In this module, we will complete the standalone server configuration overview by looking at the Web server configuration.
- JBoss AS ships with an enterprise-ready Web server, designed for medium and large applications, based on the Jakarta Tomcat.
- At first, we will learn how to configure static and dynamic resources on JBoss Web server.
- We will then explore the structure of a web application by creating, packaging, and deploying a sample AS 7 project

Configuring the Web container

- JBoss AS has always been bound to the popular Apache Tomcat engine for rendering web applications
- The new release of JBoss Web is actually based on a fork of Apache Tomcat 7.0, which implements the Servlet 3.0, the JavaServer Pages 2.2 specifications from the Java Community Process, and includes a Java Server Faces 2.1 compliant implementation.
- On the top of the Web container, additional frameworks can be run, such as the Web service engine, which is based on the Apache CXF framework and helps you build and develop services using frontend programming APIs, such as JAX-WS and JAX-RS.

Where is the Web server configuration?

- A standard Tomcat Web server uses a file named **`server.xml`** to define the configuration of its services.
- JBoss AS 7 configuration is centralized into a single file named standalone.xml. So the Web server configuration is now also contained in this file.

```
<subsystem xmlns="urn:jboss:domain:web:1.1"
           default-virtual-server="default-host">
  <connector name="http"
             protocol="HTTP/1.1"
             scheme="http" socket-binding="http"/>
  <virtual-server name="default-host"
                  enable-welcome-root="true">
      <alias name="localhost"/>
      <alias name="example.com"/>
   </virtual-server>
</subsystem>
<socket-binding-group name="standard-sockets"
                      default-interface="public">
   <socket-binding name="http" port="8080"/>
   <socket-binding name="https" port="8443"/>
. . .
</socket-binding-group>
```

Configuring the Web server connectors

- A connector is basically a Java object
- It provides an interface to Web server clients.
- The default configuration provides just a quintessential connector configuration, leaving the remaining attributes to default values.
  - It could be just fine for simple applications.
- If you want to configure applications that require a minimum of complexity, you should configure by yourself the connector attributes with the most appropriate values.

Configuring the Web server connectors

- The following is a connector configuration that shows off all the available connector attributes:

```
<subsystem xmlns="urn:jboss:domain:web:1.1">
  <connector enable-lookups="false"
   enabled="true" executor="http-executor"
   max-connections="200" max-post-size="2048"
   max-save-post-size="4096" name="http"
   protocol="HTTP/1.1" proxy-name="proxy"
   proxy-port="8081" redirect-port="8443"
   scheme="http"    secure="false"
   socket-binding="http" />
</subsystem>
```

Configuring the Web server connectors

> One important aspect that refers to the Web server Thread Pool is the executor attribute

```
<subsystem xmlns="urn:jboss:domain:threads:1.0">
  <bounded-queue-thread-pool
            name="http-executor"
            blocking="true">
     <core-threads count="10" per-cpu="20"/>
     <queue-length count="10" per-cpu="20"/>
     <max-threads count="10" per-cpu="20"/>
     <keepalive-time time="10" unit="seconds"/>
  </bounded-queue-thread-pool>
</subsystem>
```

Configuring the Web server connectors

> All the other connector settings refer to the equivalent properties used by Tomcat/earlier JBoss Web server

Parameter	Description
enable-lookups	Set to true, if you want calls to request. getRemoteHost() to perform DNS lookups, in order to return the actual hostname of the remote client. Set to false to skip the DNS lookup and return the IP address in String form instead.
enabled	Determines if the connector is enabled (default) or not.
executor	The executor represents a Thread Pool that can be shared between components in the JBoss Web server.

Parameter	Description
max-connections	The maximum number of request processing Threads to be created by this connector, which therefore determines the maximum number of simultaneous requests that can be handled.
max-post-size	Sets a limit on the maximum size of HTTP POST requests it accepts. By default, it's 2 MB.
max-save-post-size	Maximum size of a POST that will be saved by the container during authentication, which is 4KB by default.
name	The name of the connector
protocol	This attribute value must be HTTP/1.1 to use the HTTP handler, which is the default. When set to AJP/1.3, the Apache JServ Protocol (AJP) will be used.

Parameter	Description
proxy-name	The server name to which we require our requests to this connector were proxied.
proxy-port	The server port to which we require our requests to this connector were proxied.
redirect-port	If this connector is supporting non-SSL requests and a request is received for which a matching <security-constraint> requires SSL transport, Catalina will automatically redirect the request to the port number specified here.
scheme	Set this attribute to the name of the protocol you wish to have returned by calls to request.getScheme(). For example, you would set this attribute to https for an SSL connector. The default value is http

Configuring the HTTP connector

- Out of the box, JBoss Web server uses the Java-based HTTP/1.1 (**Coyote**) connector for parsing HTTP requests and passes the request to the Web server engine.
- The Java-based Coyote connector is a mature and extremely stable component and should be the choice for most situations, especially if you are just beginning to use JBoss AS.
- On highly loaded systems, however, you can opt for the newer **APR** HTTP connector, which potentially offers the highest level of performance improvement, by leveraging proven operating system–level optimization on it.

Configuring the HTTP connector

- The APR connector optimizes performance and enhances scalability through
  - Use of a **`sendfile`** kernel mode call to send large static files directly from the buffer cache
  - Use of a single native code keep alive poller, to implement connection keep alive for a large number of connections

Installing the APR connector

- JBoss AS is a pure Java application server and so the APR connector does not ship with the standard distribution.
- When you start the application server, you might have noticed the following log message:

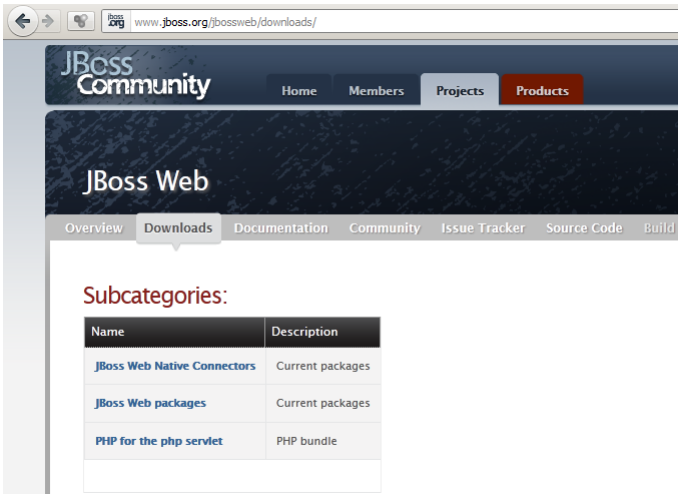
```
15:50:39,187 INFO [org.apache.catalina.core.AprLifecycleListener] (MSC service Thread 1-2) The Apache Tomcat Native library, which allows optimal performance in production environments, was not found on the java.library.path
```

- **`AprLifecycleListener`** tries to find the APR libraries at startup. Since we still haven't added them to the server installation, the Web server will fall back to the standard HTTP/1.1 connector.

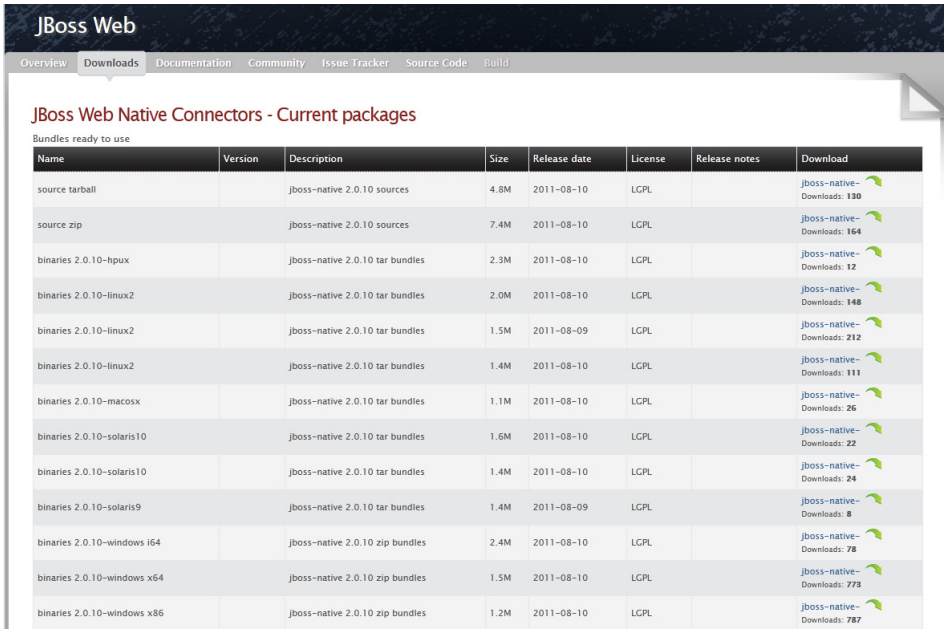
Installing the APR connector

- Download the **`JBoss`** Native connector from:

  http://www.jboss.org/jbossweb/downloads



Installing the APR connector



Name	Version	Description	Size	Release date	License	Release notes	Download
source tarball		jboss-native 2.0.10 sources	4.8M	2011-08-10	LGPL		jboss-native- Downloads: 130
source zip		jboss-native 2.0.10 sources	7.4M	2011-08-10	LGPL		jboss-native- Downloads: 164
binaries 2.0.10-hpux		jboss-native 2.0.10 tar bundles	2.3M	2011-08-10	LGPL		jboss-native- Downloads: 12
binaries 2.0.10-linux2		jboss-native 2.0.10 tar bundles	2.0M	2011-08-10	LGPL		jboss-native- Downloads: 148
binaries 2.0.10-linux2		jboss-native 2.0.10 tar bundles	1.5M	2011-08-09	LGPL		jboss-native- Downloads: 212
binaries 2.0.10-linux2		jboss-native 2.0.10 tar bundles	1.4M	2011-08-10	LGPL		jboss-native- Downloads: 111
binaries 2.0.10-macosx		jboss-native 2.0.10 tar bundles	1.1M	2011-08-10	LGPL		jboss-native- Downloads: 26
binaries 2.0.10-solaris10		jboss-native 2.0.10 tar bundles	1.6M	2011-08-10	LGPL		jboss-native- Downloads: 22
binaries 2.0.10-solaris10		jboss-native 2.0.10 tar bundles	1.4M	2011-08-10	LGPL		jboss-native- Downloads: 24
binaries 2.0.10-solaris9		jboss-native 2.0.10 tar bundles	1.4M	2011-08-09	LGPL		jboss-native- Downloads: 8
binaries 2.0.10-windows i64		jboss-native 2.0.10 zip bundles	2.4M	2011-08-10	LGPL		jboss-native- Downloads: 78
binaries 2.0.10-windows x64		jboss-native 2.0.10 zip bundles	1.5M	2011-08-10	LGPL		jboss-native- Downloads: 773
binaries 2.0.10-windows x86		jboss-native 2.0.10 zip bundles	1.2M	2011-08-10	LGPL		jboss-native- Downloads: 787

Installing the APR connector

- Once downloaded, unzip the **`JBoss Native`** into the home folder of your **`JBoss AS`** distribution.
- Add the following line to standalone.conf.bat

```
set "JAVA_OPTS=%JAVA_OPTS%
  -Djava.library.path=%JBOSS_HOME%/bin/native"
```

- Now, restart **`JBoss AS`**, which will now detect the presence of the native libraries and leverage them:

```
12:37:46,762 INFO  [org.jboss.as.connector] (MSC service thread 1-6) Starting JC
A Subsystem (JBoss IronJacamar 1.0.3.Final)
12:37:46,827 INFO  [org.jboss.as.connector.subsystems.datasources] (MSC service
thread 1-5) Bound data source [java:jboss/datasources/ExampleDS]
12:37:46,951 INFO  [org.apache.coyote.http11.Http11AprProtocol] (MSC service thr
ead 1-4) Starting Coyote HTTP/1.1 on http--127.0.0.1-8080
12:37:47,017 INFO  [org.jboss.as.deployment] (MSC service thread 1-3) Started Fi
leSystemDeploymentService for directory c:\opt32\jboss-as-web-7.0.2.Final\standa
lone\deployments
12:37:47,055 INFO  [org.jboss.as] (Controller Boot Thread) JBoss AS 7.0.2.Final
"Arc" started in 4023ms - Started 93 of 148 services (55 services are passive or
 on-demand)
```

Configuring the Web server resources

- We have covered the standard Web server configuration
- Now we will look at the additional configuration options that are available, but not explicitly shown in the `standalone.xml` configuration file.
- The Web server configuration section is divided into two broad areas:
- **Static resources configuration**

  It includes resources, such as HTML pages and images
- **JSP configuration**

  It is used for pages that are dynamically generated by the Web container

Configuring static resources

- Static resources include that part of the Web application such as HTML pages and images
- They do not require processing by the Web container
- They are just rendered to the end user.
- For large applications requiring the highest throughput, we suggest you configure Apache as dispatcher for static resources.
- For smaller projects, using JBoss for this purpose might be just as good

Configuring static resources

> A sample static resources configuration which is based on the default values

```
<subsystem xmlns="urn:jboss:domain:web:1.1"
           default-virtual-server="default-host">
  <configuration>
       <static-resources listings="false"
            sendfile="49152"
            file-encoding="utf-8"
            read-only="true" webdav="false"
            max-depth= "3" disabled="false"/>
  </configuration>
  . . .
</subsystem>
```

Configuring static resources

Property	Meaning
disabled	Setting to true disables static resources handling by connectors.
file-encoding	File encoding to be used when reading static resources. Defaults to platform file encoding.
listings	When set to true, and no welcome file is present, a directory listing is shown.
max-depth	Defines the max-depth for a WebDAV request.
read-only	When set to true, HTTP commands, such as PUT and DELETE, are rejected

Configuring static resources

Property	Meaning
`sendfile`	When sendfile is enabled, set the sendfile buffer size. Default is 49152. Higher values will cause sendfile to increase its buffer instead of sending large response using blocking writes. This may or may not increase the performance of your data transfer, depending on your network configurations.
`webdav`	This parameter lets you configure your JBoss AS as a WebDAV server (allows the users to easily upload/download documents to/from your web application)

Configuring dynamic resources

- By default, there's no **`jsp-configuration`** element.
- If you want to include it, you need to add it as a child element of the configuration section, just like for the static resources

```
<configuration>
   <jsp-configuration
      check-interval="1"
      development="false"
      disabled="false"
      display-source-fragment="true"
      dump-smap="false"
      error-on-use-bean-invalid-class-attribute="false"
      generate-strings-as-char-arrays="false"
      java-encoding="UTF8"
      keep-generated="true"
      mapped-file="true"
      modification-test-interval="4"
      recompile-on-fail="false"
      scratch-dir=""
      smap="true"
      source-vm="1.5"
      tag-pooling="true"
      target-vm="1.5"
      trim-spaces="false"
       x-powered-by="true"/>
</configuration>
```

Property	Meaning
check-interval	If development is enabled, this setting decides how frequently the compiles are triggered. This defaults to 300 seconds.
development	Setting this to false disables JBoss AS from checking the JSP pages for modification.
disabled	If set to true, this parameter disables the JSP configuration settings
display-source-fragment	If this parameter is set to true, a source fragment is included when displaying exception messages.
dump-smap	If this parameter is set to true, the SMAP info (source files are mapped to class files -JSR 45) will be logged to a file.
error-on-use-bean-invalid-class-attribute	When set to true, the Jasper compiler performs a stricter check on JSP by issuing a compilation error if there's an error in the useBean action.

Property	Meaning
generate-strings-as-char-arrays	This is a minor performance optimization, which causes the compiler to generate slightly more efficient char arrays when set to true.
java-encoding	Setting this to false disables JBoss AS from checking the JSP pages for modification.
keep-generated	When set to true (default), the generated source files for JSP files are not deleted.
mapped-file	When set to true, JBoss Web server generates static content with one print statement per-input-line, to ease debugging.
modification-test-interval	This sets the frequency for checking the modification of JSP files. This defaults to 4 seconds, and setting it to 0 causes the JSP to be checked on every access. Needs the development element to be true to work.

Property	Meaning
recompile-on-fail	When set to true, the Web server ignores the modificationTestInterval if there's a compilation error. Used in development mode only and is disabled by default, as compilation may be expensive.
scratch-dir	If set, this parameter determines the scratch directory that we should use when compiling JSP pages.
smap	If this parameter is set to true, the generation of SMAP info for JSR45 debugging is enabled. Otherwise it's suppressed.
source-vm	Selects which JDK versions the source files are compatible with (default value: 1.5).
target-vm	Selects which JDK version the generated files are compatible with (default value: 1.5).
tag-pooling	The enablePooling attribute specifies if pooling of Tag library classes is to be enabled (true) or not (false). This parameter defaults to true.
trim-spaces	To remove useless white space bytes from the response, set this to true.

MODULE 7

CONFIGURING A JBOSS EAP 7 DOMAIN

Purpose

These are the topics we are going to cover in this module:

> Introduction to JBoss AS domain

> How to configure the domain components

> What is the criteria for choosing a domain or a standalone

Introducing the JBoss AS domain

From the process point of view, a domain is made up of four elements:

1. Domain controller
   - It is the management control point of your domain.
   - An AS instance running in domain mode will have at most one process instance acting as a domain controller.
   - The domain controller holds a centralized configuration, which is shared by the node instances belonging to the domain.

Introducing the JBoss AS domain

2. Host controller
   - Is responsible for
     - coordinating with a domain controller the lifecycle of server processes
     - the distribution of deployments, from the domain controller to the server instances.

Introducing the JBoss AS domain

3. Process controller
   - It's a very lightweight process
   - Primary function is to spawn server processes and host controller processes, and manage their input/output streams.
   - It also allows the host controller to be patched and restarted without impacting the associated servers.

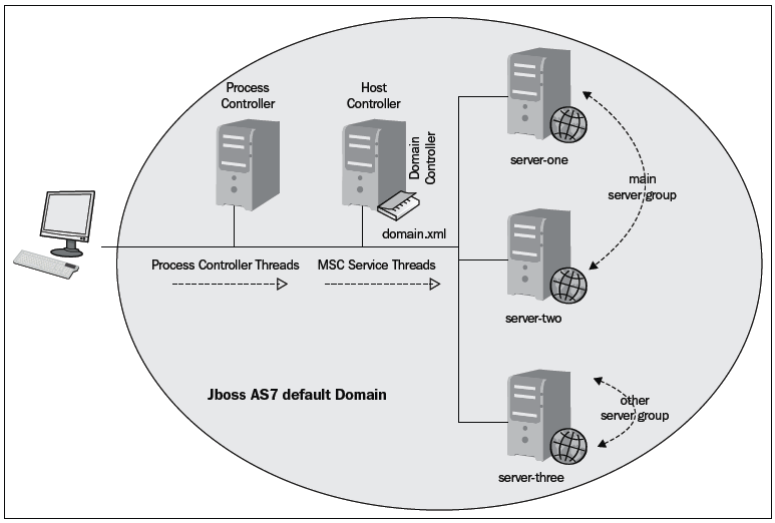
Introducing the JBoss AS domain

4. Application server nodes
   - They are regular Java processes that map to instances of the application server.
   - Each server node, in turn, belongs to a domain group.
   - Domain groups are explained in detail when we discuss the domain configuration file.

Understanding the default domain configuration

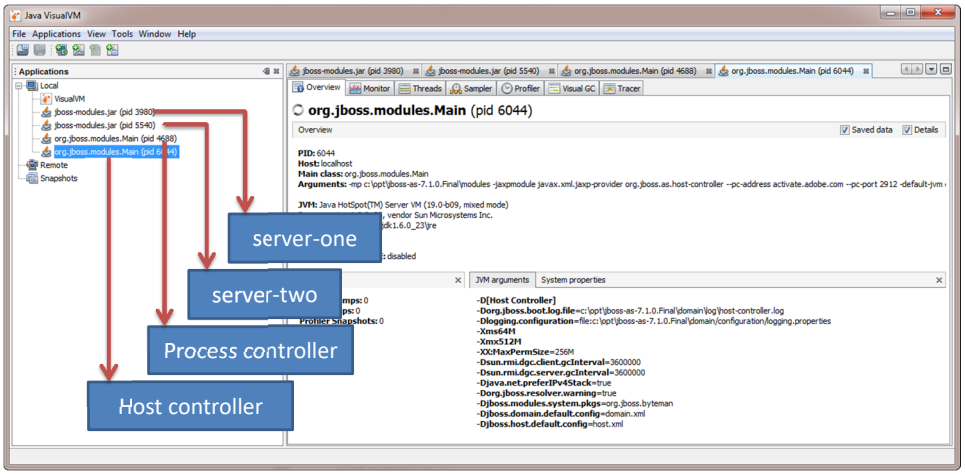
> Out of the box, the domain configuration includes a basic configuration made up of the following elements:
  - One process controller that starts the other JVM processes
  - One host controller that acts as domain controller.
  - Three server nodes. The first two are part of the main server group, and the third one (inactive) is part of the other server group.

Understanding the default domain configuration



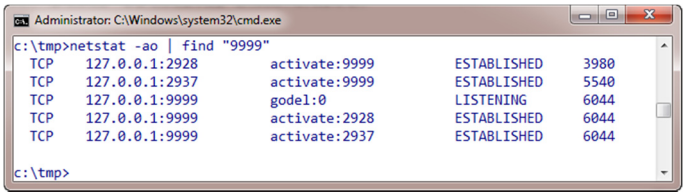
The default domain configuration

> If you want to see this configuration from the JVM point of view, then you can use the **VisualVM** utility to have a look at the low-level details of your domain.



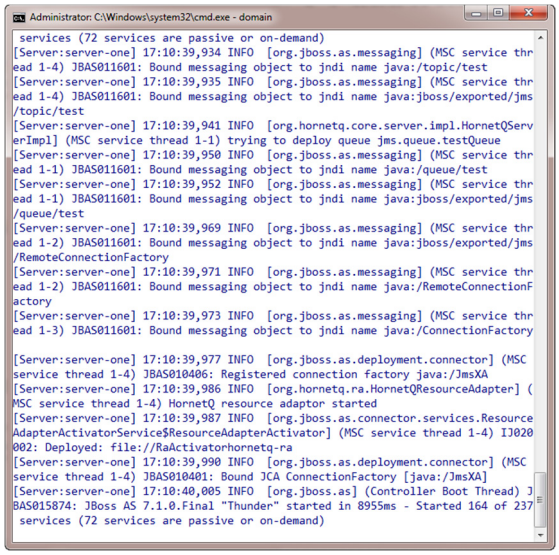
The default domain configuration

> On a Windows machine you would execute



Starting up a domain

> Launch the script file **`JBOSS_HOME\bin\domain.bat`**



Stopping a domain

> In order to stop the application server domain, you can use the command-line client and issue the **`shutdown`** command on the host controller:

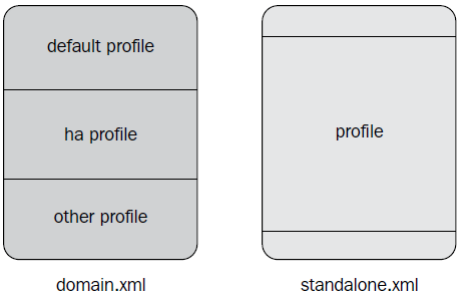
```
jboss-cli.bat --connect command=/host=master:shutdown
```

Configuration of an AS Instance in Domain Mode

> An individual server's config comes from 2 sources
  - **`domain/configuration/domain.xml`** on host with DC
    - Elements that are consistent across the domain
  - **`domain/configuration/host.xml`** on each host
    - Elements specific to the host the server runs on
> Host Controller process combines **`domain.xml`** data + **`host.xml`** data to derive server config(s)

Configuring the **`domain.xml`** file

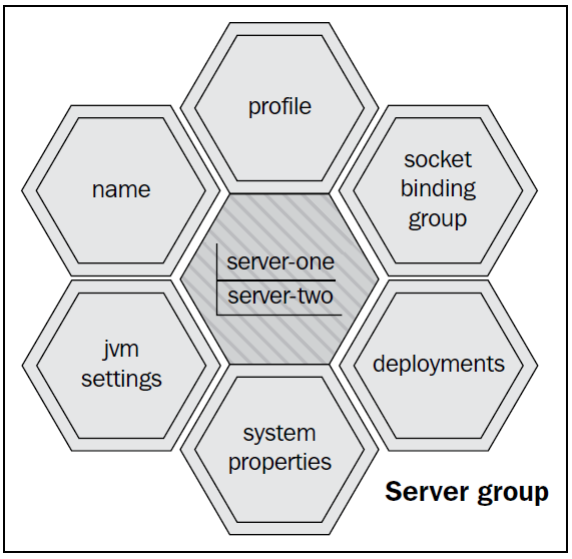
> The domain.xml contains the domain subsystems' configuration that is shared by all servers in the domain.
> The content of the file follows the same structure of the standalone file, with an obvious and important difference:
  - A domain can have *several profiles*
  - By default, two profiles are defined:
    - a ***default*** profile
    - a ***ha*** profile



Configuring the **`domain.xml`** file

- Each AS domain can be further split into server groups, each one bound to a different profile.
- The concept of server groups can be seen as a set of servers managed as a single unit by the domain.
- You can actually use server groups for fine-grained configuration of nodes
  - For example, each server group is able to define its own settings, such as customized JVM settings, socket bindings interfaces, or deployed applications.

Configuring the **`domain.xml`** file



Configuring the **domain.xml** file

```
<server-group name="custom-server-group" profile="default">
  <deployments>
    <deployment name="sample.war_v1" runtime-name="sample.war"
                hash="ABCDEFG1234567890ABC"/>
  </deployments>
  <jvm name="default">
    <heap size="512m" max-size="1g"/>
  </jvm>
  <socket-binding-group ref="standard-sockets"/>
  <system-properties>
    <property name="foo" value="bar" boot-time="true"/>
    <property name="key" value="value" boot-time="true"/>
  </system-properties>
</server-group>
```

Configuring the **host.xml** file

- The other domain configuration file is **host.xml**
- It is placed in **JBOSS_HOME\domain\configuration** folder.
- The term **host** here means a physical structure that is the portion of the domain which is contained in your AS distribution

MODULE 8

DEPLOYING APPLICATIONS ON JBOSS EAP 7

Objectives

In this module, you will learn

> Which resources can be deployed on the application server
> How to deploy applications on **JBoss AS** standalone
> How to deploy applications on **JBoss AS** domain
> Understanding the **JBoss AS 7** classloading architecture

Deploying resources on AS

There are basically three file types that we work with in Java Enterprise applications:

1. JAR file

   This is the most basic package that can be used for both application and common resources.

2. WAR file

   This archive is used for packaging web applications.

3. EAR file

   This package acts as a container for Enterprise applications.

Deploying resources on AS

> JBoss AS is also able to process the following archives:

> **RAR file**

- This is a resource adapter file
- Itis used to define a resource adapter component
- The resource adapter subsystem is provided by IronJacamar project (http://www.jboss.org/ironjacamar)

> **SAR file**

This enables the deployment of Service Archives containing MBean Services, as supported by previous versions of the application server

The JAR archive

- A **Java Archive** (**JAR**) file is used for organizing many files into one. The actual internal physical layout is much like a ZIP file, and as a matter of fact, it uses the same algorithm as the ZIP utility for compressing files.
- A JAR is generally used to distribute Java classes and associated metadata.
- In Java EE applications, the JAR file often contains utility code, shared libraries, and EJBs.

The WAR archive

- A **Web Application Archive** (**WAR**) file is essentially an archive used to encapsulate a web application.
- The web application usually includes a collection of **Java Server Pages** (**JSP**), Servlets, Java classes, XHTML, and HTML files, plus other file types, depending on the technology used.
- Since Java EE 6, a web application is now able to package EJBs in a WAR, using the same packaging guidelines that apply to web application classes.
  - You can place EJB classes
    - under the WEB-INF/classes directory
    - as a JAR file within the WEB-INF/lib directory.

The EAR archive

- An **Enterprise Archive** (**EAR**) file represents an application archive, which acts as a container for a set of modules.
- An EAR file can contain
  - One or more web modules packaged in WAR files
  - One or more EJB modules packaged in JAR files
  - One or more application client modules
  - Additional JAR files required by the application
  - JBoss specific archives such as Service Archives (.sar)

The EAR archive

- The advantage of using an Enterprise Archive file
  - It helps to distribute all applications components using a single archive, instead of distributing every single module.
  - Applications within an EAR archive will be loaded by a single classloader. Thus, by default, each module will have visibility on other modules packed in the archive

The EAR archive

> The isolation level of applications modules contained in the EAR archive is controlled by the ear-subdeployments-isolated element of the main configuration file (**`standalone.xml`**/**`domain.xml`**)

```
<subsystem xmlns="urn:jboss:domain:ee:1.0">
    <ear-subdeployments-isolated>
        false
    </ear-subdeployments-isolated>
</subsystem>
```

Deploying applications on AS standalone

> Deployment applications on JBoss AS has been traditionally a fairly simple task on JBoss AS

> Deploying applications on AS 7 can be achieved using several different strategies. The first distinction is about the automation of the deployment:

– Automatic deployment

This is triggered by the ***deployment scanner*** when a change happens in one deployed resource

– Manual deployment

This does not rely on the deployment scanner to trigger a deployment, but on a set of marker files to decide if the user needs to start/restart an application

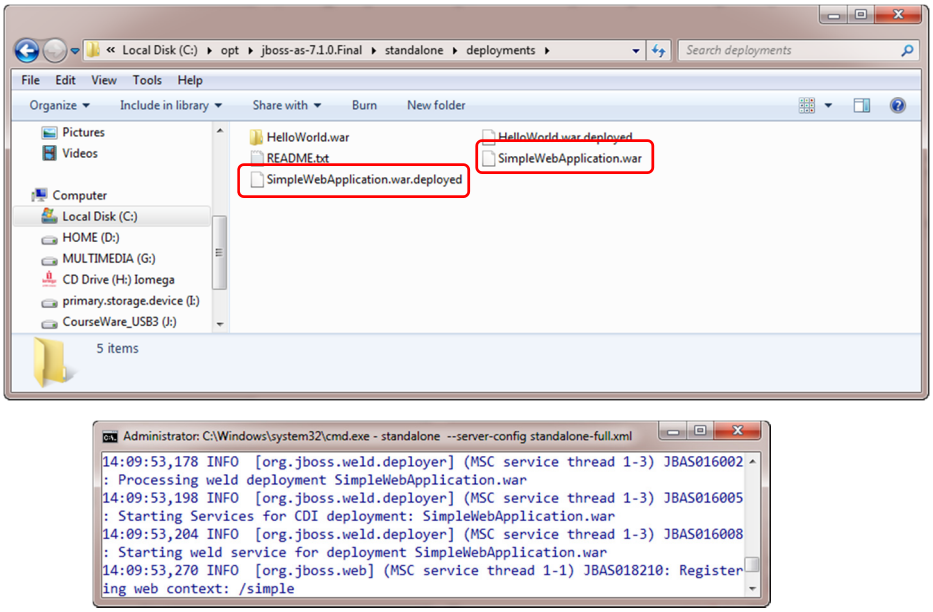
PART 1

AUTOMATIC DEPLOYMENT

Automatic application deployment

- Automatic application deployment is a simple process that consists of dropping an application into the deployments folder, which is located in the path **`JBOSS_HOME\standalone\deployments`**.
- By default, every application archive (WAR, JAR, EAR, and SAR) that is placed into this folder is automatically deployed on the server.

Automatic application deployment



Automatic application deployment

> The service that scans for deployed resources is called the **deployment scanner**

> It is configured within the **`standalone.xml`** configuration file.

```
<subsystem xmlns="urn:jboss:domain:deployment-scanner:1.0">
  <deployment-scanner name="default"
                      path="deployments"
                      scan-interval="5000"
            relative-to="jboss.server.base.dir"/>
</subsystem>
```

Deploying applications on a custom folder

- If you want to change the default deployment path, you need to operate on the **relative-to** and **path** properties.
- If you provide both properties, the **deployment** folder will be a sum of both properties.
- For example:

```
<paths>
  <path name="as7deployment" path="C:/var/as7" />
</paths>
<subsystem xmlns="urn:jboss:domain:deployment-scanner:1.0">
  <deployment-scanner name="default"
                      path="deployments" relative-to="as7deployment"
                      scan-enabled="true" scan-interval="5000"
                      deployment-timeout="60"/>
</subsystem>
```

Changing the behavior of the scanner

- We can change the behavior of the deployment scanner.
- The properties that control the auto-deploy feature are `auto-deploy-zipped` and `auto-deploy-exploded`

```
<deployment-scanner scan-interval="5000"
   relative-to="jboss.server.base.dir"
    path="deployments"
   auto-deploy-zipped="true"
   auto-deploy-exploded="false"/>
```

- You could set the `auto-deploy-exploded` to `true`, to achieve automatic redeployment of exploded archives:

```
<deployment-scanner scan-interval="5000"
    relative-to="jboss.server.base.dir"
    path="deployments"
    auto-deploy-zipped="true"
    auto-deploy-exploded="true"/>
```

PART 2

DEPLOYING AN APPLICATION USING THE COMMAND LINE INTERFACE

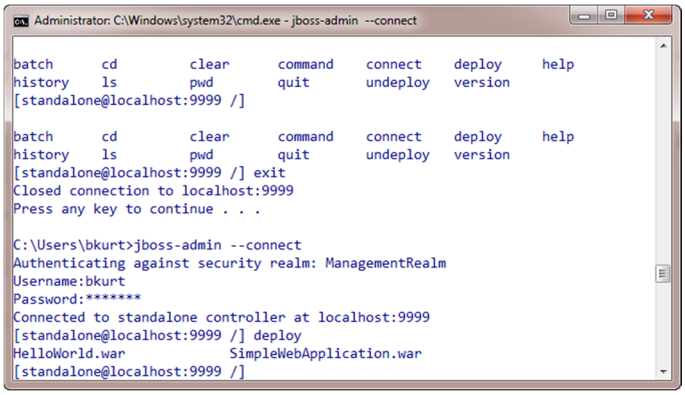
Deploying an application using the CLI

> Direct copy of the application archives is the favorite choice for many developers, as it can be performed automatically by the development environment.

> However, there are many advantages of using the CLI
  – It offers a wide choice of additional options when deploying
  – Enables remote deployment

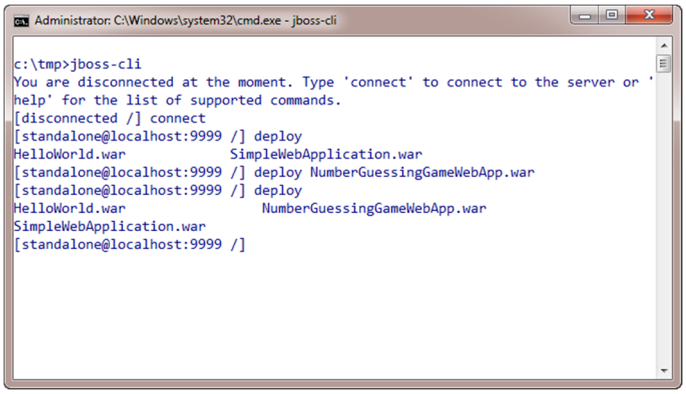
Deploying an application using the CLI

> Log into the application server, either local or remote, and issuing the **`deploy`** shell command



Deploying an application using the CLI

> If you feed to the shell a resource archive, such as .war, it will deploy it on the standalone server right away:



Deploying an application using the CLI

> If you want to just perform the deployment of the application and defer to the activation at a later time, you have to add the **`--disabled`** switch:

```
[standalone@localhost:9999 /] deploy ../game.war --disabled
'game.war' deployed successfully.
```

> In order to activate the application, simply issue another deploy command without the **`--disabled`** switch:

```
[standalone@localhost:9999 /] deploy --name=game.war
'game.war' deployed successfully.
```

Deploying an application using the CLI

> Re-deploying the application requires an additional flag to the deploy command.

> Use the –f argument to force the application redeployment.

```
[localhost:9999 /] deploy -f ../game.war
'game.war' re-deployed successfully.
```

> Un-deploying the application can be done through the **`undeploy`** command

– It takes as argument the application deployed.

```
[localhost:9999 /] undeploy game.war
'game.war' undeployed successfully.
```

Deploying to an AS domain using the CLI

> Start by launching the CLI and connect to the domain controller:

```
[disconnected /] connect
```

> When you are deploying an application using domain mode, you have to specify to which server group the deployment is associated.

> The CLI lets you choose between two options

1. Deploy to ***all server groups***.
2. Deploy to ***a single server group***.

Deploy to all server groups

> When choosing this option, the application will be deployed to all the available server groups.

> The --all-server-groups flag can be used for this purpose.

For example:

```
[domain@localhost:9999 /] deploy ../application.ear --all-server-groups
Successfully deployed application.ear
```

> If, on the other hand, you want to un-deploy an application from all server groups belonging to a domain, you have to issue the undeploy command:

```
[domain@localhost:9999 /] undeploy application.ear --all-relevant-server-groups
Successfully undeployed application.ear
```

Deploy to a single server group

> The other option lets you perform a selective deployment of your application, just on the server groups you have indicated.

```
[domain@localhost:9999 /] deploy application.ear --server-groups=mainserver-group
Successfully deployed application.ear
```

> You are not limited to a single server group, but you can separate multiple server groups with a comma; for example:

```
[domain@localhost:9999 /] deploy application.ear --server-groups=mainserver-group,other-server-group
Successfully deployed application.ear
```

Deploy to a single server group

> Now, suppose we want to un-deploy the application from just one server group.

> There can be two possible outcomes.

1. If the application is available just on that server group, you will successfully complete the un-deployment:

```
[domain@localhost:9999 /] undeploy as7project.war --server-groups=main-server-group
Successfully undeployed as7project.war.
```

Deploy to a single server group

> There can be two possible outcomes.

2. if your application is available on other server groups, the following error will be returned by the CLI:

```
[domain@localhost:9999 /] undeploy application.ear --server-groups=main-server-group
Undeploy failed: {"domain-failure-description" => {"Composite operation failed and was rolled back. Steps that failed:" => {"Operation step-3" => "Cannot remove deployment application.ear from the domain as it is still used by server groups [other-server-group]"}}}
```

   – When you are removing an application from a server group, the domain controller checks that the application is not referenced by any other server group, otherwise the previous command will fail.

Deploy to a single server group

> You can, however, instruct the domain controller to un-deploy the application, without deleting the content also:

```
[domain@localhost:9999 /] undeploy application.ear --server-groups=main-server-group --keep-content
Successfully undeployed application.ear.
```

CLI deployment options

Command	Options	Effect
deploy	--all-server-groups	Deploys an application to all server groups
undeploy	--server-groups	Deploys an application to one or more server groups
undeploy	--all-relevant-server-groups	Un-deploys and removes an application from all server groups
undeploy	--server-groups	Un-deploys an application from one server group. Fails if it's referenced in another server group
undeploy	--server-groups –keep-content	Un-deploys an application from one server group, without deleting it

PART 3

MANUAL DEPLOYMENT

Manual application deployment

- The deployment scanner will not directly deploy the applications that are placed in the deployments folder.
- It will use a set of marker files, which are used to trigger application re-deployment and capture the result of the operation.
- Marker files provide the new default approach for JBoss AS 7 exploded deployments.
- They consist either in empty files, with a suffix, which are placed by the user, or in files that are added by the container to indicate the outcome of an operation.
- Unix users can trigger deployment by simply using the touch command:

```
touch MyApplication.war.dodeploy
```

Manual application deployment

- Windows users can trigger deployment by simply using the touch command:

```
copy /b MyApplication.war.dodeploy +,,
```

Manual application deployment

On Linux operating system

> Perform a safe re-deployment of a web application named **`large.war`**

```
touch $JBOSS_HOME/standalone/deployments/large.war.skipdeploy
cp -r large.war/ $JBOSS_HOME /standalone/deployments
rm $JBOSS_HOME /standalone/deployments/large.war.skipdeploy
```

Manual application deployment

Marker	Created by	Description
.dodeploy	User	Creating this file triggers application deployment. Touching this file causes application re-deployment.
.skipdeploy	User	Application auto-deployment is disabled as long as this file exists
.deployed	JBoss AS	Application is deployed. Removing it causes un-deployment of the application.
.undeployed	JBoss AS	Application has been un-deployed. Removing it causes re-deployment of the application

Manual application deployment

Marker	Created by	Description
.failed	JBoss AS	Application deployment failed.
.isdeploying	JBoss AS	Application deployment in progress
.isundeploying	JBoss AS	Application un-deployment in progress
.pending	JBoss AS	One condition prevents application deployment (for example, file copying in progress).

PART 4

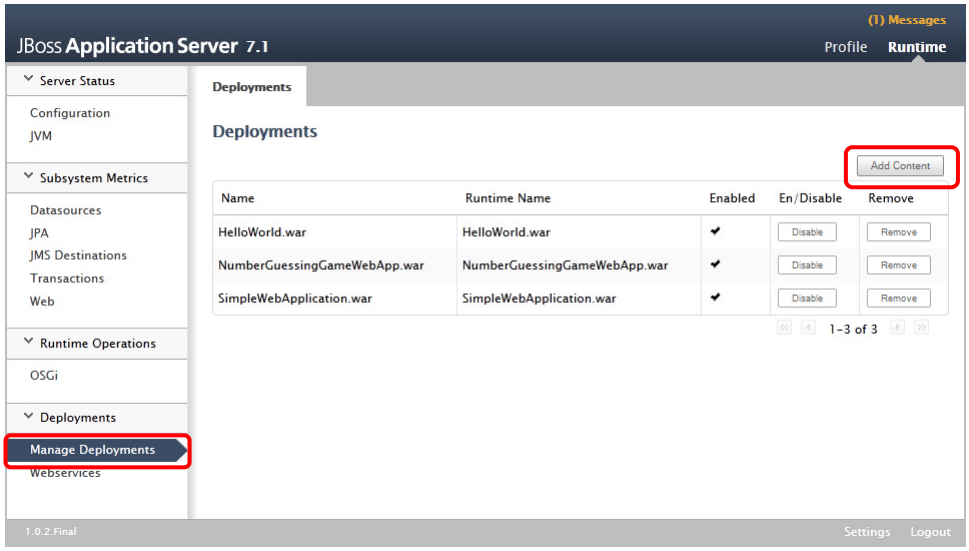
DEPLOYMENT USING THE WEB ADMIN CONSOLE

Deployment using the Web admin console

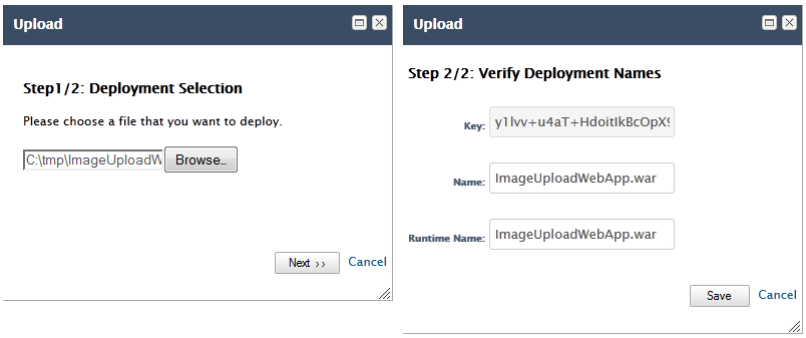
> Server deployments are managed by the application server by selecting the "**Runtime**" upper menu, and choosing the **Deployments** | **Manage Deployments** label.

> If you want to add a new application to AS7, just click on the **Add Content** button in the mid-right area of your console:

Deployment using the Web admin console



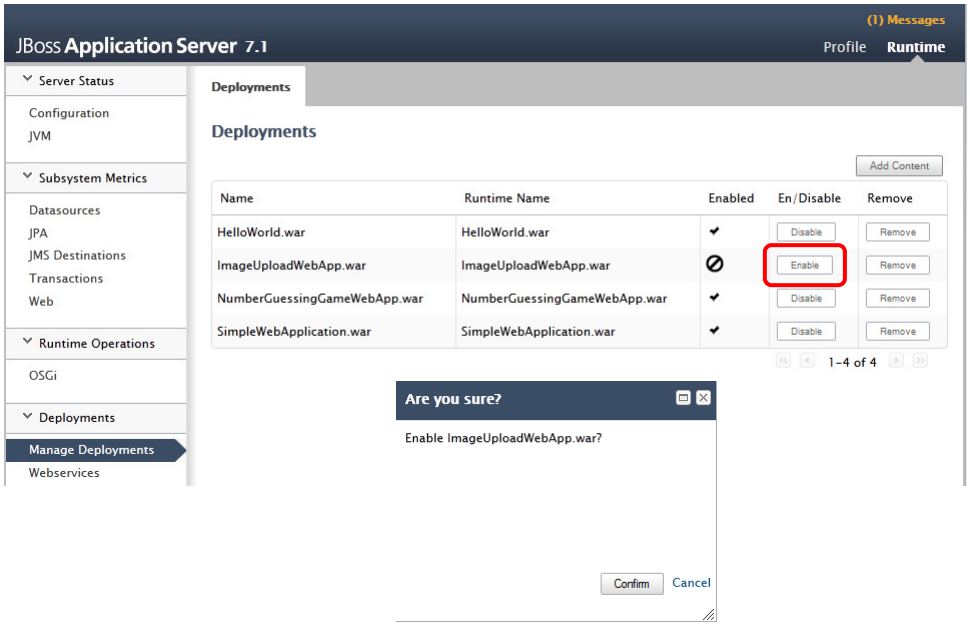
Deployment using the Web admin console



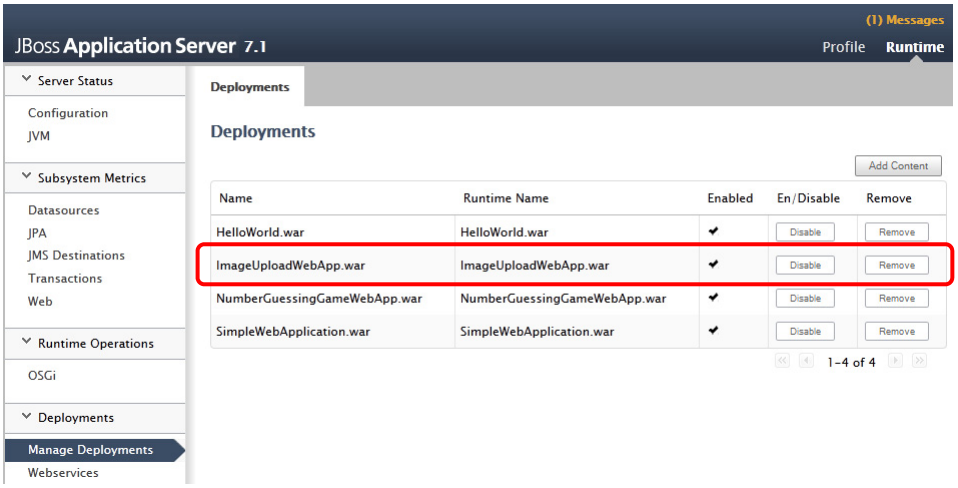
Deployment using the Web admin console



Deployment using the Web admin console



Deployment using the Web admin console



PART 5

JBOSS EAP 7 CLASSLOADING

JBoss EAP 7 Classloading

> As mandated by Java EE specifications, an application server should ideally give its deployed applications the freedom to use whatever utility library and whatever version of it, regardless of the presence of concurrent applications that want to use the same library.
> This is also known as namespace isolation (Java EE 5 specifications, section EE.8.4).
> However, loading classes from different namespaces can raises some issues, which are not easy to solve.

JBoss EAP 7 Classloading

> What happens if I pack a newer version of an utility library with my application, while an older version of the same library was loaded by the application server?

> How can I use two different versions of the same utility library, simultaneously, within the same instance of the application server?

JBoss EAP Classloading History

> The 4.x releases of the application server used a **`UnifiedClassLoader`**, which aimed to reduce communications overhead between running applications, as class data could be shared by reference or simple copies.

> One of the major outstanding issues not resolved with the **`UnifiedClassLoader`** is **classloading dependencies**.

- The idea being that if one application (A) uses the classes of another application (B), the system should know to re-deploy A when B gets re-deployed, otherwise it will be referencing stale classes.

Jboss EAP Classloading History

> Since JBoss AS 5.0 was introduced, a new classloader was based on the new **Virtual File System** (**VFS**).
> The VFS was implemented to simplify and unify file handling within the application server.
> The new classloader, named the VFS classloader, uses VFS to locate JAR and class files.
> Even though this represented a significant change in how classes are loaded in JBoss AS 5.0, the resulting behavior is much the same as for prior versions of JBoss AS.

JBoss EAP 7 Classloading

> Classloading in AS 7 marks a radical departure from previous attempts.
> Classloading is now based on the JBoss modules project.
> Any application that is deployed is a module.
> Which is the module name assigned to a deployed application?
> How are dependencies between modules handled by the AS?

Getting to know module names

> Applications that are packaged as top-level archives (such as WAR, JAR, and SAR) are assigned the following module name:

`deployment.[archive name]`

> For example, a web application named **`WebExample1.war`** will be deployed as the following module name:

`deployment.WebExample1.war`

Getting to know module names

> On applications that contain nested modules (such as the EAR archive), every single archive will be assigned a module name using this classification:

deployment.[**ear archive name**].[**sub deployment archive name**]

> The same web application, if contained in the archive **`EnterpriseApp.ear`**, will be deployed with the following name:

deployment.EnterpriseApp.ear.WebExample1.war

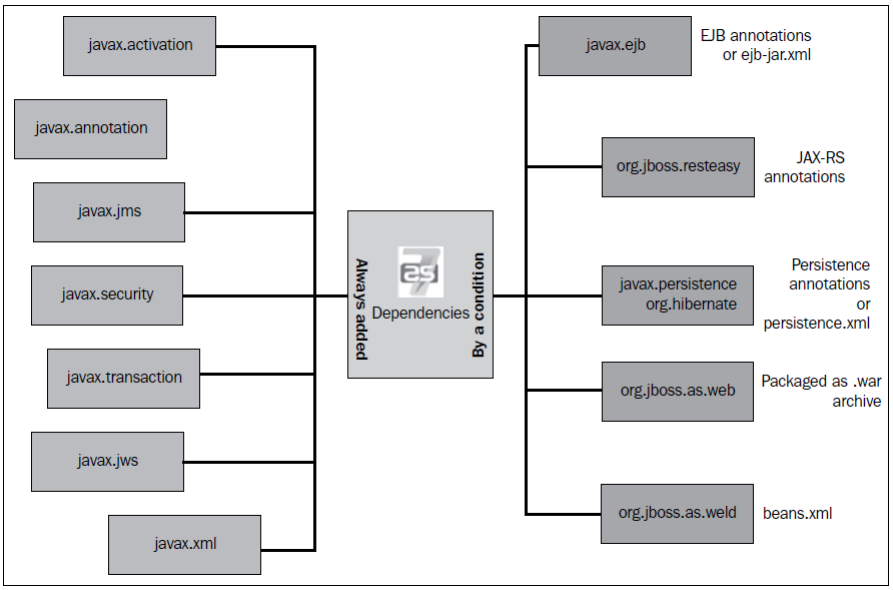
Finding the isolation level

- Using the application server modules is pretty easy:
- ***Add a dependency to the required module and the AS will use it***.
- The application server automatically adds dependencies, or they need to be signaled by the user:
  - The core module libraries (namely the Enterprise classes) are qualified as implicit dependencies, so they are automatically added to your application when the deployer detects their usage
  - The other module libraries need to be explicitly declared by the user in the application's MANIFEST file or in a custom JBoss deployment file named **`jboss-deployment-structure.xml`**.

Implicit dependencies

- Pointing out dependencies for API that are commonly used by the Enterprise application can be tedious.
- They are automatically added for you by the application server.
- Some are added when the application server detects some annotations or configuration files that are typical of that module.
  - For example, adding a **`beans.xml`** file triggers the automatic **Weld** dependency.

> A synthetic view of the modules that are implicitly added to your application



Explicit dependencies

> Modules that are not qualified as implicit dependencies need to be declared by the user.

> The simplest and recommended approach to achieve it is including in the **`META-INF/MANIFEST.MF`** the Dependencies: [module] declaration.

> For example, to make your application dependent on **`log4j`**, just include in your manifest file:

```
Dependencies: org.apache.log4j
```

> Please note that the module name does not always match with the package name of the library.

> The actual module name is specified in the **`module.xml`** file by the name attribute of the module element.

Explicit dependencies

- You are not limited to a single dependency, as you can add multiple dependencies separated by a comma.
- For example, in order to add a dependency on both **log4j** and **Apache Velocity API**, you would use the following:

```
Dependencies: org.apache.log4j,org.apache.velocity
```

Setting up global modules

- If you define a section named global-modules within your **standalone.xml/domain.xml**, then you will make the module accessible to other AS modules.
- For example, instead of declaring a dependency on **log4j**, you could alternatively use the following section:

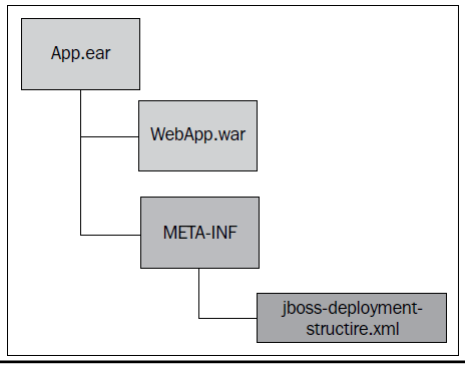
```
<subsystem xmlns="urn:jboss:domain:ee:1.0">
  <global-modules>
    <module name="org.apache.log4j" />
  </global-modules>
</subsystem>
```

Advanced deployment strategies

- If you are using a complex archive configuration, such as an EAR archive with several modules and dependencies, it would prove useful to define your classloading strategy in a single file.
- The configuration file **`jboss-deployment-structure.xml`** can do exactly this.
- The advantages of using this file are many:
  - You can define the dependencies of all application modules in a single file
  - You can load the modules classes using a finer grained manner by including/excluding all or part of modules
  - You can define classloading isolation policy for your applications packaged in an Enterprise archive

Setting up a single module dependency

- We have already learnt how to activate a **`log4j`** dependency, using the **`Dependencies`** attribute in the archive's **`MANIFEST`** file.
- The same effect can be achieved by using the **`jboss-deployment-structure.xml`** file.
- Let's recap the archive structure:



jboss-deployment-structure.xml

```
<jboss-deployment-structure>
  <sub-deployment name="WebApp.war">
    <dependencies>
       <module name="org.apache.log4j" />
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Excluding server automatic dependencies

- we have shown how the application server is able to trigger some dependencies, automatically, when some conditions are met.
  - For example, if you deploy a JSF application (containing the **`faces-config.xml`** file), then the JSF 2.1 API implementation is automatically added.
- This might not always be the desired option, for example, because you want to provide another release implementation for that module.

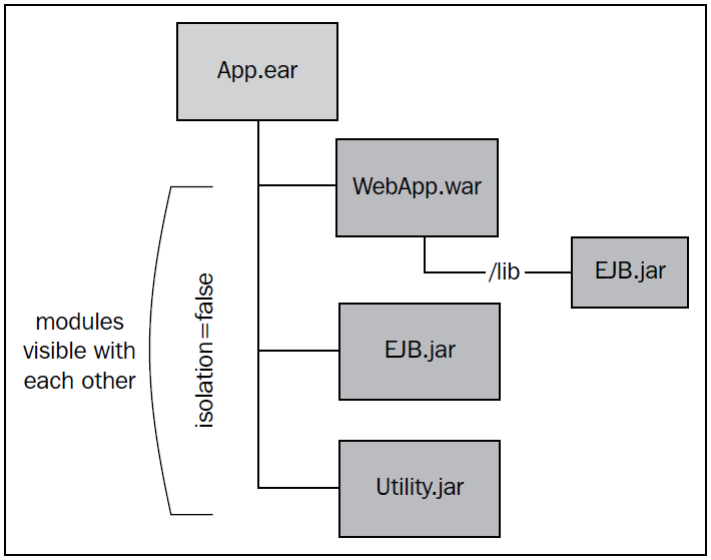
Excluding server automatic dependencies

```
<jboss-deployment-structure>
  <deployment>
    <exclusions>
       <module name="javax.faces.api" />
       <module name="com.sun.jsf-impl" />
    </exclusions>
    <dependencies>
       <module name="javax.faces.api"
               slot="1.2"/>
       <module name="com.sun.jsf-impl"
               slot="1.2"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Isolating sub-deployments

- Supposing you have an application that is made up of
  - a web application,
  - an EJB module,
  - a JAR file containing utility classes.
- All sub-deployments are placed at the root of the archive, so that they will be able to see each other.
- This can be convenient, however, supposing that your web application itself contains some implementations of the same EJB.
- That's absolutely possible since Java EE 6 specification allows your web application to include EJB classes within the **`WEB-INF/classes`** or **`WEB-INF/lib`** folder.

Isolating sub-deployments



How does the classloader solves this conflict?

Isolating sub-deployments

The application server classloader has a priority list when loading classes that are used to avoid any conflict between loaded classes.

> The **maximum** priority is given to modules, automatically, by the container, including the Java EE APIs. Libraries that are contained in the modules folder are included in this category.

> **Then**, libraries that are indicated by the user within the MANIFEST.MF of the packaged archive as Dependencies (or in the jboss-deploymentstructure.xml file).

> **Next**, libraries that are packed within the application itself, such as classes contained in WEB-INF/lib or WEB-INF/classes.

Isolating sub-deployments

The application server classloader has a priority list when loading classes that are used to avoid any conflict between loaded classes.

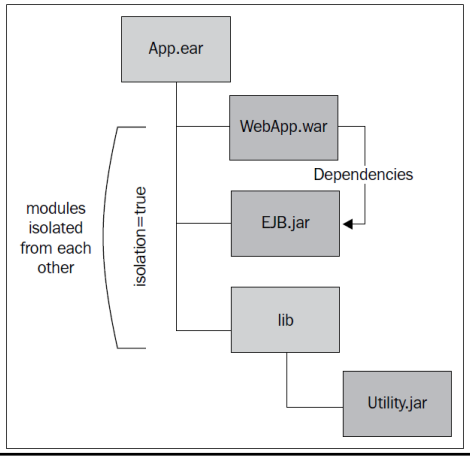
> **Finally**, libraries that are packed within the same EAR archive (in the EAR's lib folder)

Isolating sub-deployments

> If you want to keep deployment isolated, but allow visibility for some of them

Isolating sub-deployments

> Move the library to the EAR/lib folder, so that it will be picked up as a separate module
> Specify a dependency using Dependencies or using Class-Path in the MANIFEST.MF file of the calling application



jboss-deployment-structure.xml

```
<jboss-deployment-structure>
   <ear-subdeployments-isolated>
    true
   </ear-subdeployments-isolated>
   <sub-deployment name="WebApp.war">
     <dependencies>
       <module name=
           "deployment.App.ear.EJB.jar"/>
     </dependencies>
   </sub-deployment>
</jboss-deployment-structure>
```

MODULE 9

MANAGING THE APPLICATION SERVER

Objectives

Upon completion of this module, you should be able to

> Use the **Command Line Interface** (**CLI**)

> Use the **Web admin console**

The Command Line Interface (CLI)

- Terminals and consoles were one of the earliest types of communication interfaces between a system administrator and the machine.
- Due to this long time presence, most system administrators prefer to use the raw power of the command line for performing management tasks.
- One of the most evident advantages of using a low-level interface, such as a shell, is that tasks can often be executed as a part of batch processing or macros, for repetitive actions.

The Command Line Interface (CLI)

- As we have introduced at the beginning of this training, the CLI is located in the **`JBOSS_HOME/bin`** folder and wrapped by **`jboss-admin.bat`** in 7.0 and **`jboss-cli.bat`** in 7.1

```
You are disconnected at the moment. Type 'connect' to connect to the
server or 'help' for the list of supported commands.
[disconnected /] connect
Connected to standalone controller at localhost:9999
```

The Command Line Interface (CLI)

> The default port is **9999**

> You can adjust the default port in the **standalone.xml/domain.xml** configuration file

```
<socket-binding name="management-native" interface=
"management" port="9999"/>
<socket-binding name="management-http" interface= "management"
port="9990"/>
```

The Command Line Interface (CLI)

> A handy switch is **--connect**

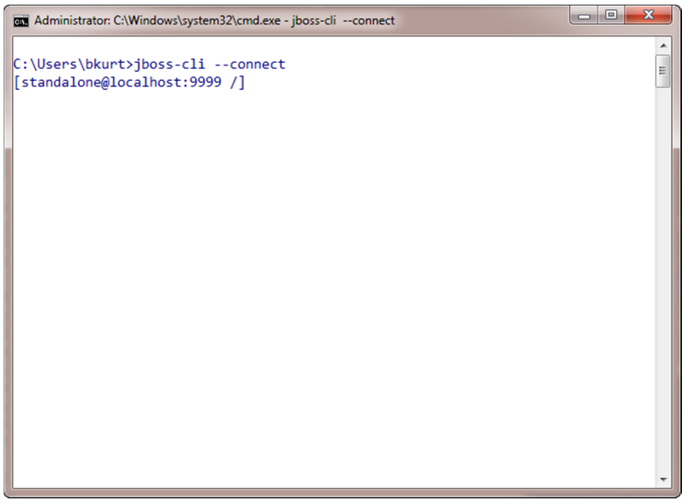
– used to automatically hook up to your standalone/domain controller:

```
jboss-cli.bat --connect
```

> The corresponding command for exiting the CLI is **quit**

```
[standalone@localhost:9999 /] quit
Closed connection to localhost:9999
```

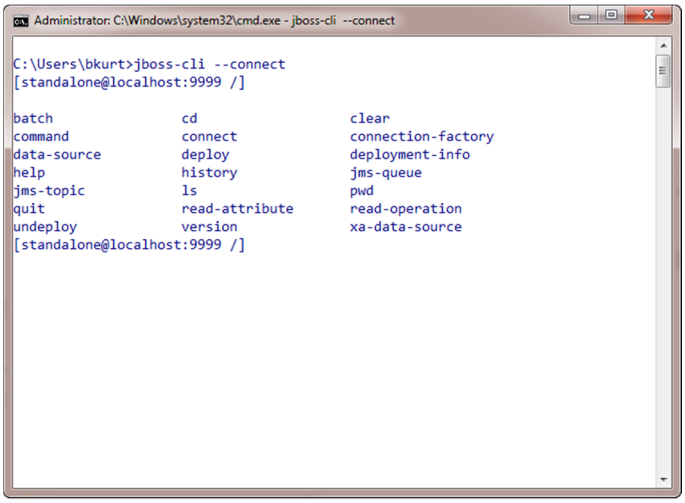
The Command Line Interface (CLI)



How to use the CLI

- One of the most interesting features of the CLI is its embedded intelligence
- It helps us to find the correct spelling of resources and commands, by simply pressing the *Tab* key.
- Once you have successfully connected, press the *Tab* key, and it will expand to show the list of available commands and resources

How to use the CLI



How to use the CLI

> There are about 25 commands available.

> We can, however, distinguish all the interactions that happen with the CLI into two broad categories:

1. Operations

   These include the resource path (address) on which they are executed.

2. Commands

   These don't include the resource path and they can, thus, execute an action independently from the path of the current resource.

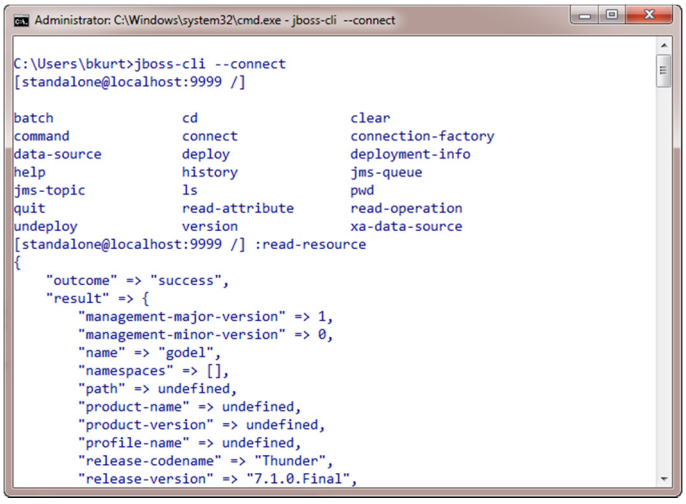
Navigating through the resources and executing operations

- Operations are strictly bound to an application server resource path.
- The path along the tree of resources is represented by the "/" character
  - represents the root of the tree, just like for an Unix-Linux file system.
- When you are executing operations on the AS resources, you have to use a well-defined syntax:

```
[node-type=node-name (,node-type=node-name)*]
: operation-name [( [parameter-name=parameter-value (,parameter-name=parameter-value)*] )]
```

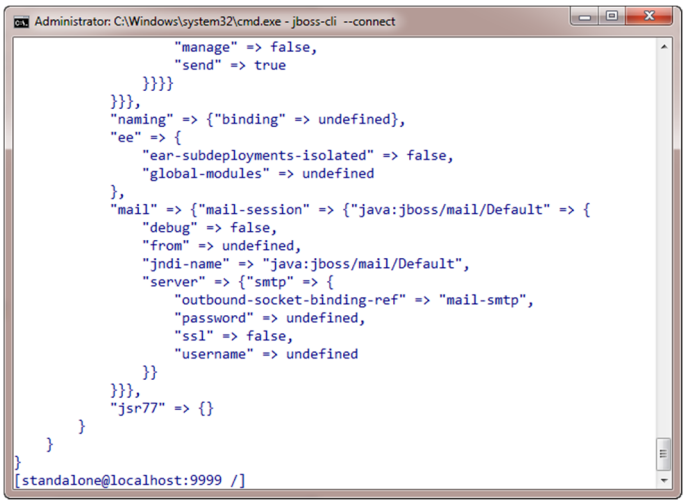
Operations

- :read-resource



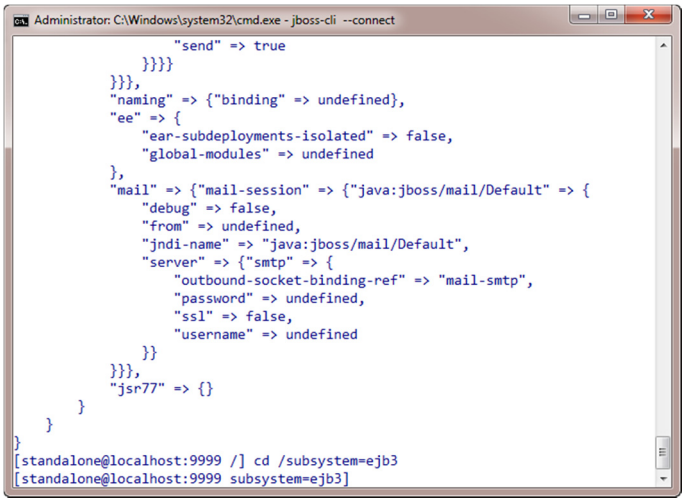
Operations

> :read-resource(recursive="true")



Operations

> cd /subsystem=ejb3



Operations

> cd /subsystem=ejb3

> :read-resource

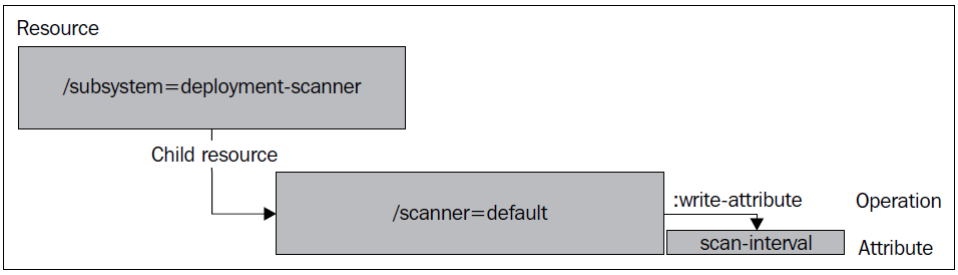


Operations

> /subsystem=deployment-scanner/scanner=default:**write-attribute**(name=scan-interval,value=2500)

{"outcome" => "success"}

> Here, we are telling the CLI to navigate into the deployment-scanner sub-system, under the default **`scanner`** resource, and set the scan-interval attribute to 2500 ms, using the **`write-attribute`** operation.



Operations

> In order to move through the resources path, you can either state the full tree path or use the **`cd`** command or also the equivalent **`cn`** (change node) to navigate to the path and then issue the desired command.

> For example, the earlier attribute can also be rewritten as:

[standalone@localhost:9999 /] **cd** /subsystem=deployment-scanner/scanner=default

[standalone@localhost:9999 scanner=default] :**write-attribute**(name=scan-interval,value=2500)

{"outcome" => "success"}

Does modified attributes survive server restart?

> When using CLI, every change is persisted into the server configuration file, thus, you must be responsible when you are using this instrument.
> More power asks more responsibility indeed!
> Nevertheless, you can, at any time, take a snapshot of your server configuration.

Singleton Session Bean Characteristics

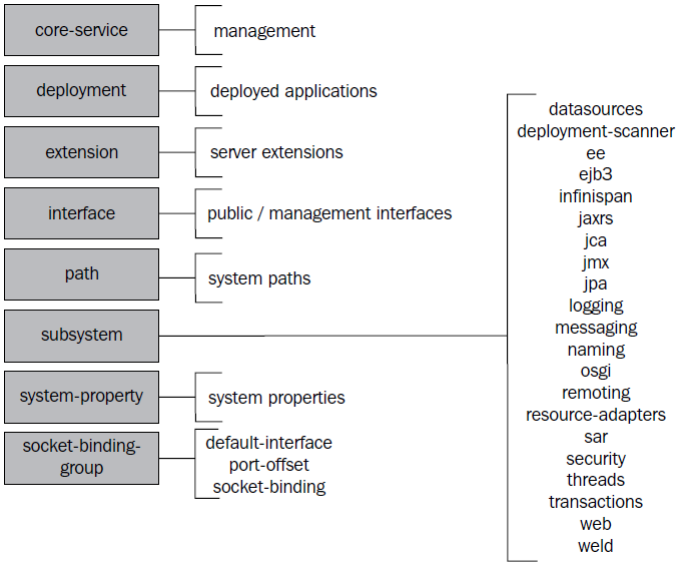
> Just like for the operating system shell, by issuing **cd ..** you move the resource pointer to the parent resource:

```
[standalone@localhost:9999 scanner=default] cd ..
[standalone@localhost:9999 subsystem=deployment-scanner]
```

> You can, at any time, check the resource path where you are located, by issuing either an empty cd command or just **pwd**, as you do for an Unix shell:

```
[standalone@localhost:9999 scanner=default] pwd
/subsystem=deployment-scanner/scanner=default
```

Bird's eye view of the AS tree or resources



Operations which you can issue on resource

> Operations are triggered by the **:** character

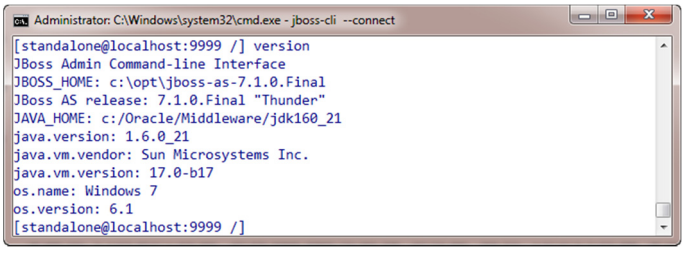
Command	Meaning
`read-resource`	Reads a model resource's attribute values along with either basic or complete information about any child resources.
`read-resource-description`	Outputs a description for the selected resource
`read-operation-names`	Read the available operation names on the node.
`read-operation-description`	Outputs a description for the available operations.

Operations which you can issue on resource

Command	Meaning
`read-children-names`	Gets the name of all children under the selected resource
`read-children-resources`	Reads information about all of a resource's children that are of a given type
`read-children-types`	Provides the list of the children located under the selected resource
`read-attribute`	Gets the value of an attribute for the selected resource
`write-attribute`	Writes an attribute for the selected resource

Executing commands with the CLI

- The CLI includes a set of actions as well, which are not bound to your navigation path across the AS tree
- They can be issued anywhere to create and modify resources.
- For example, the version command can be issued to retrieve some basic information about the application server and the environment where JBoss AS is running:



Adding a JMS destination

> You can add a JMS queue with the **`add-jms-queue`** command

add-jms-queue [--profile=profile_name]
--name=queue_name
[-- entries=entry(,entry)*]
[--selector=selector_name]
[-- durable=(true|false)]

> The only mandatory element is **name**, which specifies the queue name
> The optional entries element includes a comma-separated list of JNDI names the queue should be bound under.
  – If not present, the name will be used as the JNDI name

Adding a JMS destination

> The following command creates a new JMS queue named queue1 and bound under the JNDI queues/queue1 namespace:

add-jms-queue --name=queue1 --entries=queues/queue1

Creating and modifying datasources

> The CLI provides an useful **`data-source`** command to create a datasources.

data-source [--profile=<profile_name>] **add**/**remove** --jndi-name=<jndi_name>
--driver-name=<driver_name>
--pool-name=<pool_name>
--connection-url=<connection_url>

Creating and modifying datasources

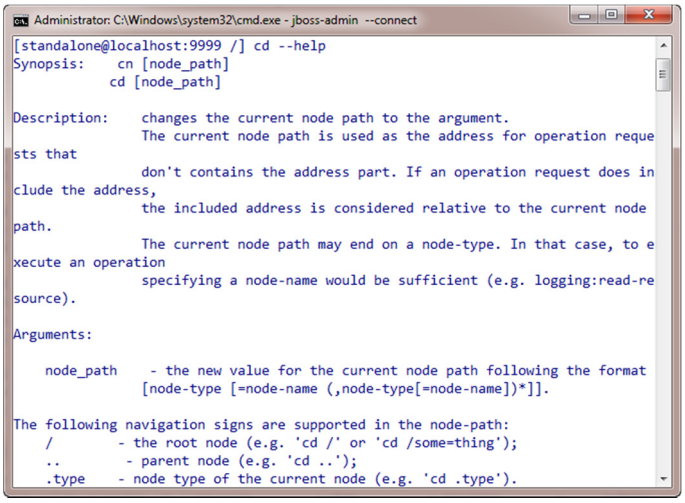
> data-source **add** --**jndi-name**=java:/worldDS --**pool-name**=**world_pool** --**connection-url**=**jdbc:mysql://localhost:3306/world** --**driver-name**=**mysql-connector-java-5.1.13-bin.jar** --**user-name**=root --**password**=root --**max-pool-size**=**30**

> The data-source command can be used as well to remove a datasource from the configuration:

[standalone@localhost:9999 /] data-source **remove** --jndi-name=**java:/worldDS**

Getting help from the CLI

> Besides the tab completion functionality, the CLI has also got a man page for each command, just like the Unix shell.



Executing CLI scripts in batch

> The batch mode allows the execution of multiple CLI commands as an atomic unit.

> Just as an ordinary transaction,

– If any of the commands or operations fail, the changes are rolled back.

– If the execution ends without any error, changes are committed.

> Not every command can be part of a batch. For example, navigation commands, such as **`cd`**, **`pwd`**, or **`help`** are excluded

– They do not reflect any change into the server configuration.

Executing CLI scripts in batch

```
[standalone@localhost:9999 /] batch
[standalone@localhost:9999 /#] add-jms-queue --name=queue1 --
entries=queues/queue1
[standalone@localhost:9999 /#] deploy MDBApplication.jar
[standalone@localhost:9999 /#] run-batch
```

> Before executing the batch, you can, at any time, get the list of batch commands

```
[standalone@localhost:9999 /] list-batch
#1 add-jms-queue --name=queue1 --entries=queues/queue1
#2 deploy MDBApplication.jar
```

Executing CLI scripts in batch

> Batch scripts can be quite useful as part of a script, which can be stored on the file system and executed when needed.

> The application server distribution provides into the **`JBOSS_HOME/bin/scripts`** folder two examples of batch files

– They are used to secure the management interfaces.

Advanced batch commands

- Script batching can indeed be more complex than just starting and executing a list of commands.
- As a matter of fact, by pressing the *Tab* completion key when you are in batch mode, you should see several additional commands available.
- One of the most useful ones is **holdback-batch**
  – temporarily pausea the batch of commands:
  [standalone@localhost:9999 /#] **holdback-batch**
- In order to continue your batch of commands, just issue the batch command again:
  [standalone@localhost:9999 /] **batch**

Advanced batch commands

- It's even possible to save the batch by assigning an unique name, so that you can have multiple save points in your scripts:
  [standalone@localhost:9999 /#] **holdback-batch step1**
- Later on, you can continue the execution, by specifying the holdback name:
  [standalone@localhost:9999 /] **batch step1**
- When executed with the –l parameter, the batch command provides the list of batch files that are held:
  [standalone@localhost:9999 /] **batch –l**
  step1

Advanced batch commands

Command	Description
batch	Starts a batch of commands. When the batch is paused, reactivates the batch
list-batch	Lists the commands that have been added to the batch
run-batch	Executes the currently active batch of commands and exits **`batch`** mode
holdback-batch	Saves the currently active batch and exits the **`batch`** mode, without executing the batch. The held back batch can later be re-activated, by invoking batch commands
clear-batch	Removes all the existing command lines from the currently active batch. The CLI stays in the **`batch`** mode after the command is executed

Advanced batch commands

Command	Description
discard-batch	Discards the currently active batch. All the commands added to the batch will be removed, the batch will be discarded and the CLI will exit the batch mode
edit-batch-line	Replaces existing command line from the currently active batch, with the specified line number with the new one
remove-batch-line	Removes an existing command line specified with line number argument from the currently active batch
move-batch-line	Moves existing line from the specified position to the new position, shifting the lines between the specified positions

Executing scripts in a file

> Until now, we have seen CLI commands as part of an interactive session.

> You can, however, execute commands in no-interactive fashion, adding them in a file, just as a shell script.

> Suppose you have created a sample **test.cli** file used to issue a re-deploy command:

```
connect
deploy Utility.jar --force
```

> Then, as Windows user, launch the CLI with the --file parameter:

```
jboss-cli.bat --file=test.cli
```

Executing scripts in a file

> Another way to execute commands in a non-interactive way, is passing the **--commands** parameter to the CLI, containing the list of commands lines separated by a comma.

> For example, the previous script can be also executed this way:

```
jboss-cli.bat --commands="connect,deploy Utility.jar --force"
Connected to standalone controller at localhost:9999
'Utility.jar' re-deployed successfully.
Closed connection to localhost:9999
```

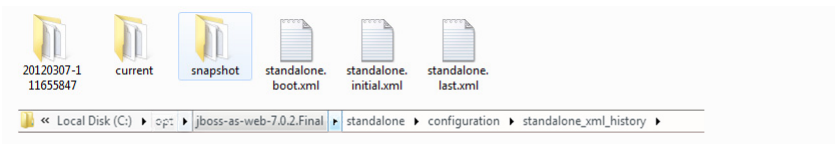
Redirecting non-interactive output

- When you are executing the CLI in a non-interactive way, you can redirect the output to a file
  - otherwise be printed on the screen
- Just as for a shell command, use the > operator to redirect the output:

```
jboss-cli.bat --file=test.cli > out.log
```

Taking snapshots of the configuration

- Everyone makes mistakes, but many of them are preventable.
- Whenever you are performing massive changes to your configuration, it's always a good idea to save copies of your work.
- One of the advantages of using the CLI, is the ability to create snapshots of the configuration
  - They are stored in its **`history`** folder.
  - Standalone servers have a history folder named **`standalone_xml_history`**

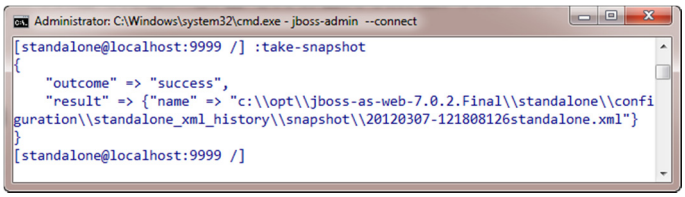


Taking snapshots of the configuration

- The domain configuration, on the other hand, provides two backup directories both for the domain configuration file and the host configuration file.
- These folders are named **`domain_xml_history`** and **`host_xml_history`**, respectively.
- The **`standalone.boot.xml`** file contains the AS configuration that was used for the last successful boot of the server
  - overwritten every time we boot the server successfully.

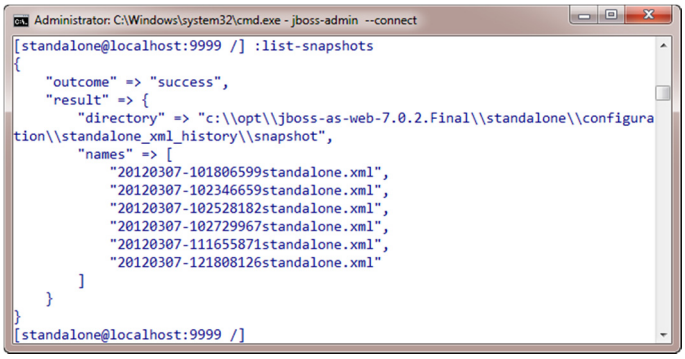
Taking your own snapshots

- you can also take snapshots on demand whenever you need it.
- Snapshots created by the user are stored directly in the snapshot folder.
- In order to take a snapshot of the configuration, just issue the **`take-snapshot`** command and the CLI will back up your configuration:



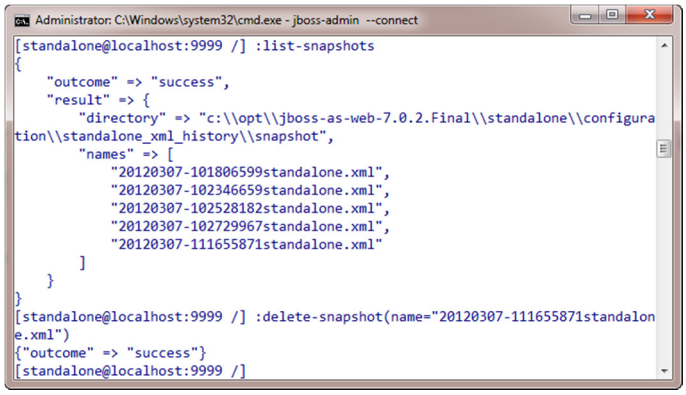
Taking your own snapshots

> You can check the list of available snapshots by using the **`list-snapshots`** command:



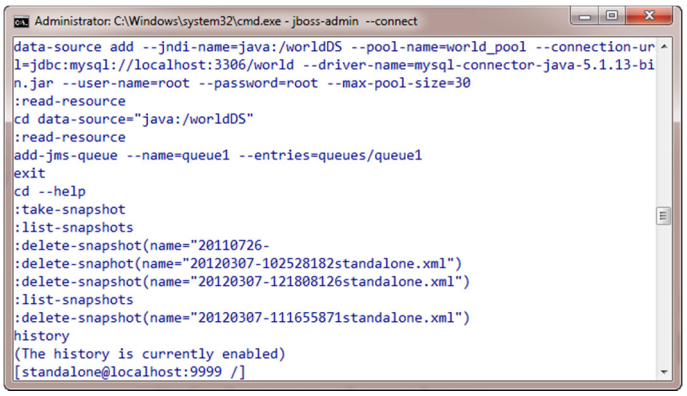
Taking your own snapshots

> You can, at any time, delete a particular snapshot using the delete-snapshot command
  – requires the snapshot name as parameter



History of CLI

- All commands which are executed within a CLI session are stored in an history, much like shell commands.
- CLI commands are kept in memory and also persisted on the file system in a file named **`.jboss-cli-history`** placed in the user's home directory.



Web admin console

- Historically, the JBoss AS has always provided a Web-based application to perform some administration and management tasks.
- Releases 4.x and earlier used the **`jmx-console`** to read/write and display the value of MBeans.
- The **`jmx-console`** was indeed a useful tool, however, it also required some degree of experience in order to get started with it.
- The information contained in this application was fragmented across many MBeans.
  - For example, the datasource information was contained into four Mbeans, thus making it cumbersome to manage this resource.

Web admin console

- The 5.x and 6.x release proposed a simpler to use approach made up of the Admin console
  - built as a Seam-Based Web application
  - Although the new Admin console was a neat and simple application, it consumes a pretty good amount of memory and start up time

The new Web admin console

- The new AS 7 Web console is a completely new Web application
  - powered by the **Google Web Toolkit** (**GWT**) that uses the HTTP management API to configure a management domain or standalone server.
- As many GWT applications, it uses
  - a JSON-encoded protocol and a de-typed
  - RPC style API to describe and execute management operations against a managed domain or standalone server

Accessing the admin console

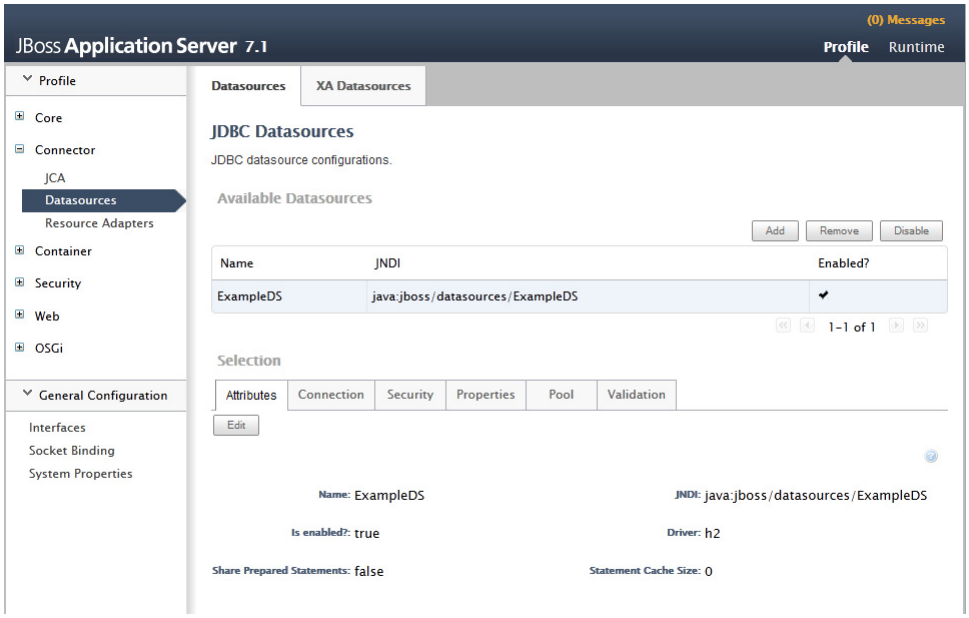
- **JBoss AS 7**, by default, does not use anymore port 8080 to serve the admin console.
- You can access it at the **http:// localhost:9990/console** as configured in your **standalone.xml/domain.xml**.

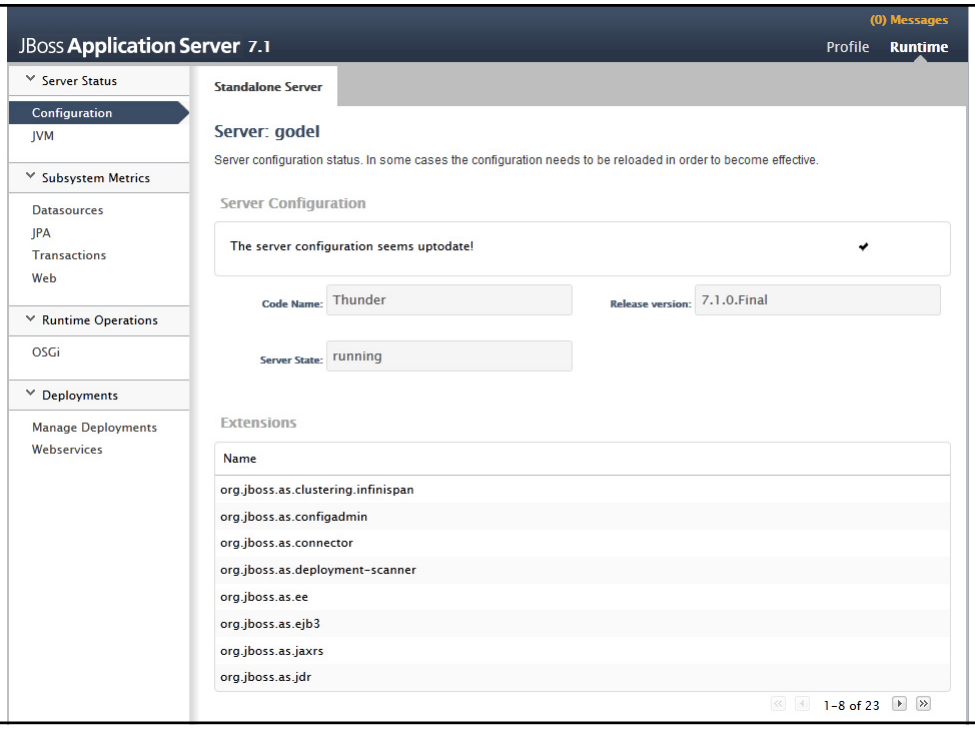
```
<socket-binding name="management-http"
                interface="management"
                port="9990"/>
```

Accessing the admin console

- Since **JBoss AS *7.1.0*** you need to add at least one user to access the Web console.
- To add a new user execute the **add-user.bat** script within the bin folder of your **AS 7** installation and enter the requested information.
- See "Module 11 – Security" for more details about it.

Accessing the admin console





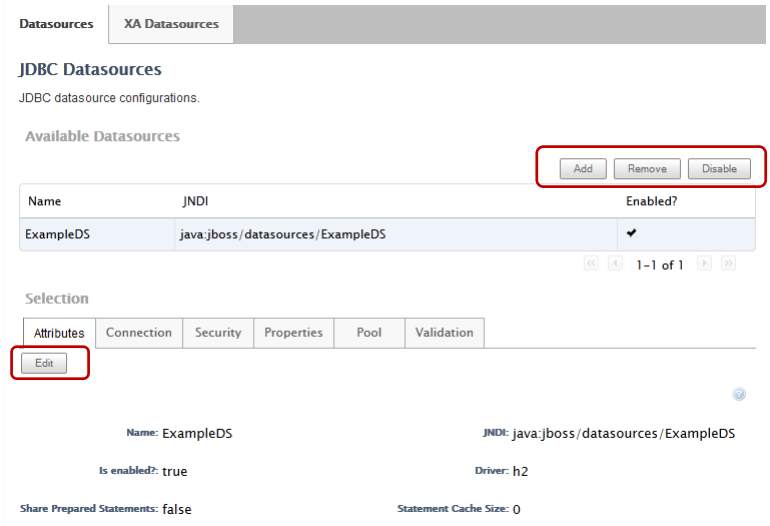
Configuring server profiles

- The server profile configuration is located in the left area of the Web application. When running in standalone mode, it can be located under the "Profile" label.
- When running domain mode, you will need to navigate through the "Profiles" tab and choose one through the "Profile" combo box.
- Expanding the Profile label, you can see a set of subsystems which can be configured through the interface
- We will just cover the following handy topics:
  - Configuring datasources
  - Configuring JMS resources
  - Configuring Socket Binding groups

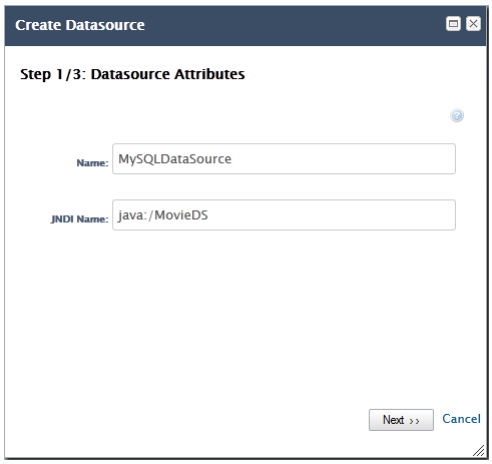
Configuring datasources

- Clicking on the **Datasources** link, you will switch the main panel to the **Datasource Configurations** panel.
- This panel contains two upper tabs for configuring **Datasources** and **XADatasources**.

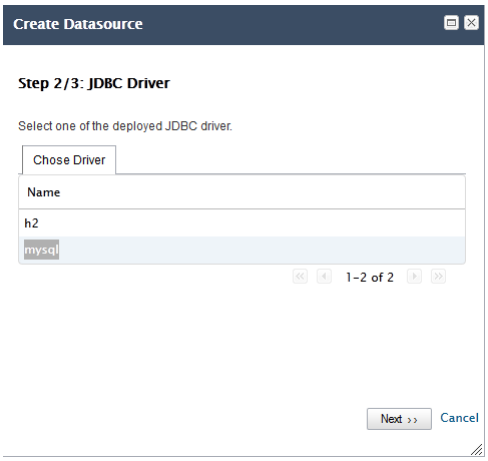
Configuring datasources



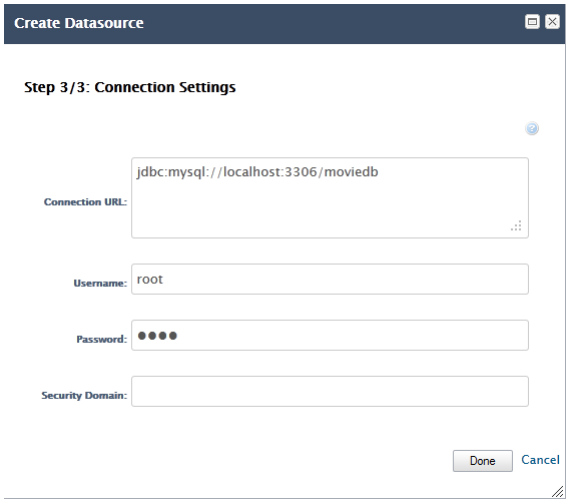
Configuring datasources



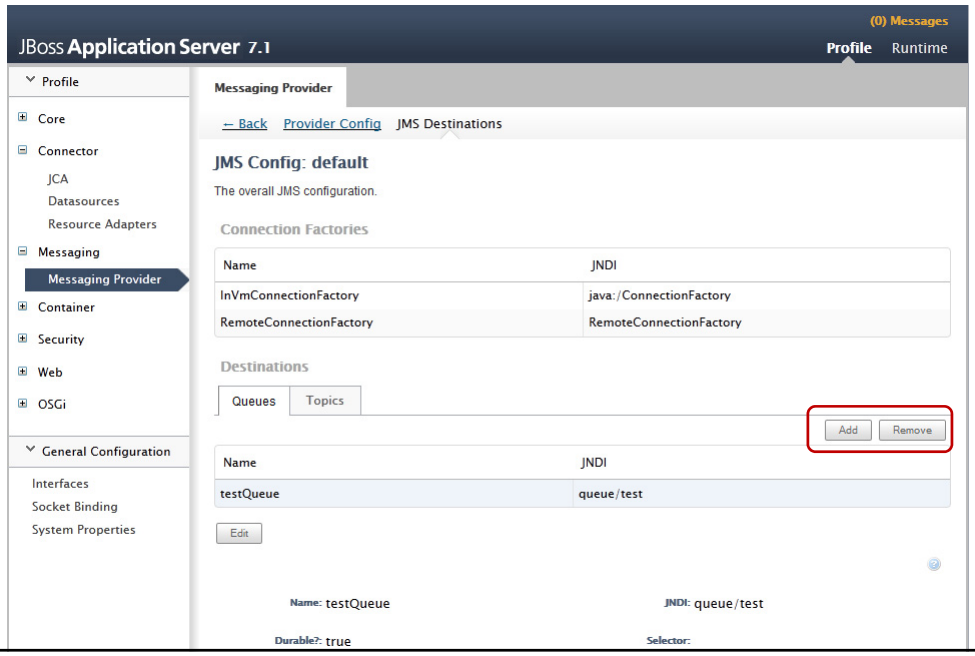
Configuring datasources



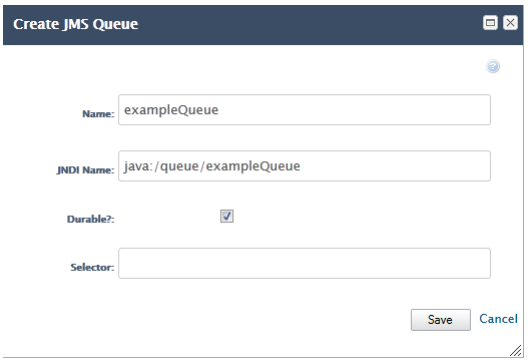
Configuring datasources



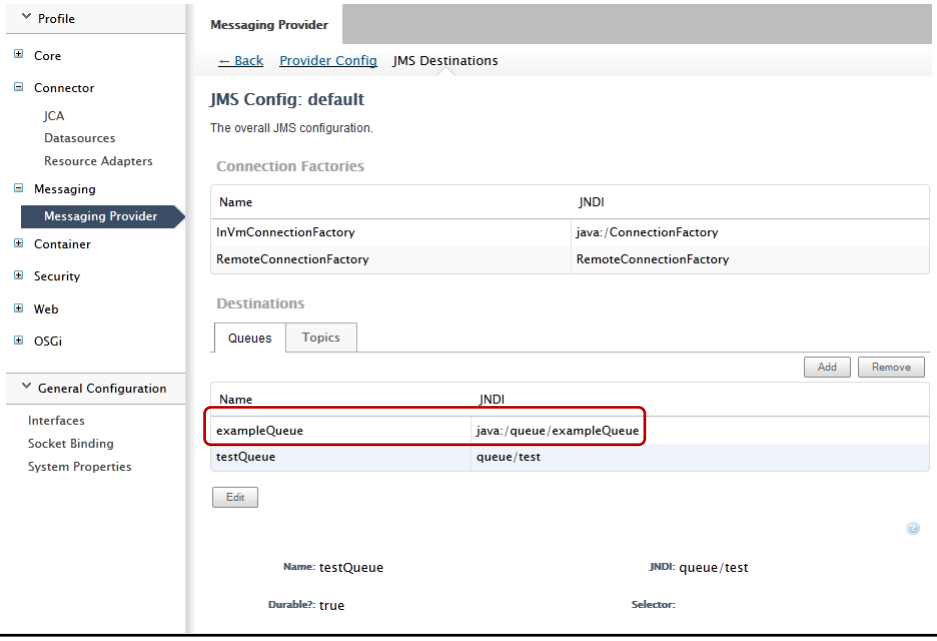
Configuring JMS destinations



Configuring JMS destinations



Configuring JMS destinations



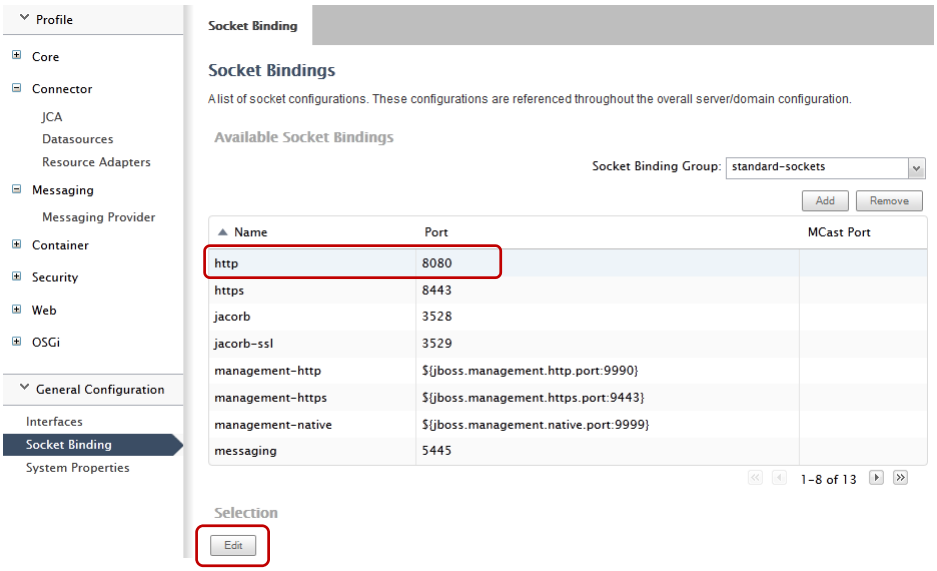
Configuring Socket Binding groups

> Changing the Socket Bindings of the application server can be used to solve port conflicts with other applications or even other instances of JBoss AS.

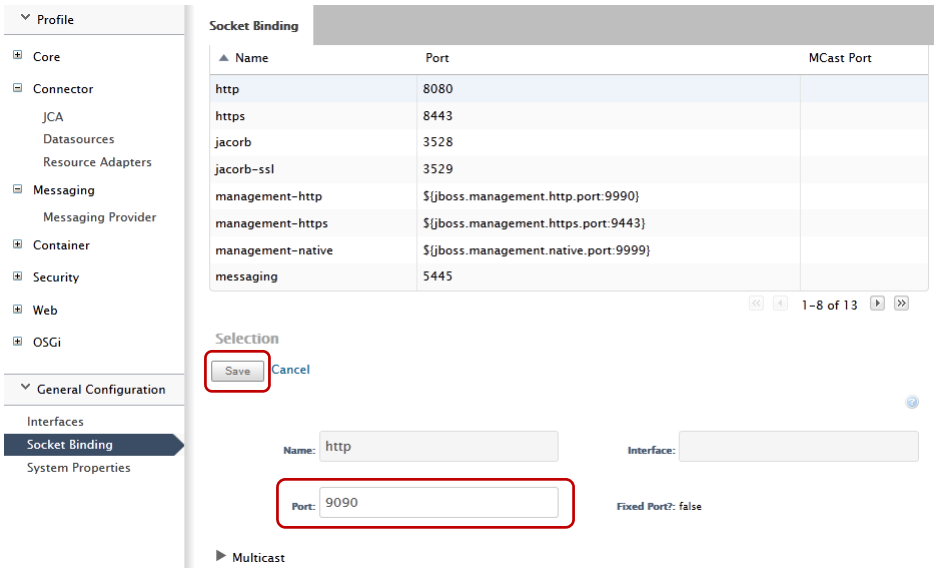
> If you are running in **domain** mode, the best thing you can do is specify a port offset for your servers

> If, however, you are running in standalone mode and you have just to change one or more port address, than you might easily change it from the Web console.

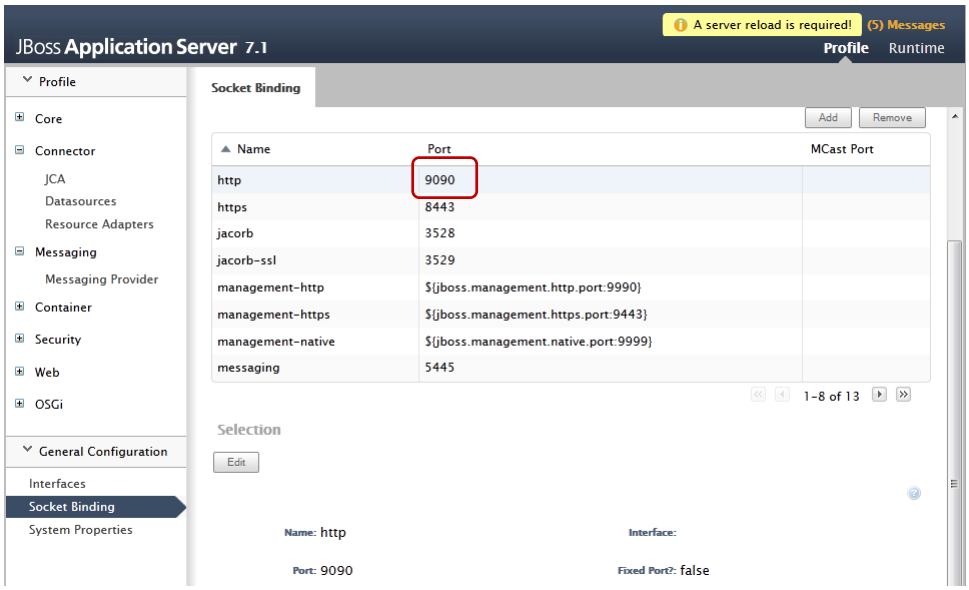
Configuring Socket Binding groups



Configuring Socket Binding groups



Configuring Socket Binding groups



MODULE 10

CLUSTERING

Objectives

> We will study how to effectively configure the cluster-building blocks:
  - The **`JGroups`** subsystem which is used for the underlying communication between nodes
  - The **`Infinispan`** subsystem which handles the cluster consistency using its advanced data grid platform
  - The **`Messaging`** subsystem which uses the **`HornetQ`** clusterable implementation

Overview

> This module will cover AS 7 clustering capabilities that serve as an essential component to providing **scalability** and **high availability** to your applications.
> Advantages of clustering
  - You can spread the traffic load across several AS instances.
  - **Load balancing** is an orthogonal aspect of any Enterprise application that is generally achieved using a properly configured web server in front of the application server

Setting up **JBoss** clustering

> AS 7 ships with a separated configuration file named **`standalone-ha.xml`**, which contains all the clustering subsystems.
> On the other hand, a domain server is able to store multiple profiles into the core **`domain.xml`** configuration file, hence you can use this file both for clustered domains and for non-clustered domain servers.

Setting up a Cluster of Standalone Servers

> A cluster of standalone servers is a common option for a development or test environment.
> Nothing prevents you from using it also on your production system; however, you would need to arrange some tools for managing your cluster nodes, which are built-in domain of servers.

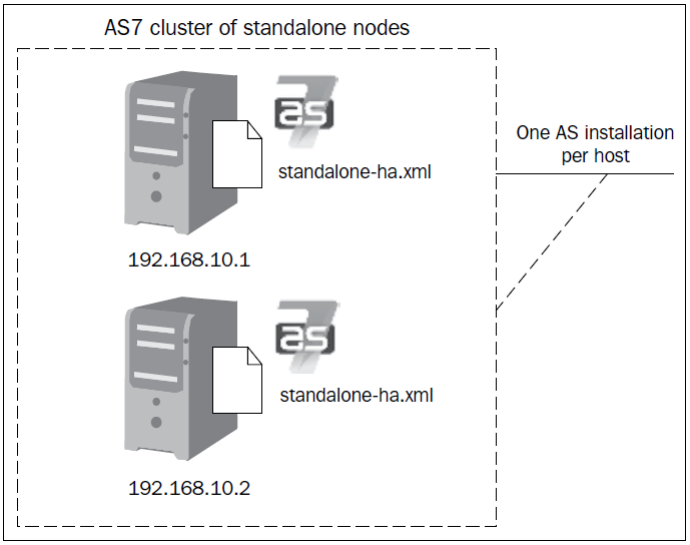
Setting up a Cluster of Standalone Servers

> The possible cluster configurations for standalone servers can be broken in two main scenarios:
  - Cluster of AS nodes running on ***different*** machines
  - Cluster of AS nodes running on the ***same*** machine

Cluster of nodes running on different machines

> If each server application is installed on a dedicated machine you are planning a **horizontal-scaled** cluster.
> In terms of configuration, this one requires the least effort
  1. Bind the server to its IP address in the single standalone servers
  2. start the server using a clustered-compliant configuration

Cluster of nodes running on different machines



Cluster of nodes running on different machines

> Open the standalone-ha.xml file on each AS 7 distribution and reach to the interfaces section.

> Within the nested interface element insert the IP address of the standalone server.

Example: The 1st machine (**192.168.10.1**)

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.10.1"/>
  </interface>
  <interface name="public">
    <inet-address value="192.168.10.1"/>
  </interface>
</interfaces>
```

Example: The 2nd machine (**192.168.10.2**)

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.10.2"/>
  </interface>
  <interface name="public">
    <inet-address value="192.168.10.2"/>
  </interface>
</interfaces>
```

Starting the Cluster

> To start the cluster, you have to indicate that your standalone servers will be using the **standalone-ha.xml** configuration file.

```
standalone.bat --server-config=standalone-ha.xml
```

Starting the Cluster

> Starting from Release 7.0.2, the **–b** option has been resurrected to provide the server binding address.
> In addition, you can use the **–bmanagement** flag to specify the management-interface address.
> The previous configuration can be rewritten for the first server as:

```
standalone.bat --server-config=standalone-ha.xml –b 192.168.10.1
-bmanagement 192.168.10.1
```

> For the second server as:

```
standalone.bat --server-config=standalone-ha.xml –b 192.168.10.2
-bmanagement 192.168.10.2
```

Starting the Cluster

> In a matter of seconds, your servers will be running, however we didn't find any details about clustering nodes in the console, why?

> One of the most evident differences between JBoss AS 7 and earlier releases is that the clustering services are started on demand and stop once you don't need them anymore.

> Hence, simply starting the clustering, without any clustered-enabled application, will not initiate any service or channel.

Starting the Cluster

> In order to verify our installation, we will deploy a bare bones, cluster-enabled, web application named **`example.war`**.

> To enable clustering of your web applications, you must mark them as **`distributable`** in the **`web.xml`** descriptor:

```
<web-app>
   <distributable/>
</web-app>
```

> When deploying the application on both machines, you will find out that clustering services are now started and that each machine is able to find out the cluster members.

Deployment to Cluster

> If you are going to distribute your application to a set of standalone nodes, your best strategy is creating a CLI script that deploys your application to all your server nodes.

> For example, create a file named deploy.cli:

```
connect 192.168.10.1
deploy example.war
connect 192.168.10.2
deploy example.war
```

> Now you can execute the following script

```
jboss-cli.bat --user=username
--password=password --file=deploy.cli
```

Clustering with server nodes on the same machine

> The second variant of the standalone configuration comes into picture when your server nodes are located (all or some of them) on the same machine.

> This scenario applies generally when you are scaling up your architecture by adding more hardware resources to your system.

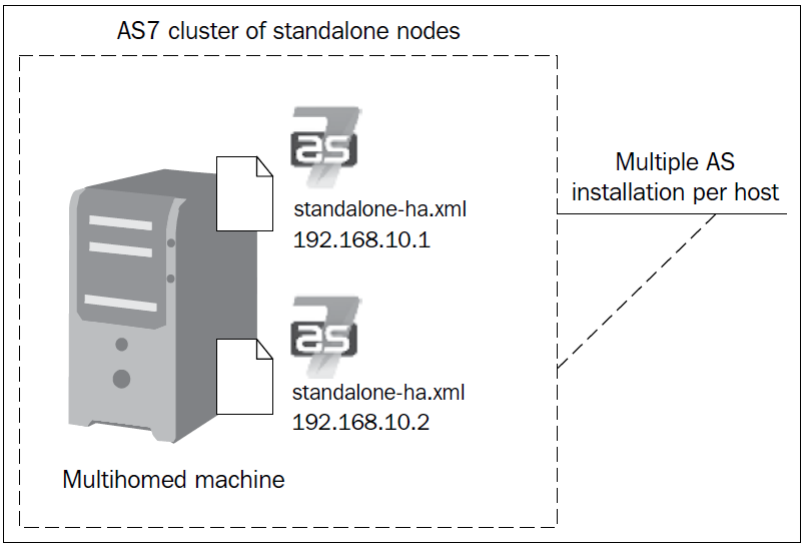
Clustering with server nodes on the same machine

> Configuring server nodes on the same machine obviously requires duplicating your AS7 distribution on your file system.

> In order to avoid a port conflict between server distributions, you have to choose between two options:

1. Defining ***multiple IP*** address on the same machine.
2. Defining a ***port offset*** for each server distribution.

Clustering with server nodes on the same machine

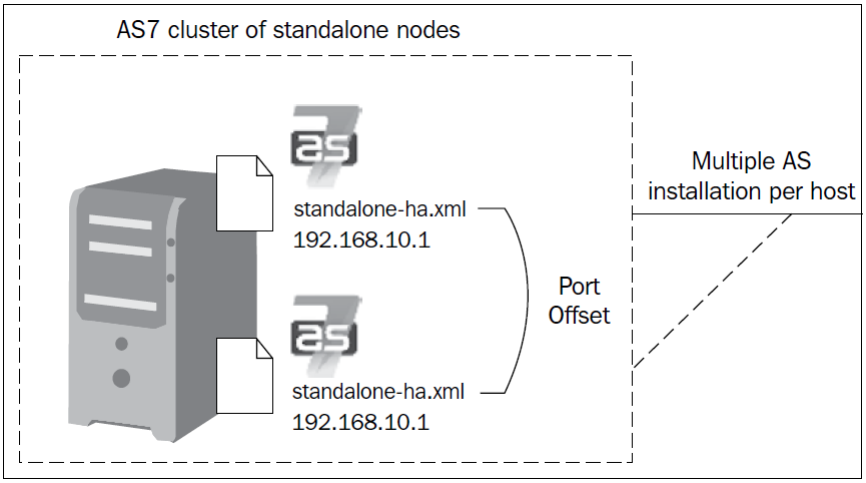


Setting up a cluster on the same machine using port offset

> Configuring multi-homing is not always a viable choice because it requires some degree of administration experience or maybe because you are lacking in permissions on your machine, in which case, you could still set up a clustering configuration by defining a port offset for your each cluster member.

> By defining a port offset for each server, the entire default server binding interfaces will shift to a fixed amount, and hence you will not have two servers engaging the same IP address and the same port.

Setting up a cluster on the same machine using port offset



Setting up a cluster on the same machine using port offset

> When using this configuration, you can bind each server network address to the same IP address.

> For all your server distributions, configure the **standalone-ha.xml** as follows:

```
<interfaces>
  <interface name="management">
   <inet-address value="192.168.10.1"/>
  </interface>
  <interface name="public">
   <inet-address value="192.168.10.1"/>
  </interface>
</interfaces>
```

Setting up a cluster of domain servers

> When you are configuring a domain server, you will find that the clustering subsystem is already part of the main configuration file, **domain.xml**.

> JBoss EAP 7 domain deals with clustering just as a different profile used by the application server.

> Opening the **domain.xml** file, you will see that the application server ships with two profiles:

- The **default profile**, which can be used for *non-clustered* environments
- The **ha profile** for *clustered* environments

Setting up a cluster of domain servers

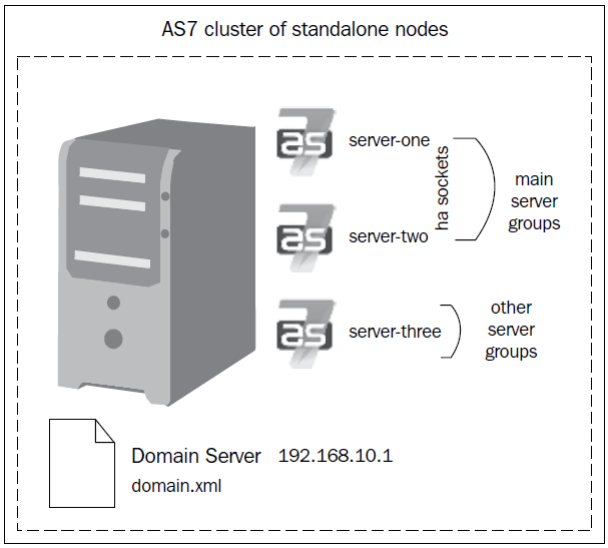
> In order to use clustering on a domain, you have at first to configure your server groups to point to the ***ha profile***.

> Let's build an example configuration which uses two server groups

Setting up a cluster of domain servers

```
<server-groups>
   <server-group name="main-server-group" profile="ha">
      <jvm name="default">
         <heap size="64m" max-size="512m"/>
      </jvm>
      <socket-binding-group ref="ha-sockets"/>
   </server-group>
   <server-group name="other-server-group" profile="ha">
      <jvm name="default">
         <heap size="64m" max-size="512m"/>
      </jvm>
      <socket-binding-group ref="ha-sockets"/>
   </server-group>
</server-groups>
```

Setting up a cluster of domain servers



Troubleshooting clustering

> The clustering communication is carried out, by default, using UDP and multicasts information around the cluster.

> If there are problems, typically it is because of one of the following reasons:

– The nodes are behind a firewall. If your nodes are on different machines then it is possible that the firewall is blocking the multicasts

– You are using a home network or are behind a gateway.

- Typically home networks will redirect any UDP traffic to the Internet Service Provider, which is then either dropped by the ISP or just lost. To fix this, you will need to add a route to the firewall/gateway that will redirect any multicast traffic back to local network

Troubleshooting clustering

- **JGroups** ships with two test programs that can be used to test multicast communication: **McastReceiverTest** and **McastSenderTest**.
- Start **McastReceiverTest**, for example:

```
java -classpath jgroups-3.2.7.Final-redhat-1.jar
org.jgroups.tests.McastReceiverTest
-mcast_addr 224.10.10.10 -port 5555
```

- Then, start **McastSenderTest**:

```
java -classpath jgroups-3.2.7.Final-redhat-1.jar
org.jgroups.tests.McastSenderTest
-mcast_addr 224.10.10.10 -port 5555
```

Troubleshooting clustering

- The **jgroups-3.2.7.Final-redhat-1.jar** can be located in the **JBOSS_HOME**/modules/**system/layers/base/**org/jgroups/main path of your server distribution.

MODULE 11

LOAD BALANCING WEB APPLICATIONS

Objectives

- In the previous module, we studied the first important advantage of clustered applications:
  - The ability to maintain unaffected services for any predefined number of clients.
  - Also known as high availability.
- In this module, we will cover the second main concern of clustering:
  - The ability to make several servers participate in the same service and do the same work.
  - In other words, how to load-balance the number of requests across the available servers.

Content

> First, we will study the advantages of using a web server in front of your web applications
> Then we will study the following topics
  – Connecting JBoss AS 7 using **`mod_jk`** and **`mod_proxy`**
  – Connecting JBoss AS 7 using **`mod_cluster`** API

Using Apache web server with **`JBoss`** AS 7

> In real-world projects, it's common to find Apache web server as a front door to your application server.
> The advantages of such an architecture are
  – **Speed**
    • Apache is generally faster at serving static content than **`JBoss`** Web server.
  – **Security**
    • The application server, which contains sensitive data, can then be placed in a protected area and, from a security point of view, you only need to worry about the Apache server.
    • Apache becomes a smart proxy server.

Using Apache web server with **JBoss** AS 7

> The advantages of such an architecture are

– **Load balancing and clustering**

- By using Apache as a frontend you can handle traffic to multiple **JBoss** Web server instances.
- If one of your **JBoss** AS fails, the communication transparently continues to another node in the cluster.

Using Apache web server with **JBoss** AS 7

Connecting Apache and **JBoss** AS can be done by means of several libraries

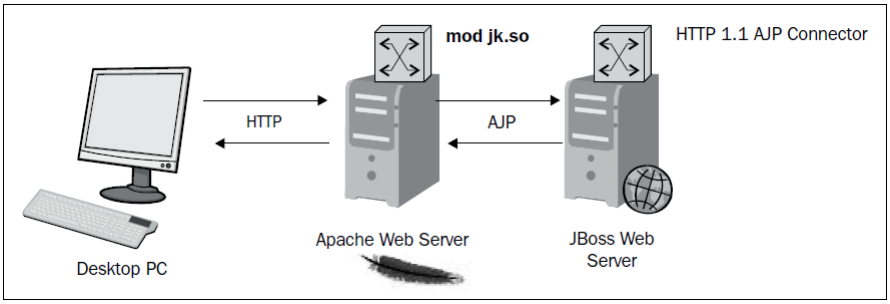
1. In the past, most projects have adopted either Tomcat's **mod_jk** library or Apache's **mod_proxy** libraries.
   – Because the installation of either **mod_jk** or **mod_proxy** does not differ from earlier AS releases, we will just include a quick setup guide for your reference.
2. If you are planning to set up a high-performance, dynamic cluster of web servers, we suggest you migrate to the newer **mod_cluster** API, which is discussed in the second part of this module.

Configuring **mod_jk**

- **mod_jk** is the most-used solution for fronting JBoss AS with Apache web server.
- All requests first come to the Apache web server.
- The Apache web server accepts and processes any static resource requests, such as requests for HTML pages or graphical images.
- Then, with the help of **mod_jk**, the Apache web server redirects requests for any JSP or Servlet component to a **JBoss** Web server instance(s).
- Sending the request over the network using the **AJP** protocol performs this redirection.

Apache - JBoss AS Connectivity

Scenario 1

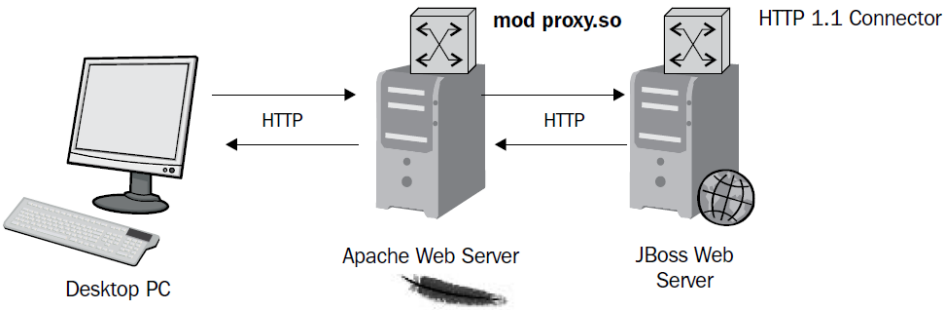


Configuring mod_proxy

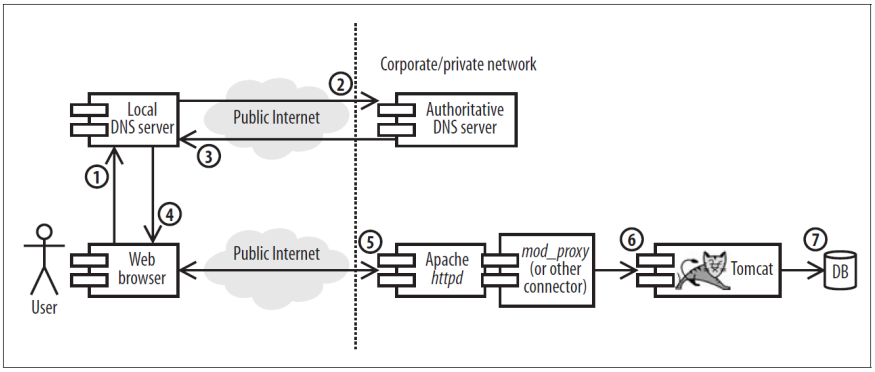
- Since **Apache 1.3**, there's support for an optional module, named **`mod_proxy`**, that configures Apache to act as a proxy server.
- This can be used to forward requests for particular web applications such as Tomcat or **`JBoss`**, without having to configure a web connector such as **`mod_jk`**.

Configuring mod_proxy

Scenario 2



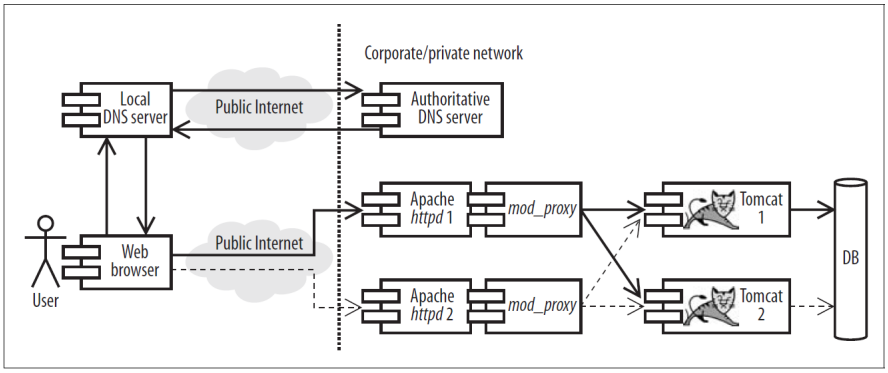
The Communication Sequence of an HTTP Request



Clustering Technologies

- *DNS request distribution*
- *TCP Network Address Translation (NAT) request distribution*
- **`mod_proxy_balancer`** *load balancing and failover*
- *JDBC request distribution and failover*

Clustering Technologies



DNS Request Distribution

> Request distribution can be done at the authoritative DNS server.

> This is a Wide Area Network (WAN) clustering solution that can distribute requests across server machines at one or more data centers.

> When the browser's local DNS asks for an IP address from the web site's authoritative DNS, and there are two machines in the cluster that run web servers, which IP address should the authoritative DNS reply with?

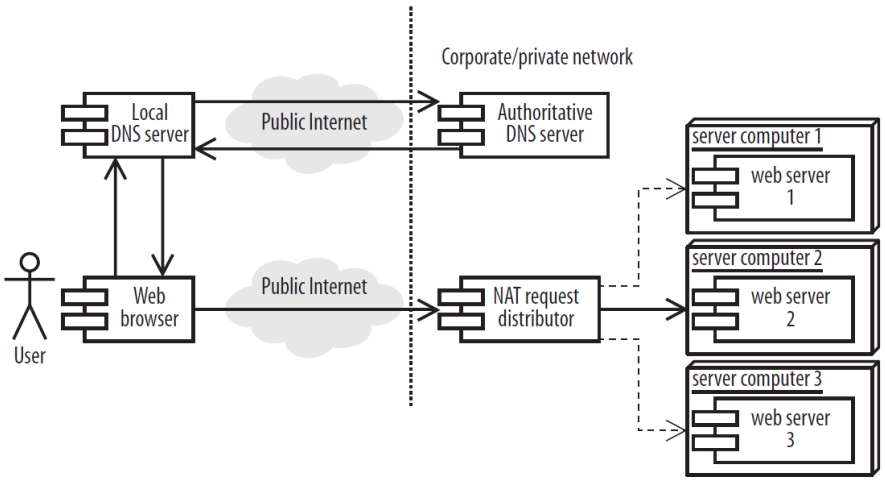
DNS Request Distribution

- DNS can give multiple answers to a single question—it can give both IP addresses to the browser, but the browser will use only one of the addresses.
- To balance the load a bit, most DNS server software will give out the list of IP addresses in a different, circular order every time a request is made.
- This means that no specific IP address stays at the top of the list, and therefore the browsers will use the IP addresses in a circular order (DNS *round-robin).*
- *DNS* round-robin is simple and relatively easy to configure, but it has many drawbacks.

DNS Request Distribution

- *DNS* round-robin is simple and relatively easy to configure, but it has many drawbacks.
  - *It does not take load into account*
  - *It is not fault tolerant*
  - *It knows nothing about congested networks, nor downed network links*

TCP NAT Request Distribution



TCP NAT Request Distribution

> There are many NAT request distributor implementations available, both commercially and as open source software that runs on commodity computer hardware.

> Here are several free implementations:

– *Linux IPTables*

– *The Linux Virtual Server Project's VS-NAT*

http://www.linuxvirtualserver.org

– *IP Filter*

http://cheops.anu.edu.au/~avalon

`mod_proxy` Load Balancing and Failover

> Distribute requests to one or more Tomcat instances
> Detect JBoss instance failure
> Detect when a JBoss instance comes back up after failing
> Manually mark a JBoss instance available or unavailable

`mod_proxy` Load Balancing and Failover

/etc/httpd/conf.d/proxy-balancer.conf

```
<IfModule !proxy_module>
 LoadModule proxy_module modules/mod_proxy.so
</IfModule>
<IfModule !proxy_http_module>
 LoadModule proxy_http_module modules/mod_proxy_http.so
</IfModule>
<IfModule !proxy_balancer_module>
 LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
</IfModule>
<IfModule !status_module>
 LoadModule status_module modules/mod_status.so
</IfModule>
```

mod_proxy Load Balancing and Failover

```
<IfModule proxy_balancer_module>
 ProxyRequests off
<Proxy balancer://tccluster>
 BalancerMember http://tc1:8080 loadfactor=1 max=150 smax=145
 BalancerMember http://tc2:8080 loadfactor=1 max=150 smax=145
 Order Deny,Allow
 Allow from all
</Proxy>
<Location /balancer-manager>
 SetHandler balancer-manager
 Order Deny,Allow
 Allow from all
</Location>
```

mod_proxy Load Balancing and Failover

```
<Location /game>
 ProxyPass balancer://tccluster/game stickysession=jsessionid
 ProxyPassReverse balancer://tccluster/game
 Order Deny,Allow
 Allow from all
</Location>
<Location /examples>
 ProxyPass balancer://tccluster/examples stickysession=jsessionid
 ProxyPassReverse balancer://tccluster/examples
 Order Deny,Allow
 Allow from all
</Location>
</IfModule>
```

Load Balancer Manager

Load Balancer Manager for 10.0.0.8

Server Version: Apache/2.2.17 (Win32) mod_jk/1.2.32
Server Built: Oct 18 2010 01:58:12

LoadBalancer Status for balancer://tccluster

StickySession	Timeout	FailoverAttempts	Method
jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.0.0.8:8080			1	0	Err	6	2.5K	6.9K
http://10.0.0.9:8080			1	0	Ok	3	1.2K	4.5K

Load Balancer Manager

Load Balancer Manager for 10.0.0.8

Server Version: Apache/2.2.17 (Win32) mod_jk/1.2.32
Server Built: Oct 18 2010 01:58:12

LoadBalancer Status for balancer://tccluster

StickySession	Timeout	FailoverAttempts	Method
jsessionid	0	1	byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
http://10.0.0.8:8080			1	0	Err	6	2.5K	6.9K
http://10.0.0.9:8080			1	0	Ok	3	1.2K	4.5K

Edit worker settings for http://10.0.0.8:8080

Load factor: 1
LB Set: 0
Route:
Route Redirect:
Status: Disabled: ○ | Enabled: ◉
Submit

Distributed Java Servlet Containers

- The Java Servlet Specification version 2.2 defined and specified the semantics of distributed servlet containers and the servlet specifications 2.3 through 2.5 further clarified them.
- The specifications define the behavior and leave much of the implementation detail up to the servlet container authors.

Distributed Java Servlet Containers

- Marking the web application as being distributable in the application's **`web.xml`** *file* means that it will be deployed and run in a special way on a distributed servlet container:

```
<distributable/>
```

Session Affinity

- When you have your cluster set up to examine the HTTP session cookie and **`jvmRoute`** and send all dynamic content requests from the same session to the same Tomcat instance, you're using the ***session affinity request distribution model***.
- *A*ll requests from the same session are served by the same Tomcat instance.

Session Affinity

- ***mod_proxy_balancer*** *supports Tomcat session affinity*
- *When Apache httpd forwards a request to* ***mod_proxy_balancer****,* ***mod_proxy_balancer*** *examines the* session cookie and the **jvmRoute** and forwards the request to the same Tomcat instance.
- For web applications that are marked **distributable**, this model is the only model that should be used.
- When all requests belonging to one HTTP session are served by one Tomcat instance, **session replication** is *not necessary* for the application to function under normal circumstances.

Load-balancing with **mod_cluster**

> **mod_cluster** is an HTTP-based load balancer, which, like **mod_jk**, can be used to forward a request to a set of application server instances.

> The advantage of using **mod_cluster** against **mod_jk** and **mod_proxy** can be summarized in these key points:

1. Dynamic clustering configuration.
2. Server-side pluggable load metrics.
3. Lifecycle notifications of application status.

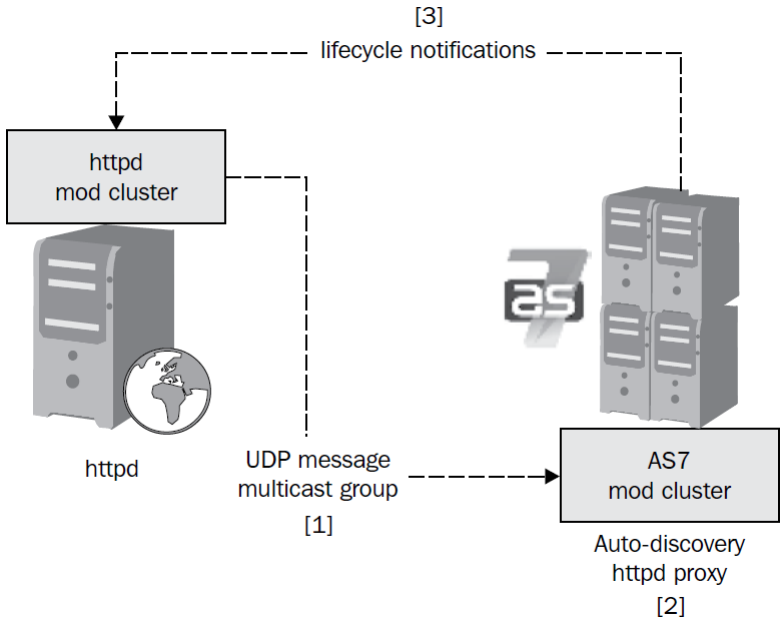
Load-balancing with **mod_cluster**

> When using **mod_cluster**, you are able to dynamically add or remove nodes to your cluster because cluster nodes are discovered through an advertising mechanism.

> In practice, the **mod_cluster** libraries on the httpd side send UDP messages on a multicast group, which is subscribed by AS7 nodes.

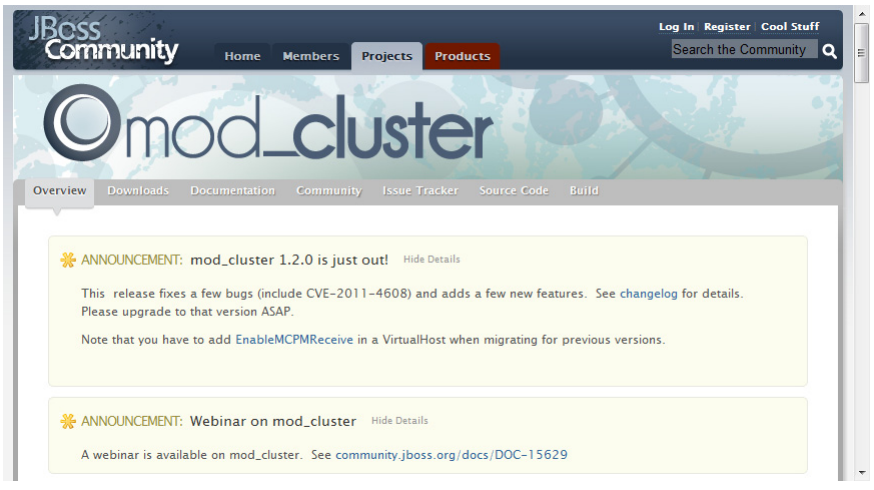
> This allows AS7 nodes the automatic discovery of httpd proxies where application lifecycle notifications are sent.

Load-balancing with `mod_cluster`



`mod_cluster`

> Download **`mod_cluster`** at **`http://www.jboss.org/mod_cluster/`**



Installing `mod_cluster`

> **`mod_cluster`** is implemented as a core AS 7 module, which is a part of the distribution and, on the **httpd** side, as a set of libraries installed on the Apache web server.

> On the **`JBoss AS 7`** side, you can find already bundled the mod cluster's 1.1.3 module as part of the clustered configuration file.

> You can find it either in the standalone-ha.xml file or in the domain.xml configuration file:

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">
    <mod-cluster-config advertise-socket="modcluster"/>
</subsystem>
```

Installing `mod_cluster`

> This is just a bare-bones declaration that references its socket-binding through the advertise-socket element:

```
<socket-binding name="modcluster"
                port="0"
                multicast-address="224.0.1.105"
                multicast-port="23364"/>
```

Installing **mod_cluster**

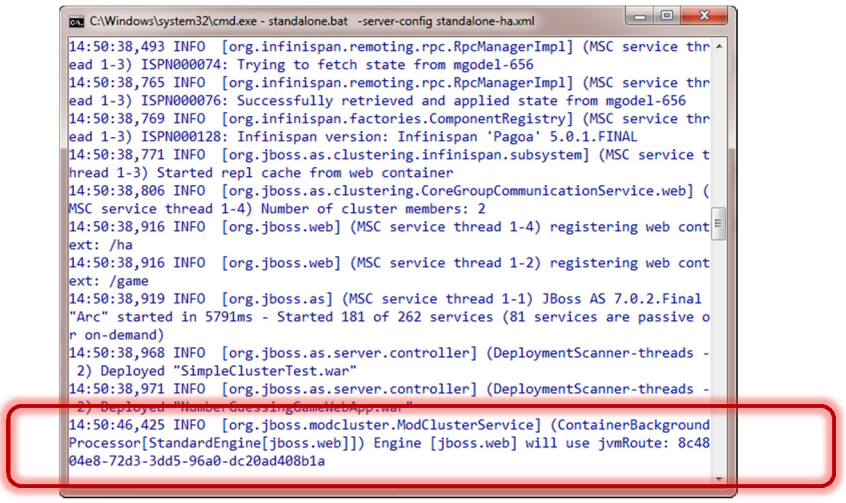
> This is the list of modules that are needed for mod_cluster to get working:

LoadModule **proxy_module** modules/**mod_proxy**.so
LoadModule **proxy_ajp_module** modules/**mod_proxy_ajp**.so
LoadModule **slotmem_module** modules/**mod_slotmem**.so
LoadModule **manager_module** modules/**mod_manager**.so
LoadModule **proxy_cluster_module** modules/**mod_proxy_cluster**.so
LoadModule **advertise_module** modules/**mod_advertise**.so

Installing **mod_cluster**

```
Listen 10.0.0.8:80
<VirtualHost 10.0.0.8:80>
   <Location />
     Order deny,allow
         Allow from all
   </Location>
   KeepAliveTimeout 60
   MaxKeepAliveRequests 0
   ManagerBalancerName mycluster
   EnableMCPMReceive
   ServerAdvertise On
</VirtualHost>
```

> Now, restart Apache web server and the single application server nodes.
> If you have correctly configured the mode cluster on the httpd side, you will see the following message:



Adding native management capabilities

```
<Location /mod_cluster_manager>
     SetHandler mod_cluster-manager
     Order deny,allow
     Deny from all
     Allow from 10.0.0.
</Location>
```

Adding native management capabilities



mod_cluster/1.2.0.Final

Auto Refresh show DUMP output show INFO output

Node 8c4804e8-72d3-3dd5-96a0-dc20ad408b1a (ajp://10.0.0.8:8009):

Enable Contexts Disable Contexts
Balancer: mycluster,LBGroup: ,Flushpackets: Off,Flushwait: 10000,Ping: 10000000,Smax: 65,Ttl: 60000000,Status: OK,Elected: 0,Read: 0,Transferred: 0,Connected: 0,Load: 1

Virtual Host 1:

Contexts:

/ha, Status: ENABLED Request: 0 Disable
/game, Status: ENABLED Request: 0 Disable
/, Status: ENABLED Request: 0 Disable

Aliases:

default-host
localhost
example.com

Node 4e4da0fe-dea1-3ffe-8bf4-40b0fcceb1d7 (ajp://10.0.0.9:8009):

Enable Contexts Disable Contexts
Balancer: mycluster,LBGroup: ,Flushpackets: Off,Flushwait: 10000,Ping: 10000000,Smax: 65,Ttl: 60000000,Status: OK,Elected: 0,Read: 0,Transferred: 0,Connected: 0,Load: 1

Virtual Host 1:

Contexts:

/game, Status: ENABLED Request: 0 Disable
/, Status: ENABLED Request: 0 Disable

Aliases:

default-host
localhost
example.com

MODULE 12

MONITORING AND TUNING JBOSS EAP 7

Tuning JBoss EAP

> A poorly configured server environment can affect your user experience significantly, and eventually, on your application value.
> The amount of configuration elements, which can influence your server performance are quite a lot
> However, some of them deserve a special attention:
  – JVM tuning
  – Application server resource pooling
  – Logging
  – Caching data

JVM Tuning

> The JBoss EAP 7 runs within a Java Virtual Machine (JVM), hence it's natural that the AS can give a better performance with proper configuration of the JVM parameters.
> JVM tuning has evolved over the years and actually changed with each version of Java. Since the release 5.0 of the J2SE, the JVM is able to provide some default configuration ("Ergonomics") which is consistent with your environment.
> However, the smarter choice provided by Ergonomics is not always the optimal and without an explicit user setting, the performance can fall below your expectations.

JBOSS EAP 7 PERFORMANCE TUNING

JVM TUNING

Objectives

> After completing this lesson, you should be able to:

- Describe HotSpot generational garbage collector components and operation
- Monitor the JVM garbage collector with command-line tools
- Monitor the JVM garbage collector with GUI tools like JConsole and VisualVM
- Monitor the JVM JIT compiler
- Define application throughput and responsiveness

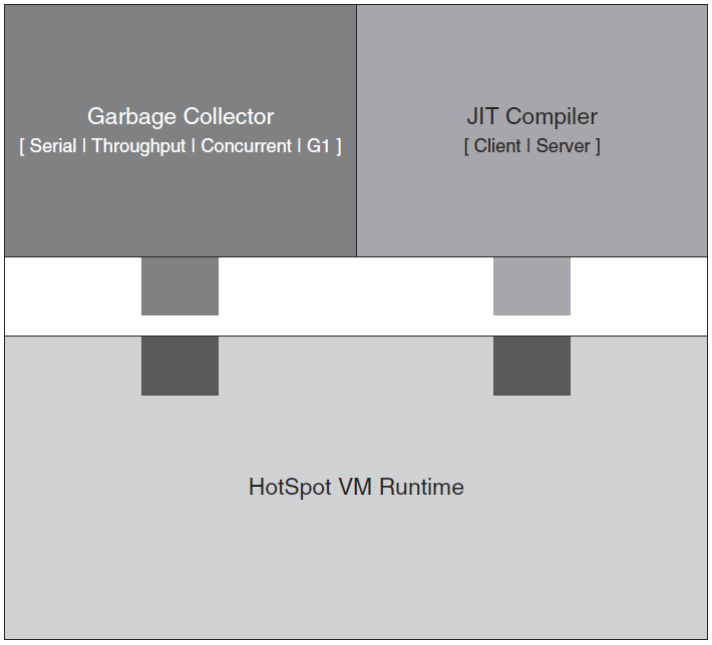
What to Monitor

- The two major areas to monitor at the JVM level:
  - Garbage collector
  - JIT compilation
- Data of interest
  - Frequency and duration of collections
  - Java heap usage
  - Number of application threads

JVM

- Since its introduction in 1995, Java Virtual Machines has evolved substantially
  - The integration of JIT compilers
  - More sophisticated garbage collectors,
  - Improvements in the JVM runtime environment
- Even with the many enhancements added to modern JVMs, performance and scalability remain important to application stakeholders
- One of challenges introduced by modern JVMs is many users of Java technology see a JVM as a black box, which can make it a difficult task to improve the performance or scalability of a Java application

HotSpot VM High Level Architecture



Pluggable Architecture

- Garbage Collector
  - Serial GC
  - Throughput
  - Concurrent
  - G1
- JIT Compiler
  - Client
  - Server

HotSpot architecture

> HotSpot is Sun's Java Virtual Machine technology, the foundation for Java Standard Edition
> HotSpot technology: the VM ***continuously analyses*** the program performance for **hot spots**, frequent part of the program which are executed
> Hot spots are then ***highly optimized***, excluding the less performance code

HotSpot architecture

> HotSpot **Client** VM: used for light applications, ***reducing*** **memory footprint** and **startup time**
> HotSpot **Server** VM: used for enterprise applications, ***maximizing*** the **speed**

> Both options: -server or -client ***share the same*** HotSpot runtime, but use ***different compilers*** internally !

HotSpot architecture

> ***-client***

does less computation, analyzing and compiling the code, resulting in a ***faster startup***

> ***-server***

advanced static analysis and compilation techniques resulting in ***faster execution time*** for ***long lasting*** applications

32-bit JVM

```
cmd> java -version
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) Client VM (build 23.25-b01, mixed mode, sharing)
cmd> java -client -version
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) Client VM (build 23.25-b01, mixed mode, sharing)
cmd> java -server -version
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) Server VM (build 23.25-b01, mixed mode)
```

64-bit JVM

cmd> java -version

java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) **64-Bit Server** VM (build 23.25-b01, mixed mode)

cmd> java -client -version

java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) **64-Bit Server** VM (build 23.25-b01, mixed mode)

cmd> java -server -version

java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) **64-Bit Server** VM (build 23.25-b01, mixed mode)

JVM Modes

> **-Xmixed**
mixed mode execution (default)

> **-Xint**
interpreted mode execution only

> **-Xcomp**
compiled mode execution only

Sharing

> **`-Xshare:off`**
  do not attempt to use shared class data
> **`-Xshare:auto`**
  use shared class data if possible (default)
> **`-Xshare:on`**
  require using shared class data, otherwise fail.

Heap Size

> 32-bit JVM
  – Windows
    • 1.5 GB
  – Linux
    • 2.5 GB (Older kernels)
    • 3.0 GB (Recent kernels)
  – Solaris
    • 3.3 GB

Heap Size

> 64-bit JVM
  - Much larger amounts of memory
  - A performance penalty due to an increase in size of the internal HotSpot VM's representation of Java objects:
    - ***O**rdinary **o**bject **p**ointers* (***oops***)
    - Decreases CPU cache efficiency by **8%** to **15%**
  - **-XX:+UseCompressedOops**
    - yield 32-bit JVM performance with the benefit of larger 64-bit Java heaps
    - pack a 64-bit pointer into 32 bits by relying on alignment and possibly having an offset
    - Until heap size up to 32GB

HotSpot VM Runtime

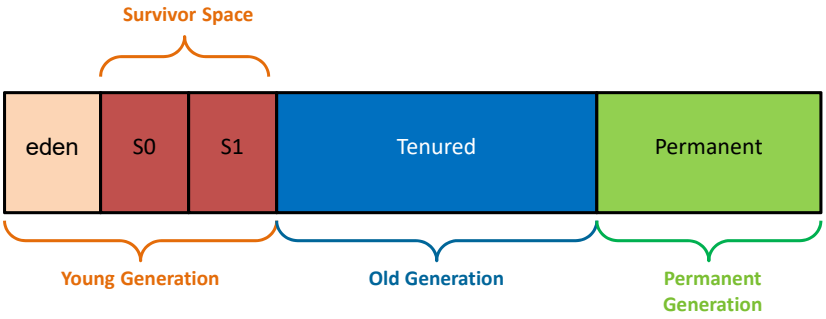
> The VM Runtime is an ***often overlooked*** part of the HotSpot VM
  - Garbage collectors and JIT compilers tend to get more attention than the VM Runtime

> However, the VM Runtime provides the ***core functionality*** of the HotSpot VM
  - **command line arguments**,
  - **VM life cycle**,
  - **class loading**,
  - **byte code interpreter**,
  - **exception handling**,
  - **synchronization**
  - **thread management**,
  - **Java Native Interface**,
  - **VM fatal error handling**,
  - **C++ heap management**

HotSpot GC Basics

– HotSpot uses what is termed “generational collectors.”

– HotSpot Java heap is allocated into generational spaces.

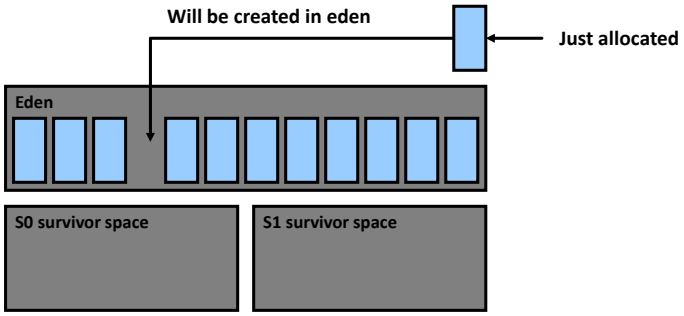


The Young GC Process: Part 1

> New objects are allocated to the eden space.

> When eden space is full, a minor garbage collection is triggered:

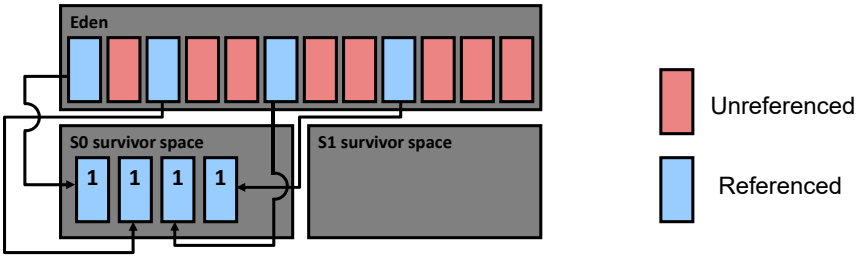
– “Stop the World” event



The Young GC Process: Part 2

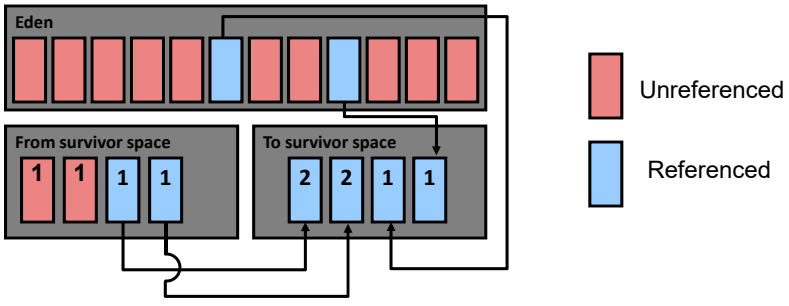
> Unreferenced objects are garbage collected.

> Referenced objects are copied to survivor space and have their age incremented.



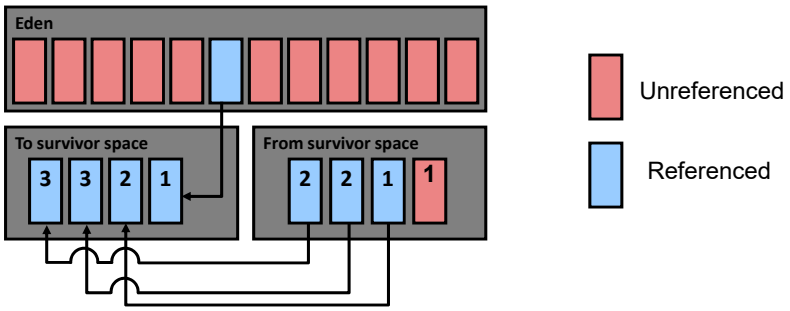
The Young GC Process: Part 3

> Next minor GC
  - Referenced objects from last GC become “from” survivor space.
  - Referenced objects are copied to the “to” survivor space.
  - Surviving object ages are incremented.



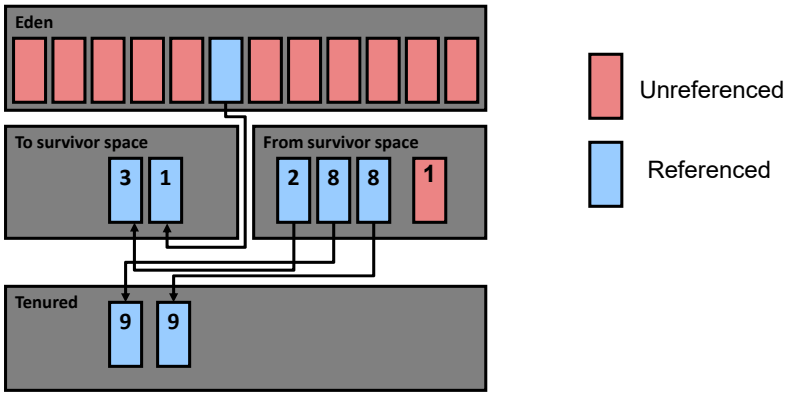
The Young GC Process: Part 4

> Next minor GC
  – Referenced objects from last GC become “from” survivor space.
  – Referenced objects are copied to the “to” survivor space.
  – Surviving object ages are incremented.



The Young GC Process: Part 5

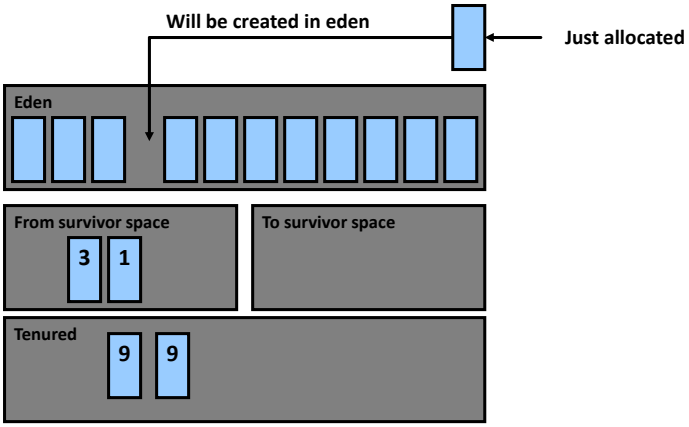
> Promoted to Old Space
  – When age threshold is reached, objects are eventually promoted to tenured space.



The Young GC Process: Summary

> Process repeats at each minor GC.

> When objects reach an age threshold, they are copied to old generation.



Young Generation Recap



> Young generation
  – Eden
  – Survivor spaces
  – Objects age here

> Minor garbage collections are always “Stop the World” events

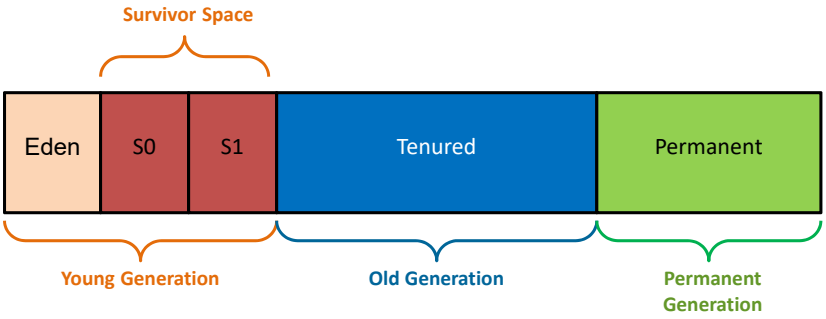
> Minor garbage collections can be
  – Single-threaded events
  – Multithreaded (Parallel) events

Tenured (old) Generation

> Contains objects that have reached the minor collection age threshold

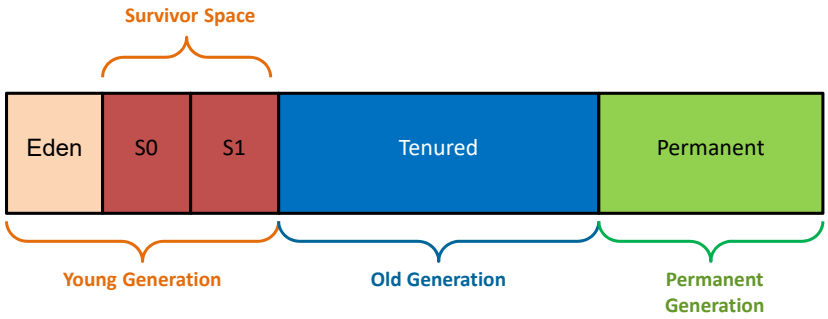
> Full garbage collections can be:

- Single-threaded
- Mostly concurrent
- Multithreaded (Parallel)



Permanent Generation

> Contains metadata required by the JVM:
  - Class objects and methods
  - Interned Strings



Tools for Monitoring GC

> **-verbose:gc**
> **-XX:+PrintGCTimeStamps**
> **-XX:+PrintGCDetails**
> **-XX:+PrintGCApplicationStoppedTime**
> **-XX:+PrintGCApplicationConcurrentTime**
> **jstat, jps**
> **jconsole**
> Java VisualVM - **jvisualvm**
> VisualGC
> DTrace (Java HotSpot JDK 6 contains samples.)

Using -verbose:gc



> What does **-verbose:gc** do?
> What to look for?

```
host:~ $ java -client -verbose:gc -Xmx12m -Xms3m -Xmn1m
-XX:PermSize=20m -XX:MaxPermSize=20m
-jar /usr/java/demo/jfc/Java2D/Java2Demo.jar
...
[GC 1884K->1299K(5056K), 0.0031820 secs]
...
```

> Adding **-XX:+PrintGCTimeStamps**

```
host:~ $ java -client -verbose:gc -XX:+PrintGCTimeStamps
 -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m
-jar /usr/java/demo/jfc/Java2D/Java2Demo.jar
...
3.791: [GC 1884K->1299K(5056K), 0.0031820 secs]
...
```

Additional -verbose:gc Print Options

> **-XX:+PrintGCDateStamps** (Java 6u4 and later)

```
host:~ $ java -client -verbose:gc -XX:+PrintGCDateStamps
 -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m
-jar /usr/java/demo/jfc/Java2D/Java2Demo.jar
...
2008-06-10T06:12:47.513-0500: [GC 10308K->2725K(101376K),
0.0320270 secs]
...
```

> **-XX:+PrintGCDateStamps** and **XX:+PrintGCTimeStamps**

> **PrintGCTimeStamps** output after date & time.

```
host:~ $ java -client -verbose:gc -XX:+PrintGCDateStamps
-XX:+TimeGCDateStamps -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m
-XX:MaxPermSize=20m -jar /usr/java/demo/jfc/Java2D/Java2Demo.jar
...
2008-06-10T06:21:29.711-0500: 5.551:[GC 10377K->2786K(101376K),
0.0271350 secs]
...
```

Using -XX:+PrintGCDetails

> **-XX:+PrintGCDetails**

```
host:~ $ java -client -XX:+PrintGCDetails -Xmx12m -Xms3m -Xmn1m
-XX:PermSize=20m -XX:MaxPermSize=20m
-jar /usr/java/demo/jfc/Java2D/Java2Demo.jar
...
[GC [DefNew: 490K->64K(960K), 0.0032800 secs] 5470K->5151K(7884K),
0.0033270 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

[Full GC (System) [Tenured: 5087K->5151K(6924K), 0.0971070 secs]
6047K->5151K(7884K), [Perm : 11178K->11178K(16384K)],
0.0972120 secs] [Times: user=0.10 sys=0.01, real=0.10 secs]

...
```

Printing Pause Time

> `-XX:+PrintGCApplicationStoppedTime`
> `-XX:+PrintGCApplicationConcurrentTime`
> Helpful when tuning pause time–sensitive applications

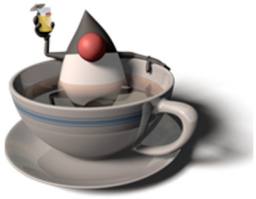


Using **`jps`**

`jps`

> Command-line utility to find running Java processes
> Included in the HotSpot JDK
> Capable of local and remote monitoring
> Example:

```
jps [-q] [-mlvV] [<hostid> where <hostid> =
  <hostname>[:<port>]
```



Using **jps**

> -q

Suppress the output of the class name, JAR file name, and arguments passed to the main method, producing only a list of local VM identifiers.

> -m

Output the arguments passed to the main method. The output may be null for embedded JVMs.

> -l

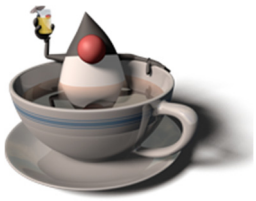
Output the full package name for the application's main class or the full path name to the application's JAR file.

> -v

Output the arguments passed to the JVM.

Using **jps**

```
> jps
> jps -l
> jps -m
> jps -v
> jps -q
```



Using `jcmd`

- The **`jcmd`** command is new in JDK 7 and provides many of the features of **`jps`** plus some additional information as well.
- The **`jcmd`** command allows you to query various aspects of specific virtual machines.

Using `jcmd`

```
> jcmd
> jcmd -l
> jcmd 10672 VM.version
> jcmd 10672 VM.system_properties
> jcmd 10672 VM.flags
> jcmd 10672 VM.uptime
> jcmd 10672 GC.class_histogram > tmp.txt
> jcmd 7140 PerfCounter.print
```

Using **jstat**

jstat

> Command-line utility that runs on a local or remote JVM

> Included in the HotSpot JDK

> Example:

```
jstat -<option> [-t] [-h<lines>] <vmid> [<internal>
  [<count>]]
```

> Garbage collection options:

```
-gc, -gccapacity, -gccause, -gcnew, -gcnewcapacity, -
  gcold, -gcoldcapacity, -gcpermcapacity, -gcutil
```



Using **jstat**

jcmd

jstat -gcutil 7140 5000

jstatd.policy:

```
grant codebase "file:${java.home}/../lib/tools.jar"{
    permission java.security.AllPermission;
};
```

jstatd -J-Djava.security.policy=jstatd.policy

jstat -gcutil 2874@remotehost 5000

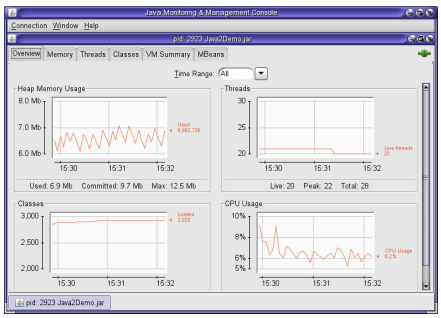


Using `jconsole`

`jconsole`

> Graphical Monitoring and Management Console
  - Included in the HotSpot JDK
  - Can attach multiple JVMs, locally or remotely
  - Graphical view of
    - heap memory
    - threads
    - CPU usage
    - class loading

> Multiple `jconsole` sessions can attach to a single JVM



Using **`VisualVM`**

`VisualVM`

> Graphical JVM monitoring tool

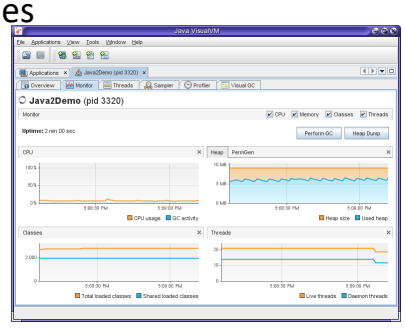
> Included with JDK 6 as of update 7

> Open source project

> Integrates with other JDK tools

> Includes analysis and troubleshooting abilities

> Can be enhanced with plugins
  - Plugin Center
  - https://visualvm.dev.java.net



Using `VisualGC`

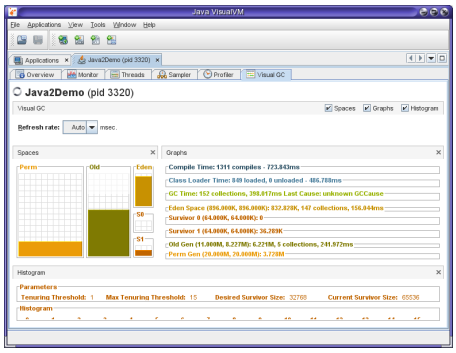
`VisualGC`

> Stand-alone GUI or VisualVM plug-in

> Not included in HotSpot JDK

> Visually observes garbage collection behavior

> Also includes class loading and JIT compilation information



Oracle Java Mission Control

> The former JRockit Mission Control (JRMC) is now called Java Mission Control (JMC)

> Tools suite
  - Monitor
  - Profiler
  - Eliminates memory leaks in your Java application without

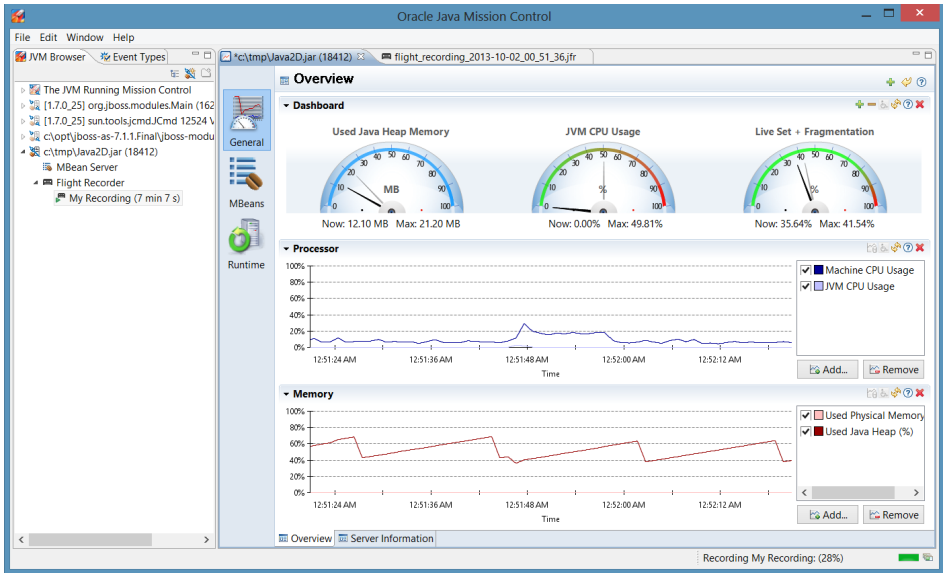
> Available in jdk7u40

> Java Flight Recorder (JFR)

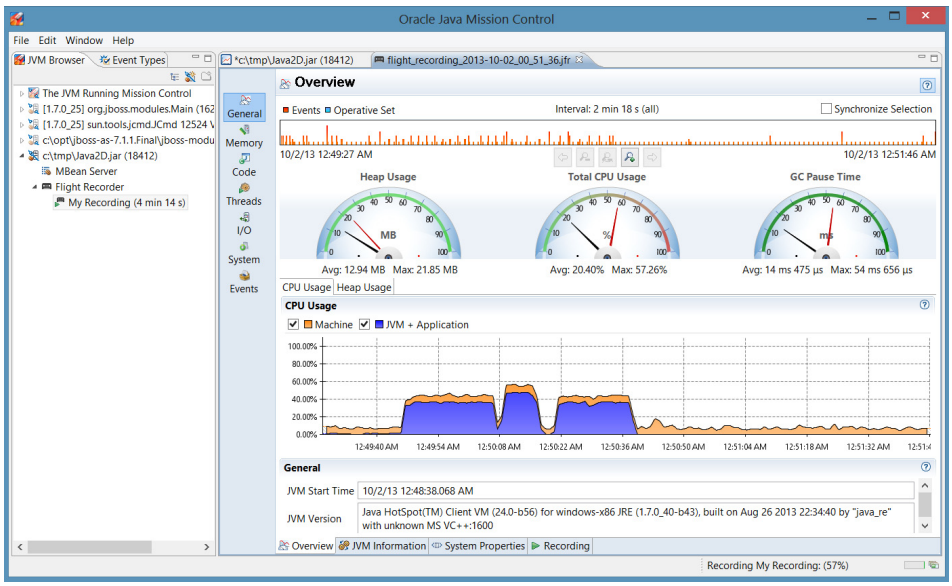
> **`-XX:+UnlockCommercialFeatures`**

> **`-XX:+FlightRecorder`**

Oracle Java Mission Control

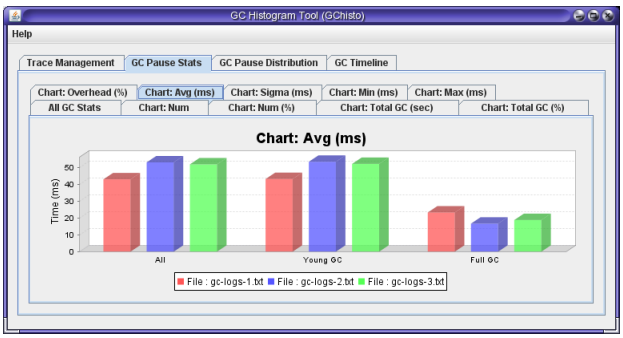


Oracle Java Mission Control



Using `GCHisto`

- Stand-alone GUI
- Open Source project: http://gchisto.dev.java.net
- Not included with JDK
- Summarizes GC activity obtained from GC logs
  - Can Compare Multiple Logs



What is the `.hotspot_compiler` File?

- File turns off JIT compilation in specific methods
  - Used very rarely
- When to use the `.hotspot_compiler` file
  - JIT compiler in an endless loop attempting a “heroic” optimization which will not converge
  - JIT compiler in a de-optimization – re-optimization cycle
  - JIT compiler producing “bad” code resulting in a core dump or other severe problem

Using `.hotspot_compiler` File

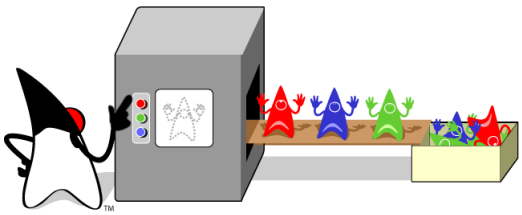
- The `.hotspot_compiler` file must be placed in the directory where the Java command is launched.
- The `.hotspot_compiler` file format
  - `exclude A/B/C/D methodName`
  - `exclude java/util/HashMap clear`
- Can specify on the command line

```
-XX:CompileCommand=exclude, java/util/HashMap,clear
```



Focusing on Throughput

- Throughput
  - Have as highest priority the raw throughput of the information or data being processed
  - Maximize application throughput even at the expense of responsiveness
  - Will tolerate high pause times in order to maximize throughput



Focusing on Responsiveness

- Responsiveness
  - Have as highest priority the servicing of all requests within a predefined maximum time
  - Raw throughput of data or speed of processing requests are secondary to max response time goal
  - Are sensitive to GC pause time



Command Line Options

- The HotSpot VM Runtime parses the many command line options and configures the HotSpot VM based on those options
- A number of command line options and environment variables can affect the performance characteristics of the HotSpot VM
- Some of these options are consumed by the HotSpot VM launcher such as
  - the choice of JIT compiler
  - choice of garbage collector
  - Java heap Size

Command Line Options

- There are three main categories of command line options:
  - standard options,
  - nonstandard options,
  - developer options

Standard command line options

- Standard command line options are expected to be accepted by all Java Virtual Machine implementations as required by the *Java Virtual Machine Specification*
- Standard command line options are stable between releases.
- However, it is possible for standard command line options to be deprecated in subsequent releases after the release in which it was first introduced.

Nonstandard command line options

- Nonstandard command line options begin with a **-X** ***prefix***.
- Nonstandard command line options are
  - not guaranteed to be supported in all JVM implementations,
  - not required to be supported in all JVM implementations.
- Nonstandard command line options are also subject to change without notice between subsequent releases of the Java SDK

Developer command line options

- Developer command line options in the HotSpot VM begin with a **–XX** ***prefix***.
- Developer command line options often have specific system requirements for correct operation and may require privileged access to system configuration parameters.
- Like nonstandard command line options, developer command line options are also subject to change between releases without notice

Developer command line options

> For developer command line options (-XX options) with boolean flags, a + or - before the name of the options indicates a true or false value, respectively, to enable or disable a given HotSpot VM feature or option.

> For example,

-XX:+AggressiveOpts

sets a HotSpot internal boolean variable to ***true*** to ***enable*** additional performance optimizations.

-XX:-AggressiveOpts

sets the same internal variable to ***false*** to ***disable*** additional performance optimizations.

Developer command line options

> Developer command line options (the **-XX** options) that take an additional argument

-XX:OptionName=*N*

where ***N*** is some ***numeric*** value.

> Almost all developer command line options that take an additional argument, accept an integer value along with a suffix of **k** (kilo), **m** (mega), or **g** (giga)

> There is also a small set of developer command line options that accept data passed in directly after the name of the flag without any delineation.

– The approach depends on the particular command line option and its parsing mechanism.

VM Life Cycle

- The HotSpot VM Runtime is responsible for launching the HotSpot VM and the shutdown of the HotSpot VM.
- The component that starts the HotSpot VM is called the launcher
- There are several HotSpot VM launchers
  - the **`java`** command on Unix/Linux
  - the **`java`** and **`javaw`** commands on Windows.
  - the **`javaws`** which is used by Web browsers to launch applets
    - **`ws`** stands for web start

VM Life Cycle

The launcher executes a sequence of operations to start the HotSpot VM

1. Parse command line options.
   - Some of the command line options are consumed immediately by the launcher such as -client or – server, which determines the JIT compiler to load.
   - Other command line options are passed to the launched HotSpot VM.
2. Establish the Java heap sizes and the JIT compiler type (client or server) if these options are not explicitly specified on the command line.
   - If Java heap sizes and JIT compiler are not explicitly specified as a command line option, these are ergonomically established by the launcher.

VM Life Cycle

3. Establish environment variables such as **`LD_LIBRARY_PATH`** and **`CLASSPATH`**.
4. If the Java **`Main-Class`** is not specified on the command line, the launcher fetches the **`Main-Class`** name from the JAR's manifest
5. Create the HotSpot VM using the standard Java Native Interface method **`JNI_CreateJavaVM`** in a newly created nonprimordial thread.
6. Once the HotSpot VM is created and initialized, the Java **`Main-Class`** is loaded and the launcher gets the Java main method's attributes from the Java **`Main-Class`**.
7. The Java main method is invoked in the HotSpot VM using the Java Native Interface method **`CallStaticVoidMethod`** passing it the marshaled arguments from the command line.

How to print the list of all options

> -XX:+PrintFlagsFinal

> -XX:+PrintFlagsInitial

> -XX:+UnlockExperimentalVMOptions

> -XX:+UnlockDiagnosticVMOptions

```
java -XX:+UnlockDiagnosticVMOptions
     -XX:+PrintFlagsFinal -version
```

Monitoring JIT Compilation

> Tools for monitoring JIT Compilation
  - **`jstat`**
  - **`jconsole`**
  - VisualVM
  - VisualGC
  - **`-XX:+PrintCompilation`** (can be mildly intrusive, 2% CPU)
  - DTrace (HotSpot JDK 6 contains samples)



> Data of interest
  - Frequency, duration, possible opt/de-opt cycles, failed compilations

Using `-XX:+PrintCompilation`

> **`-XX:+PrintCompilation`** Sample
  - Blue text added to output
  - Shows an opt/de-opt cycle

```
1 java.util.Properties$LineReader::readLine (452 bytes)
2 java.lang.String::hashCode (60 bytes)
3 java.lang.String::equals (88 bytes)
3 made not entrant  (2)  java.lang.String::equals (88 bytes)
4 java.lang.Object::<init> (1 bytes)
5 java.lang.String::indexOf (151 bytes)
6 java.lang.String::equals (88 bytes)  <---------- redoing 3
6 made not entrant  (2)  java.lang.String::equals (88 bytes)
7 java.lang.String::indexOf (151 bytes)
8 java.lang.String::equals (88 bytes)  <---------- redoing 6
8 made not entrant  (2)  java.lang.String::equals (88 bytes)
...
```

-XX:+PrintAssembly

> **-XX:+PrintAssembly**

prints assembly code for bytecoded and native methods

> **-XX:+PrintNMethods**

prints native methods as they are generated

> **-XX:+PrintNativeNMethods**

prints native method wrappers as they are generated

> **-XX:+PrintSignatureHandlers**

prints native method signature handlers

> **-XX:+PrintAdapterHandlers**

prints adapters (i2c, c2i) as they are generated

-XX:+PrintAssembly

> **-XX:+PrintStubCode**

prints stubs: de-opt, uncommon trap, exception, safepoint, runtime support

> **-XX:+PrintInterpreter**

prints interpreter code

PrintAssembly

> Put the dll (**hsdis-i386.dll**) on the classpath

```
java -XX:+UnlockDiagnosticVMOptions
     -XX:+PrintAssembly
     -jar Java2D.jar
```



GARBAGE COLLECTION SCHEMES

Objectives

> After completing this lesson, you should be able to describe:
  - What garbage collection is
  - The advantages of generational garbage collectors over non-generational garbage collectors
  - GC performance metrics
  - The various garbage collection algorithms
  - Various types of garbage collectors
  - JVM Ergonomics

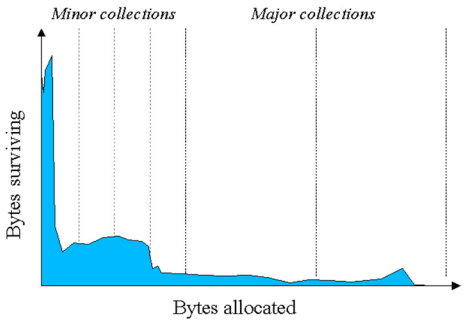
Garbage Collection

> Garbage = memory occupied by dead objects (unreachable)
> Garbage collection = Reclamation of garbage
> Invented by John McCarthy in 1958 as part of Lisp
> Available in Java, Modula3, Prolog , SmallTalk etc

Garbage Collection

> Generational Hypothesis
  – Young objects are more likely to die than old objects (infant mortality)
> Basis for generational GC algorithms which divide the heap into New and Old segments based on object age.
> Entire heap does not need to be garbage collected at every instance.

Garbage Collection



Garbage Collection Basics

> Garbage collection is a way in which Java recollects the space occupied by loitering objects.

> In general, a garbage collector is responsible for three tasks:

– Allocating memory for new objects

– Ensuring that any referenced objects (live objects) remain in memory

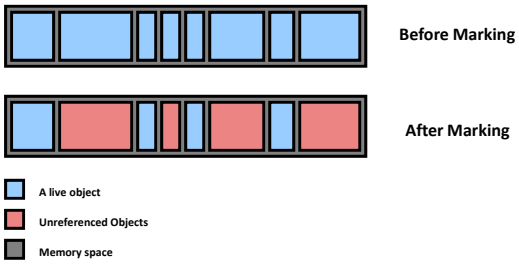
– Recovering memory used by objects that are no longer reachable (dead objects)



Garbage Collection Basics

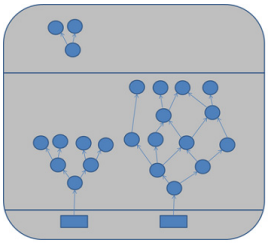
> The garbage collector first performs a task called marking.

> Each object the garbage collector meets is marked as being used, and will not be deleted in the sweeping stage.



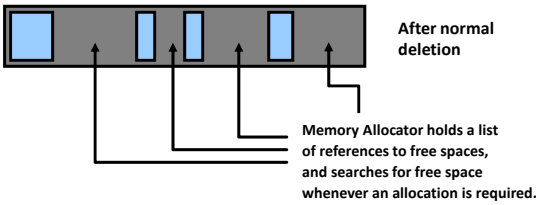
Root Objects

> A simple Java application has the following GC roots:
  - System classes (which have static fields)
  - Thread stacks
  - JNI global references
  - The finalizer queue
  - The interned String pool



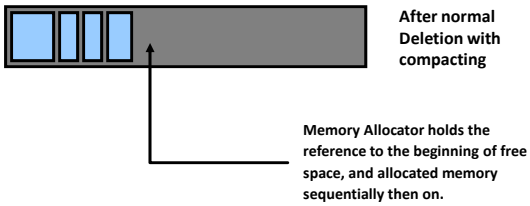
Normal Deletion

> The *sweeping* stage is where the deletion of objects take place.
> The traditional way is to let the allocator methods use complex data structures to search the memory for the required free space.



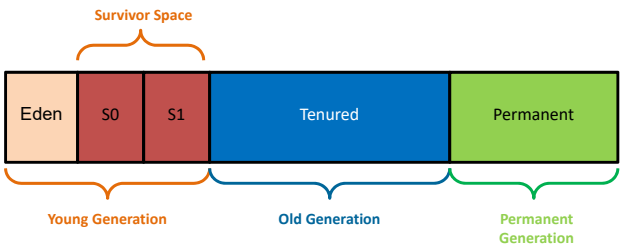
Deletion with Compacting

> Compact the memory by moving objects close to each other.
> Object allocation is faster.



Generational Garbage Collection

> HotSpot uses what is termed “generational collectors.”
> HotSpot Java heap is allocated into generational spaces.



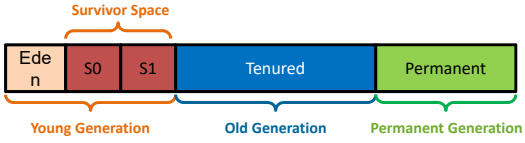
Why Generational GC?

> Non-Generational GC will just iterate over every object in the heap:
  – Increased objects reduces performance
  – Ignores typical object behavior
> Generational GC improves upon this.



Generational Garbage Collection: Major Spaces

> The memory space is divided into three sections:
> Young generation (for young objects)
  – Eden
  – A "from" survivor space
  – A "to" survivor space
> Tenured (old) generation (for old objects)
> Permanent generation (meta data, classes, and so on)



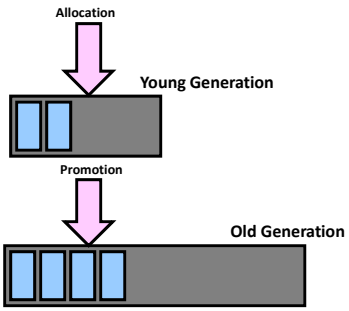
Features of Young Generational Space

- GCs occur relatively frequently.
- GCs are fast and efficient because young generation space is usually small and likely to contain a lot of short-lived objects.
- Objects that survive some number of young generation collections are promoted to old generation heap space.

Features of Old Generation Space

- Typically larger than young generation heap space
- Its occupancy grows more slowly.
- GCs are infrequent but takes significantly longer time to complete than young generational heap space.
  - GCs in old generation space should be minimized.

Generational Garbage Collection



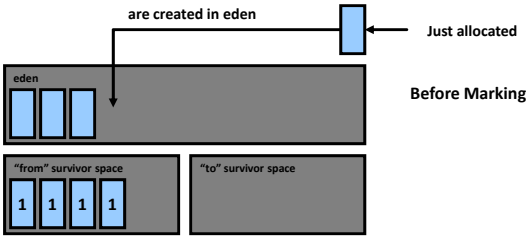
Garbage Collection Notations

> When the young generation space is full, and young generation GC occurs:
  – Young generation GC is called minor GC or nursery GC.
> When the old generation does not have enough space for the objects that are being promoted, a full GC occurs:
  – Full GC is also called major GC.

Generational Garbage Collection: Young Collection

> Young generation area consists of an area called eden plus two smaller survival spaces.

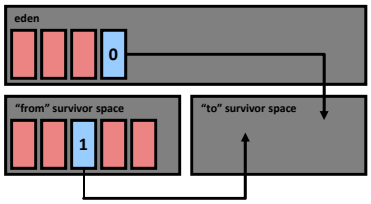
> Newly created objects are created in eden.



Generational Garbage Collection: Young Collection

> When the eden space is full, a minor garbage collection event occurs. Live objects in the eden space are copied to a "to" a survivor space.

> Additionally, objects in the "from" survivor space are copied to a "to" survivor space.



GC Performance Metric

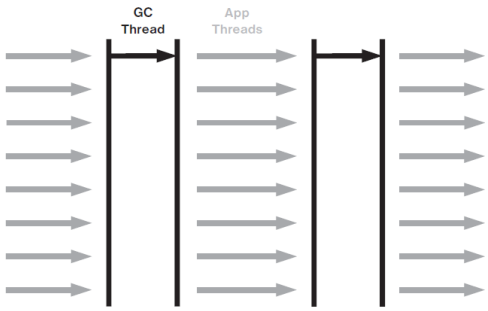
> The following are the different ways to measure GC Performance:
  - Throughput – % of time not spent in GC over a long period of time
  - Responsiveness – application unresponsive because of GC
  - Footprint – overall memory a process takes to execute
  - Promptness – time between object death, and time when memory becomes available
> There are many right ways to size generations, so make the call based on your applications usage.

Choices of Garbage Collecting Algorithms

> Serial versus Parallel
> Stop the World versus Concurrent
> Compacting versus Non-compacting versus Copying

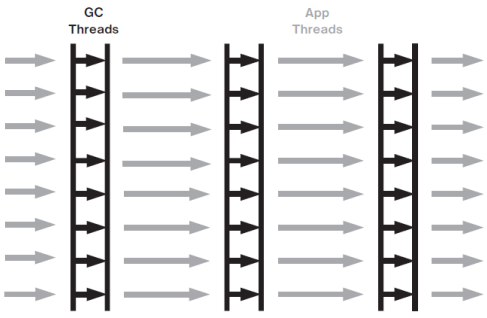
Serial GC

> In the serial collector, one hardware thread handles the GC task



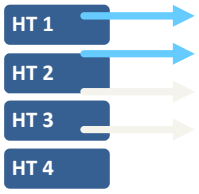
Parallel GC

> In the parallel collector, multiple hardware threads handle the GC task simultaneously.

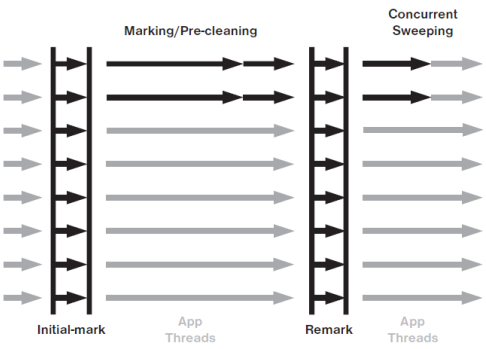


Stop the World Versus Concurrent

> Stop the World
  - Execution of the application is completely suspended during garbage collection
  - Simpler to implement
  - More pause time
> Concurrent (Mostly)
  - One or more GC tasks
    - can be executed concurrently
    - with the application.
  - Less pause time
  - Additional overhead



Concurrent GC



Compacting Versus Non-Compacting Versus Copying

- Compacting
  - Move all live objects together and completely reclaim the remaining memory
- Non-compacting
  - Release the space “in-place”
- Copying (sometimes also called Evacuating)
  - Copies objects to a different memory area
  - The source area is empty and available for fast and easy further allocations.

Types of GC Collectors

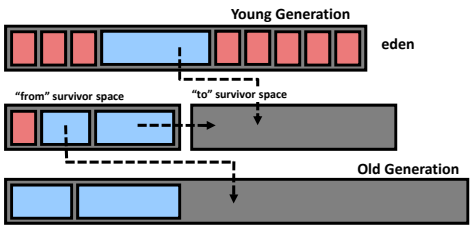
- Below are the different types of GC Collectors:
  - Serial Collector
  - Parallel Collector (Throughput Collector)
  - Parallel Compacting Collector
  - Concurrent Mark-Sweep Collector
  - G1
  - Z

Serial Collector

- Default for client style machines in Java SE 5 and 6
- Both young generation GC and old generation GC are done serially (using a single CPU).
- Serial collectors are used for most of the applications that:
  - Are running on client-style machines
  - Do not have low pause time requirements
- Can be explicitly requested with:
  - **`-XX:+UseSerialGC`**

Serial Collector on Young Generation: Before

- Live objects are copied from eden to an empty survival space.
- Relatively young live objects in the occupied (From) survival space are copied to the empty (To) survival space whereas relatively old ones are copied to the old generation space directly.

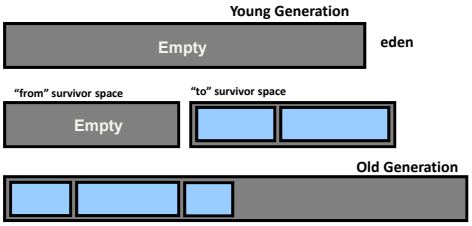


Serial Collector on Young Generation: After

> Both eden and the formerly occupied survival space are empty.

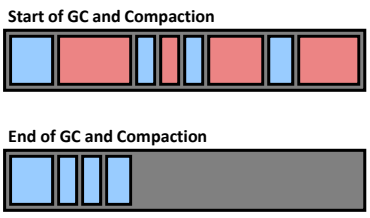
> The formerly empty survival space contains live objects.

> Survivor spaces switch roles.



Serial Collector on Old Generation

The old and permanent generations are collected through a serial mark-sweep-compact collection algorithm.



Parallel Collector: Throughput Matters!

> Uses multiple hardware threads for young generation GC, and thus increases the throughput

> Parallel collectors are used for Java applications that run on machines:

– With a lot of physical memory

– With multiple threads

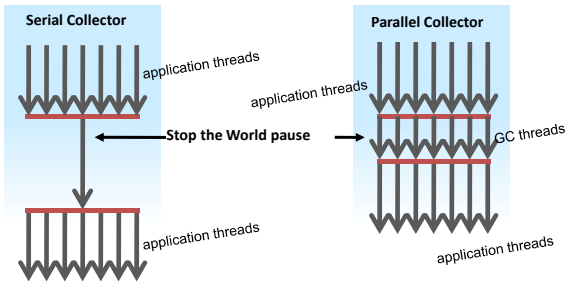
– That do not have low pause time requirements

> Can be explicitly requested with

– **`-XX:+UseParallelGC`**

Parallel Collector on Young Generation

> Uses a parallel version of young generation collection algorithm of a serial collector:



Parallel Compacting Collector

> In the purist sense, the difference between a parallel collector and a parallel compacting collector is “compacting.”
> Parallel collector is enabled via -XX:+UseParallelGC or -XX:+UseParallelOldGC
> Young generation in HotSpot is considered a copy collector; therefore, there is no need for compaction.
> -XX:+UseParallelGC as a parallel collector and -XX:+UseParallelOldGC as a parallel compacting collector

Concurrent Mark-Sweep (CMS) Collector

> Also called as Low Pause Collector
> CMS collector is used when an application has shorter pause time and can share processor resources with GC when the application is executing.
> Can be explicitly requested with
  – **`-XX:+UseConcMarkSweepGC`**
  – **`-XX:ParallelCMSThreads=<n>`**

CMS Collector Phases

> Concurrent collector cycle contains the following phases:
- Initial mark
- Concurrent mark
- Remark
- Concurrent sweep
- Concurrent reset

> During a concurrent collector cycle, a Java application is paused during the "initial mark and "remark" phases.

CMS Collector on Old Generation

> Initial mark phase
- Objects in the tenured generation are "marked" as reachable including those objects which may be reachable from young generation.
- Pause time is typically short in duration relative to minor collection pause times.

> Concurrent mark phase
- Traverses the tenured generation object graph for reachable objects concurrently while Java application threads are executing.

CMS Collector on Old Generation

- Remark
  - Finds objects that were missed by the concurrent mark phase due to updates by Java application threads to objects after the concurrent collector had finished tracing that object
- Concurrent sweep
  - Collects the objects identified as unreachable during marking phases
- Concurrent reset
  - Prepares for next concurrent collection

Concurrent Mark and Sweep - Problems

- 64 – bit environments
- Larger heap sizes (Xmx > 6 GB)
- GC pauses grow beyond acceptable levels in live enterprise environments due to fragmentation.
- Unpredictable GC pause times

G1GC

Garbage First GC

> This is meant to be a long term replacement for CMS.
> Fully supported since JDK 7 update 4.
> It is a compacting collector.
> It allows specifying desired pause times.

Garbage First GC

> It attempts to meet these specified pause times with a high probability.
> The maximum pause time can be specified using
  `-XX:MaxGCPauseMillis`.
> For example, **`-XX:MaxGCPauseMillis=100`** will set the request for maximum GC pause time to **`100`** ms.

Garbage First GC

> The JVM heap is partitioned into a set of uniformly sized heap regions.
> The default size of each region is determined based on the actual heap size. The region size varies between 1Mb–32Mb
> The size can also be specified explicitly using
  `-XX:G1HeapRegionSize`=*<size in Mb>*.

G1 Design

> The region size calculation is based on the average of the initial and maximum Java heap sizes such that there are about 2000 regions for that average heap size.
> As an example, for a 16GB Java heap with -Xmx16g -Xms16g command-line options, G1 will choose a region size of 16GB/2000 = 8MB.

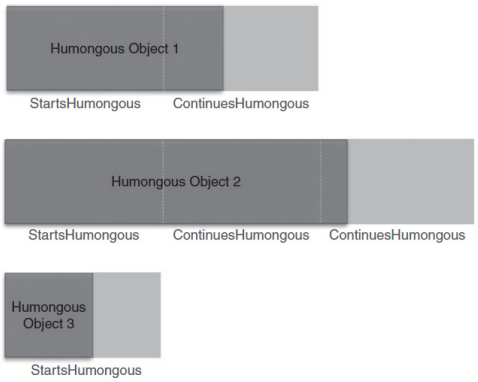
G1 Design

> Each region has an associated remembered set (a collection of the locations that contain pointers into the region, shortened to RSet).
> The total RSet size is limited but noticeable, so the number of regions has a direct effect on HotSpot's memory footprint.
> The total size of the RSets heavily depends on application behavior.
> At the low end, RSet overhead is around 1 percent
> At the high end 20 percent of the heap size.

Humongous Objects

- G1 deals specially with large object allocations
  - G1 calls “humongous objects.”
  - A humongous object is an object that is 50 percent or more of a region size.
- When a humongous object allocation occurs, G1 locates a set of consecutive available regions that together add up to enough memory to contain the humongous object.
- The first region is tagged as a “humongous start” region
- The other regions are marked as “humongous continues”
- If there are not enough consecutive available regions, G1 will do a full GC to compact the Java heap.
- Humongous regions are considered part of the old generation

Humongous Objects



Garbage First GC

> G1 performs a concurrent marking phase across all regions to determine the live-ness of objects
> G1 concentrates its collection and compaction activity on the areas of the heap that are likely to be full of reclaimable objects, hence the name **Garbage First**.
> G1 selects the number of regions to collect based on the specified pause time target.

Garbage First GC

> Concurrent mark phases:
  - **Initial mark**
  - Concurrent root region scan
  - Concurrent mark
  - **Remark**
  - Cleanup

Garbage First GC

> The regions in the **collection set** are garbage collected using **evacuation**.
> G1 copies objects from one or more regions of the heap to a single region on the heap, and in the process both compacts and frees up memory.
> The evacuation is performed in parallel on multiprocessor environments.

Garbage First GC

> During evacuation, all application threads are stopped.
> Parallel GC threads copy live objects from the identified **collection set** of regions to the target region.
> Application threads resume after the evacuation pause.

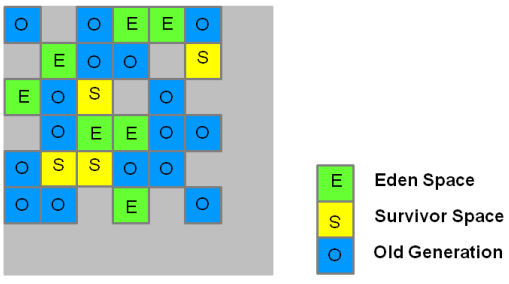
Recommended Use Cases for G1

> The first focus of G1 is to provide a solution for users running applications that require large heaps with limited GC latency. This means heap sizes of around 6GB or larger, and stable and predictable pause time below 0.5 seconds.

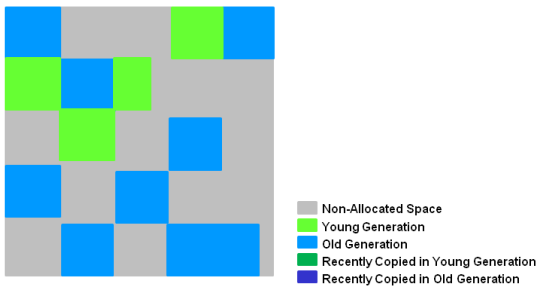
> Applications running today with either the CMS or the ParallelOldGC garbage collector would benefit switching to G1 if the application has one or more of the following traits.

- Full GC durations are too long or too frequent.
- The rate of object allocation rate or promotion varies significantly.
- Undesired long garbage collection or compaction pauses (longer than 0.5 to 1 second)

G1 Heap Allocation



Young Generation in G1



A Young GC in G1

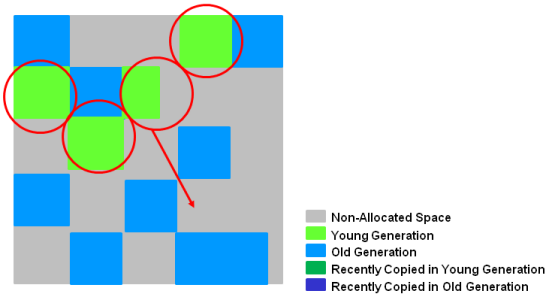
> Live objects are evacuated (i.e., copied or moved) to one or more survivor regions.

> If the aging threshold is met, some of the objects are promoted to old generation regions.

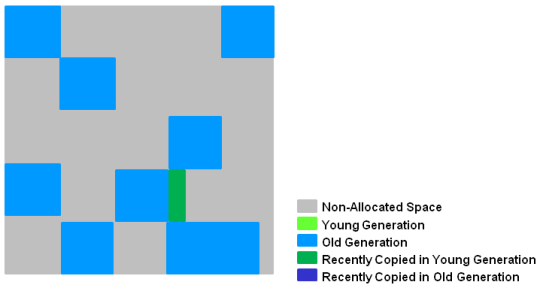
> This is a stop the world (STW) pause.

> Eden size and survivor size is calculated for the next young GC.

A Young GC in G1



End of a Young GC with G1



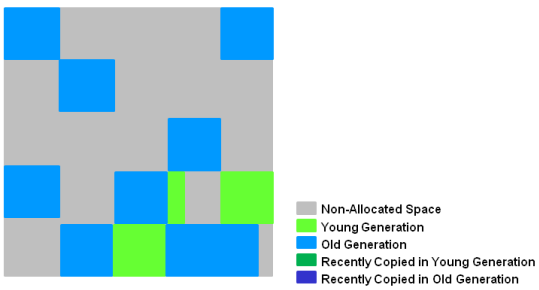
Old Generation Collection with G1

Phase	Description
(1) Initial Mark *(Stop the World Event)*	This is a STW event. With G1, it is piggybacked on a normal young GC. Mark survivor regions (root regions) which may have references to objects in old generation.
(2) Root Region Scanning	Scan survivor regions for references into the old generation. This happens while the application continues to run. The phase must be completed before a young GC can occur.
(3) Concurrent Marking	Find live objects over the entire heap. This happens while the application is running. This phase can be interrupted by young generation garbage collections.
(4) Remark *(Stop the World Event)*	Completes the marking of live object in the heap. Uses an algorithm called snapshot-at-the-beginning (SATB) which is much faster than what was used in the CMS collector.
(5) Cleanup *(Stop the World Event and Concurrent)*	•Performs accounting on live objects and completely free regions(STW) •Scrubs the Remembered Sets. (STW) •Reset the empty regions and return them to the free list. Concurrent)
(*) Copying *(Stop the World Event)*	These are the STW pauses to evacuate/copy live objects to new unused regions. This can be done with young generation regions which are logged as [GC pause (young)]. Or both young and old generation regions which are logged as [GC Pause (mixed)].

Initial Marking Phase

> Initial marking of live object is piggybacked on a young generation garbage collection.

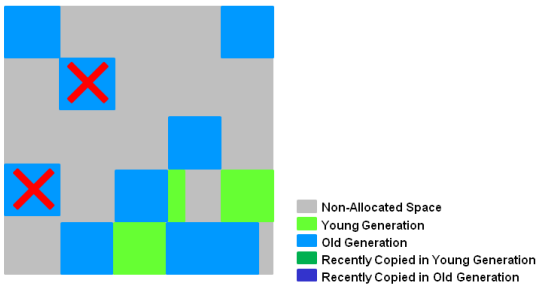
> In the logs this is noted as GC pause (young)(initial-mark).



Concurrent Marking Phase

> If empty regions are found (as denoted by the "X"), they are removed immediately in the Remark phase.

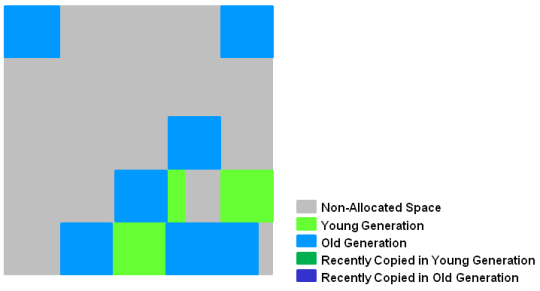
> Also, "accounting" information that determines liveness is calculated.



Remark Phase

> Empty regions are removed and reclaimed.

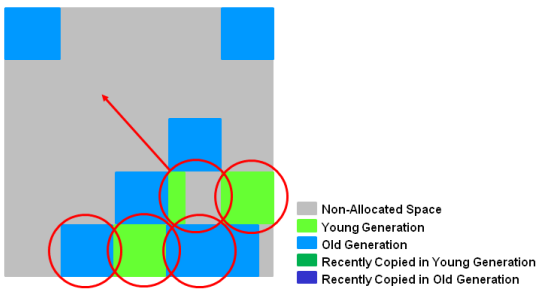
> Region liveness is now calculated for all regions.



Copying/Cleanup Phase

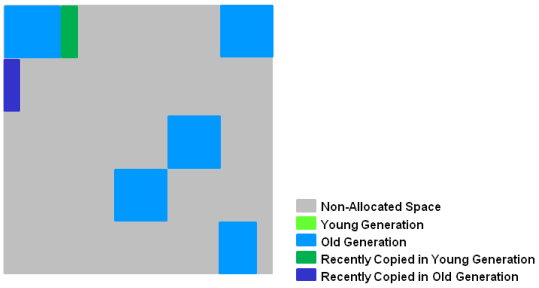
> G1 selects the regions with the lowest "liveness“
  – Those regions which can be collected the fastest.
> Then those regions are collected at the same time as YGC.
> This is denoted in the logs as [GC pause (mixed)].
> Both young and old generations are collected at the same time.

Copying/Cleanup Phase



After Copying/Cleanup Phase

> The regions selected have been collected and compacted into the dark blue region and the dark green region shown in the diagram.



G1GC

COMMAND LINE OPTIONS AND BEST PRACTICES

Basic Command Line

> To enable the G1 Collector use: -XX:+UseG1GC

```
java -Xmx50m -Xms50m -XX:+UseG1GC
-XX:MaxGCPauseMillis=200 -jar Java2Demo.jar
```

Key Command Line Switches

> **-XX:+UseG1GC**
  - Tells the JVM to use the G1 Garbage collector.

> **-XX:MaxGCPauseMillis=200**
  - Sets a target for the maximum GC pause time.
  - A soft goal: JVM will make its best effort to achieve it.
  - The pause time goal will sometimes not be met.
  - The default value is 200 milliseconds.

Key Command Line Switches

> **-XX:InitiatingHeapOccupancyPercent=45**
  - Percentage of the (entire) heap occupancy to start a concurrent GC cycle.
  - It is used by G1 to trigger a concurrent GC cycle based on the occupancy of the entire heap, not just one of the generations.
  - A value of 0 denotes 'do constant GC cycles'.
  - The default value is 45 (i.e., 45% full or occupied).

Best Practices

Do not set young generation size

> Explicitly setting young generation size via **-Xmn** interferes with the default behavior of the G1 collector.
  - G1 will no longer respect the pause time target for collections.
    - Setting the young generation size disables the pause time goal.
  - G1 is no longer able to expand and contract the young generation space as needed.
    - The size is fixed, no changes can be made to the size.

Best Practices

Response Time Metrics

> Instead of using average response time (ART) as a metric to set the XX:MaxGCPauseMillis=<N>, consider setting value that will meet the goal 90% of the time or more.

> This means 90% of users making a request will not experience a response time higher than the goal.

> Remember, the pause time is a goal and is not guaranteed to always be met.

What is an Evacuation Failure?

> A promotion failure that happens when a JVM runs out of heap regions during the GC for either survivors and promoted objects.

> The heap can't expand because it is already at max.

> This is indicated in the GC logs when using

-XX:+PrintGCDetails by **to-space overflow**. This is expensive!

- GC still has to continue so space has to be freed up.
- Unsuccessfully copied objects have to be tenured in place.
- Any updates to **RSets** of regions in the **CSet** have to be regenerated.
- All of these steps are expensive.

How to avoid Evacuation Failure

> To avoid evacuation failure, consider the following options.

- Increase heap size
  - Increase the **-XX:G1ReservePercent=n**, the default is 10.
  - G1 creates a false ceiling by trying to leave the reserve memory free in case more 'to-space' is desired.
- Start the marking cycle earlier
- Increase the number of marking threads using the

-XX:ConcGCThreads=n option.

Complete List of G1 GC Switches

JVM option	Description
`XX:G1HeapRegionSize=8m`	Size of heap region for G1 collector. Region size could be from 1MiB to 32MiB. Default is initial heap size / 2048.
`XX:G1ConfidencePercent=50`	Confidence coefficient for G1 pause prediction heuristics
`XX:G1ReservePercent=10`	It determines the minimum reserve we should have in the heap to minimize the probability of promotion failure
`XX:MaxGCPauseMillis=200`	GC time per MMU time slice in milliseconds
`XX:GCPauseIntervalMillis=300`	Time slice for MMU specification in milliseconds Should be greater than parameter above.
XX:InitiatingHeapOccupancyPercent=50	Percentage of the (entire) heap occupancy to start a concurrent GC cycle. It is used by GCs that trigger a concurrent GC cycle based on the occupancy of the entire heap, not just one of the generations. A value of 0 denotes 'do constant GC cycles'. The default value is 45.

G1 GC Glossary of Terms (1/3)

- Heap Region
  - G1 divides the heap into heap regions.
  - The regions for Eden, Survivor and Old are not contiguous.
- Collection Set (**CSet**)
  - The set of regions to be collected in a gc pause.
  - The number of young/survivor/old regions in CSet is decided dynamically according to pause time goal and other heuristics.

G1 GC Glossary of Terms (2/3)

- Humongous Objects
  - If the size of the object is larger than ½ of the region size, then g1 treats this object as Humongous Object.
  - Regions used for humongous objects are HS (Humongous Start) and HC (Humongous Continuous).
  - If HS is not big enough for this object, one or multiple HC regions will be used.
- Young GC
  - The collection set of the young gc includes only young/survivor regions.
- Mixed GC
  - The collection set of the mixed gc includes both young/survivor regions, but also old regions.

G1 GC Glossary of Terms (3/3)

> Remember Set (RSet)
  - The RSet is like a card table for each region.
  - It records references from other regions to objects in this region.
  - A regions' Remember Set consists of card addresses.
  - These cards contain locations of references that point into this region.

G1GC

LOGGING GC WITH G1

Setting the Log Detail

> You can set the detail to three different levels of detail.

1. -XX:+PrintGCDetails sets the detail level to finer. The options shows the following information:
   - Average, Min, and Max time are displayed for each phase.
   - Root Scan, RSet Updating (with processed buffers information), RSet Scan, Object Copy, Termination (with number of attempts).
   - Also shows “other” time such as time spent choosing CSet, reference processing, reference enqueuing and freeing CSet.
   - Shows the Eden, Survivors and Total Heap occupancies.

Setting the Log Detail

2. -verbosegc (which is equivalent to -XX:+PrintGC) sets the detail level of the log to fine.
   - Sample Output

```
[GC pause (G1 Humongous Allocation) (young) (initial-mark) 24M- >21M(64M), 0.2349730 secs]
[GC pause (G1 Evacuation Pause) (mixed) 66M->21M(236M), 0.1625268 secs]
```

Garbage Collectors: Comparisons

> The following table summarizes the trade-offs between the garbage collectors:

(m.sec)	Serial GC	Parallel GC	CMS GC
Parallelism	No	Yes	Yes
Concurrency	No	No	Yes
Young GCs	Serial	Parallel	Parallel
Old GCs	Serial	Parallel	Parallel & Conc

Ergonomics: What It Does

> Evaluates the system and auto-magically choose defaults for the HotSpot JVM. No tuning required!

> Relies on definition of “server class machine”

– 2 or more processor cores and 2 or more GB of physical RAM

> Server class machines will use -server JIT compiler

Ergonomics: What It Does

> As of JDK 5, server class machines set:
- Server JIT compiler
- Throughput collector
- Initial heap size (`-Xms`) 1/64th of physical memory up to max of 1 GB
- Max heap size (`-Xmx`) 1/4th of physical memory up to max of 1 GB

> If not identified as server class
- Client JIT compiler
- Default serial collector, same as before

Ergonomics

> Use -XX:+PrintCommandLineFlags to tell you what Ergonomics is choosing.

> Example:
- On 2 socket, 3.0 GHz Hyper threaded Intel, running Solaris with 4 GB RAM:

```
java -XX:+PrintCommandLineFlags -version
-XX:MaxHeapSize=1073626112  (1 GB)
-XX:+PrintCommandLineFlags
-XX:+UseParallelGC
Java HotSpot(TM) Server VM (build 1.6.0_02-b04, mixed mode)
```

The Garbage Collector

> The Garbage Collector goals
  - **Throughput**

    the total time **not** spent in garbage collection
  - **Pause Time**

    the time during the application is paused which garbage collection takes place
  - **Frequency of collection**

    how often the GC is working
  - **Footprint**

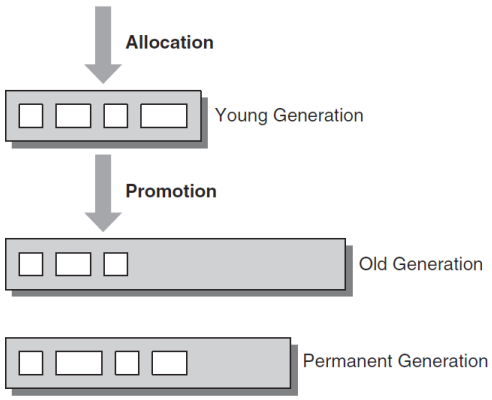
    the Heap size

Generational Garbage Collection

> The HotSpot VM uses a ***generational garbage collector***
> *A* well-known garbage collection approach that relies on the following two observations:
  - Most allocated objects become unreachable quickly.
  - Few references from older to younger objects exist.
> These two observations are collectively known as the ***weak generational hypothesis***, which generally holds true for Java applications.

Generational Garbage Collection

> To take advantage of this hypothesis, the HotSpot VM splits the heap into two physical areas

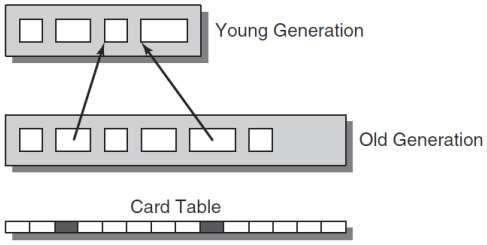


Generational Garbage Collection

> To keep minor garbage collections short, the garbage collector must be able to identify live objects in the young generation without having to scan the entire old generation.

> To achieve this, the garbage collectors in the HotSpot VM use a data structure called a ***card table***.

> The old generation is split into 512-byte chunks called *cards*.

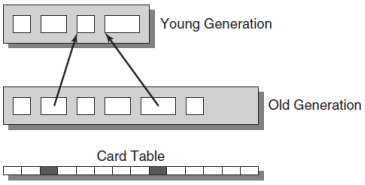


Generational Garbage Collection

> The card table is an array with one byte entry per card in the heap.

> Every update to a reference field of an object must also ensure that the card containing the updated reference field is ***marked dirty*** by setting its entry in the card table to the appropriate value.

> During a ***minor garbage collection***, only the areas that correspond to ***dirty cards*** are scanned to potentially discover old-to-young generation references



The Garbage Collector

> **Serial Collection**
  – Only one thing happens at the time. If the Serial collector is used on a multi CPU machine only one CPU is utilized

> **Parallel Collection**
  – The garbage collector job is spread across multiple jobs and multiple CPUs if available
  – The execution is faster but at the expenses of fragmentation

The Garbage Collector

> **Stop-the-World Collection**
  - the execution of the application is totally suspended during this phase.
  - very simple, the entire Heap is frozen

> **Concurrent Collection**
  - many garbage collection jobs can be executed at the same time the application is running
  - some of the work is done concurrently some is done as a short Stop-of-World collections
  - operates on live and large heaps

The Garbage Collector

> **Compacting Collection**
  - The memory is compacting, resulting in easy and fast allocations
  - Moves all live objects to a contiguous space

> **Non-Compacting Collection**
  - Does not move all live objects
  - Finishes faster
  - Expensive allocations in this mode. Searches for contiguous are of memory

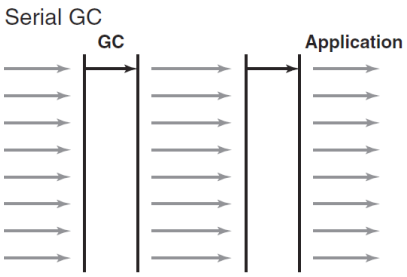
Serial Collector

> Young and Old generations are collected serially

> default on client type of hardware or manually using

-XX:+UseSerialGC option

> the application is paused during this operation, the collection is being done ***serially***, using ***one CPU***, ***stop-the-world*** method



HotSpot GC collectors

Young collector	Old collector	JVM option
Serial (DefNew)	Serial MSC	-XX:+Use**Serial**GC
Parallel scavenge (PSYoungGen)	Serial MSC (PSOldGen)	-XX:+Use**Parallel**GC
Parallel scavenge (PSYoungGen)	Parallel MSC (ParOldGen)	-XX:+Use**ParallelOld**GC
Serial (DefNew)	Concurrent MS	-XX:+Use**ConcMarkSweep**GC -XX:-Use**ParNew**GC
Parallel (ParNew)	Concurrent MS	-XX:+Use**ConcMarkSweep**GC -XX:+Use**ParNew**GC
G1		-XX:+Use**G1**GC

GC Logging Options 1/2

JVM option	Description
`verbose:gc` or `XX:+PrintGC`	Print basic GC info
`XX:+PrintGCDetails`	Print more elaborated GC info
`XX:+PrintGCTimeStamps`	Print timestamps for each GC event (seconds count from start of JVM)
`XX:+PrintGCDateStamps`	Print date stamps at garbage collection events
`XX:+PrintGCTaskTimeStamps`	Print timestamps for individual GC worker thread tasks (very verbose)
`Xloggc:`	Redirects GC output to a file instead of console
`XX:+PrintTenuringDistribution`	Print detailed demography of young space after each collection
`XX:+PrintTLAB`	Print TLAB allocation statistics

GC Logging Options 2/2

JVM option	Description
`XX:+PrintReferenceGC`	Print times for weak/soft/JNI/etc reference processing during STW pause
`XX:+PrintJNIGCStalls`	Reports if GC is waiting for native code to unpin object in memory
`XX:+PrintGCApplicationStoppedTime`	Print pause summary after each stop-the-world pause
`XX:+PrintGCApplicationConcurrentTime`	Print time for each concurrent phase of GC
`XX:+PrintClassHistogramAfterFullGC`	Prints class histogram after FGC
`XX:+PrintClassHistogramBeforeFullGC`	Prints class histogram before FGC
`XX:+HeapDumpAfterFullGC`	Creates heap dump file after FGC
`XX:+HeapDumpBeforeFullGC`	Creates heap dump file before FGC
`-XX:+HeapDumpOnOutOfMemoryError`	Creates heap dump in out-of-memory condition
`XX:HeapDumpPath=<path>`	Specifies path to save heap dumps

GC Logging Options for CMS

JVM option	Description
`-XX:PrintCMSStatistics`	Print additional CMS statistics if n>=1
`-XX:+PrintCMSInitiationStatistics`	Print CMS initiation details
`-XX:PrintFLSStatistics`	Print additional info concerning free lists
`-XX:PrintFLSCensus`	Print additional info concerning free lists
`-XX:+PrintPromotionFailure`	Print additional diagnostic information following promotion failure
`-XX:+CMSDumpAtPromotionFailure`	Dump useful information about the state of the CMS old generation upon a promotion failure.
`-XX:+CMSPrintChunksInDump`	In a CMS dump enabled by option above, include more detailed information about the free chunks.
`-XX:+CMSPrintObjectsInDump`	In a CMS dump enabled by option above, include more detailed information about the allocated objects.

Miscellaneous GC options

JVM option	Description
`-XX:+DisableExplicitGC`	JVM will ignore application calls to System.gc()
`-XX:SoftRefLRUPolicyMSPerMB=1000`	Soft references time to live in milliseconds per MB of free space in the heap
`OutOfMemoryError` conditions	
`-XX:+UseGCOverheadLimit`	Use policy to limit of proportion of time spent in GC before an `OutOfMemory` error is thrown
`-XX:GCTimeLimit=98`	Limit of proportion of time spent in GC before an `OutOfMemory` error is thrown (used with GCHeapFreeLimit)

JVM Sizing Options 1/2

JVM option	Description

`-Xms2048m -Xmx2g`
or
`-XX:InitialHeapSize=2g`
`-XX:MaxHeapSize=2g`
	Initial and max size of heap space (young space + tenured space). Permanent space does not count to this size.

`-XX:NewSize=64m` `-XX:MaxNewSize=64m`	Initial and max size of young space.
`-XX:NewRatio=3`	Alternative way to specify young space size. Sets ration of young vs tenured space (e.g. -XX:NewRatio=2 means that young space will be 2 time smaller than tenure space).
`-XX:SurvivorRatio=15`	Sets size of single survivor space as a portion of Eden space size (e.g. -XX:NewSize=64m -XX:SurvivorRatio=6 means that each survivor space will be 8m and eden will be 48m).
`-XX:PermSize=256m` `-XX:MaxPermSize=256m`	Initial and max size of permanent space.

JVM Sizing Options 2/2

JVM option	Description

`-Xss256`
or
`-XX:ThreadStackSize=256`
`(sizes are in Kbytes)`
	Sets size of stack area dedicated to each thread. Thread stacks do not count to heap size.

`-XX:MaxDirectMemorySize=2g`	Maximum size of off-heap memory available for JVM

Young collection tuning 1/2

JVM option	Description
`-XX:InitialTenuringThreshold=8`	Initial value for tenuring threshold (number of collections before object will be promoted to tenured space).
`-XX:MaxTenuringThreshold=15`	Max value for tenuring threshold.
`-XX:PretenureSizeThreshold=2m`	Max object size allowed to be allocated in young space (large objects will be allocated directly in old space). Thread local allocation bypasses this check so if TLAB is large enough object exciding size threshold still may be allocated in young.
`-XX:+AlwaysTenure`	Promote all objects surviving young collection immediately to tenured space (equivalent of `-XX:MaxTenuringThreshold=0`)
`-XX:+NeverTenure`	Objects from young space will never get promoted to tenured space while survivor space is large enough to keep them.

Young collection tuning 2/2

JVM option	Description
Thread local allocation blocks	
`-XX:+UseTLAB`	Use thread local allocation blocks in young space. Enabled by default.
`-XX:+ResizeTLAB`	Allow JVM to adaptively resize TLAB for threads.
`-XX:TLABSize=1m`	Initial size of TLAB for thread
`-XX:MinTLABSize=64k`	Minimal allowed size of TLAB

CMS Tuning options for initial mark phase

JVM option	Description
`XX:+UseCMSInitiatingOccupancyOnly`	Only use occupancy as a criterion for starting a CMS collection.
`XX:CMSInitiatingOccupancyFraction=70`	Percentage CMS generation occupancy to start a CMS collection cycle. A negative value means that CMSTriggerRatio is used.
`XX:CMSBootstrapOccupancy=10`	Percentage CMS generation occupancy at which to initiate CMS collection for bootstrapping collection stats.
`XX:CMSTriggerRatio=70`	Percentage of MinHeapFreeRatio in CMS generation that is allocated before a CMS collection cycle commences.
`XX:CMSTriggerPermRatio=90`	Percentage of MinHeapFreeRatio in the CMS perm generation that is allocated before a CMS collection cycle commences, that also collects the perm generation.
`XX:CMSWaitDuration=2000`	Once CMS collection is triggered, it will wait for next young collection to perform initial mark right after. This parameter specifies how long CMS can wait for young collection.

CMS Tuning options for remark phase

JVM option	Description
`XX:+CMSParallelRemarkEnabled`	Whether parallel remark enabled (only if ParNewGC), enabled by default
`XX:+CMSParallelSurvivorRemarkEnabled`	Whether parallel remark of survivor space enabled (effective only with option above), enabled by default
`XX:+CMSScavengeBeforeRemark`	Force young collection before remark phase.
`XX:+CMSScheduleRemarkEdenSizeThreshold`	If Eden used is below this value, don't try to schedule remark
`XX:CMSScheduleRemarkEdenPenetration=20`	The Eden occupancy % at which to try and schedule remark pause
`XX:CMSScheduleRemarkSamplingRatio=4`	Start sampling Eden top at least before young generation occupancy reaches 1/ of the size at which we plan to schedule remark

CMS Tuning options for Parallel Execution

JVM option	Description
`-XX:+UseParNewGC`	Use parallel algorithm for young space collection.
`-XX:+CMSConcurrentMTEnabled`	Use multiple threads for concurrent phases.
`-XX:ConcGCThreads=2`	Number of parallel threads used for concurrent phase.
`-XX:ParallelGCThreads=2`	Number of parallel threads used for stop-the-world phases.
`-XX:+UseParNewGC`	Use parallel algorithm for young space collection.
`-XX:+CMSConcurrentMTEnabled`	Use multiple threads for concurrent phases.

JBOSS EAP 7 PERFORMANCE TUNING

TUNING THE DATABASE CONNECTION POOL

Tuning the database connection pool

- Establishing a JDBC connection with a DBMS can be quite slow.
- If your application requires database connections that are repeatedly opened and closed; this can become a significant performance issue.
- The connection pools in JBoss AS datasources offer an efficient solution to this problem.
- When a client closes a connection from a data source, the connection is returned to the pool and becomes available for other clients; therefore, the connection itself is not closed.
- The cost of opening and closing pooled connections can be measured in terms of nanoseconds.

Tuning the database connection pool

```
<datasources>
 <datasource ... >
   ...
 <pool>
    <min-pool-size>10</min-pool-size>
    <max-pool-size>30</max-pool-size>
    <prefill>true</prefill>
    <use-strict-min>true</use-strict-min>
  </pool>
  <timeout>
  <idle-timeout-minutes>0</idle-timeout-minutes>
  <query-timeout>600</query-timeout>
  </timeout>
 </datasource>
```

Adjusting the pool size

> To determine the proper sizing, you need to monitor your connection usage.
> If you have access to the Command Line Interface you can monitor the runtime properties of your datasource.

```
[standalone@localhost:9999 /]/subsystem=datasources/data-
source="java:/MySqlDS":read-resource(include-runtime=true)
.............
```

Adjusting the pool size

```
{
"outcome" => "success",
"result" => {
"ActiveCount" => "10",
"AvailableCount" => "29",
"AverageBlockingTime" => "0",
"AverageCreationTime" => "56",
"CreatedCount" => "10",
"DestroyedCount" => "0",
"MaxCreationTime" => "320",
"MaxUsedCount" => "5",
"MaxWaitCount" => "0",
"MaxWaitTime" => "1",
. . . .
}
}
```

Monitor your connections using DB Tools

Database	Command
Oracle	SELECT * from V$SESSION;
MySQL	SHOW FULL PROCESSLIST;

JBOSS EAP 7 PERFORMANCE TUNING

TUNING EJB CONNECTION POOL

Tuning the database connection pool

- Establishing a JDBC connection with a DBMS can be quite slow.
- If your application requires database connections that are repeatedly opened and closed; this can become a significant performance issue.
- The connection pools in JBoss AS datasources offer an efficient solution to this problem.
- When a client closes a connection from a data source, the connection is returned to the pool and becomes available for other clients; therefore, the connection itself is not closed.
- The cost of opening and closing pooled connections can be measured in terms of nanoseconds.

JBOSS EAP 7 PERFORMANCE TUNING

TUNING WEB SERVER THREAD POOL

Tuning the database connection pool

- Establishing a JDBC connection with a DBMS can be quite slow.
- If your application requires database connections that are repeatedly opened and closed; this can become a significant performance issue.
- The connection pools in JBoss AS datasources offer an efficient solution to this problem.
- When a client closes a connection from a data source, the connection is returned to the pool and becomes available for other clients; therefore, the connection itself is not closed.
- The cost of opening and closing pooled connections can be measured in terms of nanoseconds.

JBOSS EAP 7 PERFORMANCE TUNING

TUNING WEB SERVER THREAD POOL

Tuning the database connection pool

- Establishing a JDBC connection with a DBMS can be quite slow.
- If your application requires database connections that are repeatedly opened and closed; this can become a significant performance issue.
- The connection pools in JBoss AS datasources offer an efficient solution to this problem.
- When a client closes a connection from a data source, the connection is returned to the pool and becomes available for other clients; therefore, the connection itself is not closed.
- The cost of opening and closing pooled connections can be measured in terms of nanoseconds.

JBOSS EAP 7 PERFORMANCE TUNING

TUNING LOGGING

Tuning the database connection pool

- Establishing a JDBC connection with a DBMS can be quite slow.
- If your application requires database connections that are repeatedly opened and closed; this can become a significant performance issue.
- The connection pools in JBoss AS datasources offer an efficient solution to this problem.
- When a client closes a connection from a data source, the connection is returned to the pool and becomes available for other clients; therefore, the connection itself is not closed.
- The cost of opening and closing pooled connections can be measured in terms of nanoseconds.

JBOSS EAP 7 PERFORMANCE TUNING

TUNING CACHE

Tuning the database connection pool

- Establishing a JDBC connection with a DBMS can be quite slow.
- If your application requires database connections that are repeatedly opened and closed; this can become a significant performance issue.
- The connection pools in JBoss AS datasources offer an efficient solution to this problem.
- When a client closes a connection from a data source, the connection is returned to the pool and becomes available for other clients; therefore, the connection itself is not closed.
- The cost of opening and closing pooled connections can be measured in terms of nanoseconds.