RED HAT®
TRAINING

redhat.

Comprehensive, hands-on training that solves real world problems

OpenShift Enterprise Development

Student Workbook

DO290-OSE3.0-en-2-20160126

OPENSHIFT ENTERPRISE DEVELOPMENT

OpenShift Enterprise 3.0 DO290
OpenShift Enterprise Development
Edition 2 20160126

Authors: Douglas Silva, Fernando Lozano, Jim Rigsbee, Ricardo Taniguchi, Zachary Gutterman

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

Document Conventions

Notes and Warnings

Note

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

Comparison

"Comparisons" look at similarities and differences between the technology or topic being discussed and similar technologies or topics in other operating systems or environments.

References

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

Insight

"Insights" point out some extra information about a step a student just took in a task, or give students a peek behind the scenes that will make them more effective later on.

Best Practice

"Best Practices" call out a common best practice that developers or administrators use.

Important

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

Warning

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

About This Course

OpenShift Enterprise Development

OpenShift Enterprise Development (DO290) prepares the system administrator to install, configure, and manage OpenShift Enterprise by Red Hat instances. OpenShift Enterprise by Red Hat, Red Hat's platform-as-a-service (PaaS) offering, provides pre-defined deployment environments for applications of all types through its use of container technology. This enables an environment that supports DevOps principles such as reduced time to market and continuous delivery. Students will learn how to install and configure an instance of OpenShift Enterprise by Red Hat, test the instance by deploying a real world application, and manage projects/ applications through hands-on labs.

OpenShift Enterprise is a key technology in Red Hat's DevOps story. Using PaaS has proven to accelerate time to market for many organizations already. This course provides the knowledge necessary to take advantage of the platform in a popular DevOps environment. This course is part of a series of DevOps courses that center around the use of OSE to enable continuous delivery a key goal in the DevOps philosophy. Red Hat is committed in being a leader in providing prescriptive approaches to DevOps using Red Hat and other open source technologies.

Objectives

- Adapt and deploy applications in OpenShift to support continuous integration implementing practices adopted by DevOps.
- Learn how to customize OpenShift configuration files and environment to allow the development of legacy applications as well as new applications.

Audience

- Application developers.
- Software architects.
- System administrators that deploy applications.

Prerequisites

Students should meet one or more of the following prerequisites:

- Developer in one of the languages supported by OpenShift Enterprise such as Java EE, Python, Ruby, Node.js, and others.
- Java EE programming skills are helpful but not required.

Orientation to the Classroom Lab Environment

In this course, students will do most hands-on practice exercises and lab work with a computer system, which will be referred to as **workstation**. This machine has the host name workstation*X*, where the *X* in the computer's host name will be a number that will vary from student to student. The machine has a standard user account, *student*, with the password *student*. Access to the *root* account is available from the *student* account, using the **sudo** command.

Important

Instructions on how to control your stations vary depending on whether you are taking this course in a physical classroom or in a virtual classroom.

Read the *Instructor-Led Training (ILT)* section if you are taking this course in a classroom which is using local computer hardware and an in-person instructor.

Read the *Virtual Training (VT)* section if you are taking this course using a remote classroom accessed through your web browser and have a remote instructor.

Instructor-Led Training (ILT)

In an Instructor-Led Training classroom, students will be assigned a physical computer ("foundation*X*"), which will be used to access the lab environment. The **workstation** system is a virtual machine running on that host. Students should log into the physical machine as user *kiosk* with the password *redhat*.

Each student is on the IPv4 network 172.25.*X*.0/24, where the *X* matches the number of their **workstation** system. The instructor runs a central utility server, **classroom**, which acts as a router for the classroom networks and provides DNS, DHCP, HTTP, and other content services.

Classroom Machines

Machine name	IP addresses	Role
classroom.example.com	172.25.254.254	Classroom utility server
workstation.pod*X*.example.com	172.25.*X*.9	Student computer
node.pod*X*.example.com	172.25.*X*.10	OSE node
master.pod*X*.example.com	172.25.*X*.11	OSE master

Controlling your station

On foundation*X*, a special command called **rht-vmctl** is used to work with the **workstation** machine. The commands in the following table should be run as the *kiosk* user on foundation*X*, and can be used with **workstation**.

rht-vmctl Commands

Action	Command
Start **workstation** machine	`[kiosk@foundationX]$ rht-vmctl start workstation`

Action	Command
View "physical console" to log in and work with **workstation** machine	`[kiosk@foundationX]$ rht-vmctl view workstation`
Reset **workstation** machine to its previous state and restart virtual machine	`[kiosk@foundationX]$ rht-vmctl reset workstation`

At the start of a lab exercise, if the instruction "reset your workstation" appears, that means the command **rht-vmctl reset workstation** should be run in a prompt on the foundation*X* system as user *kiosk*. This is an uncommon occurrence in most classes. Only do this at the advice of the instructor.

Virtual Training (VT)

In virtual training, students will see four computer systems that are used by the instructor. These are known as **Classroom**, **Workstation**, **Node**, and **Master** and will primarily be used by the instructor for lab demonstrations.

Virtual Training Machines

Machine name	IP addresses	Role
classroom.example.com	172.25.254.254	Classroom utility server
workstation.pod*X*.example.com	172.25.*X*.9	Student computer
node.pod*X*.example.com	172.25.*X*.10	OSE node
master.pod*X*.example.com	172.25.*X*.11	OSE master

Controlling your station

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

Machine States

State	Description
none	Your machine has not yet been started. When started, your machine will boot into a newly initialized state (the desk will have been reset).
starting	Your machine is in the process of booting.
running	Your machine is running and available (or, when booting, soon will be.)
stopping	Your machine is in the process of shutting down.
stopped	Your machine is completely shut down. Upon starting, your machine will boot into the same state as when it was shut down (the disk will have been preserved).
impaired	A network connection to your machine cannot be made. Typically this state is reached when a student has corrupted networking or firewall rules. If the condition persists after a machine reset, or is intermittent, please open a support case.

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

Machine Actions

Action	Description
Start Station	Start ("power on") the machine.
Stop Station	Stop ("power off") the machine, preserving the contents of its disk.
Reset Station	Stop ("power off") the machine and select "Reset" to reset your image, resetting the disk to its initial state. **Caution: Any work generated on the disk will be lost.**

Internationalization

Language support

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

Per-user language selection

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

Language settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the **Region & Language** application. Run the command **`gnome-control-center region`**, or from the top bar, select **(User) > Settings**. In the window that opens, select **Region & Language**. The user can click the **Language** box and select their preferred language from the list that appears. This will also update the **Formats** setting to the default for that language. The next time the user logs in, these changes will take full effect.

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

Note

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

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

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

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

```
[user@host ~]$ LANG=fr_FR.utf8 date
```

```
jeu. avril 24 17:55:01 CDT 2014
```

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

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

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

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

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

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

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

Note

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

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

System-wide default language settings

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

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

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

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

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

Important

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

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

Language packs

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

References

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

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

Language Codes Reference

Language Codes

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

CHAPTER 1

DEVELOPMENT ENVIRONMENT REVIEW

Overview	
Goal	Describe the developer's tools, agile practices, and deployment environment that will be utilized in this course.
Objectives	• Describe the high-level activities of this course. • Describe the development tools and deployment environment.
Sections	• Course Overview • Development Environment (and Guided Exercise)

Course Overview

Objectives

After completing this section, students should be able to describe the high-level activities of this course.

Traditional deployment vs. PaaS

In the traditional development, the developers and sysadmins must setup their environments. Any discrepancy from each setup may require investigation and may cause some reconciliation, normally executed during the deployment process. Unfortunately, due to the nature of the lack of standardization and manual intervention, the reconciliation process is not documented and automatized, causing pain to all the parts that are responsible for deploy an application.

According to Wikipedia, "Platform as a service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app."

References

Platform as a Service Wikipedia definition
https://en.wikipedia.org/wiki/Platform_as_a_service

A **PaaS** platform provides:

- *Preconfigured runtimes*: Developers can deploy their own applications with minimal setup configuration using a set of tools such as a web server, an application server and web framework to provide databases, messaging, business process management, and enterprise integration patterns, among other middleware services.
- *Simplified workflow for production environment deployment*: Since a PaaS allows standardized and robust deployment workflow using hooks, any update made to the source code can be deployed into the production environment in a fast and safe way, following Continuous Integration (CI) and Continuous Delivery (CD) processes.
- *Unified patch management systems*: A PaaS will be responsible for managing the overlying infrastructure to run an application, and it will be able to manage and install patches to it.
- *Efficiency to manage hardware resources*: It allows system administrators to focus on managing application uptime, performance, and scalability using internal tools for monitoring purposes.

Improving developer's productivity

OpenShift Enterprise by Red Hat is the PaaS distributed by Red Hat to create a private PaaS in an existing infrastructure. It improves developer productivity because it provides:

- *Faster deployment process to production*: OpenShift Enterprise eliminates the wait time for the environment provisioning and testing. It allows developers to focus on developing business logic, instead of integrating runtime environments and other dependencies.

- *Self-service platform*: Developers can create applications from templates or from their own source code control repositories. System administrators can define resource quotas for users and projects.
- *Polyglot, multilanguage support*: Support for Java, Node.js, PHP, Perl, and Ruby directly from Red Hat, plus many others from partners, OpenShift Commons, and the larger Docker community. MySQL, PostgreSQL, and MongoDB databases directly from Red Hat and others from partners and the Docker community. Red Hat also supports JBoss Middleware products such as JBoss EAP, Active MQ, and Fuse running natively on OSE.
- *Automation*: Application life-cycle management features to automatically apply changes to source code under version control and security errata from the base OS or dependency libraries to running applications. Scale and fail over applications based on scheduling and policy. Orchestrate composite applications built from independent components or services.
- *User interfaces*: Easy-to-use Web frontend for deploying and monitoring applications. CLI interface for remote management of applications and resources. Eclipse IDE and JBoss Developer Studio plug-ins so developers can stay with their familiar tools. REST API for integration with third- party or in-house tools.
- *Collaboration*: Share projects and customized runtimes within an organization or with the larger community.
- *Scalable*: Provides a web scale distributed application platform including elasticity to handle increased traffic on demand and high availability so an application survives events such as the loss of a physical machine.
- *Container portability*: Applications and services are packaged using standard container images and composite applications are orchestrated using Kubernetes templates that can be deployed to other platforms built on those base technologies, such as Red Hat Atomic.
- *Open Source*: No vendor lock-in.
- *Choice of cloud*(or no cloud): Deploy OpenShift Enterprise on bare-metal servers, hypervisors from multiple vendors and most, if not all, IaaS cloud providers.
- *Enterprise grade*: Rely on Red Hat support for OpenShift Enterprise itself, selected container images, and application runtimes. Trust third-party container images, runtimes, and applications certified by Red Hat. Run specialized or third-party applications in a hardened, secure environment with high availability provided by OpenShift Enterprise.

Customers who prefer not having to manage their own datacenters can refer to **OpenShift Online**, a public cloud platform provided by Red Hat. Both OpenShift Enterprise and OpenShift Online are based on the **OpenShift Origin** open source software project, which itself builds on a number of other Open Source Projects such as **Atomic**, **Docker**, and **Kubernetes**.

References

More information about OSE upstream projects can be found in:

OpenShift product family
http://www.openshift.com

OpenShift Online
http://www.openshift.org

Atomic
http://www.projectatomic.io

Kubernetes
http://kubernetes.io

Docker
http://docker.com
http://github.com/docker/docker

Applications run on OpenShift Enterprise as **containers**, which are isolated partitions inside a single OS. Containers provide many of the same benefits as virtual machines, such as security, storage, and network isolation, but require fewer hardware resources and are quicker to launch and terminate.

OpenShift Enterprise is an enabler for microservices architectures, while also supporting more traditional workloads. Many organizations will also find OSE-native features sufficient to enable a DevOps process, while others will find it is easy to integrate with both standard and custom CI/CD tools.

Note

This course will use Java EE and a relational database, but OSE 3 supports a substantial number of environments such as Node.js, Ruby, PHP, etc.

Course flow

This course will present how a traditional development process looks and how to adopt the DevOps processes using OpenShift Enterprise.

In order to move the mindset to a DevOps process, an existing JavaEE project (the bookstore application) will be updated throughout the course using tools that will allow developers to create, manage, and update applications.

Gradually, the course will introduce OpenShift to improve the software development life cycle (SDLC) and eliminate some tasks that usually require manual intervention, such as testing and deployment using CI tools with agile development processes such as test-driven development (TDD).

Finally, the bookstore application will be deployed in OpenShift Enterprise using the same set of tools used during the traditional development process. In order to manage the development

process, agile techniques and tools will be presented and used throughout the course to accomplish the final product development.

Due to the nature of software issues, troubleshooting and container customization will be discussed to fix and deploy applications in an OpenShift Enterprise environment using best practices.

DevOps principles

Developers aspiring to practice DevOps standards must follow principles to achieve a reliable and repeatable software development process:

- *Agile development*: A developer must implement agile methodologies practices to allow faster development cycles and improve the source code quality using test-driven development principles. Implementing agile practices minimizes the amount of time needed to deploy new requests from customers. Additionally, the tests created during the development process will guarantee that any change to the source code that affects the application will be identified promptly.

- *Source Control Management (SCM) and Continuous Integration (CI)*: A SCM manage changes made to the source code, identifying which updates were executed during the software development and what changes may conflict with any update made by a developer. Any change to the SCM will trigger a CI tool to execute the testing, building, and deployment processes needed to make the software available for usage.

- *Infrastructure as code via containers*: Software may require external dependencies such as databases, messaging systems, or integration systems running to exchange data. Containers can be used to simplify the external dependencies management as well as the developed software deployment.

The bookstore application

The application used by this course is an online bookstore developed:

- *Java EE 6*: It is a multilayered application developed using Java EE technologies such as:
  - *JavaServer Faces (JSF 2)*: Responsible for creating the user interface. It is a component-based web framework with a large set of reusable components available for development.
  - *Enterprise JavaBeans (EJB 3.x)*: Implements the back-end layer of the bookstore application.
  - *Java Persistence API (JPA 2.x)*: Enables the access to the database layer of the application.
- *Rule engine*: Implements rules to manage coupons and offers to the customer.
- *HTML5*: Improves the visual aspects and usability of the existing web pages.
- *Relational database*: Stores data from customer orders as well as the products sold by the bookstore.

Development Environment

Objective

After completing this section, students should be able to describe the development tools and deployment environment.

Source control management tool: Git

Git is a distributed source control management (DSCM) tool that will be used during the development. It is the SCM tools where OpenShift will get the source code to build and deploy applications.

Installing Git
To install a Git client, use the yum installation tool from RHEL7 **`yum install git`**.

Cloning Git repositories
Different from ordinary SCM tools such as subversion or CVS, Git downloads all the contents locally, cloning the resources instead of checking them out. This can be achieve using the command **`git clone <URI>`**.

Note

A **`git clone`** command requires authentication from the developer.

Note

The **`git clone`** command generates a local repository that can be freely updated; however unlike other SCM tools, it is not associated with a single central repository.

Adding contents to a Git repository
Git identifies changes made to the existing files and will be able to commit them to the local repository. However, any new content created by a developer must be added to the local Git repository. To add it, the developer must execute the command **`git add <file1> <file2> ...`**

Committing changes to Git
Every change made to a Git-managed directory will not be automatically committed to the local repository. This approach allows developers to undo any modification made to the source code. To commit the changes to the local repository, the **`commit`** command is available and a message must be added to the commit command to guarantee traceability: **`git commit -m "This is my initial commit"`**.

Note

The commit term may cause confusion for experienced developers using other SCM tools, since it sends the changes to a remote repository. For Git, to update a remote repository, a **push** command is necessary.

Pushing changes to the remote Git repository
The commit updates the local repository but does not push changes to the remote repository. To make it possible, Git provides the **push** command: **`git push`**.

Integrated Development Environment (IDE): JBoss Developer Studio

JBoss Developer Studio (JBDS) builds upon the popular, extensible, open source Java-based IDE, Eclipse, to provide developers with unparalleled capabilities with regard to platforms and frameworks. This integrated development environment (IDE) provides tooling for such technologies as:

- Hibernate
- Java EE (EJB, JSF, CDI, JPA)
- Web services
- RichFaces
- JBoss EAP/EWP
- OpenShift

These tools provide wizards to implement best practices when developing for Red Hat's platforms.

References

Red Hat JBoss Developer Studio download link
https://access.redhat.com/jbossnetwork/restricted/softwareDownload.html?softwareId=38553

Starting JBDS
After installing JBoss Developer Studio, an icon is installed in the desktop area. To start up JBDS, it is necessary to double-click the icon.

Note

It is also possible to open a Terminal window (**Applications** > **Utilities** > **Terminal**) and run JBDS from the installation directory executing the **`./jbdevstudio`** script.

Selecting a workspace
A dialog box will be opened to select a directory. It is called workspace and is an important placeholder where all resources from multiple projects will be stored. JBDS looks for the selected directory and creates, for each project, its own subdirectory.

Database: MySQL 5.5

In order to store data from the bookstore application, a MySQL database will be used by this course. The database is already installed in the workstation machine. However, it will be installed later in a running OpenShift instance.

JBoss Developer Studio has an integrated tool to manage and customize an existing database.

Demonstration: Setting up MySQL access in JBoss Developer Studio

In this example, please follow along with the steps while your instructor demonstrates how to configure JBDS to access MySQL.

1. Open JBoss Developer Studio by double-clicking the icon in the desktop.
2. Select **/home/student/DO290/workspace** directory as the JBDS workspace.
3. Select the **Window > Perspective > Open Perspective > Other...**. Select the **Database Development** perspective and click **OK**.
4. Right-click the **Database Connections** tree item and select the **New** option.
5. Choose the MySQL option and change the name field with **bookstore** and click **Next**.
6. Click the **New Driver Definition** button (located at the right side of the **Driver** combo box). Choose the **MySQL JDBC Driver version 5.1** and open the **JAR List** tab. Click the **Clear All** button and then click the **Add JAR/Zip...** button. Choose the file located at **/home/student/DO290/lib/mysql-connector-java-5.1.36-bin.jar** and click the **OK** button.
7. Change the values from the next dialog to:
   - Database: **bookstore**
   - URL: **jdbc:mysql://localhost:3306/bookstore**
   - Username: **root**
   - Password: **redhat**
   - Save password: **checked**

   Test the connection by pressing the **Test Connection** button. Click the **Finish** button.
8. To create a table, right-click the bookstore from the **Database Connection** tab and choose **Open SQL Scrapbook** option. A new editor will be opened. Choose the following values from the dropboxes:
   - Type: **MySql_5.1**
   - Name: **bookstore**
   - Database: **bookstore**

   Type the following command to the **SQL Scrapbook** opened to create a table in the database:

```
CREATE TABLE projects (
  id int(11) NOT NULL,
  name varchar(255) DEFAULT NULL,
  code varchar(255) DEFAULT NULL,
  PRIMARY KEY (id)
```

```
);
```

Execute the SQL command by right-clicking the SQL command and selecting the **Execute All** option.

9. To populate the table, run the following command:

```
insert into projects (id, name, code) values (1,'DevOps','DO290');
```

10. To list all the items added to a table, run the following command:

```
select * from projects;
```

11. Drop the table running the following command:

```
drop table projects;
```

Application server: JBoss EAP

JBoss EAP is a JavaEE-certified application server supported by Red Hat and that will be responsible for running the bookstore application. It can be integrated with JBoss Developer Studio to permit a unified environment for development purposes. It can also be used by a developer to trigger the deployment process in OpenShift.

Demonstration: Configuring JBoss EAP in JBoss Developer Studio

In this example, please follow along with the steps while your instructor demonstrates how to configure JBDS to access JBoss EAP.

1. Double-click the JBDS icon from the desktop.
2. Choose **`/home/student/DO290/workspace`** directory as the JBDS workspace.
3. Locate the **Servers** tab from the JBoss perspective (found at the top right area from JBDS) and click the link **No servers are available. Click this link to create a new server....**
4. From the **Define a New Server** step pick the **JBoss Enterprise Application Platform 6.1+** and click **Next**. In the **Create a new Server Adapter** step, keep the default values and click **Next**. And point it to the place where EAP is installed (**`/home/student/DO290/opt/jboss-eap-6.4`**) as the **Home directory** entry and click **Finish**.
5. Right-click the server and select **Start**.
6. Note that another tab called **Console** will output the logs from EAP.
7. Stop the EAP instance, clicking the **Servers** tab and right-click the **JBoss EAP 6.1+** server. Select the **Stop** option to stop JBoss.

Building tool: Apache Maven

In order to avoid using development tools (such as an IDE) to build a deployable artifact, DevOps delegates the creation of an artifact to a build tool, such as Apache Maven, due to the amount

of plug-ins available to manage the whole build process, including compiling, testing, and deployment. In order to make Maven work, a project must follow some standards such as a directory structure and a build configuration file (**`pom.xml`**). Maven is already installed in the actual environment.

Note

Maven download site, **`http://maven.apache.org/`**

Compiling a project
Maven provides a plug-in to compile a set of Java sources called **`compile`**. To trigger it, run the Maven command line: **`mvn compile`**.

Testing a project
All the tests developed for a project can be executed with a plug-in called **surefire**. To test all the source code (compiling all the source codes), run the following command **`mvn test`**.

Package a project
Once all the tests were executed successfully, the deployable package can be built running the following command **`mvn package`**.

Deploying a project
Red Hat JBoss EAP provides a plug-in that will be responsible for deploying applications into JBoss EAP. To run it **`mvn jboss-as:deploy`**.

Note

JBoss Developer Studio embeds a plug-in to simplify Maven's usage.

Guided Exercise: Setting Up the Development Environment

In this lab, the student will clone the bookstore application's source code from a local Git repository and deploy it to the local installation of JBoss EAP.

Resources	
Files	NA
Application URL	`http://localhost:8080/bookstore`

Outcomes
You should be able to:

- Clone the source code from a Git remote repository.
- Deploy the application to the local JBoss EAP.
- Test the application.

1. Cloning the Source Code From a Git Repository
   1.1. Open a new Terminal window from the workstation VM (**Applications** > **Utilities** > **Terminal**) an execute the following command:

```
[student@workstation ~]$ cd ~/D0290/labs
[student@workstation labs]$ git clone http://workstation.podX.example.com/
bookstore.git
```

   Replace *X* by your student number.

   1.2. Open JBDS by double-clicking the icon found at the desktop. Select the directory **/home/student/D0290/workspace** as the workspace and click the **Ok** button.

   1.3. From the JBDS menu, select **File** > **Import** and choose the **Maven** > **Existing Maven Projects** and click the **Next** button. Click the **Browse** button and select the directory **/home/student/D0290/labs/bookstore**. Click the **Finish** button to import the project. Click the **Next** button.

Note

A popup may open due to problems in **Auto share git projects**. You may safely dismiss the message by clicking the **Ok** button. The side effect of this problem is that JBDS integration with Git will not work.

2. Deploy the Application
   2.1. From the **Servers** tab, click the **No servers are available. Click this link to create a new server...**. A dialog box will open. Select **Red Hat JBoss Middleware** > **JBoss Enterprise Application Platform 6.1+** option. Click the **Next** button. In the **Create a new server**

adapter step, keep the default values and click **Next**. In the **JBoss Runtime** step, change the following fields:

- Home directory: **`/home/student/DO290/opt/jboss-eap-6.4`**

Click the **Finish** button.

2.2. To deploy an application, right-click the project and select **Run As** > **Run on Server**. Click the **Finish** button from the dialog box opened. Wait for the **Console** tab to stop the log generation and the embedded browser window is opened.

Warning

The browser from JBDS must be dismissed because the browser is not using the correct context.

3. Test the Application
   Click **Applications** > **Internet** > **Firefox** and open the address **`http://localhost:8080/bookstore`**.

4. Remove the Application
   From JBoss Developer Studio, choose the **Servers** tab and expand the JBoss EAP 6.1+ and right-click the bookstore icon and click the **Remove** option to remove the app from JBoss EAP 6.4. Confirm the removal of the project by clicking **OK**.

5. Stop JBoss EAP
   From JBDS choose the **Servers** tab, right-click the **JBoss EAP 6.1+** and select the **Stop** option.

6. Delete the application
   From JBDS, right-click the bookstore application and select the Delete option. Mark the checkbox **Delete project contents on the disk (cannot be undone)** and click **Ok**. This will delete the project.

Summary

In this chapter, you learned:

- OpenShift Enterprise is a PaaS cloud platform to enable DevOps practices.
- DevOps follow agile principles to guarantee quick delivery in a reliable deployment environment.
- The set of tools needed by both agile and traditional teams are the same, but in order to improve productivity, an agile team adopts additional tools to automate the deployment process, following Continuous Integration and Continuous Delivery processes.
- Agile processes using simple project management tools such as the kanban board and user stories guarantees the visibility of the project status.

CHAPTER 2

BOOKSTORE APPLICATION REVIEW

Overview	
Goal	Describe the developer's tools, agile practices, and deployment environment that will be utilized in this course.
Objectives	• Describe the type of application and its major components. • Describe the major features of the bookstore application. • Build and deploy the application to JBoss EAP and MySQL.
Sections	• Bookstore Architecture (and Quiz) • Bookstore Features (and Quiz) • Deploying *The Shoppe* (and Guided Exercise)

Bookstore Architecture

Objectives

After completing this section, students should be able to describe the type of application and its major components.

Java EE

Bookstore is a Java EE web profile application using some of the latest APIs to develop the application. Due to the nature of Java EE, the application can be deployed at any Java EE-compliant container. For this course, it will be deployed on a JBoss EAP 6.4 instance, the same container supported by OpenShift Enterprise.

To enhance the customer's buying experience, a rule engine (Drools) responsible for calculating discounts is also deployed.

Finally, the application stores data using a relational database management system (RDBMS) called MySQL. Since the persistence is managed by a Java API called Java Persistence API (JPA) that is database-agnostic, the bookstore application can be deployed at any compatible database supported by JPA.

Presentation layer
The bookstore application uses JavaServer Faces (JSF) 2 tags. JSF is a major technology update to develop web rich-user interfaces, using a component-driven framework to guarantee tag reuse. It includes support to AJAX-compliant JavaScript frameworks to provide an interactive experience to the user.

Additionally, HTML5 tags were included to simplify the development of web pages and provide productivity to the developer and a rich interface experience to the customer.

Finally, JavaScript code was added to provide a better usability and visual enhancement. It also uses some JavaScript libraries to integrate with social networks such as Facebook, LinkedIn, and Google+, and get information about the amount of followers.

Business layer
The bookstore application connects to a database and a rule engine using Enterprise JavaBeans (EJBs). It will be responsible for providing a high-level access to both the database and the business rules engine. Due to the transactional nature of EJB, it can handle any error related to database persistence or the rule engine, avoiding inconsistent sales data.

Persistence layer
The bookstore application stores data in a RDBMS called MySQL and is accessed using the Java Persistence API (JPA). Due to the flexible nature of JPA and the way the application is coded, the bookstore may be deployed in other RDBMS as well, such as PostgreSQL or Oracle.

Rules layer
To manage and execute rules, a rule engine processor is embedded in to the bookstore application: Drools. It is a declarative rule engine that will analyze and calculate the discounts for a shopping cart.

Database

The following image illustrates the database entity-relationship model.

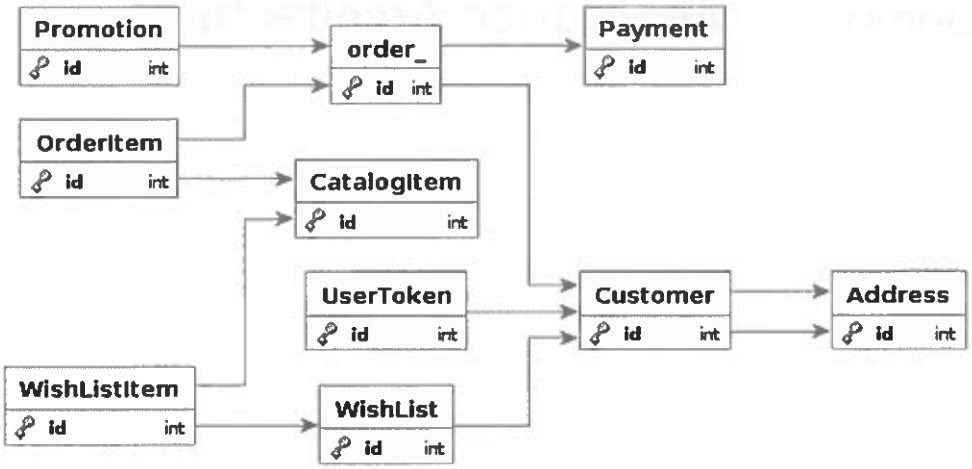

Figure 2.1:
Entity-relationship diagram for the bookstore

A **`Customer`** will have an username and password to access the site. Additionally, each customer may have multiple **`Order`**s with one or multiple **`CatalogItem`**s.

Each **`CatalogItem`** added to a shopping cart will become an **`OrderItem`**.

In order to store some items for future purchases, there is a **`WishList`** where each customer may store the set of products that may be purchased later.

The **`Promotion`**s store the valid date for a certain coupon; however, the logic behind each promotion is managed and stored at the rule engine.

Quiz: Bookstore Application Architecture Review

Choose the correct answer to the following questions:

1. The bookstore application stores its data in a/an (Select one.):

   a. Rule engine.
   b. Database.
   c. Enterprise JavaBeans.
   d. JavaServer Faces.

2. The web interface was developed using (Select three.):

   a. JavaServer Faces.
   b. HTML5.
   c. Drools.
   d. JavaScript.

3. The web application database stores data about each customer's order in the (Select one.):

   a. User table.
   b. Order table.
   c. Promotion table.
   d. Contact table.

4. The web application database stores data about each item sold in the (Select one.):

   a. **`Order`** table.
   b. **`OrderItem`** table.
   c. **`CatalogItem`** table.
   d. **`WishListItem`** table.

Solution

Choose the correct answer to the following questions:

1. The bookstore application stores its data in a/an (Select one.):

   a. Rule engine.
   b. Database.
   c. Enterprise JavaBeans.
   d. JavaServer Faces.

2. The web interface was developed using (Select three.):

   a. JavaServer Faces.
   b. HTML5.
   c. Drools.
   d. JavaScript.

3. The web application database stores data about each customer's order in the (Select one.):

   a. User table.
   b. Order table.
   c. Promotion table.
   d. Contact table.

4. The web application database stores data about each item sold in the (Select one.):

   a. `Order` table.
   b. `OrderItem` table.
   c. `CatalogItem` table.
   d. `WishListItem` table.

Bookstore Features

Objective

After completing this section, students should be able to describe the major features of the bookstore application.

Catalog

The bookstore application has a set of books and comics that can be sold to a customer. Each of them are grouped in catalogs focused on a certain type of literature. For instance, there is a children's book catalog where all the major publications are stored.

This grouping allows a faster search and provides customers a better organization of the bookstore's titles. The catalog is used for organization purposes and is indexed to simplify the search.

To make the bookstore appealing, a set of predefined catalogs are listed and can be directly accessed by any customer. They will bring a paginated list of books of that catalog, as well as the book's price and its picture.

Shopping cart

In order to select items, a shopping cart will hold the items selected by a customer throughout the shopping experience. The shopping cart will be used while the customer is buying items. As long as the customer exits the application, the shopping cart will be discarded as well as the books.

Remember me feature

To allow users to access the bookstore faster, without going through the authentication process, a customer may check the **Remember Me** checkbox to avoid the logout process. This convenience mechanism allows customer to directly access their wish lists without needing to log in again.

Promotions

Some items from the bookstore may be on sale and they are presented at the welcome page. Since the promotion may evaluate many aspects of the shopping cart, the bookstore application will process the shopping cart and the promotion codes provided by the customer using a rule engine.

Wish list

The customer may not have the resources to buy a certain item. To guarantee that a customer may buy it later, without the need to take notes, a wish list can be invoked. It can be used to store all the desired items in the customer's account from both the catalog and the shopping cart. This tool permits customers to quickly open their accounts, add books, and pay for their items.

Checkout

The checkout process allows a customer to provide their credit card information and process its purchase safely since the order will not store any sensitive data. Additionally, the books can be shipped to a different address than the credit card uses.

Administrative tools

An administrator may access all the purchases made so far in order to ship to the final destination.

Quiz: Bookstore Features

Choose the correct answer to the following questions:

1. How is promotion applied to a purchase? (Select one.)

   a. Each product will have its own promotion and will be applied individually during the checkout process.
   b. Each product will have its own promotion and will be applied individually inside the catalog.
   c. A promotion will be applied to the shopping cart and the values discounted during the checkout process.
   d. A promotion will be applied to the catalog and will be presented during the checkout process.

2. Each customer may have a wish list where all desired books may be stored. Will the books in a wish list keep any promotion applied? (Select one.)

   a. Yes
   b. No

3. Will each purchase store sensitive data from a customer such as credit card number and its verification code? (Select one.)

   a. Yes
   b. No

4. May an administrator from the bookstore application change any information from the original purchase? (Select one.)

   a. Yes
   b. No

Solution

Choose the correct answer to the following questions:

1. How is promotion applied to a purchase? (Select one.)

   a. Each product will have its own promotion and will be applied individually during the checkout process.
   b. Each product will have its own promotion and will be applied individually inside the catalog.
   c. **A promotion will be applied to the shopping cart and the values discounted during the checkout process.**
   d. A promotion will be applied to the catalog and will be presented during the checkout process.

2. Each customer may have a wish list where all desired books may be stored. Will the books in a wish list keep any promotion applied? (Select one.)

   a. **Yes**
   b. No

3. Will each purchase store sensitive data from a customer such as credit card number and its verification code? (Select one.)

   a. Yes
   b. **No**

4. May an administrator from the bookstore application change any information from the original purchase? (Select one.)

   a. Yes
   b. **No**

Deploying *The Shoppe*

Objective

After completing this section, students should be able to build and deploy the application to JBoss EAP and MySQL.

Creating the database

The bookstore application stores all products and orders in a MySQL database and it should be built before deploying the application. A SQL script is provided by the developers to create the database structure (tables and sequences) and provide a minimal runtime environment for the bookstore.

To create the database, there are two approaches to run the script:

- *Administration tools*: MySQL provides an administration tool called **mysql** that may be used to run all administration tools needed.
- *JBoss Developer Studio*: JBoss Developer Studio has a database plug-in that allows any developer to connect to and manage the contents from the database.

Using administration tools

MySQL provides a command line tool that can be used to access the database. In order to install it, RHEL7 provides an installation manager called yum that can be used to manage all the dependencies needed by the MySQL client.

```
# yum install mysql
```

Note

The yum command must be executed as root.

To access a MySQL database, the following command can be used:

```
$ mysql -uroot❶ -ppassword❷ -h127.0.0.1❸ bookstore❹
```

❶ The username created to access the database.
❷ The password.
❸ The hostname where the database server is hosted.
❹ The database name.

A command line will be presented to run data definition language (DDL) commands.

Note

Due to the amount of the commands needed to create a database, normally, an external SQL file with all the commands can be used instead. MySQL client allows to redirect the output from the file to the CLI using a syntax similar to the following:

```
$ mysql -uroot -ppassword -h127.0.0.1 bookstore < SQL-Dump.sql
```

Using JBoss Developer Studio

JBDS has a set of plugins for database development that can be used as a GUI to access most of the databases available in the market. The tools can be accessed as a group using the menu **Window** > **Show Perspective** and select the **Database Development** perspective. To issue commands to a database using JBDS, a JBDC driver is needed, and it must be installed in JBDS.

To configure a database, from the main menu, select **Window** > **Show View** > **Other...**. A dialog box called **Show View** will be opened. By selecting the **Data Management** > **Data Source Explorer** and click the **OK** button, a database connection can be created. Right-click the **Database Connections** node from the **Data Source Explorer** tab and select **New** from the context menu. By choosing the **Connection profile** from a database, JBDS will be able to connect and send commands to the database. In order to complete the setup all the database details must be passed as parameters, such as:

- Database.
- URL: JBDC compatible URL format to connect to the database.
- Username.
- Password.

JBDS offers the **Test Connection** button to verify if the database is running and the credentials are correct.

To create the database, using an SQL file open it using **File** > **Open** menu. A SQL script will be presented with SQL syntax highlighting. To execute the commands, a database must be selected from the editor's tab. The database name and the database name must be selected to run the commands. Finally, right-click the script and select **Execute All** to create the database.

Build and deploy the Java EE application

To build the bookstore application, a developer must:

- *Clone a Git repository*: The source code can be cloned in both the bash and JBDS. To clone the source code from the bash, execute a command like **`git clone <gitRepoURL>`**.
- *Build the package*: Most Java projects are now developed using Maven and they usually contains a **`pom.xml`** file in the root directory from the project. It contains all the dependencies and configuration needed to deployed a Java application. Normally, this objective can be achieved by running the command **`mvn package`** from the root directory of the project, where the **`pom.xml`** is found. The resulting package will be stored at the **`target`** directory from the root directory from the project.

- *Start the JavaEE container*: In order to test a JavaEE application, a JavaEE container must be running to deploy it. For JBoss EAP, a script is provided to start called **`standalone.sh`** from the **`$JBOSS_HOME/bin`** directory.
- *Deploy the JavaEE application*: Some JavaEE containers provide tools to deploy an application using the command line or even Maven. JBoss EAP provides a maven plugin called jboss-as that allows the remote deployment of the applications.

Note

The plugin web site: **`https://docs.jboss.org/jbossas/7/plugins/maven/latest/`**

```
$ mvn jboss-as:deploy
```

Demonstration: Deploying and Testing *The Shoppe*

In this example, please follow along with the steps while your instructor demonstrates how to deploy and run the bookstore application.

1. Clone the repository from the Git repository.

   Open a Terminal window and run the following command:

   ```
   [student@workstation ~]$ cd ~/D0290/labs
   [student@workstation labs]$ git clone http://workstation.podX.example.com/
   bookstore.git
   ```

2. Start JBoss EAP 6.4.

   From another Terminal window run the following command:

   ```
   [student@workstation ~]$ cd ~/D0290/opt/jboss-eap-6.4/bin
   [student@workstation jboss-eap-6.4]$ ./standalone.sh
   ```

   This will start the JBoss EAP 6.4 instance. Leave it running.

3. Generate the package.

   To generate the artifact that will be deployed in JBoss EAP 6.4, Apache Maven will be used. To generate it, run from the another Terminal window:

   ```
   [student@workstation ~]$ cd ~/D0290/labs/bookstore
   [student@workstation bookstore]$ mvn package
   ```

   Open the **`~/D0290/labs/bookstore/target`** directory and verify that a **`bookstore.war`** file was generated.

   Deploy the application.

   From another command line, execute the following command:

```
[student@workstation ~]$ cd ~/DO290/labs/bookstore
[student@workstation bookstore]$ mvn jboss-as:deploy
```

The last command will start the deployment process locally.

4. Test the application.

   From **Applications** > **Internet** > **Firefox** open the following URL: **`http://localhost:8080/bookstore`**. Log in as **`admin/redhat`** and add some books to the shopping cart. Additionally, check out and evaluate the order.

5. Undeploy the application.

   To remove the application from JBoss EAP 6.4, run the following command:

```
[student@workstation bookstore]$ mvn jboss-as:undeploy
```

6. Stop JBoss EAP.

   Press **`Ctrl+C`** on the Terminal window where JBoss was started.

Guided Exercise: Deploying and Testing *The Shoppe*

In this lab, the student will open the project in JBoss Developer Studio, connect to a database, and deploy the application.

Resources	
Files	`/home/student/DO290/labs/bookstore`
Application URL	`http://localhost:8080/bookstore`

Outcomes

You should be able to:

- Create the database.
- Deploy the application.

1. Clone the Bookstore Application
   Open a Terminal window and run the following commands to clone the source code. Replace *X* by your student number.

```
[student@workstation ~]$ cd ~/DO290/labs
[student@workstation labs]$ git clone http://workstation.podX.example.com/
bookstore.git
```

2. Start JBoss EAP
   To run JBoss EAP 6, open a new Terminal window and execute the following command:

```
[student@workstation ~]$ cd ~/DO290/opt/jboss-eap-6.4/bin
[student@workstation jboss-eap-6.4]$ ./standalone.sh
```

3. Build the Application
   In order to build the application, Maven will be used. For this, run the following commands from a new Terminal window:

```
[student@workstation labs]$ cd ~/DO290/labs/bookstore
[student@workstation bookstore]$ mvn clean package
```

   Check that the bookstore application will be built using Maven. At the end, a **`Build Successful`** message should be presented.

   The final bits are stored at the **`/home/student/DO290/labs/bookstore/target`** directory.

4. Deploy the application
   Run the following command from the Terminal window opened to build the application.

```
[student@workstation bookstore]$ mvn jboss-as:deploy
```

5. Test the Application
   Open a web browser and navigate to **`http://localhost:8080/bookstore`**. Create a customer, buy some books, use the coupons mentioned at the top of the page, check out, and pay for them.

6. Undeploy the application
   From the Terminal window where the **`mvn jboss-as:deploy`** was executed, run the following command to remove the bookstore application from EAP:

```
[student@workstation bookstore]$ mvn jboss-as:undeploy
```

7. Stop JBoss EAP
   Hit **`Ctrl+C`** on the Terminal window where JBoss EAP was started.

Summary

In this chapter, you learned:

- The technologies and APIs used by the bookstore were chosen to keep the environment portable to deploy in any Java EE container as well as in any database.
- JBoss EAP was the Java EE container chosen to run the bookstore application to make it compatible with the environment run in OpenShift Enterprise.
- The business requirements from the web application were defined to simplify user interaction and make it visual appealing.
- To use **`mysql`** command to connect to a remote database passing as parameters the login parameter (**`-u<username>`**) and password (**`-p<password>`**) and the database instance (the last parameter).
- To use **`mvn`** to build, package (using **`package`** as a parameter), and deploy the Java EE app in JBoss (**`mvn jboss-as:deploy`**).
- The steps needed to set up, deploy, and run the bookstore application use developer-focused tools, and to ease the deployment process in a local machine.

CHAPTER 3

INTRODUCING OPENSHIFT ENTERPRISE BY RED HAT

Overview	
Goal	List the features and architecture of the OpenShift Enterprise by Red Hat product.
Objectives	• List the typical usage of the product and its features. • Deploy applications in Docker using images. • Describe the architecture of the product.
Sections	• OpenShift Enterprise Features (and Quiz) • Container Concepts (and Guided Exercise) • OpenShift Enterprise by Red Hat Architecture (and Quiz and Guided Exercise)

OpenShift Enterprise Features

Objectives

After completing this section, students should be able to list the typical usage of the product and its features.

Taking advantage of Platform as a Service (PaaS)

According to Wikipedia, "Platform as a service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app."

References

Platform as a Service Wikipedia reference
https://en.wikipedia.org/wiki/Platform_as_a_service

A PaaS helps developers by eliminating the wait time for the environment provisioning and testing. It allows developers to focus on developing business logic, instead of integrating runtime environments and other dependencies, and usually provides easy hooks into Continuous Integration (CI) and Continuous Delivery (CD) processes.

Although most offerings in this space are marketed in a developer-centric way, PaaS also brings many benefits to system administrators and architects. A PaaS provides standardized environments for running applications, which already include patch management, monitoring, and elasticity. It allows system administrators to focus on managing application uptime, performance, and scalability, with efficient use of hardware resources, instead of spending their time procuring and provisioning development, quality assurance (QA), and production environments.

A **PaaS** provides a preconfigured OS and runtimes, over which users can deploy their own applications. A good PaaS would go beyond a basic web server, app server, and web framework to provide databases, messaging, business process management, and enterprise integration patterns, among other middleware services. Traditional PaaS offerings tend to be very restrictive on the runtime available and offer little customization for those, but OSE imposes virtually no restrictions.

Containerized applications

A container is a segregated environment provided by the operating system, where a process tree runs with restricted access to external resources. Ideally, a container may allow an administrator to monitor efficiently all the resources used by the process tree, and enable access to other resources/processes in a controlled environment. Any external access to or from external services such as databases are enabled via a virtual network adapter that is managed by the container host. Even though it may be similar to a virtual machine (VM) managed by a virtualization tool, a container is lightweight and simpler to manage since it does not require a new operating system to support the VM execution.

OpenShift Enterprise 3 (OSE 3) uses Docker as a framework to create, manage, and deploy containers to run an application. OSE 3 creates an integrated environment where integrating

and deploying image containers is simplified. It also will be responsible for deploying images to multiple nodes and keeping a uniform environment to allow large deployment scenarios supporting software-based high availability (HA). Fortunately, Docker provides a large number of container images supporting multiple programming languages, databases, and frameworks such as:

- JavaEE.
- Ruby.
- Python.
- Perl.
- PHP.
- MySQL.
- PostgreSQL.
- Sinatra.
- Ruby on Rails.

Some of the benefits OSE brings to a developer are:

- *Simplified multicontainer integration*: An application can be created using multiple containers to enable both development and production environment. This approach guarantees consistency among production and development as well, as it supports a simpler drift management.
- *Supported container images*: Container images managed by Red Hat will have an environment tested and supported to avoid security issues. Using S2I container building, these images will be consistently updated to the latest version without major integration problems.
- *Container image reusability*: Multiple applications may be deployed to OSE 3 and they will be managed transparently by OSE runtime.

OpenShift Enterprise features

OpenShift Enterprise features includes:

- *Source-to-image (S2I) container building and deployment:* Improves the container image assembly process because it performs multiple operations at once and uses the latest images to keep application consistency.
- *Ochestration for composite applications, built from multiple containers:* Supports integration among multiple containers without the need to change multiple configuration files to minimize the large set of customization needed.
- *Polyglot, variety of databases, and frameworks supported:* Support for Java, Node.js, PHP, Perl, and Ruby directly from Red Hat, plus many others from partners, OpenShift Commons, and the larger Docker community. MySQL, PostgreSQL, and MongoDB databases directly from Red Hat and others from partners and the Docker community. Red Hat also supports JBoss Middleware products such as JBoss EAP, Active MQ, and Fuse running natively on OSE.
- *High availability and failover:* Enables an environment with multiple masters to guarantee high availability and failover support in an OpenShift Enterprise deployment.

- *Extensible control through REST API:* Controls the whole OpenShift Enterprise environment using a powerful REST API that is used from the command line interface (CLI) to the web console.
- *Team project management and collaboration:* Due to the multitenant feature, multiple collaborators may develop an application running on OpenShift, using a shared source code management system. Integration with other third-party tools can be used to trigger the build process for the latest updated version.

Quiz: OpenShift Enterprise Features

Match the following items to their counterparts in the table.

Container	Docker	JBoss EAP	Multitenant	REST	S2I

Description	Name
The responsible tool for providing runtime environments for OpenShift.	
The application programming interface used to support administration tools in OpenShift.	
The feature provided by OpenShift to create images using the latest released container image.	
The JavaEE implementation provided by OpenShift.	
The segregated environment where a process runs on.	
The capability to enable multiple access levels in OpenShift.	

Solution

Match the following items to their counterparts in the table.

Description	Name
The responsible tool for providing runtime environments for OpenShift.	Docker
The application programming interface used to support administration tools in OpenShift.	REST
The feature provided by OpenShift to create images using the latest released container image.	S2I
The JavaEE implementation provided by OpenShift.	JBoss EAP
The segregated environment where a process runs on.	Container
The capability to enable multiple access levels in OpenShift.	Multitenant

Container Concepts

Objective

After completing this section, students should be able to deploy applications in Docker using container images.

Introduction to containers and Docker

One way of looking at containers is as improved chroot jails. Containers allow an operating system (OS) process (or a process tree) to run isolated from other processes hosted by the same OS. Through the use of **Linux kernel namespaces** features, it is possible to restrict a process view of:

- Other processes (including the PID number space).
- File systems.
- User and group IDs.
- IPC channels.
- Devices.
- Networking.

Other Linux kernel features complement the process isolation brought by kernel namespaces:

- **Cgroups** limit the use of CPU, RAM, virtual memory, and I/O bandwidth, among other hardware and kernel resources.
- **Capabilities** assign partial administrative faculties; for example, enabling a process to open a low network port (<1024) without allowing it to alter routing tables or change file ownership.
- **SELinux** enforces mandatory access policies even if the code inside the container finds a way to break its isolation.

Together, those features create an execution environment where a containerized application will not have access to any file, device, socket, or process outside its own container. It looks to the application as if it is running alone inside its own OS installation. If the application needs something from outside, it needs to open a network connection, as if it were on a different server.

Each container actually gets an exclusive virtual network interface and private IP address, else it would not be able to access or be accessed by the outside world. That is why many look at containers as a kind of lightweight virtualization. Like VMs, containers are a way to allow deploying an application and its dependencies as self-contained units, without interference and conflicts caused by other applications (and their dependencies) sharing the same OS installation. But while VMs needs a complete OS image, with kernel, device drivers, and commands to manage hardware (not to mention the hardware emulation layer provided by the hypervisor), a container needs only the subset directly used by the application, without administrative commands and libraries.

The following figure illustrates two apps running on the same server using a traditional OS and using containers:

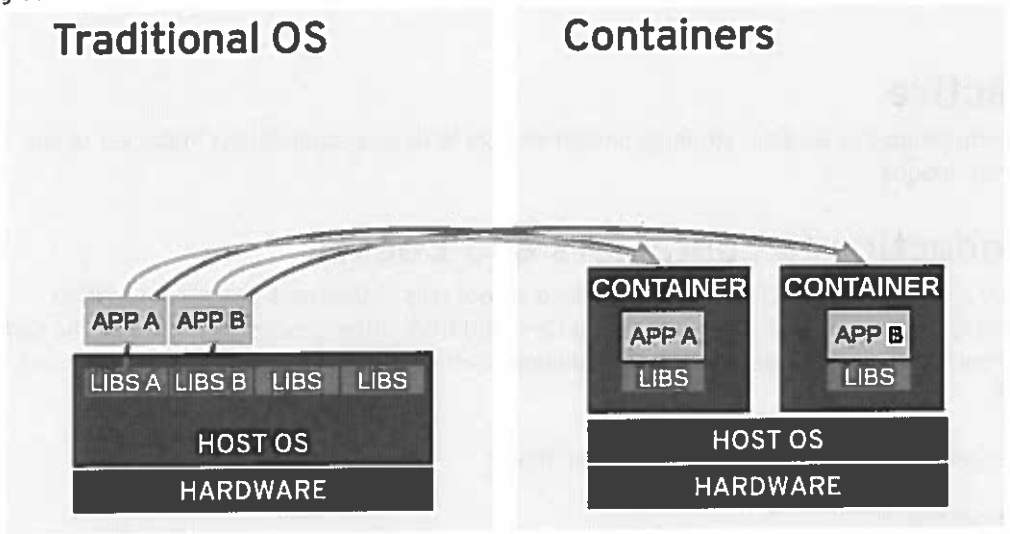

Figure 3.1: Traditional OS vs. container

A running container needs a file-system image containing not just the application process, but also all its dependencies: shared libraries, OS commands, and runtime services like app servers. This provides a third way to look at containers: as a packaging mechanism where an application and all runtime dependencies are packaged together, as a single unit.

So, if containers are actually a Linux kernel feature (or the sum of many Linux kernel features), what does **Docker** bring to the table? Docker does provide an easy CLI and API to create and manage containers. Instead of manually issuing dozens of complicated commands to configure namespaces, cgroups, SELinux, and capabilities, Docker does that with a single easy-to-use command.

Besides creating and managing containers, Docker also provides a standard **container image** format, which is actually a tar file containing the entire container file system with all libraries, commands, and working directories used by the application, plus metadata describing the image itself.

Docker formatted container images, or simply **container images**, are **immutable**. A new image should be created whenever a custom image is needed. Container images are **layered**: A complex image can be built by stacking a base OS image under an application server image and then under application-specific binaries and libraries. This allows for reuse and conserves disk space on the host. Complete images are shared between running containers, and layers are shared between images.

Docker formatted container images also provides a standard HTTP-based protocol for accessing **image registries,** which are akin to yum repositories, but hosting container images instead of RPM package files. Any organization can host its own public or private registry for sharing container images.

References

The Docker organization hosts a
> public image registry
> https://registry.hub.docker.com/

.

Red Hat provides a
> private registry for customers
> http://registry.access.redhat.com/

.

For more in-depth training about Docker containers on RHEL:

RH270 - Managing Containers with Red Hat Enterprise Linux Atomic Host
> https://www.redhat.com/en/services/training/rh270-managing-containers-red-hat-enterprise-linux-atomic-host

Managing containers with Docker commands

Docker runs as a system daemon that accepts requests through HTTP using an REST API. Docker also provides a command-line client named **docker**, which works by calling that API. To check if the Docker daemon is running, use **systemctl status docker**.

These are the essential client commands, with brief explanations about syntax and some illustrative examples. Starting with commands that deal with images and registries:

- **docker images**: List images available from the local Docker daemon:

```
# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED
         VIRTUAL SIZE
openshift/hello-openshift           latest              ccf05f873f0d        5 hours ago
           5.77 MB
registry.access.redhat.com/rhel7    latest              e3c92c6cff35        3 weeks ago
         158.2  MB
registry.access.redhat.com/rhel7    7.1-9               e3c92c6cff35        3 weeks ago
         158.2  MB
```

The REPOSITORY column contains the image name as the last path component. A Docker daemon installation comes without any image, so the images list will be empty until the sysadmin downloads some images (see the **docker pull** command).

An **image name** is prefixed by a registry name, which is usually the FDQN name of the registry host, but could be any string. An image name can also include a **tag**. Thus the full syntax for an image name is:

[*registry_name*/]**image_name**[:*tag*]

For example: **registry.access.redhat.com/rhel7:latest** and **openshift/hello-openshift:latest**

If many tags are applied to the same image, they show as if there were many distinct images. The output from **docker images** shows that two images (**registry.access.redhat.com/rhel7:latest** and **registry.access.redhat.com/rhel7:7.1-9**) have the same image ID (**ccf05f873f0d**), meaning they are exactly the same image.

Best Practice

Docker tags help with managing upgrades for an application base OS and other dependencies. An image cannot be updated (because it is immutable), and the new base image built with updates will have a new image ID, so application images that refer to the base image will not get updates, either.

But an application image build script could use the **latest** image tag, so when the application image is rebuilt, it uses the updated base OS image as a layer. But if an application needs a specific base image release and it is incompatible with later updates, it can use another tag that refers to that particular release.

The **latest** tag is special because it is assumed by most Docker commands if there are many images with the same name and registry, but no tag was specified.

Important

Most Docker commands that work on images can take either an image name or an image ID as argument.

- **docker search**: Search for images on known Red Hat container registries registries:

```
# docker search hello-openshift
```

```
NAME                              DESCRIPTION                                     STARS
     OFFICIAL   AUTOMATED
openshift/hello-openshift         Simple Example for Running a Container on ...   2

memeinstigator/hello-openshift                                                    0

jhou/hello-openshift                                                              0
```

The Docker daemon already comes preconfigured with at least the public registry, and Red Hat subscriber RPM packages also come preconfigured with the Red Hat private registry, so an empty search result means there are really no matching images.

- **docker pull**: Download an image from a registry, making it available to the local Docker daemon:

```
# docker pull openshift/hello-openshift:v0.4.3
```

```
Pulling repository registry.access.redhat.com/openshift/hello-openshift:v0.4.3
```

```
Pulling repository openshift/hello-openshift
ccf05f873f0d: Download complete
4fa448caf463: Download complete
99ce53f6077d: Download complete
Status: Downloaded newer image for openshift/hello-openshift:v0.4.3
```

For a layered image, each layer is downloaded in turn. The first layer to be downloaded is the outermost, and its ID is used to identify the whole image. In the previous command output, the image ID is *ccf05f873f0d*.

The hexadecimal layer ID is derived from an UUID, so it should be unique not only for a given Docker installation but for all Docker installations and registries everywhere.

- **`docker load`**: Load an image from a tar file in the local file system to the local Docker daemon.
- **`docker save`**: Save an image available from the local Docker daemon as a tar file in the local file system.
- **`docker rmi`**: Delete an image so it is no longer available from the local Docker daemon.
- **`docker tag`**: Add a tag to an image available from the local Docker daemon.

The following commands deal with managing containers:

- **`docker run`**: Create a new container and start a process inside the new container:

```
# docker run openshift/hello-openshift
```

```
serving on 8080
serving on 8888
^C#
```

Whatever output **`docker run`** shows is generated by the process inside the container, which is just a regular process from the host OS perspective. Killing that process stops the container. In the previous example output, the container was started with a noninteractive process, and stopping that process with **`Ctrl`**+**`C`** (**`SIGINT`**) also stopped the container.

The container image itself specifies the command to run to start the containerized process, but a different one can be specified after the container image name in **`docker run`**:

```
# docker run -it rhel7 /bin/bash
[root@8682f6516d6f /]#
```

In the previous command output, the unusual Bash prompt is particular to the **`rhel7`** image, but has no relation with the image ID. For most images, it may be difficult to know from the Bash prompt if it is running from the host or from a container.

Options **`-t`** and **`-i`** are usually needed for interactive text-based programs, so they get a proper terminal, but not for background daemons. The program must exist inside the container image.

- **`docker ps`**: List running containers.

```
# docker ps
```

```
CONTAINER ID        IMAGE                                          COMMAND
 CREATED             STATUS              PORTS                      NAMES
8682f6516d6f        registry.access.redhat.com/rhel7:7.1-9    "/bin/bash"         5
 minutes ago        Up 5 minutes                                  suspicious_brattain
```

Each container, when created, gets a **container ID** that is a hexadecimal number and looks like an image ID, but is actually unrelated. The container name (in the previous example, **suspicious_brattain**) is generated by Docker and is also unrelated to its image tags.

Stopped containers are not discarded immediately. Their local file systems and other states are preserved so they can be inspected for *post-mortem* analysis. Option **-a** lists containers that were not discarded yet:

```
# docker ps -a
```

```
CONTAINER ID        IMAGE                                          COMMAND
 CREATED             STATUS                          PORTS          NAMES
8682f6516d6f        registry.access.redhat.com/rhel7:7.1-9    "/bin/bash"         19
 minutes ago        Exited (127) 5 minutes ago                     suspicious_brattain
```

- **docker stop**: Stop a running container.

```
# docker stop 8682f6516d6f
```

```
8682f6516d6f
```

Using **docker stop** is easier than finding the container start process on the host OS and killing it.

Insight

Most Docker commands that deal with containers (except for **docker run**) output the container ID so it can be chained to other commands. For example, to stop and delete a container:

```
# docker rm $(docker stop 8682f6516d6f)
```

- **docker rm**: Remove a stopped container, discarding its state and filesystem.
- **docker logs**: Show the container standard output. It is expected that containerized applications send all their logs to standard output.
- **docker exec** : Start an additional process inside a running container:

```
# docker ps
```

```
CONTAINER ID        IMAGE                                  COMMAND
 CREATED             STATUS              PORTS              NAMES
7ed6e671a600        registry.access.redhat.com/rhel7:7.1-9   "/bin/bash"         10
 minutes ago      Up 10 minutes                           reverent_bell
```

```
# docker exec 7ed6e671a600 cat /etc/hostname
```

```
7ed6e671a600
```

As with **docker run**, **docker exec** will work only for programs and commands available inside the container image for the running container.

Warning

Common Linux troubleshooting commands like **ps** and **ping** are not included by most container images. Sysadmins may need to be creative to overcome this and other container limitations when troubleshooting failing containerized applications. The section about **persistent storage** later in this chapter will provide some troubleshooting strategies.

- **docker top**: List processes inside a running container.
- **docker inspect**: List metadata about a running or stopped container.
- **docker diff**: List files changed by a running or stopped container, relative to its start image.

Use the following commands for getting help about **docker** commands and information about the Docker daemon and its host:

- **docker**: Without any arguments, shows all **docker** commands and a brief description for each one.

```
# docker
Usage: docker [OPTIONS] COMMAND [arg...]

A self-sufficient runtime for linux containers.

Options:
...Output omitted...

Commands:
    attach    Attach to a running container
    build     Build an image from a Dockerfile
    commit    Create a new image from a container's changes
    cp        Copy files/folders from a container's filesystem to the host path
    create    Create a new container
...Output omitted...
```

- **docker <command> --help**: Show a syntax summary for **<command>**:

```
# docker stop --help
Usage: docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop a running container by sending SIGTERM and then SIGKILL after a grace period

  --help=false       Print usage
  -t, --time=10      Number of seconds to wait for the container to stop before
killing it. Default is 10 seconds.
```

- **`docker version`**: Lists internal versions for Docker client and server, and the API version.
- **`docker info`**: Lists summary information about the Docker host.

References

Getting started with docker formatted container Images on Red Hat Systems
https://access.redhat.com/articles/881893

Demonstration: Running a container image with Docker

In this example, please follow along with the steps while your instructor demonstrates how to install Docker and start a container from a base OS image.

1. Configure yum repositories for Docker packages.

   The classroom environment uses a preconfigured local repository, but a real system would need a subscription and entitlements, to the correct software channels. To see the classroom yum repo configuration, which mimics what a real system would have after being attached to the correct subscription, entitlements and channels:

   ```
   [student@workstation ~]$ cat /etc/yum.repos.d/training.repo
   ...Output omitted...
   ```

2. Verify Docker is installed.

   Docker is packaged by RHEL as a single package. Verify it is already installed:

   ```
   [student@workstation ~]$ rpm -q docker
   ```

3. Enable the Docker daemon.

   Most RHEL packages do NOT enable nor start system services after installing.

   ```
   [student@workstation ~]$ sudo systemctl enable docker
   ```

4. Start the Docker daemon:

```
[student@workstation ~]$ sudo systemctl start docker
```

5. Verify the Docker daemon is running:

```
[student@workstation ~]$ sudo systemctl status docker
docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled)
   Active: active (running) since Wed 2015-07-15 10:34:58 EDT; 2min 30s ago
     Docs: http://docs.docker.com
...Output omitted...
Jul 15 10:34:58 master.podX.example.com systemd[1]: Started Docker Application
 Container Engine.
```

If the output shows the daemon is not running or has errors, review the previous steps.

6. Check the Docker installation is already configured to use the registry preconfigured on the station VM:

```
[student@workstation ~]$ cat /etc/sysconfig/docker
```

Note the line starting with **ADD_REGISTRY**:

```
ADD_REGISTRY='--add-registry workstation.podX.example.com:5000'
```

7. Search for the base RHEL7 image.

```
[student@workstation ~]$ sudo docker search rhel7
```

```
INDEX         NAME
 DESCRIPTION   STARS     OFFICIAL   AUTOMATED
example.com   workstation.podX.example.com:5000/library/rhel7
        0
example.com   workstation.podX.example.com:5000/openshift3/mysql-55-rhel7
        0
example.com   workstation.podX.example.com:5000/openshift3/nodejs-010-rhel7
        0
example.com   workstation.podX.example.com:5000/openshift3/php-55-rhel7
        0
example.com   workstation.podX.example.com:5000/openshift3/ruby-20-rhel7
         0
```

8. Pull the RHEL7 container image to the local Docker daemon.

```
[student@workstation ~]$ sudo docker pull rhel7
```

```
Trying to pull repository workstation.podX.example.com:5000/rhel7 ...
e3c92c6cff35: Download complete
Status: Downloaded newer image for workstation.podX.example.com:5000/rhel7:latest
```

9. List the images downloaded from the **docker pull** execution. Image id and other values may be different from the output.

```
[student@workstation ~]$ sudo docker images
```

```
REPOSITORY                                      TAG                 IMAGE ID
 CREATED             VIRTUAL SIZE
workstation.podX.example.com:5000/rhel7   latest              e3c92c6cff35        4
 weeks ago          158.2 MB
```

10. Save the sample application container image to verify it is a base image with a single layer.

```
[student@workstation ~]$ sudo docker save -o rhel7.tar rhel7
[student@workstation ~]$ tar tvf rhel7.tar
```

The expected output is similar to:

```
drwxr-xr-x 0/0               0 2015-07-15 17:05
 e3c92c6cff3543d19d0c9a24c72cd3840f8ba3ee00357f997b786e8939efef2f/
-rw-r--r-- 0/0               3 2015-07-15 17:05
 e3c92c6cff3543d19d0c9a24c72cd3840f8ba3ee00357f997b786e8939efef2f/VERSION
-rw-r--r-- 0/0            1501 2015-07-15 17:05
 e3c92c6cff3543d19d0c9a24c72cd3840f8ba3ee00357f997b786e8939efef2f/json
-rw-r--r-- 0/0       166057984 2015-07-15 17:05
 e3c92c6cff3543d19d0c9a24c72cd3840f8ba3ee00357f997b786e8939efef2f/layer.tar
-rw-r--r-- 0/0             189 2015-07-15 17:05 repositories
```

Note

There should be a single directory inside the tar file because this image contains a single layer.

11. Create a container from the **rhel7** image and start a Bash shell inside.

```
[student@workstation ~]$ sudo docker run -it rhel7 /bin/bash
```

```
Usage of loopback devices is strongly discouraged for production use. Either use `--
storage-opt dm.thinpooldev` or use `--storage-opt dm.no_warn_on_loop_devices=true`
 to suppress this warning.
```

For now, disregard the warning about loopback devices. The recommended Docker configuration for production use will be shown later.

12. Start a long running process so it can be seen that a container can have multiple processes inside:

```
[root@8682f6516d6f /]# dd if=/dev/zero of=/dev/null
```

13. Open another terminal and check if the container is running:

```
[student@workstation ~]$ sudo docker ps
```

The expected output is similar to:

```
CONTAINER ID        IMAGE                                COMMAND
 CREATED             STATUS              PORTS            NAMES
8682f6516d6f        rhel7:latest                         "/bin/bash"         5
 minutes ago         Up 5 minutes                         suspicious_brattain
```

14. Open another terminal and check the processes running on a container:

```
[student@workstation ~]$ sudo docker top 8682f6516d6f
```

The expected output is similar to:

```
UID                 PID                 PPID                C
 STIME               TTY                 TIME                CMD
root                2406                980                 0
 13:10               pts/2               00:00:00            /bin/bash
root                4510                2406                99
 13:13               pts/2               00:00:14            dd if=/dev/zero of=/
dev/null
```

15. To prove each container can run different processes and have its own state, open a third terminal and start another container from the same image. Start a different long running process inside it:

```
[student@workstation ~]$ sudo docker run -it rhel7 /bin/bash
```

```
[root@0e01fb4e14dd /]# md5sum < /dev/zero > /dev/null
```

16. Return to the second terminal and inspect the running containers list again:

```
[student@workstation ~]$ sudo docker ps
```

```
CONTAINER ID        IMAGE                                COMMAND
 CREATED             STATUS              PORTS            NAMES
8682f6516d6f        rhel7:latest                         "/bin/bash"         15
 minutes ago         Up 15 minutes                        suspicious_brattain
0e01fb4e14dd        rhel7:latest                         "/bin/bash"         5
 minutes ago         Up 5 minutes                         cranky_sammet
```

Verify which processes are running on the first container ID.

Note

The container ID values may be different from the ones presented previously.

```
[student@workstation ~]$ sudo docker top 8682f6516d6f
```

```
UID                 PID                 PPID                C
 STIME               TTY                 TIME                CMD
root                2406                980                 0
 13:10               pts/2               00:00:00            /bin/bash
root                4510                2406                99
 13:13               pts/2               00:00:26            dd if=/dev/zero of=/
dev/null
```

```
[student@workstation ~]$ sudo docker top 0e01fb4e14dd
```

```
UID                 PID                 PPID                C
 STIME               TTY                 TIME                CMD
root                4465                3512                0
 13:23               pts/2               00:00:00            /bin/bash
root                4753                4465                99
 13:28               pts/2               00:00:04            md5sum
```

17. Return to the terminal running the first container and kill the long-running process using **Ctrl+C**, but do not stop the container yet; that is, do not kill the shell:

```
[root@8682f6516d6f /]# dd if=/dev/zero of=/dev/null
^C9480381+0 records in
9480381+0 records out
4853955072 bytes (4.9 GB) copied, 23.8098 s, 204 MB/s
```

Do the same for the second container.

```
[root@0e01fb4e14dd /]# md5sum < /dev/zero > /dev/null
^C
```

18. To prove the container has a different OS image than the host, compare how many files are inside **/bin** in each.

For the first container:

```
[root@8682f6516d6f /]# ls /bin | wc -l
366
```

For the host:

```
[student@workstation ~]$ ls /bin | wc -l
1688
```

The output should be different from the container. Exact numbers do not matter.

19. Check the second container, but this time using **docker exec**.

Note

Do not forget the double quotes, because the pipe must run inside the container, not inside the host.

```
[student@workstation ~]$ sudo docker exec 0e01fb4e14dd bash -c "ls /bin | wc -l"
366
```

The output should be equal to the first container, as both were created from the same base image.

20. Stop the second container, and verify it was not discarded by Docker:

```
[student@workstation ~]$ sudo docker stop 0e01fb4e14dd
0e01fb4e14dd
```

```
[student@workstation ~]$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND
 CREATED             STATUS              PORTS               NAMES
8682f6516d6f        rhel7:latest        "/bin/bash"         25
 minutes ago         Up 25 minutes                           suspicious_brattain
```

```
[student@workstation ~]$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND
 CREATED             STATUS              PORTS               NAMES
8682f6516d6f        rhel7:latest        "/bin/bash"         25
 minutes ago         Up 25 minutes                           suspicious_brattain
0e01fb4e14dd        rhel7:latest        "/bin/bash"         19
 minutes ago         Exited (137) 19 minutes ago             cranky_sammet
```

21. Stop all remaining containers and remove their states and images, leaving the host clean:

```
[student@workstation ~]$ sudo docker rmi `sudo docker images -q`
[student@workstation ~]$ sudo docker rm `sudo docker ps -aq`
```

Note

You can safely ignore "No such image" and "failed to remove images" from the last command.

Alternatively, the following commands can also be used:

```
[student@workstation ~]$ sudo docker rmi $(sudo docker images -q)
[student@workstation ~]$ sudo docker rm $(sudo docker ps -aq)
```

This concludes the demonstration.

Port mapping

Docker containers have a private, virtual network interface and a private IP address. All container private network interfaces are connected to a virtual bridge **docker0** created by the Docker daemon, and thus containers can connect to one another. Docker also creates an **iptables MASQUERADE** rule so containers have Internet access though the host.

If a container runs a network server application, one way to access it is to find the container private IP address using **docker inspect *container_id***, then run a test client from the container host. Given a running container where Apache HTTPD was started on the standard TCP port 80:

```
# docker inspect 97d581e093d6 | grep IPAddress
        "IPAddress": "172.17.0.14",
```

```
# curl http://172.17.0.4/
This is a sample welcome page.
```

But this will not work for clients who are not on the same host as the container.

A sysadmin could configure the Docker host networking to route packets from the physical network to the virtual one, but Docker provides a built-in alternative: **port mapping**. Port mapping is activated using **docker run** command option **-p *host-port*:*container-port***. This tells the **docker-proxy** process (started by the Docker daemon) to forward packets from the host's IP address and *host-port* to the container private IP address and *container-port*.

If an image named *webserver* runs Apache HTTPD, its container can be created as:

```
# docker run -p 10080:80 webserver
```

If this container Docker host is named **host.example.com**, anyone with network access to the host could try:

```
# curl http://host.example.com:10080/
This is a sample welcome page.
```

The Docker client also provides **docker ports *container-id* command** to show the ports mapped for a running container.

Building images with Dockerfiles

A container image on disk is simply a **tar** file where each folder is an image layer. An image file could be built manually just by tarring the desired files together with the metadata text files. Most base OS images are built this way.

A second way of building container images is to run a container, make modifications inside it, and them commit (save) the running container file system to a container image on disk, using the **docker commit** command.

A third way, which is the preferred one by the Docker community, is create a **Dockerfile** and process it with the **docker build** command. A Dockerfile is a plain text file containing instructions that are executed by the Docker daemon. The end result is a new container image alongside other images pulled to the same Docker daemon. A Dockerfile example follows:

```
# Dockerfile to build a basic Apache Web Server image with static content
FROM rhel7
MAINTAINER John Doe <jdoe@example.com>
EXPOSE 80
RUN yum -y install httpd
RUN echo "Sample welcome page" > /var/www/html/index.html
CMD /usr/sbin/httpd -DFOREGROUND
```

Each line contains one instruction. Most simply register metadata for the container image, while **RUN** instructions execute OS commands to add whatever is needed to the image. Which OS commands are available depends on the base image specified by the **FROM** instruction, and on previous **RUN** instructions. The **CMD** instructions tells which process to start when creating the container.

The Docker daemon runs a Dockerfile by running each instruction inside its own container. After each instruction, the container is committed. The image committed from the previous instruction is used to start the container for the next one. Those intermediate containers are removed automatically by the Docker daemon after the last instruction. The final image generated by **`docker build`** has many layers, one for each Dockerfile instruction.

The way Dockerfiles are processed makes it possible for the Docker daemon to reuse layers from previous builds, and even share layers between builds. When a change is made to a Dockerfile, and then its image is rebuilt, instructions before the change will not have to be executed again.

Very long Dockerfiles can create inefficient images because of too many layers overhead (although there are ways to deal with this). Dockerfiles are nice because they can be stored on SCM and be the source for a CI/CD process, but the image created by a Dockerfile is never a base OS image, as the **FROM** instruction is mandatory.

Persistent storage

When Docker creates a container, it creates for it a new image. This new image has an empty layer over the original shared image. All files created, deleted, or changed while the container is running are saved to this new image, while the original shared image remains unchanged. When the container is stopped, its image is kept until the container is removed. Docker even allows the stopped container to be restarted with **`docker restart`** ***`container_id`***, and this reuses the same image.

Even if it is saved on disk, storage for containers is said to be **ephemeral**; if a containerized application is stopped then restarted, it should not expect to find data from previous runs. After all, the sysadmin cannot keep too many stopped container images around, as sooner or later, there will be **`disk full errors`**.

Best Practice

A containerized application should be designed so it could be always started from a brand-new container. Restarting a container is a troubleshooting feature which is not to be abused.

Using Docker-standard ephemeral storage means containers should be stateless, and applications like database servers would not be able to run as containers. Fortunately, Docker can mount additional directories and file systems from the host when creating a container, using option **`-v`** ***`host_dir:container_dir`*** from **`docker run`**. The following example shows how

to create a container that accesses **/bin** and **/sbin** from the host and keep root privileges, so a sysadmin can troubleshoot network connectivity issues from inside the container:

```
# docker run --privileged -v /sbin:/mnt/sbin -v /bin:/mnt/bin -it rhel7 /bin/bash
[root@1284c687b924 /]# /mnt/bin/ping -c 1 www.google.com
PING www.google.com (173.194.119.17) 56(84) bytes of data.
...Output omitted...
```

```
[root@1284c687b924 /]# /mnt/sbin/ifconfig -a
eth0: flags=67<UP,BROADCAST,RUNNING>  mtu 1500
        inet 172.17.0.18  netmask 255.255.0.0  broadcast 0.0.0.0
...Output omitted...
```

If the mount point ***container_dir*** does not exist, the Docker daemon automatically creates it inside the new container image.

The following example shows that sometimes it is not so simple; a host command may need additional shared libraries, device nodes, and other features not included by the container image:

```
[root@1284c687b924 /]# /mnt/bin/ps ax
/mnt/bin/ps: error while loading shared libraries: libprocps.so.4: cannot open shared
 object file: No such file or directory
```

A nontroubleshooting use of **docker run -v**, or **Docker volumes**, is to provide persistent storage for databases and application logs. The sysadmin sets up a host directory for use by the container, then mounts it on the appropriate directory inside the container. These directory contents survive removal of the container and can be backed up using host-based tools.

Docker allows the same volume to be mounted by different containers. This has some use cases, like a volume containing web server access logs being shared by a web server container and a log analysis container. But a sysadmin has to be aware Docker itself will not manage conflicting reads to this shared volume, just like a regular OS would not. The applications running inside the containers have to support this sharing and avoid data consistency issues themselves.

Guided Exercise: Managing Containers with Docker

In this lab, you will install, configure, and manage Docker.

Resources	
Files	NA
Application URL	NA

Outcome(s)
You should be able to:
- Start a container from a sample application image.

1. Fetch Container Images
   The workstation VM is already configured with a Docker daemon that searches the classroom private registry to not need internet access.

   1.1. Pull the *hello-openshift* image using the Red Hat container registry preconfigured at **`workstation.podX.example.com:5000`**.

   ```
   [student@workstation ~]$ sudo docker pull openshift/hello-openshift
   ```

   1.2. List the available images:

   ```
   [student@workstation ~]$ sudo docker images
   ```

2. Create a New Container
   2.1. Create a container using the image just downloaded, and redirect port 18080 from the host to port 8080 from the container:

   ```
   [student@workstation ~]$ sudo docker run -p 18080:8080 openshift/hello-openshift
   ```

   This command will not return to the shell. The expected output is:

   ```
   serving on 8080
   serving on 8888
   ```

   2.2. Open another terminal to find the ID for the container just started:

   ```
   [student@workstation ~]$ sudo docker ps
   ```

   The expected output is similar to:

   ```
   CONTAINER ID        IMAGE                          COMMAND                  CREATED
           STATUS              PORTS                    NAMES
   ```

```
476a9f8bd3f7        openshift/hello-openshift   "/hello-openshift"   56 seconds
 ago       Up 55 seconds       0.0.0.0:18080->8080/tcp   thirsty_stallman
```

2.3. Find the private IP address for the container using the ID from the previous step:

```
[student@workstation ~]$ sudo docker inspect 476a9f8bd3f7 | grep IPAddress
```

The expected output is similar to:

```
        "IPAddress": "172.17.0.13",
```

2.4. Test the web application using the private IP from the previous step and the container port:

```
[student@workstation ~]$ curl http://172.17.0.13:8080/
```

The expected output is:

```
Hello OpenShift!
```

2.5. Now test the web application using the host public IP and the mapped port. Replace *X* by your student number.

```
[student@workstation ~]$ curl http://workstation.podX.example.com:18080/
```

The expected output is the same as the previous step:

```
Hello OpenShift!
```

3. Clean Up

3.1. Stop and remove the container.

```
[student@workstation ~]$ sudo docker stop 476a9f8bd3f7
[student@workstation ~]$ sudo docker rm 476a9f8bd3f7
```

3.2. Confirm there are no more containers left, either running or stopped.

```
[student@workstation ~]$ sudo docker ps -a
```

If the output shows any container, run the previous step to stop and remove each one.

This concludes this lab.

OpenShift Enterprise by Red Hat Architecture

Objective

After completing this section, students should be able to describe the architecture of the product.

Overview of OpenShift architecture

OpenShift Enterprise is a set of microservices built over Red Hat Enterprise Linux. OSE adds PaaS capabilities over Atomic like remote management, multitenancy, increased security, application life-cycle management, and self-service interfaces for developers. The following figure illustrates the OpenShift software stack, and each component is described afterward.

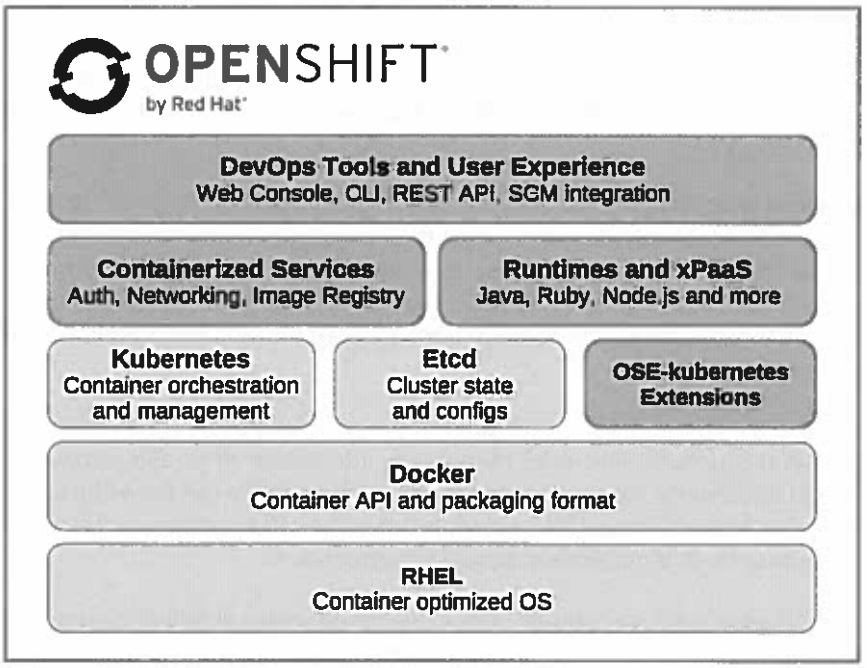

Figure 3.2: OpenShift Enterprise 3 software stack

In the figure, going from bottom to top, and from left to right, the basic container infrastructure is shown:

- The base OS must be a vanilla **RHEL**.
- **Docker** provides the basic container management API and the container image file format.
- **Kubernetes** is a Google project aimed to manage a cluster of hosts (physical or virtual) running containers. It works with **templates** that describe multicontainer applications composed of multiple **resources**, and how they interconnect. If Docker is the "core" of OSE, Kubernetes is the "heart" that keeps it moving.

- **Etcd** is a distributed key-value store, used by Kubernetes to store configuration and state information about the containers and other resources inside the OSE cloud.

OpenShift adds to the Docker + Kubernetes container infrastructure the capabilities required to provide a PaaS platform. Continuing from bottom to top and from left to right:

- **OSE-Kubernetes extensions** are additional resource types stored in Etcd and managed by Kubernetes though new controllers provided by OpenShift to manage the application life cycle: (build, deploy and updates), networking, and storage. Those additional resource types form the OSE internal state and configuration, alongside application resources managed by standard Kubernetes resources.

- **Containerized services** fulfill many PaaS infrastructure functions such as networking and authorization. Some of them run all the time, while others are started on demand. OSE leverages the basic container infrastructure from Docker and Kubernetes for most internal functions. That is, most OSE internal services run as containers orchestrated by Kubernetes.

- **Runtimes and xPaaS** are base container images ready for use by developers, each preconfigured with a particular runtime language or database. They can be used as-is or extended to add different frameworks, libraries, and even other middleware products. The xPaaS offering is a set of base images for JBoss middleware products such as JBoss EAP and ActiveMQ.

- **DevOps tools and user experience**: OSE provides web and CLI management tools for developers and system administrators, allowing the configuration and monitoring of both applications and OSE services and resources. Both web and CLI tools are built from the same REST APIs, which can be leveraged by external tools like IDEs and CI platforms. OSE also can reach external SCM repositories and container image registries and bring their artifacts into the OSE cloud.

OpenShift will not hide the core Docker and Kubernetes infrastructure from developers and system administrators. Instead it leverages them for its internal services, and allows importing raw containers and Kubernetes resources into the OSE cloud so they can benefit from added capabilities. The reverse is also true: Raw containers and resources can be exported from the OSE cloud and imported into other Docker-based infrastructures.

The main value OSE adds to Docker + Kubernetes is automated development workflows, so that application build and deployment happen inside the OSE cloud, following standard processes. A developer does not need to know the low-level Docker details; OSE will take the application, package it, and start it as a container.

Note

The Docker community itself had no features to support composite applications running as multiple interconnected containers, which are needed by both traditional layered enterprise applications and newer microservices architectures. Recently they launched the **Docker Compose** and **Docker Swarm** projects to fill this gap. But Google's **Kubernetes** was already a popular choice to fulfill this need. Kubernetes is battle-tested inside Google, where it spins more than 2 billion Docker containers daily.

Master and nodes

While the OSE software stack presents a static perspective of software packages that form OpenShift, the next figure presents a dynamic view of how OSE works:

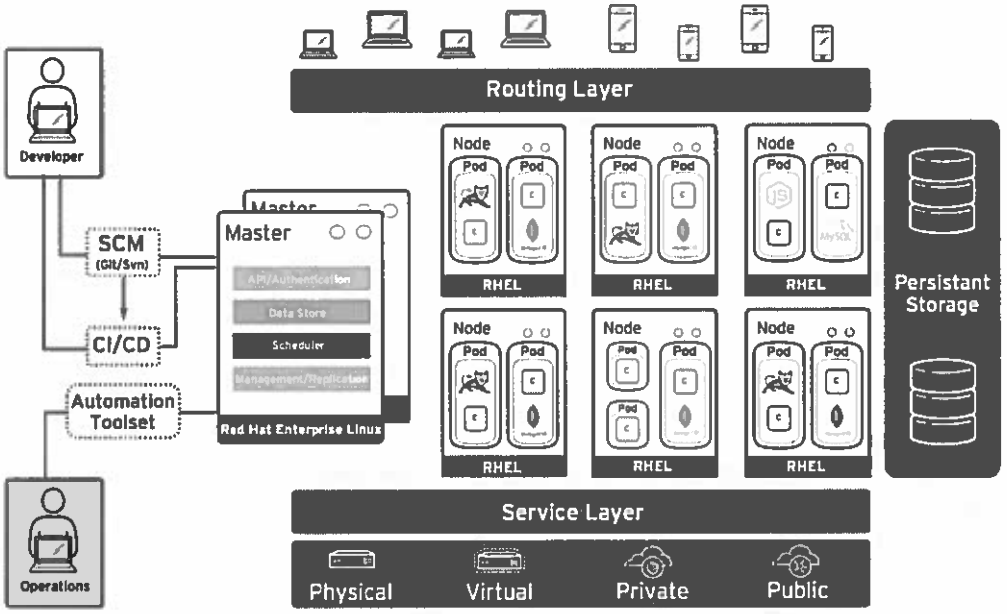

Figure 3.3: OpenShift Enterprise master and nodes

Both masters and nodes can run RHEL, but RHEL Atomic is recommended for nodes because of its smaller footprint. The **master** runs OpenShift core services such as authentication, and provides the API entry point for administration. **Nodes** run applications inside containers, which are in turn grouped into pods. This division of labor actually comes from Kubernetes, which uses the term **minions** for nodes.

OSE masters runs the **Kubernetes master** services and **Etcd** daemons, while the nodes run the Kubernetes **`kubelet`** and **`kube-proxy`** daemons. While not depicted in the figure, the masters could also be nodes themselves. **Scheduler** and **Management/Replication** in the figure are Kubernetes master services, while **Data Store** is the Etcd daemon.

The Kubernetes scheduling unit is the **pod**, which is a grouping of containers sharing a virtual network device, internal IP address, TCP/UDP ports, and persistent storage. A pod can be anything from a complete enterprise application, including each of its layers as a distinct container, to a single microservice inside a single container; for example, a pod with one container running PHP under Apache and another container running MySQL.

Kubernetes manages **replicas** to scale pods. A replica is a set of pods sharing the same definition. For example, a replica consisting of many Apache+PHP pods running the same container image could be used for horizontally scaling a web application.

OpenShift projects and applications

Over Kubernetes resources such as pod and service, OpenShift manages **projects** and **users**. A project groups Kubernetes resources so access rights can be assigned to users. A project can also be assigned a **quota** that limits its number of defined pods, volumes, services, and other resources.

There is no application concept in OSE 3. The closest equivalent would be project. Many organizations have different concepts about what an application is. For example, does a set of collaborating services form a single application, or does each service form an independent application? Do a mobile app's back-end services form an application *per se*, or are they just part of a larger application, where the mobile app is just the UI?

Even if OSE has no concept of application, it provides a **new-app** command. This creates resources inside a project, but none of them are an application resource. This command is just a shortcut to configure a project with common resources for a standard developer workflow.

Source-to-image

Developers and system administrators can use ordinary Docker and Kubernetes workflows with OpenShift, but this requires them to know how to build container image files, work with registries, and other low-level functions. OSE 3 allows developers to work with standard source control management (SCM) repositories and integrated development environments (IDE).

The **source-to-image**, or **S2I**, process in OpenShift pulls code from a SCM repository, automatically detects which kind of runtime that source code needs, and starts a pod from a base image specific for that kind of runtime. Inside this pod, it builds the application as the developer would (for example, running Maven for a Java application). If the build is successful, another image is created, layering the application binaries over its runtime, and this image is pushed to an image registry internal to OSE 3. A new pod can then be created from this image, running the application. S2I can be viewed as a complete CI/CD pipeline already built into OpenShift Enterprise.

There are many variations to CI/CD pipelines, and that is why the pipeline resources are exposed inside the project, so they can be fine-tuned to a developer's needs. For example, an external CI tool like Jenkins could be used to start the build and run tests, then label the just-built image as success or failure and promoting it to QA or production. Over time, an organization can create their own templates for those pipelines, including custom builders and deployers.

Managing OpenShift resources

OpenShift resources such as images, containers, pods, services, builders, templates, and others are stored on Etcd and can be managed by the OSE CLI, the web console, or the REST API. Those resources can be viewed as JSON or YAML text files, and shared on SCM. OSE 3 can even retrieve those resource definitions from an external SCM.

Most OpenShift operations are not imperative; OSE commands and API calls do not require an action be performed immediately. OSE commands and APIs usually create or modify a resource description stored in Etcd. Etcd then notifies Kubernetes/OSE controllers, which warn those resources about the change. Those controllers take action so, eventually, the cloud state reflects the change.

For example, if a new pod resource is created, Kubernetes will schedule and start that pod on some node, using the pod resource to determine which image to use, which ports to expose, and so on. Another example: If a template is changed so that it specifies that there should be more pods to handle the load, Kubernetes will schedule additional pods (replicas) to satisfy the updated template definition.

Warning

Although Docker and Kubernetes are exposed by OpenShift, developers and administrators should primarily use the OSE CLI and OSE APIs to manage applications and infrastructure. OSE adds additional security and automation capabilities that would have to be configured manually, or would simply be unavailable, when using Docker or Kubernetes commands and APIs directly. The access to those core components can be valuable for system administrators as a troubleshooting help.

Networking

Docker networking is very simple; it creates a virtual kernel bridge and connects each container network interface to it. Docker itself does not provide a way to allow a pod from one host to connect to a pod from another host. Docker also does not provide a way to assign a public fixed IP address to an application so external users can access it.

Kubernetes provides service and route resources to manage network visibility between pods and from the external world to them. A **service** load-balances received network requests among its pods, while providing a single internal IP address for all clients of the service (which usually are other pods). Containers and pods do not need to know where other pods are, they just connect to the service. A **route** provides an external IP to a service, making it externally visible.

Kubernetes service and route resources need external help to perform their functions. A service needs a software-defined network (SDN) that provides visibility between pods on different hosts, while a route needs something that forwards or redirects packets from the external, public IP to the service internal IP. OSE provides those as the **service layer** based on **Open vSwitch**, and as the **routing layer**, implemented by a distributed **HAProxy** farm.

Persistent storage

As a pod could be, at any time, stopped in one node and restarted on another node, plain docker storage is clearly inadequate. If a database pod, full of data, was started on another node, it would suddenly become empty.

Kubernetes provides a framework for managing external persistent storage for containers. Kubernetes recognizes a **PersistentVolume** resource, which defines a local or a network storage. A pod resource can reference a **PersistentVolumeClaim** resource in order to access a certain storage size from a PersistentVolume.

Kubernetes also specifies if a PersistentVolume resource can be shared between pods or if each pod needs its own PersistentVolume with exclusive access. When a pod moves to another node, it keeps connected to the same PersistentVolumeClaim and PersistentVolume instances. So a pod persistent storage data follow it, regardless of the node where it is scheduled to run.

OSE adds to Kubernetes a number of **VolumeProviders**, that maps PersistentVolumes to an enterprise storage, using NFS, iSCSI, or a cloud block volume service such as OpenStack Cinder.

High availability

High availability (HA) on an OpenShift Enterprise 3 cloud has two distinct aspects: HA for the OSE infrastructure itself, that is, the masters, and HA for the applications running inside the OSE cloud.

For the OSE 3.0 release, there is no out-of-the-box HA for masters yet. It has to be provided externally, using something like the **RHEL High Availability Add-on**. It is simpler than it sounds, because most OSE internal services are stateless, and their data store (Etcd) is already distributed and fault-tolerant. The OSE API service needs just an external load-balancer as it gets all state from the data store. Singleton services like the scheduler service, and stateful services like internal registry and the shared router layer, would need special attention. Note that configuring HA for OpenShift masters is outside the scope for this course.

References

For training about the RHEL High Availability Add-on:

RH436 - Red Hat Enterprise Clustering and Storage Management
http://www.redhat.com/en/services/training/rh436-red-hat-enterprise-clustering-and-storage-management

For applications, that is, pods: OSE 3 handles this out-of-the-box. If a pod is lost, whatever the reason, Kubernetes schedules another copy, and connects it to the service layer and to the persistent storage. If an entire node is lost, Kubernetes will schedule replacements for all its pods, and eventually all applications will be available again. The applications inside the pods are responsible for their own state, that is, they need to be HA by themselves, if they are stateful, employing proven techniques such as HTTP session replication or database replication.

OpenShift Enterprise version 2 vs. version 3

OpenShift Enterprise 3 is not an incremental evolution over the previous version. It is a complete rewrite. Many lessons learned from OSE 2 influenced OSE 3 design, but no code was shared. Currently there is no automated migration from OSE 2 to OSE 3, but this is expected to come in future point releases.

The following table shows how main OSE 2 concepts translate to OSE 3:

OpenShift concepts - V2 versus V3		
Operating system layer	RHEL6	RHEL7
Container model	Gears	Docker
Orchestration engine	Broker	Kubernetes
Package model	Cartridges	Container images
Routing tier	Node-level	Platform fabric
Application grouping	Domains	Projects and labels

OpenShift 2 was already based on Linux containers, but management, packaging, and orchestrating was OSE-specific, while OpenShift 3 adopts the standard Docker container model and packaging, while also adopting (and augmenting) standard Kubernetes orchestration.

Applications were the focal point with OpenShift 2. An OpenShift 2 application was a single unit, containing one web framework and at most one cartridge of each type in the same gear. In OpenShift 3, the **project** replaces applications as the focal point. It can contain any number of components of any kind. Any grouping or structure can be created inside a project by using **labels**.

In OSE 2, all building mechanics were inside the gears themselves. At runtime, the same gear would build, deploy, and run an application, and this led to downtime with OSE 2 non-scaled apps. With OSE 3, build and deploy happen on distinct pods rather than running applications. This allow more flexible developer workflows and also allows applications to stay on while the next release is built and deployed, no matter how complicated those operations may be for any particular scenario. As the build result is a new image pushed to the registry, it is easier to share the same application binaries though QA and production environments.

Another improvement in OSE 3 compared to OSE 2 is that routes are a first-class object in OSE 3, while in OSE 2 they were embedded in the cartridge logic, much like building and deploying. OpenShift 3 routing is a shared layer, allowing better control from sysadmins, while in OpenShift 2 each application had its own embedded routing.

References

OpenShift 3.0 What's New?, Chapters 3, 4 and 5

https://access.redhat.com/beta/documentation/en/openshift-enterprise-30-whats-new/whats-new

Quiz: OpenShift Enterprise Architecture

Match the following items to their counterparts in the table.

Container	Docker	Etcd	JSON	Kubernetes	Master
Node	Open vSwitch	Pod	Project	Route	S2I
Service	Volume	xPaaS			

Description	Name
Stores OpenShift cloud resource definitions.	
Defines the container image format.	
Orchestrates running composite applications.	
Provides JBoss middleware certified container images.	
Shares networking and storage configurations from enclosing pod.	
Runs the OpenShift REST API, authentication, scheduler, and configuration data store.	
Builds and deploys applications from source code.	
Runs pods, kubelet, and proxy.	

Description	Name
File format used to describe OpenShift cloud resources.	
Load-balances requests for replicated pods from the same application.	
Provides persistent storage for stateful applications like relational databases.	
Allows access to applications from external networks.	
Set of containers that must run on the same node.	
Software-defined network that allows pods from different nodes to be part of the same service.	
Can be assigned resource quotas.	

Solution

Match the following items to their counterparts in the table.

Description	Name
Stores OpenShift cloud resource definitions.	Etcd
Defines the container image format.	Docker
Orchestrates running composite applications.	Kubernetes
Provides JBoss middleware certified container images.	xPaaS
Shares networking and storage configurations from enclosing pod.	Container
Runs the OpenShift REST API, authentication, scheduler, and configuration data store.	Master
Builds and deploys applications from source code.	S2I
Runs pods, kubelet, and proxy.	Node
File format used to describe OpenShift cloud resources.	JSON
Load-balances requests for replicated pods from the same application.	Service
Provides persistent storage for stateful applications like relational databases.	Volume
Allows access to applications from external networks.	Route

Description	Name
Set of containers that must run on the same node.	Pod
Software-defined network that allows pods from different nodes to be part of the same service.	Open vSwitch
Can be assigned resource quotas.	Project

Guided Exercise: Installing OpenShift Enterprise

In this lab, you will install OpenShift Enterprise in your environment.

Resources	
Files	NA
Application URL	NA

Outcome
An OpenShift Enterprise (OSE) instance ready to be used.

1. Check Pre-Requisites
   To be able to install OSE without internet access, many container images required by OpenShift Enterprise were preloaded into the private image registry on the workstation VM. This registry is populated during the first boot of the VM workstation but the process can take a few minutes after the graphical login screen in shown in the workstation VM console.

   1.1. To be sure the private registry is ready before installing OSE, open a Terminal from the workstation VM (**Applications** > **Utilities** > **Terminal**) and check the local Docker daemon on workstation can find OpenShift Enterprise container images on the private registry, using the following command:

   ```
   [student@workstation ~]$ sudo docker search openshift3
   ```

   The expected output shows some images found in the image registry at **`workstation.podX.example.com:5000`**:

   ```
   INDEX         NAME
    DESCRIPTION   STARS     OFFICIAL   AUTOMATED
   example.com   workstation.podX.example.com:5000/openshift3/mysql-55-rhel7
                 0
   ...Output omitted
   ```

2. Install OpenShift Enterprise
   OpenShift Enterprise (OSE) will be installed on two virtual machines (VMs): master and node. These VMs are available in your environment.

   2.1. Start the node and the master VMs.

   2.2. To install OpenShift Enterprise, connect to the master VM using the following command. Replace *X* by your student number.

   ```
   [student@workstation ~]$ ssh root@master.podX.example.com
   ```

   Log into the master VM using the following credentials:
   - Username: **`root`**
   - Password: **`redhat`**

2.3. Make sure the node VM is ready to accept SSH connections from the master VM.

Log in to the node using the same user and password used to log in to the master, using the following command. Replace *X* with your student number.

```
[root@master ~]# ssh root@node.podX.example.com
```

Log out immediately to return to the master VM prompt.

```
[root@node ~]# logout
```

2.4. Execute the following command to download the installer:

```
[root@master ~]# curl -o ose-install.tar.gz \
      http://content.example.com/courses/ose/installers/ose-install.tar.gz
```

Unpack the installer:

```
[root@master ~]# tar xzf ose-install.tar.gz
```

Run the provided installation script:

```
[root@master ~]# bash installOSE.bash
```

This will install OSE for both VMs.

Note

The **installOSE.bash** Bash shell script is NOT part of the OSE official installer. It was added to perform some pre-requisite steps and some post-installation steps that are required by the classroom environment.

To get detailed instructions on how to install OSE see the official OpenShift Enterprise 3 documentation on **http://access.redhat.com** and the course **DO280 - OpenShift Enterprise Administration**.

3. Verify The OSE Installation
Check the OSE instance just installed is really working.

3.1. Check the OSE instance has two nodes using the following command:

```
[root@master ~]# oc get node
```

The expected output is:

```
NAME                     LABELS
 STATUS
master.podX.example.com   kubernetes.io/hostname=master.podX.example.com
 Ready,SchedulingDisabled
```

```
node.podX.example.com     kubernetes.io/hostname=node.podX.example.com     Ready
```

3.2. The OSE instance should have two running pods: a router and a registry pod. Check using the following command:

```
[root@master ~]# oc get pod
```

The expected output is similar to:

```
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-2-cmpdu    1/1       Running   0          20s
trainingrouter-1-mhume     1/1       Running   0          20s
```

Normal pod names get a random suffix, there should be one with "registry" and another with "router" in the name that are ready and running.

Note

The OSE internal registry that runs as a pod serves OSE processes like Source-to-Image (S2I). It is not related to the classroom private registry running on the workstation VM.

3.3. OSE creates a Software-Defined Network (SDN) to interconnect pods. Verify this is working by finding the OSE registry service name and checking it replies to API requests.

Use the following command to find the registry service IP and port:

```
[root@master ~]# oc status
```

The expected output is similar to:

```
In project default

service/docker-registry - 172.30.105.26:5000
  dc/docker-registry deploys docker.io/openshift3/ose-docker-registry:v3.0.1.0
    #2 deployed about a minute ago - 1 pod
    #1 deployed 2 minutes ago

service/kubernetes - 172.30.0.1:443

service/trainingrouter - 172.30.42.240:80
  dc/trainingrouter deploys docker.io/openshift3/ose-haproxy-router:v3.0.1.0
    #1 deployed 2 minutes ago - 1 pod

To see more, use 'oc describe <resource>/<name>'.
You can use 'oc get all' to see a list of other objects.
```

Use the IP address and port from the **`service/docker-registry`** entry to invoke an API URL using **`curl`**:

```
[root@master ~]# curl http://172.30.105.26:5000/healthz
```

The expected output is:

```
{}
```

This concludes the exercise.

Summary

In this chapter, you learned:

- OpenShift Enterprise is a PaaS cloud platform based on RHEL Atomic, containers, and Kubernetes orchestration.
- OpenShift Enterprise allows developers to focus on source code and rely on the cloud infrastructure to build and deploy containers to run applications.
- A container uses Linux kernel features to provide an isolated runtime environment for processes.
- Container images package application binaries and their dependencies as a single unit.
- To list the images available in a remote docker registry (**`docker search`**).
- To download an image to the local registry (**`docker pull`**).
- To list the images available locally (**`docker images`**).
- To start an image locally (**`docker run`**)
- To list the processes from a docker container (**`docker ps`**)
- OSE architecture employs master servers that manage node servers that run applications as containers.
- OSE services provide additional authentication, security, scheduling, networking, storage, and application life-cycle management over standard Kubernetes orchestration.

CHAPTER 4

MANAGING OPENSHIFT ENTERPRISE APPLICATIONS

Overview	
Goal	Control and maintain applications using the OpenShift Enterprise command-line interface
Objectives	• Install the CLI for remote access to an OpenShift Enterprise instance. • Execute CLI commands to manage pods, services, routes, builds, and other OSE resources.
Sections	• Installing the Command-line Tools (and Guided Exercise) • Managing an Application with the CLI (and Guided Exercise)

Installing the Command-line Tools

Objective

After completing this section, students should be able to install the CLI for remote access to an OpenShift Enterprise instance.

Locating the binaries

The OSE CLI exposes commands for managing applications, as well as lower-level tools to interact with each component of a system. The binaries for Mac, Windows, and Linux are available for download from the Red Hat Customer Portal via the following link.

Note

The binaries from multiple platforms can be downloaded from the *Red Hat Customer Portal* [https://access.redhat.com/downloads/content/290].

Installing the CLI tools

The CLI is provided as compressed files that can be decompressed to any directory. In order to make it simple for any user to access the OSE CLI, it is recommended that it is made available in a directory mapped to the environment variable called **PATH** from the OS.

Installing the CLI tools on OS X involves copying the OpenShift binary into the **`/usr/local/bin/`** folder.

Installing the CLI tools on Windows involves using the **`oc.exe`** to open an OpenShift shell.

The **`oc login`** command is the best way to initially set up the OpenShift CLI, and serves as the entry point for most users. The interactive flow helps to establish a session to an OpenShift server with the provided credentials. The information is automatically saved in a CLI configuration file that is then used for subsequent commands. To log into a remote server, use the following syntax:

```
$ oc login <hostname>:<port>
```

The following is an example displaying the interactive setup and login using the **`oc login`** command. Replace *X* with the student's number:

```
$ oc login master.podX.example.com:8443
Username: student
Authentication required
Password: ******
Login successful.

You don't have any projects. You can try to create a new project, by running

    $ oc new-project <projectname>

Welcome to OpenShift! See 'oc help' to get started.
```

Note

At any point, using the command **oc <command> --help** will display the parameters and give an explanation for how to use the command.

Guided Exercise: Installing the CLI Tools on Linux

In this lab, you will install locally the OpenShift CLI tool.

Resources	
Files	**/home/student/DO290/installers/oc-linux.tar.gz**
Application URL	NA

Outcome(s)
You should be able to run the CLI from a Terminal window.

Before you begin
OpenShift should be installed and running.

1. On the workstation VM, the binary is located in the **/home/student/DO290/installers**. To install the binary on Linux, run the following command:

```
[student@workstation ~]$ tar xzf /home/student/DO290/installers/oc-linux.tar.gz \
        -C /home/student/DO290/installers
[student@workstation ~]$ sudo cp /home/student/DO290/installers/oc /usr/local/bin/
```

2. To log into to a remote server, use the following command. Replace *X* with your student number:

```
[student@workstation ~]$ oc login https://master.podX.example.com:8443
```

The **oc login** command will warn that **The server uses a certificate signed by an unknown authority**. This is expected because the OSE installer created a self-signed SSL certificate for the OSE instance. So answer **y** to the **Use insecure connections? (y/n):** message.

Use the following credentials to access the master:

- Username: **student**
- Password: **redhat**

The expected result is:

```
Login successful.

You don't have any projects. You can try to create a new project, by running

    $ oc new-project <projectname>

Welcome to OpenShift! See 'oc help' to get started.
```

Note

The next time you log in to OSE there is no need to add the OSE master URL to the **oc login** command.

3. To log out, run the following command:

```
[student@workstation ~]$ oc logout
```

Managing an Application with the CLI

Objectives

After completing this section, students should be able to execute CLI commands to manage pods, services, routes, builds, and other OSE resources.

Describing resources and displaying status

After logging into the OpenShift CLI, there are several useful commands for viewing project status and switching between projects. The following table describes basic **oc** operations and their general syntax:

Basic CLI Operations

Operation	Syntax	Description
`types`	`oc types`	Output information about core OpenShift resource types.
`login`	`oc login <server-host-name>`	Log in to an OpenShift instance.
`logout`	`oc logout`	End the current session.
`new-project`	`oc new-project <project_name>`	Create a new project.
`new-app`	`oc new-app <application_name>`	Create a new application from the source code from a Git repository.
`new-app`	`oc new-app <application_name> -l name=<label>`	Create a new application from the source code in the current directory labelling as `<label>` all the resources for a simpler management.
`status`	`oc status`	Show an overview of the current project.
`project`	`oc project <project_name>`	Switch to a project with the name `<project_name>`. To view a list of all accessible projects, run `oc projects`.
`describe`	`oc describe <resource_type> <resource_name>`	Detail the information about an specific resource_name.

The OpenShift CLI tool provides interaction with the various resources that are managed by OpenShift. Many common **oc** operations are invoked using the following syntax:

```
$ oc <action> <resource_type> <resource_name_or_id>
```

- An `<action>` refers to an action to perform, such as a `get` or `describe`.

- The **<resource_type>** refers to the type of resource the action will be performed on. For example, a **service** or **node**.
- The **<resource_name_or_id>** refers to the name or ID of the specified **<resource_type>**.

Note

The **<resource_name_or_id>** is optional, depending on the action, and is only used to specify action on a specific resource. If an **<resource_name_or_id>** is not specified, the action will be performed on all resources that is within the scope of the user.

The **get** action returns a list of resources for the specified **resource type**. To get the list of all the resources from the current project:

```
$ oc get all
```

If the **<resource_name_or_id>** is included, then the list of results is filtered by that value. For example:

```
$ oc get services
```

To get the state about a certain resource, the **-o** parameter may be used.

```
$ oc get <resource_name_or_id> -o <yaml|json>
```

The output may be formatted using a YAML or JSON file.

To get information about a specific resource, the **describe** command returns information that is specific to the type of the resource described. With this action, the **<resource_name_or_id>** is required, as it refers to a specific resource instead of just an resource type. For example:

```
$ oc describe node <node_name_or_id>
```

Note

The **get** command is useful for getting general information about the resource. The **describe** command gives resource-specific information.

Starting and stopping resources

The following table describes the commands for starting resources:

Starting Resources

Command	Syntax	Description
create	**oc create -f <file>**	Depending on the contents of a JSON on YAML file, OpenShift will create and

Command	Syntax	Description
		start the defined resource if valid.

The following table describes the commands for stopping resources:

Stopping Resources

Command	Syntax	Description
`cancel-build`	`oc cancel-build <build-name>`	Cancel a build with the given build name. Optional flags `--dump-logs` prints out the build logs and `--restart` will create a new build with the same parameters.
`delete`	`oc delete -f <filename>`	Delete an resource using the type and ID specified in a JSON or YAML file.
`delete`	`oc delete <resource-type> <resource-id>`	Delete an resource of `<resource-type>` base on the resource's ID.
`delete`	`oc delete <resource-type> --all`	Delete all resources of a certain `<resource-type>`.
`delete`	`oc delete all -l name=<label>`	Delete all resources labeled as `<label>`.
`delete`	`oc delete all -l app=<appName>`	Delete all resources that was generated using the new-app command.

Note

The `oc delete all -l app=appName` was introduced in OSE 3.0.1.0 and it is a valuable way to delete only the application resources, not the services or secrets.

Viewing logs

Using the OpenShift CLI Tools, it is possible to view the logs for individual pods or containers. The format of the command is:

```
$ oc logs <pod_name> -c <container_name>
```

Note

If the pod only has one container, then the `<container_name>` is optional.

The following is an example of the `oc logs` usage:

```
$ oc logs -f docker-registry-1t3aw -c ruby-container
```

Note

The **-f** flag results in the logs being followed so that information continues to flow into the terminal.

Viewing project events

Viewing the OSE events associated with a project is an important step to understanding what an application is doing. In particular, if a build or a deployment of an application fails, there often are critical hints as to the root cause of the problem provided in the project events. It is a best practice to tail project events when an application is going through its build and deployment life cycles. Use the following command to view:

```
$ oc get events -w
```

Note

It is a best practice to **tail** the project events when the application is going through its build and deployment life cycles.

Executing builds

One of the fundamental capabilities of OpenShift is the ability to build applications into a container from a source. The following table describes the CLI operations for working with application builds.

Application Build CLI Operations

Operation	Syntax	Description
start-build	**oc start-build <buildConfig_name>**	Manually start the build process with the specified build configuration file.
start-build	**oc start-build --from-build=<build_name>**	Manually start the build process by specifying the name of a previous build as a starting point.
build-logs	**oc build-logs <build-name>**	Retrieve the build logs for the specified build.
new-build	**oc new-build**	Create a build configuration based on the source code in the current git repository.
cancel-build	**oc cancel-build**	Stop a build that is in progress.

Removing applications

In order to remove an individual application, it is necessary to delete all of the resources that are relevant to that particular application. The following resources must be deleted to fully remove an application:

- Routes.
- Services.
- Build configuration.
- Deployment configuration.
- Replication controller.

Note

Deleting a project will delete all resources related to the project. If there are multiple applications within the same project, resources for all applications will be deleted.

OSE resource life cycle

Kubernetes is responsible for ensuring that an application is correctly deployed at all times as per the number of replicas defined in its associated **replicationcontroller** resource.

New Linux containers to support an application can be spun up and/or down by defining the number of replicas using the **oc scale** command as follows:

```
$ oc scale --replicas=<replica count> rc <replication controller name>
```

Note

Setting the amount of replicas to zero is equivalent to a conventional **stop** command.

Editing resources

In addition to creating and deleting resources, it is also possible to modify already existing resources.

Using the **edit** operation opens an editor that allows modification to the configuration file of the resource. Once changes to the YAML or JSON file are saved, OpenShift will attempt to run **oc update** using the configuration file. If there is an error during the update process, the original configuration file will revert to its original state. OpenShift will save a copy of the modified configuration in a temporary folder that is accessible and the update can be rerun using this file.

The following table outlines usage of the **edit** operation:

Edit Operation

Syntax	Description
`oc edit <resource_type>/ <resource_type_name>`	Edit the desired resource type.
`OC_EDITOR="<text_editor>" oc edit <resource_type>/ <resource_type_name>`	Edit the desired resource type with a specified text editor.

Syntax	Description
`oc edit <resource_type>/ <resource_type_name> --output_version=<resource_type_version> -o <resource_type_format>`	Edit the desired resource in a specified format (eg: JSON).
oc exec	

An example of the command in use:

```
$ oc edit route/testRoute o json
```

It will open an editor instance for change purpose, just like the following:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
Output omitted
```

It is possible to run resource-specific commands best suited to particular tasks than the generic **`oc edit`**, such as **`oc expose`**, **`oc scale`**, and **`oc volume`**.

Executing commands

OpenShift offers the ability to execute commands or open a shell inside the pod using the **oc** client tool where all the input and output are redirected from the local developer's workstation to the OSE container. This often can be very helpful in troubleshooting a running application. Execute a command in the running container using the following **oc** utility syntax:

```
$ oc exec -p <pod_ID> -c <container_ID> <command>
```

Alternatively, the **`oc exec`** command also accepts additional arguments that enable the use of an interactive console. This is useful for more in-depth troubleshooting sessions.

For example, to launch a remote shell, execute:

```
$ oc exec <pod_ID> -i -t bash
```

To exit, just type in **`exit`** to be returned to the shell of the local environment.

Note

Using the **`oc exec`** command to access containers in a pod. Use with caution to avoid potentially corrupting the container (forcing a fresh new replacement container to be spun up).

Resource types alias

The **oc** command-line client tool also allows for listing of individual resources as per the following table:

OSE Resource	Full Command	Abbreviated Command
Builds	`oc get build`	`oc get build`
Deployment configs	`oc get deploymentconfig`	`oc get dc`
Image streams	`oc get imagestream`	`oc get is`
Pods	`oc get pod`	`oc get pod`
Replication controllers	`oc get replicationcontroller`	`oc get rc`
Routes	`oc get routes`	`oc get routes`
Services	`oc get services`	`oc get services`

As seen from this table, the command-line client tool also allows for a listing of certain resources using an abbreviated form.

Guided Exercise: Managing Applications with the CLI

In this lab, you will be introduced to additional management capabilities of OpenShift Enterprise tooling.

Resources	
Files	N/A
Application URL	`http://hello-openshift.manage-cli.cloudappsX.example.com`

Outcomes

You should be able to:

- View and manage resources via the OSE client.
- Execute commands in a container.
- Edit OSE resources using the OSE client.

Before you begin

- OpenShift Enterprise 3 installed and running.

1. Create a New Project
   1.1. Open a terminal window in the workstation VM.
   1.2. If needed, reauthenticate with the API of the OSE master node using the **`oc login`** command:

```
[student@workstation ~]$ oc login -u student -p redhat \
--server=https://master.podX.example.com:8443
```

Replace *X* with your student number.

Note

There is no need to provide the OSE master URL if this is not the first time you log in to OpenShift.

Note

Since OpenShift works with a self-signed certificate, the following message may show:

```
The server uses a certificate signed by an unknown authority. You can
 bypass the
certificate check, but any data you send to the server could be
 intercepted by others. Use
insecure connections? (y/n):
```

Please disregard this message and enter **y**.

1.3. Create a new project by running the following command:

```
[student@workstation ~]$ oc new-project manage-cli
```

2. Create a New Application

2.1. Create an instance of the Hello Openshift application provided by an image from the container registry.

```
[student@workstation ~]$ oc new-app \
      workstation.podX.example.com:5000/openshift/hello-openshift
```

2.2. Check the status of the deployment process, by running.

```
[student@workstation ~]$ watch oc status
```

Wait until a message like **#1 deployed X minutes ago** is listed.

2.3. Check which is the pod name by running:

```
[student@workstation ~]$ oc get pods
```

The output should be similar to:

```
NAME                      READY     STATUS    RESTARTS   AGE
hello-openshift-1-d62tz   1/1       Running   0          13m
```

The pod suffix normally varies.

2.4. Verify the log generated by the pod. To check what was the output from the log, run the following command:

```
[student@workstation ~]$ oc logs hello-openshift-1-d62tz
```

Change the suffix according to the output from the previous step. The following output will be presented:

```
serving on 8080
serving on 8888
```

2.5. Get information about the service hello-openshift by using the following command:

```
[student@workstation ~]$ oc get service hello-openshift
```

Take note of the service's name to expose it using a route.

Note

There is an IP address, however this is only visible from the master and the node hosts.

2.6. Create the route for the service to make it accessible using a browser and an domain. From the command line, run:

```
[student@workstation ~]$  oc expose svc hello-openshift
```

Check the URL exposed by the route.

```
[student@workstation ~]$ oc get route
NAME          HOST/PORT                                          PATH      SERVICE
   LABELS        TLS TERMINATION
hello-openshift   hello-openshift.manage-cli.cloudappsX.example.com
 hello-openshift   app=hello-openshift
```

The second column provides the address where the app is exposed.

2.7. Open a web browser and attempt to navigate to the URL of the existing **hello-openshift** route.

3. View Resources

3.1. List Resources

Run the following command with the **hello-openshift** application created to get information from all resources.

```
[student@workstation ~]$ oc get all
```

3.2. View Details and Associated Events

Identify the deployment config name from the **hello-openshift** application and get its details:

```
[student@worstation ~]$ oc describe dc hello-openshift
```

The **oc describe** command provides a concise view of the details of that resource.

Note

With some resources, the **oc describe** command will list the event history associated with that resource.

4. Events and Logging
   4.1. Project Events
   Open a new terminal window dedicated to continuous tailing of your project events.

   Execute:

```
[student@worstation ~]$ oc get events -w
```

   In your original terminal window, manipulate your application (i.e., create it, start-build, change replica count, etc).

   If failure occurs during the life cycle of your application deployed to OSE, refer back to the event log and hone in on messages that could assist in troubleshooting the root problem.

Note

The **oc get events -w** command will continue running until terminated using **Ctrl+C**

5. Remove Applications
   Execute the following command:

```
[student@workstation ~]$ oc delete project manage-cli
```

   This concludes the exercise.

Summary

In this chapter, you learned:

- How to download the OpenShift Enterprise 3 CLI tools binaries and how to install them for Linux, Windows and OS X.
- The structure and functions of commands of the CLI tools for managing resources, starting builds (**`oc start-build`**), and deleting unwanted resources (**`oc delete <resource> <resourceName>`**).
- Deleting an individual application in OpenShift Enterprise can be executed using the label facility **`oc delete all -l app=<appName>`**..
- Editing resources can be done using the CLI by calling **`oc edit`** to edit the configuration file, which is then automatically updated by OpenShift.

CHAPTER 5

DEPLOYING APPLICATIONS ON OPENSHIFT ENTERPRISE

Overview	
Goal	Define, build, and deploy an application on OpenShift Enterprise.
Objectives	• Determine the correct source images to build the bookstore application. • Define a Node.js application and deploy it in OSE. • Define, build, and deploy the MySQL pod for the bookstore application. • Define, build, and deploy the JBoss EAP pod for the bookstore web application.
Sections	• Selecting the Appropriate Source Images (and Guided Exercise) • Defining an Application (and Guided Exercise) • Creating the Bookstore Database (and Guided Exercise) • Deploying the Bookstore Web Application (and Guided Exercise)

Selecting the Appropriate Source Images

Objective

After completing this section, students should be able to determine the correct source images to build the bookstore application.

Available languages, platforms, and templates

A **template** is an OpenShift Enterprise (OSE) resource used to describe a set of resources and facilitate creating applications. The list of resources can include anything the user has permission to create within a project, such as services, build configurations, and deployment configurations.

OpenShift provides a number of out-of-the-box templates to make it easy to quickly get started creating a new application using different languages. Some templates are annotated as **Instant Apps** that provide complete sample applications a developer can reuse and modify to fit his or her needs.

Among the templates provided out of the box by OSE are the ones intended to create applications using popular programming languages and web frameworks such as:

- Rails (Ruby)
- Django (Python)
- Node.js
- CakePHP (PHP)
- Dancer (Perl)

Most templates rely on prebuilt **Source-to-Image** (S2I) **builder** images that include the programming language runtime and its dependencies. Most of those builder images can be used by themselves, without the corresponding template, to create simpler applications or to add more OSE resources later.

In addition to language-specific builder images, OpenShift also provides container images for middleware and database images. The following middleware images are provided by the OSE **xPaaS** and have to be used alongside the corresponding templates:

- JBoss EAP 6
- JBoss A-MQ
- Apache Tomcat 6, 7, and 8

The following database container images are available to be used either by themselves or as part of a template:

- MongoDB
- MySQL
- PostgreSQL

JBoss EAP 6 image

Red Hat JBoss Enterprise Application Platform (EAP) is available as a xPaaS container image that is designed for use with OpenShift. Developers can quickly build, scale, and test applications deployed across hybrid environments. The following parameters can be configured when building from a default EAP 6 template.

EAP 6 Environment Variables

Variable Name	Description
EAP_RELEASE	EAP Release Version
APPLICATION_NAME	The name for the application
APPLICATION_HOSTNAME	Custom hostname for service routes. Leave blank for a default hostname
GIT_URI	Git source URI for application
GIT_REF	Git branch/tag reference
GIT_CONTEXT_DIRECTORY	Path within Git project to build. Leave blank for root project directory
HORNETQ_QUEUES	Queue Names
HORNETQ_TOPICS	Topic names
HORNETQ_CLUSTER_PASSWORD	HornetQ cluster admin password. Generated if left blank
GITHUB_TRIGGER_SECRET	Github trigger secret. Generated if left blank
GENERIC_TRIGGER_SECRET	Generic build trigger secret. Generated if left blank

Utilizing OpenShift Enterprise's Source-to-Image process allows for OpenShift to pull code from a SCM repository and automatically detect the kind of runtime that source code needs. The process for the S2I template is configurable, but the default process for an EAP 6 Source-to-Image works as follows:

1. If a **`pom.xml`** file exists in the source repository, a Maven build is triggered. By default, the **`mvn package`** command is used. The results of the successful build are then copied into **`EAP_HOME/standalone/deployments`**.

2. If there are any deployables in the **`deployments`** source repository directory, they are also copied into the **`EAP_HOME/standalone/deployments`** directory.

3. All files in the **`configuration`** source repository directory are copied to **`EAP_HOME/standalone/configuration`**.

Note

To use a custom JBoss EAP configuration file, it should be named **`standalone-openshift.xml`**.

4. All of the files in the **`modules`** source repository directory are copied to **`EAP_HOME/modules`**.

MySQL 5.5 image

OpenShift provides container images for running MySQL version 5.5 in either RHEL 7 or CentOS 7. Those images can provide database services based on username, password, and database name settings provided via configuration. There are two built-in templates for creating a MySQL service: ephemeral and persistent. The ephemeral template is ideal for development or testing purposes because it uses ephemeral storage for the database content. This means that while this template will require less configuration, if the database pod is restarted for any reason, then all data will be lost. The persistent MySQL template uses a persistent volume store for the database data, which means the data will survive a pod restart. Using the persistent volume requires a persistent volume pool to be defined in the OpenShift deployment.

When building from a template, the following values can be specified. If no user or password is supplied, then one will be automatically generated. In addition, labels can be added at this point to help distinguish and organize different objects and pods. Labeling can be useful for distinguishing versions or environments.

MySQL Environment Variables

Variable Name	Description
MYSQL_USER	Specifies the user name for the database user that will be created
MYSQL_PASSWORD	Password for the MYSQL_USER
MYSQL_DATABASE	Name of the database to which MYSQL_USER has full rights

Guided Exercise: Exploring Available Quickstart Templates and Builders

In this lab, you will navigate the web console and identify the available instant app templates and builders from OpenShift Enterprise.

Resources	
Files	NA
Application URL	`https://master.podX.example.com:8443`

Outcome
You should be able to access and list the instant app templates and builders available at the OpenShift Enterprise web console.

Before you begin
OpenShift Enterprise (OSE) must be installed and started.

1. View All The Instant App Templates Available
   Open Firefox (**Applications** > **Internet** > **Firefox**) and access the web console using the following address:

   `https://master.podX.example.com:8443`.

   Log in as user **`student`** with password **`redhat`** that was created by this class custom OSE installation script.

Note

Accept any certificate issues from the site. Since this is a closed network, there is no need to worry about any certificate problems.

1.1. From the web welcome page, click the **New Project** button, and create a project with the following parameters:
- Name: **`exploring`**
- Display Name: **`exploring`**
- Description: Keep it blank.

Click the **Create** button. A project overview page will be presented.

1.2. In order to view which templates are available to create new applications, click the **Add to Project** button. There a few instant app templates available to create an application, among them:
- cakephp-example.
- cakephp-mysql-example.
- dancer-example.

- dancer-mysql-example.
- nodejs-example.
- nodejs-mongodb-example.

1.3. Scroll down to find the **Show All Templates** button. Click it to see other templates that are not annotated as instant apps.

1.4. Scroll up back to the text field, and type the following as the **Repository URL**, replacing *X* with your student number:

`http://workstation.podX.example.com/nodejstest.git`

And click the **Next** button.

1.5. The page shows some of the available S2I builders that OpenShift thinks may be adequate to build the application provided during the previous step.

Click the **Don't see the image you are looking for?** link to see other container images available.

Hover the mouse pointer over the exclamation mark icon to see a pop-up advising some container images on the list are NOT builder images and so are not able to build applications from source code.

1.6. DO NOT click any of the images. This exercise does not create any application, it is just intended to show templates, builders, and other images that came out of the box with OpenShift Enterprise.

Click the **OPENSHIFT ENTERPRISE** link on the top of the window to go to the project list page.

2. Clean Up
The web console from the first OSE 3 releases do not allow for removing projects. This has to be done from the command line.

2.1. Open a terminal on the workstation machine and log in as the **student** user:

```
[student@workstation ~]$ oc login -u student -p redhat
```

2.2. Remove the project by running the following command:

```
[student@workstation ~]$ oc delete project exploring
```

3. List All of the Builders
To find what builders are available to create an application, execute the following command:

```
[student@workstation ~]$ oc get is -n openshift
```

This will list all the predefined image streams. Some of them refer to builder images, other to database container images, and some to xPaaS middleware container images.

This concludes the exercise.

Defining an Application

Objective

After completing this section, students should be able to define a Node.js application and deploy it in OSE.

Projects in OpenShift

A **project** allows a community of users to organize and manage their content in isolation from other communities. Users must be given access to projects by administrators, or if allowed to create projects, automatically have access to their own projects.

Creating a project

Creating a new project via the CLI takes three parameters:

```
$ oc new-project <project_name>❶ \
    --description="<description>"❷ \
    --display-name="<display_name>"❸
```

❶ (Required) A unique identifier for the project that is most visible when using the CLI tools.

❷ (Optional) A detailed description of the project that is also visible in the web console.

❸ (Optional) The way the project is displayed in the web console. If no **displayName** is provided, it defaults to **name**.

Note

All of these features are also available in the OpenShift Enterprise web console by clicking **New Project**.

View all projects

To list all accessible projects for the actual user:

```
$ oc get projects
```

Note

All of the projects available are displayed under the **Projects** tab in the web console.

Switching among projects

After creating a project, future **oc** commands will continue to default to the new project. To switch to another project, run:

```
$ oc project <project_name>
```

Examine the source code

The application is a Hello, world! application that uses the usual configuration files for a Node.js application.

server.js

The **server.js** file is equivalent to a controller in Node.js. It controls all the requests made to the Node.js application and forwards the user to the correct address.

```
//  OpenShift sample Node application (1)
var express = require('express');
var fs      = require('fs');
var app     = express();
var eps     = require('ejs');

app.engine('html', require('ejs').renderFile);

var port = process.env.PORT || process.env.OPENSHIFT_NODEJS_PORT || 8080;
var ip   = process.env.IP   || process.env.OPENSHIFT_NODEJS_IP || '0.0.0.0';


app.get('/', function (req, res) {(2)
    res.render('index.html');
});

// error handling
app.use(function(err, req, res, next){
  console.error(err.stack);
  res.status(500).send('Something bad happened!');
});

app.listen(port, ip);
console.log('Server running on ' + ip + ':' + port);
```

(1) Declares all dependencies and variables needed by this application. The first four lines refers to the external libraries needed by this application.

(2) Identifies which is the address used by the customer to forward to the correct web page (**index.html**).

package.json

The **package.json** file is used to manage dependencies and configuration needed by Node.js applications.

```
{
  "name": "nodejs-test",
  "version": "0.0.1",
  "description": "Node.js test app for OpenShift 3",
  "main": "server.js",
  "dependencies": {(1)
      "express": "*",
      "ejs": "*"
  },
  "engine": {(2)
    "node": "*",
    "npm": "*"
  },
  "scripts": {(3)
    "start": "node server.js"
```

```
  },
  "author": "Jim Rigsbee <jrigsbee@redhat.com>",
  "license": ""
}
```

1. Identifies which dependencies and versions are needed by this application.
2. Lists the Node.js version that this app can run.
3. Identifies which controller will be used by this application.

Applications in OpenShift

Each project may have multiple applications that can be managed individually. In order to create a new application, the **oc new-app** command is provided.

```
$ oc new-app http://workstation.podX.example.com/nodejstest.git❶ \
    --context-dir=<dir>❷ \
    --name=nodejstest❸
```

1. (Required) The Git repository (either local or remote) can be passed as a parameter to the **oc create-app** command. When using a remote URL to point to a Git repository, adding **#[reference]** points to a specific subdirectory, such as a branch or version.
2. (Optional) The specific subdirectory within the repository that should be cloned.
3. (Optional) Application name.

Additionally, the **-e** tag allows for passing in environment-specific variables.

Based on the existence of a **Dockerfile** in the repository, the **oc new-app** command will utilize the file to execute a build. Otherwise, OpenShift will attempt to automatically build an image based on the language in the application's source.

Note

Internally, a project with a source code provided by a developer is built using the Source-to-Image (S2I) feature. S2I is responsible for automatically building and deploying a project using its source code. As part of the S2I functionality, a **BuildConfig** object, a pod, a **DeploymentConfig** object, and some services are created.

Languages Detected by new-app

Language	Files
Ruby	**Rakefile**, **Gemfile**, **config.ru**
Java EE	**pom.xml**
Node.js	**app.json**, **package.json**
PHP	**index.php**, **composer.json**
Python	**requirements.txt**, **config.py**
Perl	**index.pl**, **cpanfile**

Once the **new-app** command is executed, it will attempt to:

- *Build the source into a new application*: Internally, OpenShift creates a **`BuildConfig`** to build the application.
- *Create a new container image based on an **`ImageStream`** compatible with the source code*: Using the **`BuildConfig`**, OSE generates a new container image, based on **`ImageStream`**s dynamically generated during the creation process.
- *Construct a deployment configuration that deploys that new image*: A **`DeploymentConfig`** is generated without a persistent storage configured.
- *Deploy a pod*: A pod is a collection of containers and volumes that are bundled and scheduled together because they share a common resource, usually a file system or IP address. A pod is an OSE 3 resource inherited directly from the Kubernetes project.
- *Configure a service*: A set of services are configured to provide load-balanced access to the deployment that is running the image. Optionally, an OSE route can be created to expose the service to clients outside the OSE environment.

Creating, building, and deploying an application from source is simple in the OpenShift Enterprise web console.

To create an application using the web console:

1. Click **Create...** in the web console dashboard.
2. Enter the URL to the Git repository.
3. Click **next**.
4. On the next page, select the **Node.js** build image.
5. Uncheck **Create a route to the application.**
6. Click **Create** and wait for the build to start.

Creating routes

OpenShift exposes the application as a service and it can be accessed via an IP address and a port managed by Kubernetes. However, OpenShift may expose the application at a host name so that external clients can reach the service by its name. Creating routes can be done using the CLI by running the following command:

```
$ oc expose service <name> ❶ \
    --hostname=<URL> ❷
```

❶ The service name that will be exposed using a FQDN.

❷ The URL that will be used to expose the service.

Alternatively, a route can be defined by either YAML or JSON. For example:

```
apiVersion: v1
kind: Route
metadata:
  name: route-name
spec:
  host: www.cloudappsX.example.com
```

```
    to:
      kind: Service
      name: service-name
```

After creating the route YAML or JSON description file, use the following command to let OpenShift create the route:

```
$ oc create -f <routeFile> -n <projectName>
```

Demonstration: Creating a Node.js application

In this example, please read these steps while your instructor demonstrates how to create a Node.js application using the CLI.

Note

For an Internet connected environment, clone the Git repository, customize a builder script and commit it to Git. To achieve this goal, execute the following steps (they might not be needed if the **nodejstest** was previously cloned):

```
$ git clone http://workstation.podX.example.com/nodejstest.git
$ cd nodejstest
```

Open a text editor and change the **.sti/bin/assemble** file to remove the parameter **--registry** from the command **npm install**.

In order to make it available for OpenShift, add, commit and push the changes to the remote repository.

```
$ git add .
$ git commit -m "Internet based config"
$ git push
```

1. *Log into the web console*: From the workstation machine, open a web browser (**Applications** > **Internet** > **Firefox**) and open the **https://master.podX.example.com:8443** address. Log in using the following credentials:
   - Username: **student**
   - Password: **redhat**
2. *Create a project for the Node.js application*: Click the **New Project** button from the welcome page and use the following values for the project:
   - Name: **osev3-devops-course**
   - Description: **OSEv3 DevOps Course**
   - Description: <leave it blank>

   Click the **Create** button.
3. *Create an application*: Click the **Add to Project** button. In the **Create Using Your Code** text field add the following address: **http://workstation.podX.example.com/**

nodejstest.git, and click **Next**. Two builder images will be presented. Select the **nodejs:0.10** builder. Leave the default values from the wizard and click the **Create** button.

Note

Replace *X* in the Git URL by your student number.

4. *Monitor deployment of Node.js application*: The OSE web console allows viewing streamed events of a project. The project event streams can be monitored to see when OSE 3/ Kubernetes detects that it needs to start the important OSE resources such as the **BuildConfig** resource for this application. In the OSE **Project Summary** page, navigate to **Browse** > **Events** and wait until a slurry of new events related to the new application begin to stream. The first events in the OSE deployment workflow should be related to starting of the **BuildConfig** resource for the new application.

5. *View build*: The **nodejstest** application makes use of OSE 3's S2I workflow. Once the new events related to the **nodejstest** application are detected, navigate to **Browse** > **Builds**. A variety of important information regarding the **BuildConfig** resource for the **nodejstest** application will be listed, including the source repo cloning process.

   OSE will automatically trigger a build of the **BuildConfig** resource. Once completed, the details related to that build will be appended to the bottom of this same page.

6. *View pod*: OSE 3 will deploy the Node.js application into a Node.js container. To see details of this pod, open the browser, and navigate to **Browse** > **Pods**. The panel provides important information, such as the status of the pod and the node that the pod is currently running on.

7. *View service and route*: The application should be exposed as a service on port 8081 and the HTTP requests from external clients (i.e., such as from a browser) must be routed to that service. Subsequently, a OSE route will be necessary. For the Node.js application, the OSE web console has already created the necessary service and route resources. No further configuration is needed. To view the Node.js applications service and route, in the OSE project summary page, navigate to **Browse** > **Services**. Notice the internal IP address and port assigned to the application's service and the application route.

8. *Test Node.js web app*: The **nodejstest** application provides a simple HTTP response: **This is a test for NodeJS**.

   To view the Node.js application, click the link of the route specified in the **services** panel.

This concludes this demo.

Guided Exercise: Creating a Node.js Application

In this lab, you will deploy a simple Node.js-based web application to OpenShift Enterprise (OSE) 3 using OSE CLI tooling.

Resources	
Files	NA
Application URL	`http://workstation.podX.example.com/nodejstest.git` `http://nodejstest.cloudappsX.example.com`

Outcome(s)
You should be able to identify the concepts of **`BuildConfigs`**, **`pods`**, **`service`**, and **`routes`**.

Before you begin
OpenShift Enterprise 3 must be installed and started.

1. Log in as the student user
   Log in as the **`student`** user created by the custom OSE installation script:

```
[student@workstation ~]$ oc login -u student -p redhat
```

Note

If this is the first time you login to OSE, add the master URL to the command above: **`http://master.podX.example.com:8443`** replacing *X* by your student number.

2. Create a Project
   Run the following command to create a new project via CLI:

```
[student@workstation ~]$ oc new-project hello-openshift \
    --description="This is an example project to demonstrate OpenShift v3"
    --display-name="Hello OpenShift"
```

3. List The Projects
   Run the following command to check if the project was created:

```
[student@workstation ~]$ oc get projects
```

4. Check Out The Node.js Application
   From a Terminal window, run the following command to get the sample Node.js application available in a Git repository. Replace *X* by your student number:

```
[student@workstation ~]$ git clone http://workstation.podX.example.com/
nodejstest.git
```

Note

For an Internet connected environment, you will need to customize a builder script and commit it to Git. To achieve this goal, execute the following steps

- Open a text editor and change the **.sti/bin/assemble** file to remove the parameter **--registry** from the command **npm install**.
- In order to make it available for OpenShift, add, commit and push the changes to the remote repository.

```
$ git add .
$ git commit -m "Internet based config"
$ git push
```

5. Examine a Sample Node.js Application
   Within the sample application, there is a view (**view/index.html**), a Node.js dependency JSON file (**package.json**), and the main JavaScript file (**server.js**).

   The **package.json** file outlines metadata for a Node.js application, such as the application name, version number, description, and the main file for the program. In addition to storing metadata, the **package.json** file also manages dependencies for the application by listing every dependency in the **dependencies** section of the file.

   As described in the **package.json** file, the **server.js** file is where the application begins its execution. In this instance, the application is configured to render the **index.html** file.

6. Create, Build, And Deploy The Application
   Using the CLI, the following command creates the application from a Git repository. Replace *X* by your student number:

```
[student@workstation ~]$ oc new-app \
    http://workstation.podX.example.com/nodejstest.git
```

7. Wait For The Build To Finish And The Application Be Started
   Wait until the build starts and finishes, and the application pod is deployed, ready and running. Use the **oc get builds** and **oc get pods** commands to verify these conditions.

Note

The build may take a few moments to start, be patient.

8. Test The Application With Its Route
   An OpenShift **route** exposes a service at a host name so that external clients can reach the service with its name. Creating routes can be done in the CLI by using the following command. Replace *X* with your student number:

```
[student@workstation ~]$ oc expose service nodejstest \
    --hostname=nodejstest.cloudappsX.example.com
```

   Then open a web browser and visit the **--hostname** URL to see the application welcome page.

9. Clean Up
   From the command line, remove the project:

```
[student@workstation ~]$ oc delete project hello-openshift
```

Creating the Bookstore Database

Objectives

After completing this section, students should be able to define, build, and deploy the MySQL pod for the bookstore application.

Create the bookstore database

The back end of the bookstore application will be a MySQL database. The process for creating a MySQL pod as a back end to an application includes:

- Creating a new project.
- Setting up a persistent volume.
- Building the MySQL pod.
- Exposing the MySQL pod to application pods using a service.
- Exposing the MySQL server to remote administration through port forwarding.
- Defining the database schema.

Creating a new project

A new project can be created either in the OpenShift web console or using the OpenShift CLI. In the web console, click **Create...** next to **projects** and fill out the **Name**, **Display Name**, and **Description** for the bookstore project. Verify in the web console that the new project appears in the projects dropdown and that it is selected.

The command in the CLI should be executed on the master node and it should look like the following:

```
$ oc new-project <project_name> --description="<description>" \
    --display-name="<display_name>"
```

Verify, using the following command, that the project has been created:

```
$ oc get projects
```

In order to select the project, use the following command:

```
$ oc project <project_name>
```

Setting up a persistent volume

Due to the ephemeral nature of container storage, all stored data from the database will be lost. To avoid this, it is necessary to:

- Set up a persistent volume on the master machine, using an NFS share volume for the database.
- Create the persistent volume.
- Create a persistent volume claim.

The NFS share volume must be stored at the master machine since it will be shared among all the nodes. To set up the NFS share volume, some administrative tasks should be executed.

The process for creating a new NFS share is:

1. Create a new directory for the volume on the NFS server machine:

```
# mkdir /var/export/remembermysql
```

2. Give **nfsnobody** user and group permissions:

```
# chown nfsnobody:nfsnobody /var/export/remembermysql
# chmod 700 /var/export/remembermysql
```

3. Add the folder to **/etc/exports** with option **all_squash**, for example:

```
/var/export/remembermysql   *(rw,sync,all_squash)
```

4. Export the folder:

```
# exportfs -a
```

To create the persistent volume claim(PVC), the CLI must be used since there is no web console support. A JSON file must be created to set up the PVC. The following JSON code is an example of how to create it:

```
{
   "apiVersion": "v1",
   "kind": "PersistentVolumeClaim",
   "metadata": {
       "name": "mysql",
       "labels": {
           "application": "mysql"
        }
   },
   "spec": {
        "accessModes": [
            "ReadWriteOnce"
        ],
        "resources": {
           "requests": {
              "storage": "1Gi"
           }
        }
   }
}
```

1. Edit the following line:

```
"server" : "master.podX.example.com"
```

2. Create the persistent volume as an OSE **instance administrator** user from the resource definition file:

```
$ oc create -f \
  /root/DO290/labs/deploy-template/mysqlPV.json
```

Important

Only OSE administrators can create PVs. Persistent volumes are not tied to any project, but are available to be claimed by project.

To use a persistent volume (PV), a project needs a **persistent volume claim** (PVC or PVClaim). The MySQL database template already includes such a resource.

Once the project and persistent volume has been created, create a new MySQL 5.5 pod.

In the web console, use the following steps:

1. Click **Create...** within the recently created project.
2. Scroll to the bottom of the page and select **View More Templates...**.
3. Select the MySQL persistent template.
4. Click **Edit Parameters**.
5. Adjust the **MYSQL_USER** to **openshift**.
6. Change the **MYSQL_PASSWORD** to **password**.
7. Change the **MYSQL_DATABASE** to **bookstore**.

To create the MySQL 5.5 pod using the CLI:

1. Export the database template.

```
$ oc export -n openshift -o json \
  template mysql-persistent > mysql-persistent.json
```

2. Process the template to provide values for the template parameters.

```
$ oc process -f mysql-persistent.json -v \
    MYSQL_USER=openshift,MYSQL_PASSWORD=password,MYSQL_DATABASE=bookstore \
    > mysql-pod.json
```

3. Create the pod from the processed resource definition file.

```
$ oc create -f mysql-pod.json
```

Another alternative is using the **oc new-app --template** command.

Exposing the MySQL pod to application pods using a service

So other application pods can connect to the MySQL database service, an OSE service resource have to be created. Usually the web console or the **oc new-app** command will create the

service automatically, based on metadata inside the container image. It is also common for OSE templates to include one or more service resources.

If the service is not created automatically, the service resource can be created from either a JSON resource definition file or using the **oc expose** command.

A service resource creates environment variables for all pods created inside the same project. Those environment variables provide applications with IP addresses and TCP ports to connect to the database. Other information, such as login credentials, usually are provided as additional environment variables that are initialized from template parameters.

Exposing the MySQL server to remote administration through port forwarding
In order to expose the database server to the developer machine so a developer can perform DBA duties, a **port forwarding** tunnel needs to be established. The CLI can be used to forward one or more local ports to a pod. This allows for listening on a given or random port locally and having the data forwarded to and from given ports in the pod. For the MySQL pod, it listens on port 3306, so port forwarding needs to be set up from an available local port to the port the pod is listening to (3306). In the workstation, execute the following commands to create the port forwarding tunnel:

```
$ oc port-forward -p <mysql pod name> \
  <local unused port>:3306
```

Note

The port forwarding creates a tunnel and the process therefore needs to be left running while executing the next steps to create the database.

Create the MySQL database schema
Once the port forwarding tunnel is running, the database can be accessed from any client tool. For example, to use the standard MySQL command-line client, use options **-h** and **-P** to point it to the localhost and the local port allocated to the tunnel:

```
$ mysql -h127.0.0.1 -uopenshift -ppassword \
  -P13306 bookstore
```

Additionally, the MySQL database can be accessed using the database client available from the container image inside the pod:

```
$ oc exec -p mysql_pod_name -it -- \
    /bin/bash -c "mysql -h127.0.0.1 -uopenshift \
    -ppassword bookstore"
```

Guided Exercise: Creating the Bookstore Database

In this lab, you will create a database to allow the bookstore deployment.

Resources	
Files	**/home/student/DO290/labs/deploy-template/** (workstation VM)
	/root/DO290/labs/deploy-template/ (master VM)
Application URL	NA

Outcome
You should be able to deploy the MySQL database image from a template and remotely connect to the MySQL server.

OpenShift Enterprise 3 must be installed and started.

1. Create a Persistent Volume
   To create the persistent volume, the CLI must be used since there is no web console support. A JSON file must be created to set up the persistent volume.

   Use the provided script to SSH into the master VM and create the persistent volume (PV).

   ```
   [student@workstation ~]$ bash /home/student/DO290/labs/deploy-template/createpv.sh
   ```

Important

The script will work only if the root password was NOT changed on the master VM.

   Alternatively, perform the follow steps to create the PV without using the provided script. They are done on the master VM as it is an administration task, not a developer task.

   SSH to master using login **root** and password **redhat**. Replace *X* with your student number.

   ```
   [student@workstation ~]$ ssh root@master.podX.example.com
   ```

   Edit the **mysqlPV.json** file located at **/root/DO290/labs/deploy-template/mysqlPV.json**, updating *X* from the following line to use the station number:

   ```
   "server" : "master.podX.example.com"
   ```

   Open the **mysqlPV.json** file and evaluate its contents. The PV will be created pointing to a NFS share called **/var/export/bookstorevol** from a machine called **master.podX.example.com**. It will allocate 1GiB for usage. Any PVC removed will be recycled by the PV.

Create the persistent volume using the resource definition file. The **root** user on the OSE instance master is automatically logged in as an OSE instance administrator.

```
[root@master ~]# oc create -f /root/D0290/labs/deploy-template/mysqlPV.json
```

Log out of the master VM machine and return to the workstation VM.

```
[root@master ~]# logout
```

2. Create a New Project
A new project will be created to contain the MySQL pod and related resources.

This and the following steps are done on the workstation VM.

2.1. Log in as the **student** user created by the custom OSE installation script:

```
[student@workstation ~]$ oc login -u student -p redhat
```

2.2. Create the project to contain the MySQL pod and related resources.

```
[student@workstation ~]$ oc new-project bookstoredb \
    --description="Bookstore POD" \
    --display-name="bookstore"
```

3. Create a Database Pod Using Persistent Storage
Create the **PersistentVolumeClaim** (PVC) and the database pod using the CLI.

This and the following steps are done on the workstation VM.

3.1. Export the OSE standard **mysql-persistent** template to a JSON file:

```
[student@workstation ~]$ oc export -n openshift -o json \
    template mysql-persistent > mysql-persistent.json
```

3.2. Process the exported template to provide values for the template parameters:

```
[student@workstation ~]$ oc process -f mysql-persistent.json -v \
MYSQL_USER=openshift,MYSQL_PASSWORD=password,MYSQL_DATABASE=bookstore \
> mysql-pod.json
```

3.3. Create the database server pod using the processed resource definition file:

```
[student@workstation ~]$ oc create -f mysql-pod.json
```

Insight

The standard **mysql-persistent** template already includes a PVC resource to use the persistent volume created earlier.

3.4. Confirm a PVC was created and bound to the PV created earlier.

```
[student@workstation ~]$ oc get pvc
```

The expected output is:

```
NAME      LABELS                                    STATUS    VOLUME
mysql     map[template:mysql-persistent-template]   Bound     mysqldb-volume
```

4. Create a Tunnel to Expose the MySQL Server
Create a tunnel to the database server to allow for DBA operations.

4.1. Find the name of the MySQL pod:

```
[student@workstation ~]$ oc get pods
```

4.2. Create the port-forwarding tunnel:

```
[student@workstation ~]$ oc port-forward -p mysql_pod_name 13306:3306
```

The **oc port-forward** command does NOT return to the Bash prompt. It has to be running to keep the tunnel open.

Important

If for any reason the **oc port-forward** command returns to the prompt, run it again.

Open another Terminal window to continue with the following steps.

5. Create the MySQL Database Schema
Connect to the MySQL server and the **bookstore** database to create tables and test data using the provided SQL script.

5.1. Execute the **bookstore.sql** script.

```
[student@workstation ~]$ mysql -h127.0.0.1 -uopenshift \
    -ppassword -P13306 bookstore < \
    /home/student/DO290/labs/deploy-template/bookstore.sql
```

5.2. Connect to the MySQL pod and the MySQL server to verify the database:

```
[student@workstation ~]$ mysql -h127.0.0.1 -uopenshift \
    -ppassword -P13306 bookstore
```

5.3. Run a SQL query on the **Promotion** table:

```
mysql [bookstore]> select count(*) from Promotion;
```

The expected output is:

```
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.01 sec)
```

5.4. Exit the MySQL client:

```
mysql> exit
```

5.5. Return to the terminal where you left the **oc port-forward** command running and kill it using **Ctrl+C**.

6. Clean up
This database pod will not be used by the following labs, so it can be removed to keep things tidy.

There are steps to perform from the workstation VM and steps to perform from the master VM.

6.1. Delete the **bookstoredb** project. That removes all resources inside it.

```
[student@workstation ~]$ oc delete project bookstoredb
```

6.2. Use the provided Bash script to delete the PV on the master VM:

```
[student@workstation ~]$ bash /home/student/DO290/labs/deploy-template/
deletepv.sh
```

Alternatively, open a SSH session to the master VM and delete the PV. Replace *X* with the student's number:

```
[student@workstation ~]$ ssh root@master.podX.example.com
[root@master ~]# oc delete pv mysqldb-volume
[root@master ~]# logout
```

This concludes the exercise.

Deploying the Bookstore Web Application

Objectives

After completing this section, students should be able to define, build, and deploy the JBoss EAP pod for the bookstore web application.

Create the pod, service, and route

Running an application from an OpenShift instance is different from a normal approach since the developer will not need to create a WAR or EAR package. The developer only will need to commit changes to a Git repository.

On the other hand, an administrator must configure a project and an application within OpenShift to clone or pull, build, and deploy the source code from the Git repository to a JBoss EAP instance. Since there are multiple artifacts involved, such as a database and an application server, they can be cumbersome, and it is instead preferred to begin using a predefined template that includes the pod structures that make configuration much simpler. To simplify the configuration of all these elements into a single application, OSE provides templates that allows an administrator to deploy applications passing parameters for the environment.

The template works in a similar fashion as a single-tiered application. In order to deploy the bookstore application, a MySQL back end and an EAP 6 application server are both required. The EAP 6 application will serve the files and maintain the configuration of the application, and the MySQL image will hold all of the data for the application. It is possible to individually create a separate EAP pod and a separate MySQL pod and configure them to interact with each other.

To create a Java EE application with external database support, a custom template will be used. The custom template is based on the **`eap6-sti`**; however it does not have a persistent storage. In order to make it work, some tweaks were needed.

Note

Creating the necessary pod to host the MySQL and EAP 6 images can be accomplished utilizing the **eap6-mysql-persistent-sti** template. This template includes both the MySQL 5.5 image and the JBoss EAP 6.4 image.

Demonstration: Deploying an application using templates

In this example, please read these steps while your instructor demonstrates how to deploy the bookstore application using the CLI.

1. *Configure the access to the remote server*: On the workstation VM, execute the following to authenticate the OSE 3 client with the remote OSE 3 master. Replace the *X* with the student's number:

```
[student@workstation ~]$ oc login -u student -p redhat \
  --server=https://master.podX.example.com:8443
```

Once successfully authenticated, your OSE 3 client certificates will be contained in a file at **~/.kube/config**. As valid and current client certificates have been created for the authenticated user, the OSE 3 client can use these certificates when communicating with the remote OSE 3 master.

Note

Periodically, the client may need to reauthenticate with the OSE 3 master. If so, rerun the **oc login** command to regenerate fresh client certificates.

2. *Create project*: Run the following command from the workstation:

```
[student@workstation ~]$ oc new-project bookstore-demo
```

To view details of the project, execute:

```
[student@workstation ~]$ oc project
```

This command will return the name of the project.

Using the project name, view details of the project by executing the following:

```
[student@workstation ~]$ oc describe project bookstore-demo
```

3. *Evaluate the contents from the **database.sh** file*: Open a text editor and evaluate the contents from the **/home/student/DO290/labs/bookstore-demo/database.sh** file.

```
oc new-app --template=mysql-ephemeral❶ -p \
DATABASE_SERVICE_NAME=bookstoredb,\❷
MYSQL_USER=openshift,\❸
MYSQL_PASSWORD=password,\❹
MYSQL_DATABASE=bookstore \❺
--name=bookstoredb
```

❶ OSE will use the template called mysql-ephemeral to build a MySQL pod.
❷ The database name will be bookstoredb
❸ The login to access the MySQL database.
❹ The password to access the MySQL database instance.

4. *Create ephemeral database pod*: use the provided script to create a MySQL database pod using ephemeral storage so the bookstore application can be quickly tested:

```
[student@workstation ~]$ bash /home/student/DO290/labs/bookstore-demo/database.sh
```

In a few moments the oc get pods command will show the is a database pod running with name **bookstoredb-1-****.

Insight

The database template parameter **DATABASE_SERVICE_NAME** value has to match the EAP template parameter **DB_APPLICATION_UPPER_NAME** from the EAP template in the next steps.

5. *Generate OSE 3-specific objects from the **bookstore** application*: Review the contents of the **eap6-training-sti.json** template file:

```
[student@workstation ~]$ less /home/student/DO290/labs/bookstore-demo/eap6-training-
sti.json
```

Insight

The **eap6-training-sti.json** template is based on the **eap6-basic-sti** standard OSE template, adding parameters related to the Java EE datasource and the MySQL database.

6. Using the **eap6-training-sti.json** template resource definition file, create a list of OSE 3 bookstore resources (in JSON format) and save the output to a new resource definition the file with the contents from the template filled. The following listing is just an example do not execute it:

```
[student@workstation ~]$ oc process -f /home/student/DO290/labs/bookstore-demo/eap6-
training-sti.json -v \
APPLICATION_NAME=bookstore,APPLICATION_UPPER_NAME=BOOKSTORE(1),\
APPLICATION_HOSTNAME=bookstore.cloudappsX.example.com(2),\
GIT_URI=http://workstation.podX.example.com/bookstore.git,\
DB_JNDI=java:/jboss/datasources/mysql,\
DB_USERNAME=openshift(3),\
DB_PASSWORD=password(4),\
DB_DATABASE=bookstore(5),\
DB_APPLICATION_UPPER_NAME=BOOKSTOREDB(6) \
> DO290/labs/bookstore-demo/bookstore.json
```

(1) The **APPLICATION_UPPER_NAME** should be identical to the **APPLICATION_NAME**, but in uppercase, since it will be used internally by the build process to refer to all the resources used by this project.
(3) The **DB_USERNAME** must be identical to the one created from the previous demo.
(4) The **DB_PASSWORD** must be identical to the one created from the previous demo.
(5) The **DB_DATABASE** must be identical to the one created from the previous demo.
(6) The **DB_APPLICATION_UPPER_NAME** should be identical to the name of the service created previously.
(2) The route that will be used by this application.

This big command is on file process.sh in this demo lab folder. Edit it to replace X by your student number on the **GIT_URI** and **APPLICATION_HOSTNAME** template parameters. Then execute the script as follows:

```
[student@workstation ~]$ bash /home/student/DO290/labs/bookstore-demo/process.sh
```

7. *Create the resources from the template*: Those resources can now be created in OSE 3 by running:

```
[student@workstation ~]$ oc create -f \
    /home/student/DO290/labs/bookstore-demo/bookstore.json
```

OSE 3/Kubernetes will take a few moments to automatically trigger execution of the **bookstore's BuildConfig**.

7.1. Tail the project event log:

```
[student@workstation ~]$ oc get events
```

The OSE 3 Kubernetes scheduler will generate a variety of events that will be listed in the event log. Exit the log view by hitting **Ctrl+C**.

7.2. Observe the existence of a S2I **build** resource:

```
[student@workstation ~]$ oc get builds
```

A build entitled **bookstore-1** will be listed.

7.3. View the build logs associated with **bookstore-1** build object:

```
[student@workstation ~]$ oc build-logs bookstore-1 -w
```

7.4. The bookstore web application uses Maven as its build tool.

As part of the build process, Maven will download all the dependencies from the project. This should be evident while tailing the **bookstore-1** build log. After several minutes, the last dependency will have been downloaded, the **bookstore** application will have been built and pushed to the internal OSE container registry, and log messages as follows will appear in the build log:

```
...
sti.go:388] Copying all WAR and EAR artifacts from /home/jboss/source/
deployments directory into /opt/eap/standalone/deployments for later
 deployment...
sti.go:388] '/home/jboss/source/deployments/ROOT.war' -> '/opt/eap/standalone/
deployments/ROOT.war'
docker.go:430] Container exited
docker.go:436] Invoking postExecution function
...
sti.go:246] Successfully built 172.30.70.145:5000/devops/bookstore
cleanup.go:23] Removing temporary directory /tmp/sti307599709
sti.go:131] Using serviceaccount user for Docker authentication
```

```
sti.go:131] Using provided push secret for pushing 172.30.70.145:5000/devops/
bookstore image
sti.go:134] Pushing 172.30.70.145:5000/devops/bookstore image ...
sti.go:138] Successfully pushed 172.30.70.145:5000/devops/bookstore
```

Finally, the **bookstore-1** build will have completed:

```
[student@workstation ~]$ oc get builds
NAME          TYPE      STATUS     POD
bookstore-1   Source    Complete   bookstore-1-build
```

Note

The **ReplicationConfig** object of the **bookstore** application by default sets a **replica** = 1. Because of this, OSE 3/Kubernetes automatically creates a **bookstore** container.

8. *Monitor the deployment process*: Immediately after having finished building the **bookstore** application, the S2I workflow will add the built web artifact into a thin container image layered onto the JBoss EAP image. A container of this JBoss EAP image will subsequently be started.

   Determine the name of the pod that JBoss EAP container resides in:

   ```
   [student@workstation ~]$ oc get pods
   ```

   Identify the pod with a name of **bookstore-1-**** and tail the JBoss server log:

   ```
   [student@workstation ~]$ oc logs -f <pod_name>
   ```

9. *Test the application*: Open a browser and navigate to the **bookstore** application using the route. To get the route to the bookstore application, execute:

   ```
   [student@workstation ~]$ oc describe route bookstore-http-route
   ```

10. *Clean up*: From the workstation, delete the project:

    ```
    [student@workstation ~]$ oc delete project bookstore-demo
    ```

Guided Exercise: Deploying the Bookstore Application

In this lab, you will deploy the bookstore application using the S2I workflow.

Resources	
Files	**/home/student/DO290/labs/bookstore-template (workstation VM)**
	/root/DO290/labs/bookstore-template (master VM)
Application URL	**http://bookstore-template.cloudappsX.example.com**

Outcome
You should be able to create and deploy a Java EE application to OpenShift Enterprise using the S2I process.

Before you begin
OpenShift Enterprise 3 must be installed and started.

1. Create a Persistent Volume
   To create the persistent volume, the CLI must be used since there is no web console support. A JSON file must be created to set up the persistent volume.

   This should be executed on the master VM as root, since it is an administration task, not a developer task.

Important

Even if the NFS share directory name and persistent volume name are the same as in the previous lab, the persistent volume cannot be reused here. The PV has to be deleted, using the previous lab cleanup instructions, and recreated using these lab instructions.

Use the provided script to SSH into the master VM and create the persistent volume (PV).

```
[student@workstation ~]$ bash /home/student/DO290/labs/bookstore-template/
createpv.sh
```

Important

The script will work only if the root password was NOT changed on the master VM.

Alternatively, perform the following steps to create the PV without using the provided script.

Edit the **mysqlPV.json** file located at **/root/DO290/labs/bookstore-template/mysqlPV.json**, updating *X* from the following line to use the station number:

```
"server" : "master.podX.example.com"
```

Create the persistent volume using the resource definition file. The **root** user on the OSE instance master is automatically logged in as an OSE instance administrator.

```
[root@master ~]# oc create -f \
    /root/D0290/labs/bookstore-template/mysqlPV.json
```

1.1. Log out of the master VM machine and return to the workstation VM.

```
[root@master ~]# logout
```

2. Create a New Project
A new project will be created to contain the bookstore application pod and related resources.

From the workstation VM, execute the following commands.

2.1. Log in as the **student** user created by the custom OSE installation script:

```
[student@workstation ~]$ oc login -u student -p redhat
```

2.2. Create the project to contain the MySQL pod and related resources.

```
[student@workstation ~]$ oc new-project bookstore
```

3. Use The Standard OSE Template To Create The Application And Database Pods
Create the application pod, database pod, and related resources using the standard **eap6-mysql-persistent** template.

From the workstation VM, execute the following commands.

3.1. Export the standard **eap6-mysql-persistent** template from the **openshift** project to a local JSON file.

```
[student@workstation ~]$ oc export -n openshift -o json \
    template eap6-mysql-persistent-sti > eap6-mysql-persistent-sti.json
```

The template includes resource definitions for both the application and the database pods, alongside required services, a route, and a persistent volume claim.

3.2. Process the template to provide values for the template parameters. Replace the two occurrences of *X* with your student number.

```
[student@workstation ~]$ oc process -f eap6-mysql-persistent-sti.json -v \
APPLICATION_NAME=bookstore,\
APPLICATION_HOSTNAME=bookstore-template.cloudappsX.example.com,\
GIT_URI=http://workstation.podX.example.com/bookstore.git,\
DB_JNDI=java:/jboss/datasources/mysql,\
DB_USERNAME=openshift,\
DB_PASSWORD=password,\
DB_DATABASE=bookstore,\
```

```
EAP_HTTPS_SECRET=eap-app-secret-member,\
EAP_HTTPS_KEYSTORE=keystore.jks,\
EAP_HTTPS_NAME=jboss,\
EAP_HTTPS_PASSWORD=mykeystorepass \
> bookstore.json
```

The two occurrences of *X* are in the parameters **APPLICATION_HOSTNAME**, which is the application route FQDN, and **GIT_URI**, which refers to the Git repository containing the bookstore application source code.

Important

Do not add extra white space to the previous command. In particular, there should be no spaces before and after the commas (,). The command is available on the **process.sh** Bash script in this exercise folder, which can be edited to replace the two occurrences of *X* with your student number, and then executed to prevent typing mistakes.

3.3. The **eap6-mysql-persistent** template needs a secret to be created in the project to provide SSL certificates for **https:** access. Create the secret using the provided JSON file.

```
[student@workstation ~]$ oc create -f \
/home/student/DO290/labs/bookstore-template/secret.json
```

3.4. The bookstore application and database resources can now be created in OSE 3 by running:

```
[student@workstation ~]$ oc create -f bookstore.json
```

3.5. In a few moments, a build should start. Wait for the build to finish and both the application and the database pod are running and ready. If building and deploying the bookstore application and/or the MySQL database fails, perform the cleanup steps and start over.

To monitor the output run the following command:

```
[student@workstation ~]$ watch "oc get builds; oc get pods"
```

4. Test the Application
After the pods have finished starting up and the services are deployed, the bookstore application can be accessed using the URL **http://bookstore-template.cloudappsX.example.com**, using a web browser from the workstation VM. Replace *X* by your student number.

Insight

You may have noticed we never initialized the database during this exercise. The bookstore application detects an empty database and initializes it with the application schema and sample data.

5. Clean Up
As this project and its resources will not be reused later during this course, remove everything to keep things tidy.

There are steps to perform from the workstation VM and steps to perform from the master VM.

5.1. Delete the **bookstore** project. That removes all resources inside it.

```
[student@workstation ~]$ oc delete project bookstore
```

5.2. Open a SSH session to the master VM and delete the PV, using the provided Bash script.

```
[student@workstation ~]$ bash /home/student/DO290/labs/bookstore-template/
deletepv.sh
```

Alternatively, SSH to the master VM and delete the PV manually. Change the *X* with the student's number.

```
[student@workstation ~]$ ssh root@master.podX.example.com
[root@master ~]# oc delete pv mysqldb-volume
[root@master ~]# logout
```

This concludes the exercise.

Summary

In this chapter, you learned:

- How to create a project using the OpenShift Enterprise command-line interface (**`oc new-project`**) and the web console.
- The practicality of using templates and how to select the appropriate template and configure it for a particular project(**`oc export -n openshift -o json template <templateName>`**).
- Deploying an application using OpenShift Enterprise can be accomplished from simply supplying a link to the source repository (**`oc new-app <GitRepository>`**).
- The steps required to build a multitiered application using the OpenShift Enterprise web console.