



**RED HAT®
TRAINING**



Comprehensive, hands-on training that solves real world problems

Red Hat OpenShift Administration I

Student Workbook

RED HAT OPENSHIFT ADMINISTRATION I

OCP 3.5 DO280
Red Hat OpenShift Administration I
Edition 1 20170829 20170829

Authors: Ravishankar Srinivasan, Fernando Lozano, Ricardo Jun Taniguchi,
Richard Allred, Victor Costea, Razique Mahroua
Editor: David O'Brien, Seth Kenlon

Copyright © 2017 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are Copyright © 2017 Red Hat, Inc.

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

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

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

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

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

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

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

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Jim Rigsbee, George Hacker, Rob Locke

Document Conventions	vii
Notes and Warnings	vii
Introduction	ix
Red Hat OpenShift Administration I	ix
Orientation to the Classroom Environment	x
Internationalization	xii
1. Introducing Red Hat OpenShift Container Platform	1
Describing OpenShift Container Platform Features	2
Quiz: OpenShift Container Platform Features	5
Describing the OpenShift Container Platform Architecture	9
Quiz: OpenShift Container Platform Architecture	16
Summary	20
2. Installing OpenShift Container Platform	21
Preparing Servers for Installation	22
Guided Exercise: Preparing for Installation	28
Running the Installer	32
Guided Exercise: Running the Installer	37
Executing Postinstallation Tasks	42
Guided Exercise: Completing Postinstallation Tasks	47
Summary	54
3. Describing and Exploring OpenShift Networking Concepts	55
Describing OpenShift's Implementation of software-defined networking	56
Guided Exercise: Exploring Software Defined Networking	62
Creating Routes	68
Guided Exercise: Creating a Route	74
Lab: Exploring OpenShift Networking Concepts	79
Summary	86
4. Executing Commands	89
Configuring Resources with the CLI	90
Guided Exercise: Managing an OpenShift Instance Using oc	98
Executing Troubleshooting Commands	105
Guided Exercise: Troubleshooting Common Problems	111
Lab: Executing Commands	117
Summary	124
5. Controlling Access to OpenShift Resources	125
Securing Access to OpenShift Resources	126
Guided Exercise: Managing Projects and Accounts	132
Managing Sensitive Information with Secrets	140
Guided Exercise: Protecting a Database Password	144
Managing Security Policies	148
Quiz: Managing Security Policies	153
Lab: Controlling Access to OpenShift Resources	155
Summary	167
6. Allocating Persistent Storage	169
Provisioning Persistent Storage	170
Guided Exercise: Implementing Persistent Database Storage	176
Configuring the OpenShift Internal Registry for Persistence	184
Guided Exercise: Creating a Persistent Registry	187

Lab: Allocating Persistent Storage	192
Summary	202
7. Managing Application Deployments	203
Scaling an Application	204
Guided Exercise: Scaling An Application	208
Controlling Pod Scheduling	215
Guided Exercise: Controlling Pod Scheduling	220
Managing Images, Image Streams, and Templates	225
Guided Exercise: Managing Image Streams	231
Lab: Managing Application Deployments	237
Summary	245
8. Installing and Configuring the Metrics Subsystem	247
Describing the Architecture of the Metrics Subsystem	248
Quiz: The Metrics Subsystem	253
Installing the Metrics Subsystem	257
Guided Exercise: Installing the Metrics Subsystem	262
Summary	271
9. Managing and Monitoring OpenShift Container Platform	273
Limiting Resource Usage	274
Guided Exercise: Limiting Resource Usage	281
Upgrading OpenShift Container Platform	290
Quiz: Upgrading OpenShift	303
Monitoring Applications with Probes	305
Guided Exercise: Monitoring Applications with Probes	310
Monitoring Resources with the Web Console	316
Guided Exercise: Exploring Metrics with the Web Console	322
Lab: Managing and Monitoring OpenShift	330
Summary	339
10. Comprehensive Review: Red Hat OpenShift Administration I	341
Comprehensive Review	342
Lab: Installing OpenShift	345
Lab: Deploy an Application	366

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.



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.



References

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

Introduction

Red Hat OpenShift Administration I

DO280: Red Hat OpenShift Administration I is a hands-on, lab-based course that teaches system administrators how to install, configure, and manage Red Hat OpenShift Container Platform clusters. OpenShift is a containerized application platform that allows enterprises to manage container deployments and scale their applications using Kubernetes. OpenShift provides predefined application environments and builds upon Kubernetes to provide support for DevOps principles such as reduced time to market, infrastructure as code, continuous integration (CI), and continuous delivery (CD).

Students will learn how to install, configure, manage, and monitor an OpenShift cluster. Through hands-on labs, students will test the OpenShift cluster by deploying different types of applications.

Objectives

- Install, configure, monitor, and manage an OpenShift cluster.
- Install, configure, and manage persistent storage for an OpenShift cluster.
- Deploy applications on an OpenShift cluster using Source-to-Image (S2I) builds.

Audience

- System administrators
- Architects and developers who want to install and configure OpenShift

Prerequisites

Students should meet the following prerequisites:

- RHCSA certification or equivalent Linux administration experience is required.
- *Introduction to Containers, Kubernetes, and Red Hat OpenShift (DO180)* is highly recommended, or equivalent experience with containers, Kubernetes, and OpenShift basics.

Orientation to the Classroom Environment

In this course, the main computer system used for hands-on learning activities is **workstation**. This is a virtual machine (VM), which has the host name **workstation.lab.example.com**. 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.

Apart from being the main student machine, this VM will also host a number of auxiliary services that are required for the classroom environment:

- A private docker registry containing the images needed for the course.
- A Git server that stores the source code for the applications developed during the course.
- A DNS server that serves the sub-domain **.cloudapps.lab.example.com*.

Additionally, three more virtual machines which will be referred to as **master** (**master.lab.example.com**), **node1** (**node1.lab.example.com**), and **node2** (**node2.lab.example.com**) are provided to each student. These machines serve as the OpenShift *master* and *nodes* respectively, and each has a root user account, *root*, with the password *redhat*. All four of these systems are in the **lab.example.com** DNS domain.

The following table lists the virtual machines that are available in the classroom environment:

Classroom Machines

Machine name	IP addresses	Role
content.example.com , materials.example.com , classroom.example.com ,	172.25.254.254, 172.25.252.254, 172.25.253.254	Classroom utility server
workstation.lab.example.com , workstationX.example.com	172.25.250.254, 172.25.252.X	Student graphical workstation
master.lab.example.com	172.25.250.10	OpenShift Container Platform master
node1.lab.example.com	172.25.250.11	OpenShift Container Platform node
node2.lab.example.com	172.25.250.12	OpenShift Container Platform node

One additional function of **workstation** is that it acts as a router between the network that connects the student machines and the classroom network. If **workstation** is down, other student machines will only be able to access systems on the student network.

There are several systems in the classroom that provide supporting services. Two servers, **content.example.com** and **materials.example.com** are sources for software and lab materials used in hands-on activities. Information on how to use these servers will be provided in the instructions for those activities.

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, resetting the disk to its initial state. Caution: Any work generated on the disk will be lost.
Refresh	Refresh the page will re-probe the machine state.
Increase Timer	Adds 15 minutes to the timer for each click.

The Station Timer

Your Red Hat Online Learning enrollment entitles you to a certain amount of computer time. In order to help you conserve your time, the machines have an associated timer, which is initialized to 60 minutes when your machine is started.

The timer operates as a "dead man's switch," which decrements as your machine is running. If the timer is winding down to 0, you may choose to increase the timer.

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

INTRODUCING RED HAT OPENSHIFT CONTAINER PLATFORM

Overview	
Goal	List the features and describe the architecture of the OpenShift Container Platform.
Objectives	<ul style="list-style-type: none">• Describe the typical use of the product and list its features.• Describe the architecture of OpenShift.
Sections	<ul style="list-style-type: none">• Describing OpenShift Container Platform Features (and Quiz)• Describing the OpenShift Container Platform Architecture (and Quiz)
Lab	None

Describing OpenShift Container Platform Features

Objectives

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

Features of Red Hat OpenShift Container Platform

Red Hat OpenShift Container Platform (OpenShift) is a container application platform that provides developers and IT organizations with a cloud application platform for deploying new applications on secure, scalable resources with minimal configuration and management overhead.

Built on Red Hat Enterprise Linux, Docker, and Kubernetes, OpenShift provides a secure and scalable multitenant operating system for today's enterprise-class applications, while providing integrated application runtimes and libraries. OpenShift brings a robust, flexible, and scalable container platform to customer data centers, enabling organizations to implement a platform that meets security, privacy, compliance, and governance requirements.

Customers who prefer not to manage their own OpenShift clusters can use Red Hat OpenShift Online, a public cloud platform provided by Red Hat. Both OpenShift Container Platform 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 Docker and Kubernetes.



References

More information about OpenShift upstream projects can be found at:

OpenShift product family:

 | <http://www.openshift.com>

Kubernetes:

 | <http://kubernetes.io>

Docker:

 | <http://docker.com>

Applications run as *containers*, which are isolated partitions inside a single operating system. Containers provide many of the same benefits as virtual machines, such as security, storage, and network isolation, while requiring far fewer hardware resources and being quicker to launch and terminate. The use of containers by OpenShift helps with the efficiency, elasticity, and portability of the platform itself as well as its hosted applications.

The main features of OpenShift are listed below:

- **Self-service platform:** OpenShift allows developers to create applications from templates or from their own source code management repositories using Source-to-Image (S2I). System administrators can define resource quotas and limits for users and projects to control the use of system resources.

- **Multi-language support:** OpenShift supports Java, Node.js, PHP, Perl, and Ruby directly from Red Hat, and many others from partners 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 middleware products such as Apache httpd, Apache Tomcat, JBoss EAP, ActiveMQ, and Fuse running natively on OpenShift.
- **Automation:** OpenShift provides application life-cycle management features to automatically rebuild and redeploy containers when upstream source or container images change. Scale out and fail over applications based on scheduling and policy. Combine composite applications built from independent components or services.
- **User interfaces:** OpenShift provides a web UI for deploying and monitoring applications, and a CLI for remote management of applications and resources. It supports the Eclipse IDE and JBoss Developer Studio plug-ins so that developers can stay with familiar tools, and a REST API for integration with third-party or in-house tools.
- **Collaboration:** OpenShift allows you to share projects and customized runtimes within an organization or with the larger community.
- **Scalability and High Availability:** OpenShift provides container multitenancy and a distributed application platform including elasticity to handle increased traffic on demand. It provides high availability so that applications can survive events such as the loss of a physical machine. OpenShift provides automatic discovery of poor container health and automatic redeployment.
- **Container portability:** In OpenShift, applications and services are packaged using standard container images and composite applications are managed using Kubernetes. These images can be deployed to other platforms built on those base technologies.
- **Open source:** No vendor lock-in.
- **Security:** OpenShift provides multi-layered security using SELinux, role-based access control, and the ability to integrate with external authentication systems such as LDAP and OAuth.
- **Dynamic Storage Management:** OpenShift provides both static and dynamic storage management for container data using the Kubernetes concepts of Persistent Volumes and Persistent Volume Claims.
- **Choice of cloud** (or no cloud): Deploy OpenShift Container Platform on bare-metal servers, hypervisors from multiple vendors, and most IaaS cloud providers.
- **Enterprise Grade:** Red Hat provides support for OpenShift, selected container images, and application runtimes. Trusted third-party container images, runtimes, and applications are certified by Red Hat. You can run in-house or third-party applications in a hardened, secure environment with high availability provided by OpenShift.
- **Log Aggregation and Metrics:** Logging information from applications deployed on OpenShift can be collected, aggregated, and analyzed in a central location. OpenShift enables you to collect metrics and runtime information in real time about your applications and helps optimize the performance continuously.

OpenShift is an enabler for microservice architectures, while also supporting more traditional workloads. Many organizations will also find OpenShift native features sufficient to enable a

DevOps process, while others will find it is easy to integrate with both standard and custom Continuous Integration/Continuous Deployment tools.



References

OpenShift Container Platform product documentation:

<https://access.redhat.com/documentation/en/openshift-container-platform/>

Quiz: OpenShift Container Platform Features

Choose the correct answers to the following questions:

1. Which of the following two statements about OpenShift are correct? (Choose two.)
 - a. Applications run as virtual machines on OpenShift. VMs provide security, storage, and network isolation for applications.
 - b. Applications run as containers on OpenShift. Containers provide security, storage, and network isolation for applications.
 - c. OpenShift uses a proprietary application packaging and deployment format that is not portable and works only on OpenShift.
 - d. Applications and services are packaged using standard container images that can be deployed to other platforms.
2. Which of the following three statements about OpenShift are true? (Choose three.)
 - a. It can only run on bare-metal physical servers.
 - b. It provides certified container images for many popular application runtimes.
 - c. Developers can create and start cloud applications directly from a source-code repository.
 - d. It allows easy integration with third-party tools through a REST API.
 - e. Only RHEL-based containers can run on OpenShift.
 - f. It is based on proprietary code available exclusively to Red Hat subscribers.
3. Which of the following four environments can support an OpenShift deployment? (Choose four.)
 - a. Bare-metal servers running RHEL 7.
 - b. Bare-metal servers running Windows Server.
 - c. Popular public IaaS cloud providers.
 - d. Popular private IaaS cloud environments.
 - e. Popular public PaaS cloud providers.
 - f. Virtual servers hosted by popular hypervisors.
4. Which of the following two statements about OpenShift are true? (Choose two.)
 - a. Only Java based applications are supported on OpenShift.
 - b. You can deploy Wordpress blog software on OpenShift (Wordpress is built on Apache, MySQL, and PHP).
 - c. NoSQL databases are not supported. Only relational databases such as MySQL and PostgreSQL are supported.
 - d. NoSQL databases such as MongoDB are supported.
5. Which of the following two statements about OpenShift high availability and scaling are true? (Choose two.)
 - a. High availability is not provided by default. You need to use third-party high availability products.

- b. High availability is provided by default.
- c. High availability and scaling is restricted only to Java based applications.
- d. OpenShift can scale up and scale down based on demand.
- e. OpenShift cannot automatically scale up or down. An administrator has to stop the cluster and manually scale the applications.

Solution

Choose the correct answers to the following questions:

1. Which of the following two statements about OpenShift are correct? (Choose two.)
 - a. Applications run as virtual machines on OpenShift. VMs provide security, storage, and network isolation for applications.
 - b. **Applications run as containers on OpenShift. Containers provide security, storage, and network isolation for applications.**
 - c. OpenShift uses a proprietary application packaging and deployment format that is not portable and works only on OpenShift.
 - d. **Applications and services are packaged using standard container images that can be deployed to other platforms.**
2. Which of the following three statements about OpenShift are true? (Choose three.)
 - a. It can only run on bare-metal physical servers.
 - b. **It provides certified container images for many popular application runtimes.**
 - c. **Developers can create and start cloud applications directly from a source-code repository.**
 - d. **It allows easy integration with third-party tools through a REST API.**
 - e. Only RHEL-based containers can run on OpenShift.
 - f. It is based on proprietary code available exclusively to Red Hat subscribers.
3. Which of the following four environments can support an OpenShift deployment? (Choose four.)
 - a. **Bare-metal servers running RHEL 7.**
 - b. Bare-metal servers running Windows Server.
 - c. **Popular public IaaS cloud providers.**
 - d. **Popular private IaaS cloud environments.**
 - e. Popular public PaaS cloud providers.
 - f. **Virtual servers hosted by popular hypervisors.**
4. Which of the following two statements about OpenShift are true? (Choose two.)
 - a. Only Java based applications are supported on OpenShift.
 - b. **You can deploy Wordpress blog software on OpenShift (Wordpress is built on Apache, MySQL, and PHP).**
 - c. NoSQL databases are not supported. Only relational databases such as MySQL and PostgreSQL are supported.
 - d. **NoSQL databases such as MongoDB are supported.**
5. Which of the following two statements about OpenShift high availability and scaling are true? (Choose two.)
 - a. High availability is not provided by default. You need to use third-party high availability products.
 - b. **High availability is provided by default.**

- c. High availability and scaling is restricted only to Java based applications.
- d. **OpenShift can scale up and scale down based on demand.**
- e. OpenShift cannot automatically scale up or down. An administrator has to stop the cluster and manually scale the applications.

Describing the OpenShift Container Platform Architecture

Objectives

After completing this section, students should be able to describe the architecture of Red Hat OpenShift Container Platform.

Overview of OpenShift Container Platform Architecture

OpenShift Container Platform is a set of modular components and services built on top of Red Hat Enterprise Linux, Docker, and Kubernetes. OpenShift adds capabilities such as remote management, multitenancy, increased security, application life-cycle management, and self-service interfaces for developers. The following figure illustrates the OpenShift software stack:

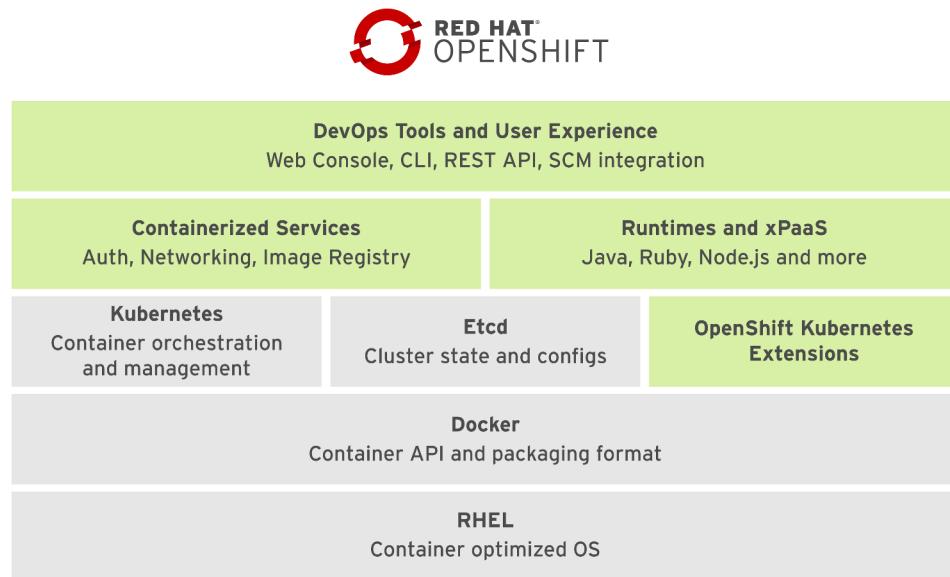


Figure 1.1: OpenShift architecture

In the above figure, going from bottom to top, and from left to right, the basic container infrastructure is shown, integrated and enhanced by Red Hat:

- The base OS is Red Hat Enterprise Linux (RHEL).
- **Docker** provides the basic container management API and the container image file format.
- **Kubernetes** manages a cluster of hosts (physical or virtual) that run containers. It works with resources that describe multi-container applications composed of multiple resources, and how they interconnect.
- **Etcd** is a distributed key-value store, used by Kubernetes to store configuration and state information about the containers and other resources inside the OpenShift cluster.

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

- **OpenShift-Kubernetes extensions** are additional resource types stored in Etcd and managed by Kubernetes. These additional resource types form the OpenShift internal state and configuration, alongside application resources managed by standard Kubernetes resources.
- **Containerized services** fulfill many infrastructure functions, such as networking and authorization. Some of them run all the time, while others are started on demand. OpenShift uses the basic container infrastructure from Docker and Kubernetes for most internal functions. That is, most OpenShift internal services run as containers managed 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**: OpenShift provides a Web UI and CLI management tools for developers and system administrators, allowing the configuration and monitoring of applications and OpenShift 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. OpenShift can also reach external SCM repositories and container image registries and bring their artifacts into the OpenShift cloud.

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

The main value that OpenShift adds to Docker + Kubernetes is automated development workflows, so that application building and deployment happen inside the OpenShift cluster, following standard processes. Developers do not need to know the low-level Docker details. OpenShift takes the application, packages it, and starts it as a container.



Note

Until recently, the Docker community had no features to support composite applications running as multiple, interconnected containers, which are needed by both traditional, layered enterprise applications, and by newer microservice architectures. The community launched the Docker Swarm project to address this gap, but Kubernetes was already a popular choice to fulfill this need. Kubernetes has been deployed in real world production environments, where it manages more than 2 billion Docker containers daily.

Master and Nodes

An OpenShift cluster is a set of *node* servers that run containers and are centrally managed by a set of *master* servers. A server can act as both a master and a node, but those roles are usually segregated for increased stability.

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

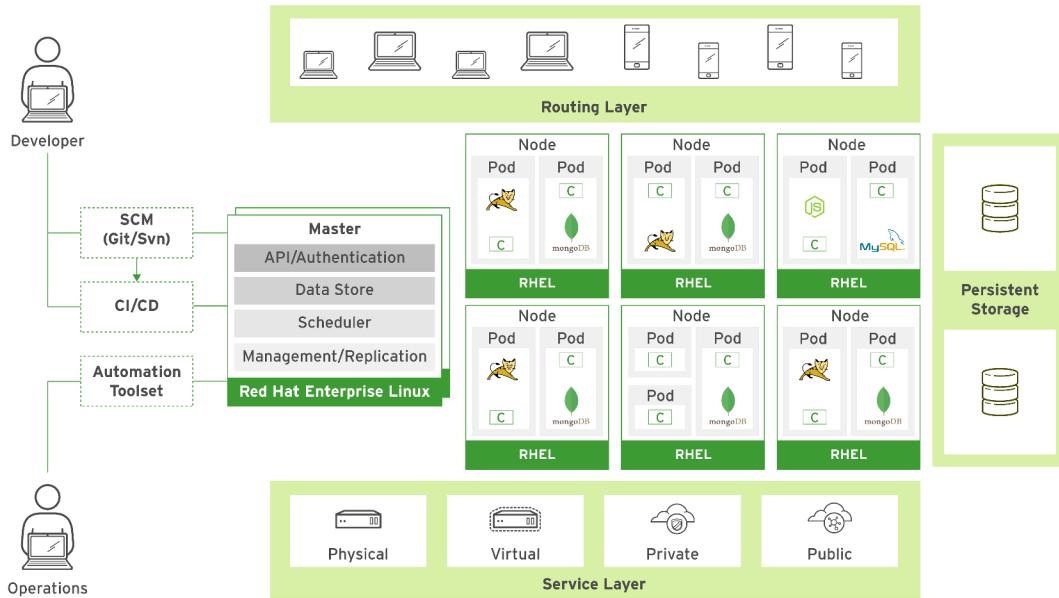


Figure 1.2: OpenShift Container Platform master and nodes

The master runs OpenShift core services such as authentication, and provides the API entry point for administration. The 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.

OpenShift masters run the **Kubernetes master** services and **Etcd** daemons, while the nodes run the Kubernetes **kubelet** and **kube-proxy** daemons. While not shown in the figure, the masters are also 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 and PHP pods running the same container image could be used for horizontally scaling a web application.

OpenShift Projects and Applications

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

There is no concept of an application in OpenShift. The OpenShift client provides a **new-app** command. This command creates resources inside a project, but none of them are application

resources. This command is a shortcut to configure a project with common resources for a standard developer workflow. OpenShift uses *labels* to categorize resources within the cluster. By default, OpenShift uses the *app* label to group related resources together into an *application*.

Building Images with 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. OpenShift allows developers to work with standard source control management (SCM) repositories and integrated development environments (IDEs).

The Source-to-Image (S2I) process in OpenShift pulls code from an SCM repository, automatically detects which kind of runtime that source code needs, and starts a pod from a base image specific to that kind of runtime. Inside this pod, OpenShift builds the application the same way that 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 OpenShift. 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.

There are many variations to CI/CD pipelines, and 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 such as Jenkins could be used to start the build and run tests, then label the newly built image as a success or a failure, 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 OpenShift CLI, the web console, or the REST API. These resources can be viewed as JSON or YAML text files, and shared on an SCM system like Git or Subversion. OpenShift can even retrieve these resource definitions directly from an external SCM.

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

For example, if a new pod resource is created, Kubernetes schedules and starts that pod on a node, using the pod resource to determine which image to use, which ports to expose, and so on. As a second example, if a template is changed so that it specifies that there should be more pods to handle the load, OpenShift schedules 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 OpenShift CLI and OpenShift APIs to manage applications and infrastructure. OpenShift 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. Access to those core components can be valuable for system administrators during troubleshooting.

OpenShift Networking

Docker networking is very simple. Docker creates a virtual kernel bridge and connects each container network interface to it. Docker itself does not provide a way to allow a pod on one host to connect to a pod on another host. Neither does Docker provide a way to assign a public fixed IP address to an application so that external users can access it.

Kubernetes provides service and route resources to manage network visibility between pods and route traffic from the external world to the pods. 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 a fixed unique DNS name for a service, making it visible to clients outside the OpenShift cluster.

Kubernetes service and route resources need external help to perform their functions. A service needs software-defined networking (SDN) which will provide visibility between pods on different hosts, and a route needs something that forwards or redirects packets from external clients to the service internal IP. OpenShift provides an SDN based on **Open vSwitch**, and routing is provided by a distributed **HAProxy** farm.

Persistent Storage

Pods might be stopped on one node and restarted on another node at any time. Consequently, plain Docker storage is inadequate because of its default ephemeral nature. If a database pod was stopped and restarted on another node, any stored data would be lost.

Kubernetes provides a framework for managing external persistent storage for containers. Kubernetes recognizes a *PersistentVolume* resource, which can define either local or network storage. A pod resource can reference a *PersistentVolumeClaim* resource in order to access storage of a certain 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 stays connected to the same PersistentVolumeClaim and PersistentVolume instances. This means that a pod's persistent storage data follows it, regardless of the node where it is scheduled to run.

OpenShift adds to Kubernetes a number of *VolumeProviders*, which provide access to enterprise storage, such as NFS, iSCSI, Fibre Channel, Gluster, or a cloud block volume service such as OpenStack Cinder.

OpenShift also provides dynamic provisioning of storage for applications via the *StorageClass* resource. Using dynamic storage, you can select different types of back-end storage. The back-

end storage is segregated into different "tiers" depending on the needs of your application. For example, a cluster administrator can define a StorageClass with the name of "fast," which makes use of higher quality back-end storage, and another StorageClass called "slow," which provides commodity-grade storage. When requesting storage, an end user can specify a **PersistentVolumeClaim** with an annotation that specifies the value of the StorageClass they prefer.

OpenShift High Availability

High Availability (HA) on an OpenShift Container Platform cluster has two distinct aspects: HA for the OpenShift infrastructure itself (that is, the masters); and HA for the applications running inside the OpenShift cluster.

OpenShift provides a fully supported native HA mechanism for masters by default.

For applications, or "pods," OpenShift handles this by default. If a pod is lost for any reason, Kubernetes schedules another copy, connects it to the service layer and to the persistent storage. If an entire node is lost, Kubernetes schedules replacements for all its pods, and eventually all applications will be available again. The applications inside the pods are responsible for their own state, so they need to maintain application state on their own (for example, by employing proven techniques such as HTTP session replication or database replication).

Image Streams

To create a new application in OpenShift, in addition to the application source code, a base image (the S2I builder image) is required. If either of these two components are updated, a new container image is created. Pods created using the older container image are replaced by pods using the new image.

While it is obvious that the container image needs to be updated when application code changes, it may not be obvious that the deployed pods also need to be updated if the builder image changes.

An *image stream* comprises any number of container images identified by *tags*. It presents a single virtual view of related images. Applications are built against image streams. Image streams can be used to automatically perform an action when new images are created. Builds and deployments can watch an image stream to receive notifications when new images are added, and react by performing a build or deployment, respectively. OpenShift provides several image streams by default, encompassing many popular language runtimes and frameworks.

An *image stream tag* is an alias pointing to an image in an image stream. It is often abbreviated to *istag*. It contains a history of images represented as a stack of all images that the tag ever pointed to. Whenever a new or existing image is tagged with a particular istag, it is placed at the first position (tagged as *latest*) in the history stack. Images previously tagged as *latest* will be available at the second position. This allows for easy rollbacks to make tags point to older images again.



References

Additional information about the OpenShift architecture can be found in the *OpenShift Container Platform Architecture* document at

|| https://access.redhat.com/documentation/en-us/openshift_container_platform/

Quiz: OpenShift Container Platform Architecture

Match the items below to their counterparts in the table.

Container	Docker	Etcd	Image Stream	JSON
Kubernetes	Master	Node	Open vSwitch	PersistentVolume
Pod	Project	Route	S2I	Service
xPaaS				

Description	Name
Stores OpenShift cluster resource definitions	
Defines the container image format	
Manages and schedules application pods in the OpenShift cluster	
Provides JBoss middleware certified container images	
Shares networking and storage configurations from the enclosing pod	
Runs the OpenShift REST API, authentication, scheduler, and configuration data store	
Builds and deploys applications from source code	

Description	Name
Runs pods, kubelet, and proxy	
File format used to describe OpenShift cluster resources	
Load-balances requests for replicated pods from the same application	
Provides persistent storage for stateful applications such as relational databases	
Dynamically provisions storage for applications based on storage tiers	
A named alias for a set of related container images	
Allows access to applications from external networks	
A set of containers that must run on the same node	
Software-defined networking that allows pods from different nodes to be part of the same service	
Can be assigned resource quotas	

Solution

Match the items below to their counterparts in the table.

Description	Name
Stores OpenShift cluster resource definitions	Etcd
Defines the container image format	Docker
Manages and schedules application pods in the OpenShift cluster	Kubernetes
Provides JBoss middleware certified container images	xPaaS
Shares networking and storage configurations from the 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 cluster resources	JSON
Load-balances requests for replicated pods from the same application	Service
Provides persistent storage for stateful applications such as relational databases	PersistentVolume

Description	Name
Dynamically provisions storage for applications based on storage tiers	StorageClass
A named alias for a set of related container images	Image Stream
Allows access to applications from external networks	Route
A set of containers that must run on the same node	Pod
Software-defined networking that allows pods from different nodes to be part of the same service	Open vSwitch
Can be assigned resource quotas	Project

Summary

In this chapter, you learned:

- Red Hat OpenShift Container Platform is a container application platform based on Red Hat Enterprise Linux (RHEL), containers, and Kubernetes.
- OpenShift Container Platform allows developers to focus on source code and rely on the container platform infrastructure to build and deploy containers to run applications.
- The OpenShift architecture employs **master** servers that manage **node** servers that run applications as containers.
- OpenShift provides additional authentication, security, scheduling, networking, storage, logging, metrics, and application life-cycle management over default Kubernetes features.
- OpenShift provides built-in high availability (HA) for masters and pods.



CHAPTER 2

INSTALLING OPENSHIFT CONTAINER PLATFORM

Overview	
Goal	Install OpenShift and configure the cluster.
Objectives	<ul style="list-style-type: none">• Prepare the servers for installation.• Run the OCP installer to configure the cluster.• Execute postinstallation tasks and verify the cluster configuration.
Sections	<ul style="list-style-type: none">• Preparing Servers for Installation (and Workshop)• Running the Installer (and Guided Exercise)• Executing Postinstallation Tasks (and Guided Exercise)
Lab	<ul style="list-style-type: none">• None

Preparing Servers for Installation

Objective

After completing this section, students should be able to prepare the servers for installation.

General Installation Overview

Red Hat OpenShift Container Platform is delivered by Red Hat as a mixture of RPM packages and container images. The RPM packages are downloaded from standard Red Hat repositories (that is, Yum repositories) using subscription manager, and the container images come from the Red Hat private container registry.

OpenShift Container Platform installations require multiple servers, and these can be any combination of physical and virtual machines. Some of them are *masters*, others are *nodes*, and each type needs different packages and configurations. To make bootstrapping an OpenShift cluster easier, Red Hat provides an Ansible-based installer that can be run either interactively by answering a series of questions, or in an automated non-interactive fashion using an answer file containing configuration details of the environment.

Before running the installer, there are preinstallation tasks that a system administrator needs to perform, and postinstallation tasks after the install to get a fully functional OpenShift Container Platform cluster. Red Hat offers two different methods for installing OpenShift Container Platform. The first method uses a quick installer, which can be used for simple cluster setups. The second method is designed for more complex installations, and uses Ansible Playbooks to automate the process.

To speed up the preinstall and postinstall tasks, we will use *Ansible* in this course to automate the configuration of an OpenShift cluster. You will get a chance to perform these tasks manually in the comprehensive review lab at the end of this course. The next section briefly introduces Ansible concepts and how to install and use it.

What is Ansible?

Ansible is an open source automation platform which is used to customize and configure multiple servers in a consistent manner. OpenShift uses Ansible to perform the installation.

For the purpose of this course, Ansible is responsible for:

- Preparing the hosts for OpenShift installation, such as package installation, disabling services, and customizing configurations.
- Installing OpenShift.
- Running postinstallation tasks needed by OpenShift to start the required services and containers correctly.

Ansible requires a playbook and an inventory to automate tasks.

A playbook is a YAML file that defines a set of tasks, and these tasks automate a process. An Ansible Playbook has the following format:

```
- name: "Server installation"①
```

```

hosts: servers 2
remote_user: root 3
vars:
  local_docker: "workstation.lab.example.com:5000"
tasks:
  - name: Create /root/.ssh
    file:4
      path: /root/.ssh
      mode: 0700
      owner: root
      group: root
      state: directory
    when: inventory_hostname in groups['servers']
...

```

- 1** The playbook name.
- 2** The hostgroup alias where the specified tasks are executed.
- 3** The username used on each host from the group alias to execute a certain task.
- 4** The module that executes the playbook.

Each host is defined in an inventory file that resembles the following excerpt:

```
[servers] 1
server1.lab.example.com
server2.lab.example.com
```

- 1** Defines a group name for a set of hosts.

In the previous playbook, the hosts named **server1.lab.example.com** and **server2.lab.example.com** execute the tasks in the playbook.

Each task in a playbook calls an Ansible module. It controls services, packages, files, or system commands on one or more hosts.

Ansible playbooks are *idempotent*. Ansible uses marker files to minimize problems with failures in multi-step tasks.

Some modules support idempotency with the **when** attribute in the playbook.

Installing Ansible

Ansible is installed using the **yum install** command:

```
# yum install ansible
```

Running an Ansible Playbook

To execute an Ansible Playbook, execute the following command

```
# ansible-playbook -i <inventory-file> <playbook-filename>
```

Preparing the Environment

Ensure the following prerequisites are satisfied before you begin the installation:

- The master and nodes require static IP addresses with resolvable fully qualified domain names (FQDN). The **dig** command provides a way to validate the FQDN host names and IP addresses.

```
[root@master ~]# dig master.lab.example.com
```

- The master and nodes need to be able to communicate with each other. The **ping** command provides a way to check communication between the master and node hosts.

```
[root@master ~]# ping node1.lab.example.com
```

- A wildcard DNS zone must resolve to the IP address of the node running the OpenShift router component. The **ping** or **dig** commands can be used for this purpose. Normally, the wildcard DNS zone can be tested by calling a host name that does not exist in a domain. For example, in the classroom, the wildcard domain **cloudapps.lab.example.com** is used for all applications running on OpenShift:

```
[root@master ~]# dig test.cloudapps.lab.example.com
```

- The **NetworkManager** service should be started and enabled on all masters and nodes. To verify, the **systemctl status NetworkManager** command can be executed on all hosts that are part of an OpenShift cluster. Use the **systemctl start NetworkManager** and **systemctl enable NetworkManager** commands to start and enable the NetworkManager service.
- The **firewalld** service should be stopped and disabled on all masters and nodes. If firewalld is still running, use **systemctl stop firewalld** and **systemctl disable firewalld** to stop and disable the service permanently. Run the **systemctl status firewalld** command on all OpenShift hosts to verify that the service is stopped and disabled.
- Docker 1.12 or later must be installed and configured on all masters and nodes. Use **yum install docker** to install Docker. You can verify the Docker version using the **docker version** command on all hosts.

A system administrator with RHCSA or equivalent skills should be able to configure the servers and ensure that they meet most of the above requirements. For the wildcard DNS zone, they might require help from an experienced system administrator with advanced knowledge of DNS server administration.

The wildcard DNS zone is needed by the OpenShift router. Because the OpenShift router runs on an OpenShift node, system administrators need to plan in advance to configure the DNS with the IP address of the correct node.



Note

If a system administrator forgets to disable the **firewalld** service, the OpenShift installer does it automatically.

The OpenShift Container Platform installer has the following additional prerequisites:

- The master, which runs the OpenShift installer, must have passwordless SSH access to all masters and nodes.

Passwordless SSH access is configured by generating an SSH key pair using **ssh-keygen**, and then copying the public key to each target host using **ssh-copy-id**.

Generate an SSH key pair on the **master** using the **ssh-keygen** command, and copy the public key to all hosts.

```
[root@master ~]# ssh-keygen -f /root/.ssh/id_rsa -N ''
...
[root@master ~]# ssh-copy-id root@master
...
[root@master ~]# ssh-copy-id root@node1
...
[root@master ~]# ssh-copy-id root@node2
```

- Each host on the OpenShift cluster (that is, masters and nodes) needs to be registered using Red Hat Subscription Management (RHSM), not RHN Classic, and attached to valid OpenShift Container Platform subscriptions. To register and attach the hosts, use the **subscription-manager register** and **subscription-manager attach** commands.

Besides the standard RHEL repository (*rhel-7-server-rpms*), the *rhel-7-server-extras-rpms*, *rhel-7-fast-datapath-rpms*, and *rhel-7-server-optional-rpms* repositories must also be enabled. OpenShift Container Platform packages are provided by the repository *rhel-7-server-ose-3.5-rpms*. To enable the required repositories, use the **subscription-manager repos --enable** command.

Enable only the Yum repositories required by the OpenShift Container Platform installer.

```
[root@master ~]# yum-config-manager --disable "*"
...
[root@master ~]# yum-config-manager \
    --enable "rhel_dvd" \
    --enable "rhel-7-server-extras-rpms" \
    --enable "rhel-7-server-updates-rpms" \
    --enable "rhel-7-server-ose-3.5-rpms" \
    --enable "rhel-7-fast-datapath-rpms"
```

Enable these repositories on all masters and nodes in the OpenShift cluster.

Running Ansible Preparation Tasks

To run the preinstallation tasks in an automated manner, the following Ansible Playbook is provided in the classroom:

```
tasks:
  - name: Create /root/.ssh
    file: ①
      path: /root/.ssh
      mode: 0700
      owner: root
      group: root
      state: directory
    when: inventory_hostname in groups['masters']
```

```
- name: Copy lab_rsa to /root/.ssh/id_rsa
  copy:2
    src: /home/student/.ssh/lab_rsa
    dest: /root/.ssh/id_rsa
    mode: 0600
    owner: root
    group: root
    force: no
    when: inventory_hostname in groups['masters']

- name: Copy lab_rsa.pub to /root/.ssh/id_rsa.pub
  copy:3
    src: /home/student/.ssh/lab_rsa.pub
    dest: /root/.ssh/id_rsa.pub
    mode: 0644
    owner: root
    group: root
    force: no
    when: inventory_hostname in groups['masters']

- name: Deploy ssh key to root at all nodes
  authorized_key:4
    user: root
    key: "{{ lookup('file', '/home/student/.ssh/lab_rsa.pub') }}"

- name: Install docker
  yum:5
    name: docker
    state: latest

- name: Customize /etc/sysconfig/docker-storage-setup
  copy:6
    src: files/docker-storage-setup
    dest: /etc/sysconfig/docker-storage-setup

- name: Verify existence of /dev/docker-vg/docker-pool
  stat:7
    path: /dev/docker-vg/docker-pool
    register: docker_vg_status

- name: Run docker-storage-setup
  command: /usr/bin/docker-storage-setup8
  when: docker_vg_status.stat.islnk is not defined

- name: Start and enable docker
  service:9
    name: docker
    state: started
    enabled: true

- name: Install required packages
  yum:10
    name: "{{ item }}"
    state: latest
  with_items:
    - wget
    - git
    - net-tools
```

```

- bind-utils
- iptables-services
- bridge-utils
- bash-completion
- kexec-tools
- sos
- psacct
- atomic-openshift-docker-excluder
- atomic-openshift-excluder

- name: Install OpenShift tools
  yum: ⑪
    name: atomic-openshift-utils
    state: latest
  when: inventory_hostname in groups['masters']

```

- ① Creates the **/root/.ssh** directory and changes its permission to 0700.
- ② Copies the **/home/student/.ssh/lab_rsa** file to the **/root/.ssh** directory and changes the file permission to 0600.
- ③ Copies the **/home/student/.ssh/lab_rsa.pub** file to the **/root/.ssh** directory and changes the file permission to 0644.
- ④ Deploys the SSH key to all hosts in the master's group from the inventory.
- ⑤ Installs docker using **yum**.
- ⑥ Customizes the docker volume group configuration **docker-storage-setup** script.
- ⑦ Starts docker to load the changes and use the new persistent storage.
- ⑧ Verifies whether the **docker-pool** volume group was created.
- ⑨ Starts the **docker** service.
- ⑩ Installs packages needed by OpenShift.
- ⑪ Installs the *atomic-openshift-utils* package on the **master** host.



References

- | DO407: Automation with Ansible I
<https://www.redhat.com/en/services/training/do407-automation-ansible-i>

Guided Exercise: Preparing for Installation

In this exercise, you will prepare the master and node hosts for the OpenShift Container Platform installation.

Resources	
Files:	/home/student/D0280/labs/install-prepare/ prepare_install.yml /home/student/D0280/labs/install-prepare/ inventory

Outcomes

You should be able to customize and run the Ansible Playbook to prepare the master and node hosts for the OpenShift Container Platform installation.

Before you begin

To verify that the **master**, **node1**, and **node2** VM's are started, and to download the files needed by this exercise, open a terminal and run the following command:

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

Steps

1. Check the Ansible version installed on the workstation host.

- 1.1. Open a terminal (**Applications > Favorites > Terminal**) on the **workstation** host and run the following command to verify if the **ansible** package is installed:

```
[student@workstation ~]$ rpm -qa | grep ansible
```

The expected output is:

```
ansible-2.2.3.0-1.el7.noarch
```

- 1.2. Check that the **ansible** command is working correctly. From the same terminal window, run the following command:

```
[student@workstation ~]$ ansible --version
```

The expected output is:

```
ansible 2.2.3.0
  config file = /home/student/ansible.cfg
  configured module search path = Default w/o overrides
```

2. Briefly review the Ansible Playbook responsible for installing all dependencies needed by OpenShift.

-
- 2.1. Open the **/home/student/D0280/labs/install-prepare/prepare_install.yml** file with a text editor.

The playbook is composed of a set of tasks as shown below:

```
- name: Create /root/.ssh
  file:
  ...
- name: Copy lab_rsa to /root/.ssh/id_rsa
  copy:
  ...
- name: Copy lab_rsa.pub to /root/.ssh/id_rsa.pub
  copy:
  ...
- name: Deploy ssh key to root at all nodes
  authorized_key:
  ...
- name: Install docker
  yum:
  ...
- name: Customize /etc/sysconfig/docker-storage-setup
  copy:
  ...
- name: Verify existence of /dev/docker-vg/docker-pool
  stat:
  ...
- name: Run docker-storage-setup
  command: /usr/bin/docker-storage-setup
  ...
- name: Start and enable docker
  service:
  ...
- name: Install required packages
  yum:
  ...
- name: Install OpenShift tools
  yum:
  ...
```

The Ansible Playbook defines 11 tasks which takes care of the following preparation requirements for OpenShift:

- Generate the SSH keys.
- Copy the SSH keys to each node.
- Install **docker** on each node.
- Create a volume group for storing docker images.
- Restart the docker service to accept the configuration changes.
- Install the packages required to run the OpenShift Container Platform installer.

3. Briefly review the **inventory** file.

- 3.1. Tasks are executed on systems defined in an *inventory* file. Open the **/home/student/D0280/labs/install-prepare/inventory** file with a text editor.

```
[workstations]
workstation.lab.example.com

[masters]
master.lab.example.com

[nodes]
master.lab.example.com
node1.lab.example.com
node2.lab.example.com
```

The inventory file lists three groups of systems:

- workstations: The group of VMs used by Ansible to start the preparation process.
- masters: The group of VMs used as master hosts by OpenShift.
- nodes: The group of VMs used as node hosts by OpenShift.

3.2. The fully qualified domain names (FQDNs) from the inventory file are defined on a DNS server, which is running on the workstation. Identify the IP address of **node1**:

```
[student@workstation ~]$ host node1.lab.example.com
node1.lab.example.com has address 172.25.250.11
```

4. Briefly review the docker storage configuration files.
- 4.1. The docker storage configuration file is located at **/home/student/D0280/labs/install-prepare/files/docker-storage-setup** and it is copied to **/etc/sysconfig/** during the Ansible Playbook execution.
5. Run the Ansible Playbook to prepare the VMs for OpenShift.

From the workstation VM, run the following commands:

```
[student@workstation ~]$ cd D0280/labs/install-prepare
[student@workstation install-prepare]$ ansible-playbook -i inventory \
    prepare_install.yml
```

After executing the commands, the following output is expected:

```
PLAY RECAP ****
master.lab.example.com      : ok=12    changed=9     unreachable=0    failed=0
node1.lab.example.com       : ok=8     changed=6     unreachable=0    failed=0
node2.lab.example.com       : ok=8     changed=6     unreachable=0    failed=0
```



Note

The counts in the play recap summary maybe different on your system. Just make sure that there are no errors during the execution of the playbook.

This concludes the guided exercise.

Running the Installer

Objective

After completing this section, students should be able to run the Red Hat OpenShift Container Platform installer to configure a master and nodes.

Installing and Configuring a Master and Nodes

The `atomic-openshift-installer` utility was installed as part of the `atomic-openshift-utils` RPM.

To start the installer in interactive mode, run the `atomic-openshift-installer install` command from the master host as `root`. The Red Hat OpenShift Container Platform installer first displays a message about the prerequisites and then prompts for answers to some configuration questions:

- The user for SSH access, previously configured for passwordless login. Usually this is `root`, but if not, it needs unrestricted and passwordless `sudo` access.
- Whether to install Red Hat OpenShift Container Platform or a standalone registry.
- The master host list, as either host names or IP addresses, and whether to run master services in a container or as a native OS process.
- The node host list, also as either host names or IP addresses, and whether to run node services in a container or as a native OS process.



Important

All master nodes must also be in the nodes list, otherwise they will not be part of the pod's virtual network, or SDN. The OpenShift Container Platform installer will label nodes that are also masters as `Unschedulable`, so they are never used to run pods.

- A DNS subdomain for exposed routes.

The OpenShift Container Platform installer then prints a summary of the selected options and asks for confirmation.

After confirmation, the installer uses embedded Ansible Playbooks to install and configure each component needed for OpenShift masters and nodes. This configuration is enough to start working with OpenShift, but a real production setup usually requires some customization.

When the installation is complete, the OpenShift client `oc` will be available on the master, alongside the OpenShift administrator command `oadm`. The local OS `root` user on the master is now configured to run the OpenShift client and administrator commands as the *cloud administrator*.

Some OpenShift internal services, such as the internal registry and the router, are configured by default by the installer. Run the `oc get nodes` and `oc get pods` commands to verify that the installation was successful.

Introduction to YAML

The answer file used by the OpenShift Container Platform unattended installer uses YAML format, so it is important to have a basic understanding of YAML to comprehend the composition of the answer file.

YAML was designed primarily for the representation of data structures such as *lists* and *associative arrays* in an easy to write, human-readable format. This design objective is accomplished mostly by abandoning traditional enclosure syntax, such as brackets, braces, or opening and closing tags, commonly used by other languages to denote the structure of a data hierarchy. Instead, in YAML, data hierarchy structures are maintained using indentation.

YAML files optionally begin with a three-dash start of document marker and are optionally terminated with a three-dot end of file marker. Between the beginning and ending document markers, data structures are represented using an outline format, which uses space characters for indentation. There is no strict requirement regarding the number of space characters used for indentation, except that data elements must be further indented than their parents to indicate nested relationships. Data elements at the same level in the data hierarchy must have the same indentation. Blank lines can be added for readability.



Important

Indentation can only be performed using the space character. Indentation is critical to the correct interpretation of YAML. Because tab characters are treated differently by various editors and tools, YAML does not recognize the use of these characters for indentation.

Users of the **Vim** text editor can modify its action in response to **Tab** key entries. For example, with the addition of the following line to the user's **\$HOME/.vimrc**, when **vim** detects that you are editing a YAML file, it performs a two-space indentation when the **Tab** key is pressed. It autoindents subsequent lines, and expands tab characters into spaces.

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

The following is a sample YAML file showing a simple data hierarchy.

```
---
title: My book

author:
  first_name: John
  last_name: Doe

publish_date: 2016-01-01

chapters:
  - number: 1
    title: Chapter 1 Title
    pages: 10

  - number: 2
    title: Chapter 2 Title
    pages: 7
```

...

Run the Unattended Installer With an Answer File

An alternative to the interactive installation, which prompts the user for information, is to run the installer unattended. To do this, you must define the environment configuration for the **atomic-openshift-installer** command in a YAML file. This file, known as an *answer file*, contains all the necessary configuration information about the target environment. This allows the installer to run entirely unattended from start to finish.

To run the installer unattended, use the **-u** option, and pass the path to the answer file using the **-c** option as shown in the following example:

```
[root@master ~] atomic-openshift-installer -u -c /root/installer.cfg.yml install
```

The following is a sample answer file:

```
version: v2
variant: openshift-enterprise
variant_version: 3.5 1
ansible_callback_facts_yaml: /root/.config/openshift/.ansible/callback_facts.yaml
ansible_inventory_path: /root/hosts 2
ansible_log_path: /tmp/ansible.log 3
deployment:
  ansible_ssh_user: root 4
  hosts: 5
    - connect_to: master.lab.example.com 6
      hostname: master.lab.example.com 7
      ip: 172.25.250.10 8
      public_hostname: master.lab.example.com 9
      public_ip: 172.25.250.10 10
      roles: 11
        - master
        - etcd
        - node
        - storage
    - connect_to: node1.lab.example.com
      hostname: node1.lab.example.com
      ip: 172.25.250.11
      node_labels: '{"region": "infra"}' 12
      public_hostname: node1.lab.example.com
      public_ip: 172.25.250.11
      roles:
        - node
    - connect_to: node2.lab.example.com
      hostname: node2.lab.example.com
      ip: 172.25.250.12
      node_labels: '{"region": "infra"}'
      public_hostname: node2.lab.example.com
      public_ip: 172.25.250.12
      roles:
        - node
  master_routingconfig_subdomain: cloudapps.lab.example.com 13
```

```

openshift_master_cluster_hostname: None
openshift_master_cluster_public_hostname: None
proxy_exclude_hosts: ''
proxy_http: ''
proxy_https: ''

roles: ⑯
  etcd: {}
  master: {
    'openshift_docker_additional_registries' : 'workstation.lab.example.com:5000', ⑮
    'openshift_docker_blocked_registries' : { 'docker.io',
      'registry.access.redhat.com' }, ⑯
    'openshift_docker_insecure_registries' : { 'workstation.lab.example.com:5000' ,
      '172.30.0.0/16' }
  }
  node: {
    'openshift_docker_additional_registries' : 'workstation.lab.example.com:5000',
    'openshift_docker_blocked_registries' : { 'docker.io',
      'registry.access.redhat.com' },
    'openshift_docker_insecure_registries' : { 'workstation.lab.example.com:5000' ,
      '172.30.0.0/16' }
  }
  storage: {}

```

- ① The version of Red Hat OpenShift Container Platform to install (3.5, 3.4, 3.3, 3.2 or 3.1). If not specified, it defaults to the latest version.
- ② Path to the hosts file containing the hosts in the OpenShift cluster. This file is used by the installer to set up and configure other components like the metrics subsystem post installation.
- ③ Defines where Ansible stores its log. By default this is **/tmp/ansible.log**.
- ④ Defines which user Ansible uses to create an SSH connection to remote systems during the installation. By default, this is the **root** user, but any user with **sudo** privileges can be used.
- ⑤ Defines the list of hosts to install the Red Hat OpenShift Container Platform components onto.
- ⑥ Defines the private IP address of the target host, which is *required*. This allows the installer to connect to the system and gather facts before proceeding with the installation.
- ⑦ Defines the private hostname of the target host, which is *required*. This allows the installer to connect to the system and gather facts before proceeding with the installation.
- ⑧ Defines the public IP address of the target host, which is *required only when doing an unattended installation*. This is often the same as the private IP address.
- ⑨ Defines the public hostname of the target host. This is *required only when doing an unattended installation*. This is often the same as the private hostname.
- ⑩ Defines the type of services that are installed on the host. Specified as a list, options include **master**, **node**, **storage**, and **etcd**.
- ⑪ The IP address or hostname that Ansible attempts to connect to when installing, upgrading, or uninstalling the systems. Typically this will match one of the public IP addresses or hostnames defined above.
- ⑫ Node labels can be optionally set per host.
- ⑬ The routing subdomain for routes deployed on this OpenShift cluster. Must match the DNS wildcard configuration, which is a pre-requisite for an OpenShift installation.
- ⑭ Defines a dictionary of roles across the deployment.
- ⑮ A list of additional docker registries that are needed during installation.
- ⑯ A list of docker registries, which should be ignored by the installer.



Note

For examples, and a full list of variables that can be set, refer to the *Configuring Ansible* section of the *Red Hat OpenShift Container Platform Installation and Configuration* guide which can be found at: https://access.redhat.com/documentation/en-us/openshift_container_platform/

Additionally, whenever the installer is run, either interactively or unattended, it stores a copy of the answer file as `/root/.config/openshift/installer.cfg.yml`. This autogenerated answer file is useful for capturing the configuration from a previous installation so that administrators can run the installer interactively once, and then again later, unattended, with the same configuration. Also, the `/root/hosts` file is created on the master host with information needed by OpenShift to install the services. This file adds important information needed to install some OpenShift tools.

Troubleshooting the Installation Using Log Files

The OpenShift Container Platform installer generates output to the terminal for each task executed, so if something goes wrong, there is enough information to find which prerequisite was missing. After fixing the errors, run the installer again.

OpenShift Container Platform has two main systemd service units, named **atomic-openshift-master** and **atomic-openshift-node**. The **atomic-openshift-master** service starts services such as the OAuth server, the Etcd daemon, Kubernetes master, scheduler, and controllers. The **atomic-openshift-node** service starts the virtual pod network.

Both **atomic-openshift-master** and **atomic-openshift-node** must be running on all master hosts (remember that master hosts are also considered nodes). Only the **atomic-openshift-node** service must be running on the node hosts. If there are problems starting the master or node services after installation (or later), check the **systemd** service logs using either **journalctl -u atomic-openshift-master** or **journalctl -u atomic-openshift-node**.

Some OpenShift Container Platform services, such as the router and registry, run as Kubernetes pods. For these services, find the container ID and use the **docker logs** command, or use the OpenShift Container Platform client **oc logs** command.



References

Additional information about the installation process is available in the *Quick Installation* section of the *Installation and Configuration* document which can be found at
https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Running the Installer

In this exercise, you will install Red Hat OpenShift Container Platform and configure a master and two nodes.

Resources	
Files	/home/student/D0280/labs/install-run/installer.cfg.yml

Outcomes

You should be able to:

- Install OpenShift Container Platform.
- Configure a master and two nodes.

Before you begin

The guided exercise *Preparing for Installation* should have been completed. If not, reset the **master**, **node1**, and **node2** VM's, and run the catch-up script using the following commands on the **workstation** host:

```
[student@workstation ~]$ lab install-prepare setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/prepare_install.yml
```

To verify that the **master**, **node1**, and **node2** VM's are started and to download the files needed by this guided exercise, open a terminal and run the following command:

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

Steps

1. Open three new terminals on the **workstation** host and access the **master**, **node1**, and **node2** hosts:

```
[student@workstation ~]$ ssh root@master
```

```
[student@workstation ~]$ ssh root@node1
```

```
[student@workstation ~]$ ssh root@node2
```

2. Remove the openshift package exclusions from the **master** and **node** hosts:

```
[root@master ~]# atomic-openshift-excluder unexclude
```

```
[root@node1 ~]# atomic-openshift-excluder unexclude
```

```
[root@node2 ~]# atomic-openshift-excluder unexclude
```



Important

Failure to remove the exclusions from the master and node hosts results in a failed installation.

3. Create the answer file to use with the unattended installer.

To use the unattended installer, an answer file with the necessary OpenShift Container Platform configuration information must be created. A template for this file has been provided for the exercise. Customize this answer file with values specific to the classroom environment.

Use a text editor to edit the **/home/student/D0280/labs/install-run/installer.cfg.yml** file.

Notice that in the answer file a number of placeholder variables exist, such as **\$MASTER_HOSTNAME**, which need to be replaced. Use the table below to update the answer file with the correct values for the classroom environment.

Answer File Variable Values

Variable Name	Value
\$MASTER_HOSTNAME	master.lab.example.com
\$NODE1_HOSTNAME	node1.lab.example.com
\$NODE2_HOSTNAME	node2.lab.example.com
\$ROUTING_SUBDOMAIN	cloudapps.lab.example.com
\$WORKSTATION_REGISTRY_URL	workstation.lab.example.com:5000
\$INTERNAL_REGISTRY_NETWORK_CIDR	172.30.0.0/16

4. To ensure that the answer file is valid, and to check that there are no errors in the file, run the following grading script on the **workstation** VM:

```
[student@workstation ~]$ lab install-run grade
```

If your answer file is valid, the grading script should pass, and you should see output like the following:

```
Checking the unattended installer answer file
```

- Check whether file /home/student/D0280/labs/install-run/installer.cfg.yml exists PASS
- Check for the master node FQDN..... PASS
- Check for the node1 FQDN..... PASS
- Check for the node2 FQDN..... PASS
- Check the routing subdomain..... PASS
- Check the private registry URL..... PASS
- Check the OpenShift internal registry network CIDR.... PASS

```
Overall answer file check..... PASS
```

If there are errors in your answer file and the grading script fails, you can compare your answer file with the valid answer file located at **/home/student/D0280/solutions/install-run/installer.cfg.yml**. Correct the errors and re-run the grading script to ensure that it passes.

5. Copy the completed answer file to the **master** host for use by the installer.

Run the following command to copy the completed answer file to the **master** host:

```
[student@workstation ~]$ scp ~/D0280/labs/install-run/installer.cfg.yml \
root@master:/root/custom-installer.cfg.yml
```

6. Run the OpenShift Container Platform unattended installer on the **master** host.

The **atomic-openshift-installer** command is used to start the OpenShift installer. Use the **-u** option to run the installer unattended, and the **-c** option to specify the location of the answer file. Use the **install** argument to begin the installation.

Return to the terminal window where you have an SSH session on the **master** host and run the installer command:

```
[root@master ~] atomic-openshift-installer -u -c \
/root/custom-installer.cfg.yml install
*** Installation Summary ***

Hosts:
- master.lab.example.com
  - OpenShift master
  - OpenShift node (Unscheduled)
  - Etcd
  - Storage
- node1.lab.example.com
  - OpenShift node (Dedicated)
- node2.lab.example.com
  - OpenShift node (Dedicated)

Total OpenShift masters: 1
Total OpenShift nodes: 3
...Output omitted...
```

The unattended installer prints output to the screen as each play in the installer playbook is completed. Monitor the output until all the plays are complete and you are returned to a command prompt.

When the installation is complete you should see the following output in the terminal:

```
localhost          : ok=11    changed=0    unreachable=0    failed=0
master.lab.example.com   : ok=616    changed=181   unreachable=0    failed=0
node1.lab.example.com    : ok=216    changed=58    unreachable=0    failed=0
node2.lab.example.com    : ok=216    changed=58    unreachable=0    failed=0

Installation Complete: Note: Play count is only an estimate, some plays may have
been skipped or dynamically added
```

The installation was successful!

The installer prints a summary of the ansible playbook execution in the end. The counts in the play recap summary maybe different on your system. Just make sure that there are no errors during the execution of the installer.



Important

The installation can take quite a long time. Wait for the installation playbook to complete and the command prompt to return before continuing to the next step.



Note

Detailed information about the installation process is captured in the **/tmp/ansible.log** file on the **master** host. Use this file to troubleshoot issues during the install or if the install fails.

7. Verify the installation.

The **oc** command-line interface provides many tools for monitoring the status of the OpenShift cluster. The **oc get nodes** command allows you to check the status of each host in the cluster, including the master and nodes.

In the SSH session on the **master** host, use the **oc get nodes** command to verify that the **master**, **node1**, and **node2** hosts all return a **Ready** status:

```
[root@master ~]# oc get nodes
NAME           STATUS            AGE
master.lab.example.com   Ready, SchedulingDisabled   1m
node1.lab.example.com    Ready             1m
node2.lab.example.com    Ready             1m
```

Use the **oc get pods** command to verify that the **docker-registry** and **router** pods are all up and running successfully:

```
[root@master ~]# oc get pods
NAME          READY   STATUS    RESTARTS   AGE
docker-registry-1-lfhhh  1/1     Running   0          1h
registry-console-1-deploy 1/1     Running   0          1h
registry-console-1-qq9qz  0/1     ImagePullBackOff 0          2m
router-1-jqzh0        1/1     Running   0          1h
router-1-v130k        1/1     Running   0          1h
```

The **registry-console** pod does not start because the default configuration of the OpenShift installer pulls the **registry-console** image from **registry.access.redhat.com**. If you are in an offline environment with no internet access, you will see an **ImagePullBackOff**, **ErrImagePull**, or **Error** status. You will fix this issue as part of the post installation steps.

Inspect the **/root/hosts** file on the **master** VM. It contains information about the hosts that are part of the OpenShift cluster. It is used by the **custom-installer.cfg.yml** file

to complete the installation process as well as to install some additional services managed by OpenShift, such as the monitoring tools.



Important

If the hosts file is missing or created in a different directory, then the metrics install exercise later in the course will fail. Ensure that this file is created in **/root/hosts**. If it is created in a different directory, then manually copy it to **/root/hosts**.

This concludes the guided exercise.

Executing Postinstallation Tasks

Objective

After completing this section, students should be able to execute post installation tasks and verify OpenShift cluster configuration.

When the OpenShift installation is complete, verify that the router and docker registry pods have a status of **Running** by running **oc get pods** command. The following output is expected:

```
[root@master ~]# oc get pods
NAME           READY   STATUS    RESTARTS   AGE
docker-registry-6-oystdi   1/1     Running   0          1m
registry-console-1-deploy  1/1     Running   0          20m
registry-console-1-vdj8x   0/1     ImagePullBackOff 0          1m
router-1-7n637            1/1     Running   0          1m
router-1-xkgf7            1/1     Running   0          1m
```

The **registry-console** pod does not start because the default configuration of the OpenShift installer pulls the **registry-console** image from **registry.access.redhat.com**. If you are in an offline environment with no internet access, you will see an **ImagePullBackOff**, **ErrImagePull**, or **Error** status. To fix this issue, use the **oc set** command to change the configuration for the registry console to point to your internal private registry:

```
[root@master ~]# oc set image --source=docker dc/registry-console \
  registry-console=workstation.lab.example.com:5000 \
  /openshift3/registry-console:3.5
```

Note

Ensure there is no white space around the backslash (\) character after the port number in the second line.

Run **oc get pods** and verify that all the pods have a status of **Running** and there are no errors:

```
[root@master ~]# oc get pods
NAME           READY   STATUS    RESTARTS   AGE
docker-registry-6-oystdi   1/1     Running   0          1m
registry-console-2-wijvb  1/1     Running   0          20s
router-1-7n637            1/1     Running   0          1m
router-1-xkgf7            1/1     Running   0          1m
```

Finally, verify that the **atomic-openshift-master** and **atomic-openshift-node** services are started and enabled on the **master** host, and the **atomic-openshift-node** service is started and enabled on the **node** hosts.



Important

The docker registry configured by the OpenShift installation uses ephemeral storage. Therefore, the images stored in this registry are destroyed after a reboot. Later, in the course, the storage will be changed to become a persistent one.

Fixing the ImageStream Resources for Offline Environments

The OpenShift Container Platform installer configures a number of *ImageStream* resources, which are used by *Source To Image (S2I)* builders, to pull images from the Red Hat subscriber private registry. But, in an offline environment, the ImageStream resources should pull the images from an internal registry.

To fix this issue, delete all predefined image streams:

```
[root@master ~]# oc delete is -n openshift --all
```

Create a copy of the example image stream definition files that are provided with a default OpenShift installation:

```
[root@master ~]# cp -r /usr/share/openshift/examples/image-streams \
~/openshift-examples
```

Notice that many of the example image streams are not used in the classroom because the container images they reference are not in the private registry. There is no need to remove these extra image streams.

Edit the copy that you just created to replace the Red Hat registry URL with the classroom registry, and to configure the image streams for insecure registries.

The annotation **openshift.io/image.insecureRepository** has to be added to all image streams. The **importPolicy** attribute has to be added to all tags with **from.kind** attribute set to **DockerImage**, inside all image streams.

For example, the following listing shows the changes made to the **php** image stream:

```
...
{
  "kind": "ImageStream",
  "apiVersion": "v1",
  "metadata": {
    "name": "php",
    "annotations": {
      "openshift.io/image.insecureRepository": "true",
      "openshift.io/display-name": "PHP"
    }
  },
  "spec": {
    "tags": [
      ...
      {
        ...
      }
    ]
  }
}
```

```
"name": "7.0",
"annotations": {
    "openshift.io/display-name": "PHP 7.0",
    ...
},
"from": {
    "kind": "DockerImage",
    "name": "workstation.lab.example.com:5000/rhscl/php-70-rhel7:latest"
},
"importPolicy":{
    "insecure":true
}
...
...
```

There should be no line breaks in string attribute values.

Remember to make the same changes to all other image streams and all tags in the image stream JSON files.

Recreate the image streams using the modified JSON file.

```
[root@master ~]# oc create -n openshift -f image-streams-rhel7.json
```

OpenShift will try to fetch metadata for each image referenced by each image stream. You can verify that the images available in the private registry were found using the **oc describe** command, for example:

```
[root@master ~]# oc describe is php -n openshift
...
7.0 (latest)
tagged from workstation.lab.example.com:5000/rhscl/php-70-rhel7:latest
will use insecure HTTPS or HTTP connections
...
* workstation.lab.example.com:5000/rhscl/php-70-
rhel7@sha256:3ff8e5a2d3f0933acadd30898ab5451392eb2189483bcb34c735e55de097a169
    2 minutes ago
...
```

Images not available in the private registry will display errors, such as:

```
[root@master ~]# oc describe is php -n openshift
...
5.6
tagged from workstation.lab.example.com:5000/rhscl/php-56-rhel7:latest
will use insecure HTTPS or HTTP connections
...
! error: Import failed (NotFound): dockerimage "workstation.lab.example.com:5000/
rhscl/php-56-rhel7:latest" not found
    2 minutes ago
...
```

Authentication Methods

The OpenShift Container Platform authentication is based on *OAuth*, which provides an HTTP-based API for authenticating both interactive and non-interactive clients. The OpenShift master runs an OAuth server, and OpenShift can be configured with a number of *identity providers*

which can be integrated with organization-specific identity management products. Supported OpenShift identity providers are:

- HTTP Basic, to delegate to external Single Sign-On (SSO) systems
- GitHub and GitLab, to use GitHub and GitLab accounts
- OpenID Connect, to use OpenID-compatible SSO and Google Accounts
- OpenStack Keystone v3 server
- LDAP v3 server

The OpenShift installer uses a *secure by default* approach, where

DenyAllPasswordIdentityProvider is the default provider. Using this provider, only the local root user on a **master** host can use OpenShift client commands and APIs.

Before users and developers can work with the OpenShift cluster, the system administrator needs to configure another identity provider.

Configuring **htpasswd** Authentication

OpenShift **HTPasswdPasswordIdentityProvider** validates users and passwords against a flat file generated by the Apache HTTPD **htpasswd** utility. This is not enterprise-grade identity management, but it is enough for a proof of concept (POC) OpenShift deployments.

The **htpasswd** utility saves user names and passwords in a plain text file, one record per line, with fields delimited by a colon (:). The password is hashed using MD5. If this file is changed to add or delete a user, or to change a user password, it is reread automatically by the OpenShift OAuth server.

To configure the OpenShift master to use **HTPasswdPasswordIdentityProvider**, edit the OpenShift master configuration file at `/etc/origin/master/master-config.yaml` and change the **provider:** property. Then restart the **atomic-openshift-master** service so the OpenShift API and clients use the new identity provider.

Creating a Test Application

To validate an OpenShift installation, a system administrator must test and verify all OpenShift components. It is not enough to simply start a pod from a sample container image, because this does not use OpenShift builders, deployers, the router, or the internal registry. To validate an OpenShift installation a system administrator needs to:

1. Let OpenShift build an application from source. OpenShift generates a container image from the build results and start a pod from that image.
2. Create a service so that the application can be accessed from the internal container network and from OpenShift nodes.
3. Create a route so that the application can be accessed from computers outside the OpenShift cluster's internal network.

Step 1 determines whether or not the OpenShift registry is running.

Step 2 is performed by OpenShift if the application image or build image provides metadata that specifies ports.

Step 3 is performed by application templates.



Note

The classroom setup includes a Git server prepopulated with sample applications. Use this repository to provide the source code for OpenShift builds.

Guided Exercise: Completing Postinstallation Tasks

In this exercise, you will ensure that the OpenShift master service and the registry console pod are running, and the ImageStream resources point to the classroom private registry.

Resources	
Files	/home/student/D0280/labs/install-post/install_post.yml /home/student/D0280/labs/install-post/image-streams-rhel7.json

Outcomes

You should be able to:

- Verify that the OpenShift **atomic-openshift-master** service is running on the **master** VM.
- Configure authentication on the **master** VM to enable access using the OpenShift command line tool (oc).
- Configure ImageStream resources to point to the private registry.

Before you begin

The guided exercise *Preparing for Installation* and the guided exercise *Running the Installer* should have been completed. If not, reset the **master**, **node1**, and **node2** hosts, and run the Ansible Playbook using the following commands from the **workstation** host:

```
[student@workstation ~]$ lab install-run setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/prepare_install.yml
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/install_ocp.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal and run the following command:

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

Steps

- Inspect the task that checks if the OpenShift **atomic-openshift-master** service is running on the master node.

Open the **/home/student/D0280/labs/install-post/post_install.yml** file with a text editor and look for the task named **Check for OCP Service**:

```
- name: Check for OCP Service
  service:
    name: atomic-openshift-master
    state: started
    enabled: true
  when: inventory_hostname in groups['masters']
```

The task verifies that the **atomic-openshift-master** service is started and enabled on all masters.

2. Inspect the task that executes OpenShift package exclusions on all hosts.

In the **/home/student/D0280/labs/install-post/post_install.yml** file, look for the task named **Re-add OpenShift package exclusions**.

```
- name: Re-add OpenShift package exclusions
  command: "/usr/sbin/atomic-openshift-excluder exclude"
```

All hosts listed in the **inventory** file run the **atomic-openshift-excluder exclude** command.

3. Inspect the task that updates the registry console to use an internal registry.

The registry console pod on the **master** VM is misconfigured and it must refer to an internal registry to support an offline environment. Because the registry console is running only on the master VM, the module execution is limited to the **masters** group in the Ansible **inventory** file.

```
- name: Fix registry console
  command: "oc set image --source=docker dc/registry-console registry-
console=workstation.lab.example.com:5000/openshift3/registry-console:3.5"
  when: inventory_hostname in groups['masters']"
```

4. Inspect the task that pauses playbook execution and waits for the **registry-console** pod to be re-deployed.

```
- name: Wait for registry-console to re-deploy
  pause:
    seconds: 15
    prompt: "Waiting for the registry-console to re-deploy"
```

5. Inspect the tasks responsible for updating the ImageStream resources from OpenShift.

The ImageStream resources must be updated because they download the S2I images from the Red Hat image registry, but the classroom does not have internet access.

- 5.1. To make the changes to the installed ImageStream resources, a JSON file is provided in the **/home/student/D0280/labs/install-post/files** directory. Open the **image-streams-rhel7.json** file in this directory, and check that the **from** attribute for all the image streams point to the **workstation.lab.example.com:5000** private registry instead of **registry.access.redhat.com**:

```
...
  "from": {
    "kind": "DockerImage",
    "name": "workstation.lab.example.com:5000/openshift3/ruby-20-rhel7:latest"
  },
  ...
```

The updated file is copied to the **master** VM by running the task **Edit RHEL7 Image Streams**.

```
- name: Edit RHEL7 Image Streams
  copy:
    src: files/image-streams-rhel7.json
    dest: /usr/share/openshift/examples/image-streams/image-streams-rhel7.json
  register: rhel7_is_result
  when: inventory_hostname in groups['masters']
```

- 5.2. Inspect the task responsible for deleting all the ImageStream resources deployed on OpenShift.

Check the task named **delete_openshift_is**. The task removes all ImageStream resources from OpenShift:

```
- name: delete_openshift_is
  command: "/usr/bin/oc delete is -n openshift --all"
  when: rhel7_is_result.changed and inventory_hostname in groups['masters']
```

- 5.3. Inspect the task responsible for adding the fixed ImageStream resources for RHEL 7.

In the Ansible Playbook, check the task named **create_rhel7_is**. It uses an existing file named **image-streams-rhel7.json** with the customized version of the ImageStream resources.

```
- name: create_rhel7_is
  command: "/usr/bin/oc create -f /usr/share/openshift/examples/image-streams/
image-streams-rhel7.json -n openshift"
  when: rhel7_is_result and inventory_hostname in groups['masters']
```

6. Inspect the task that enables access by users.

OpenShift is secured by default, which means that users cannot access it except from the master host. To change this configuration, you need to update the master configuration file to use httpd authentication. Authentication approaches are discussed later in the course.

- 6.1. Inspect the task responsible for installing the *httpd-tools* package.

The *httpd-tools* package provides the **htpasswd** utility, which is a tool to generate the password for OpenShift user access.

```
- name: Install httpd-tools
  yum:
    name: httpd-tools
    state: latest
  when: inventory_hostname in groups['masters']
```

- 6.2. Inspect the task responsible for enabling the **htpasswd** identity provider in OpenShift.

The **HTPasswdPasswordIdentityProvider** is an identity provider developed for OpenShift to read the output from the **htpasswd** tool. By default, OpenShift uses the

DenyAllPasswordIdentityProvider module, that blocks all access attempts from remote clients.

The following two tasks change the identity provider to **HTPasswdPasswordIdentityProvider**. Encrypted passwords for the users are stored in the **/etc/origin/openshift-passwd** file on the **master** VM:

```
- name: Add the path to the password file for the
  HTPasswdPasswordIdentityProvider
  lineinfile:
    dest: /etc/origin/master/master-config.yaml
    backup: "yes"
    line: '        file: "/etc/origin/openshift-passwd"'
    insertbefore: 'kind: DenyAllPasswordIdentityProvider'
  when: inventory_hostname in groups['masters']
  register: master_config_file_path_result

- name: Change master-config.yaml to use HTPasswdPasswordIdentityProvider
  lineinfile:
    dest: /etc/origin/master/master-config.yaml
    backup: "yes"
    regexp: 'kind: DenyAllPasswordIdentityProvider'
    line: '        kind: HTPasswdPasswordIdentityProvider'
  when: inventory_hostname in groups['masters'] and
  master_config_file_path_result.changed
  register: master_config_file_provider_result
```

6.3. Inspect the task responsible for creating users.

A variable named **users** is defined at the top of the Ansible Playbook with a list of users and their passwords. The file containing the passwords generated by the **htpasswd** command must be called **/etc/origin/openshift-passwd**.

```
- name: Allow oc and web access for users
  htpasswd:
    path: "{{ passwd_file }}"
    name: "{{ item.name }}"
    password: "{{ item.password }}"
    state: present
    create: yes
  with_items: "{{ users }}"
  when: inventory_hostname in groups['masters']
```

6.4. Inspect the task responsible for restarting the OpenShift master service.

The task named **Restart OpenShift service to apply authentication changes** restarts the master service.

```
- name: Restart OpenShift Master service to apply authentication changes
  service:
    name: atomic-openshift-master
    state: restarted
  when: inventory_hostname in groups['masters'] and
  master_config_file_result.changed
```

6.5. Inspect the task that pauses playbook execution and waits for the master services to be restarted.

```
- name: Waiting for the master to restart
  pause:
    seconds: 10
    prompt: "Waiting for the master to settle"
```

6.6. Inspect the task responsible for enabling administrative access for the **admin** user.

The task named **Give admin user proper permissions** changes the **admin** user permission to support administrative commands in an OpenShift cluster.

```
- name: Give admin user proper permissions
  command: "oc adm policy add-cluster-role-to-user cluster-admin admin"
  when: inventory_hostname in groups['masters']
```

7. Install the OpenShift client on the **workstation** VM.

OpenShift has a client (**oc**) that can be installed using YUM. It allows remote access to OpenShift masters from the command line.

7.1. Inspect the task responsible for installing the OpenShift clients.

The *httpd-tools* package provides the **htpasswd** utility, a tool to generate the password for OpenShift user access.

```
- name: "Completing the Post Installation Tasks (workstation)"
  hosts: workstations
  remote_user: root
  tasks:
    - name: Install atomic-openshift-clients
      yum:
        name: atomic-openshift-clients
        state: latest
```

8. Run the playbook.

Run the playbook by opening a new terminal (**Applications > Favorites > Terminal**) and running the following commands:

```
[student@workstation ~]$ cd D0280/labs/install-post
[student@workstation install-post]$ ansible-playbook -i inventory post_install.yml
```

The playbook outputs the following log after execution:

```
PLAY RECAP ****
master.lab.example.com : ok=15   changed=10   unreachable=0   failed=0
node1.lab.example.com  : ok=2    changed=1    unreachable=0   failed=0
node2.lab.example.com  : ok=2    changed=1    unreachable=0   failed=0
workstation.lab.example.com : ok=2    changed=1    unreachable=0   failed=0
```



Note

The counts in the play recap summary maybe different on your system. Just make sure that there are no errors during the execution of the playbook.

9. Verify the OpenShift cluster installation by deploying a sample application.
 - 9.1. On the **workstation** host, run the **oc** command to check that the OpenShift client was installed successfully.

Open a terminal window on the **workstation** VM and perform the following steps:

 - 9.2. Log in to the OpenShift master (<https://master.lab.example.com:8443>) as **developer** and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
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): y

Login successful.

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

  oc new-project <projectname>

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

- 9.3. Create the **install-post** project:

```
[student@workstation ~]$ oc new-project install-post
```

- 9.4. Deploy the application from the Git repository:

```
[student@workstation ~]$ oc new-app --name=hello -i php:7.0 \
http://workstation.lab.example.com/php-helloworld
```

- 9.5. Wait until the application pod is in *Running* state. It may take a while before the application is built and deployed:

```
[student@workstation ~]$ oc get pods -w
NAME          READY   STATUS    RESTARTS   AGE
hello-1-build  0/1     Completed  0          1m
hello-1-g13jp  1/1     Running   0          59s
```

The **-w** option *watches* the pod state in real time. Press **Ctrl+C** to exit once you confirm that the pod is running.

- 9.6. Expose the service to external users:

```
[student@workstation ~]$ oc expose svc hello
route "hello" exposed
```

9.7. Access the application using the URL from the route:

```
[student@workstation ~]$ curl \
  http://hello-install-post.cloudapps.lab.example.com
...
<p>Hello World</p>
...
```



Note

It may take a while for the application to be built from source and deployed on OpenShift. You may get an error like the following when invoking the route URL:

```
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  ...
    <h1>Application is not available</h1>
    <p>The application is currently not serving requests at this
       endpoint. It may not have been started or is still starting.</p>
  ...

```

The error is due to the fact that the application pods are not yet in *Running* state, and hence the route has no registered endpoints. Give the application some more time to deploy, and try invoking the URL again after a while.

9.8. Delete the test application project:

```
[student@workstation ~]$ oc delete project install-post
```

This concludes the guided exercise.

Summary

In this chapter, you learned how to:

- Prepare the environment and install OpenShift Container Platform (OCP) using Yum repositories from an entitled subscription and using Ansible Playbooks.
- Configure **master** and **node** servers using the unattended installer.
- Configure the OpenShift image streams for an offline installation using Ansible Playbooks.
- Configure the **HTPasswdPasswordIdentityProvider** for OpenShift for authentication.
- Validate a running OpenShift cluster by creating an application from source code and deploying it to OpenShift.



CHAPTER 3

DESCRIBING AND EXPLORING OPENSIFT NETWORKING CONCEPTS

Overview	
Goal	Describe and explore OpenShift networking concepts.
Objectives	<ul style="list-style-type: none">• Describe how OpenShift implements software-defined networking.• Describe how OpenShift routing works and create a route.
Sections	<ul style="list-style-type: none">• Describing OpenShift's Implementation of Software Defined Networking (and Guided Exercise)• Creating Routes (and Guided Exercise)
Lab	Exploring OpenShift Networking Concepts

Describing OpenShift's Implementation of software-defined networking

Objective

After completing this section, students should be able to describe how OpenShift implements software-defined networking.

Software-Defined Networking (SDN)

By default, Docker networking uses a host-only virtual bridge, and all containers within a host are attached to it. All containers attached to this bridge can communicate between themselves, but cannot communicate with containers on a different host. Traditionally, this communication is handled using port mapping, where container ports are bound to ports on the host and all communication is routed via the ports on the physical host. Manually managing all of the port bindings when you have a large number of hosts with containers is cumbersome and difficult.

To enable communication between containers across the cluster, OpenShift uses a *software-defined networking (SDN)* approach. Software-defined networking is a networking model that allows network administrators to manage network services through the abstraction of several networking layers. SDN decouples the software that handles the traffic, called the *control plane*, and the underlying mechanisms that route the traffic, called the *data plane*. SDN enables communication between the control plane and the data plane.

In OpenShift 3.5, administrators can configure three SDN plug-ins for the pod network:

- The **ovs-subnet** plug-in, which is the default. **ovs-subnet** provides a *flat* pod network where every pod can communicate with every other pod and service.
- The **ovs-multitenant** plug-in provides an extra layer of isolation for pods and services. When using this plug-in, each project receives a unique Virtual Network ID (**VNID**) that identifies traffic from the pods that belong to the project. Thanks to the VNID, pods from different projects cannot communicate with pod and services from a different project.



Note

Projects with a VNID of **0** can communicate with all other pods and vice-versa. In OpenShift Container Platform, the **default** project has a VNID of **0**.

- The **ovs-networkpolicy** is a tech-preview plug-in that allows administrators to define their own isolation policies by using the **NetworkPolicy** objects.

The cluster network is established and maintained by OpenShift SDN, which creates an overlay network using Open vSwitch. Master nodes do not have access to containers via the cluster network unless administrators configure them to act as nodes.

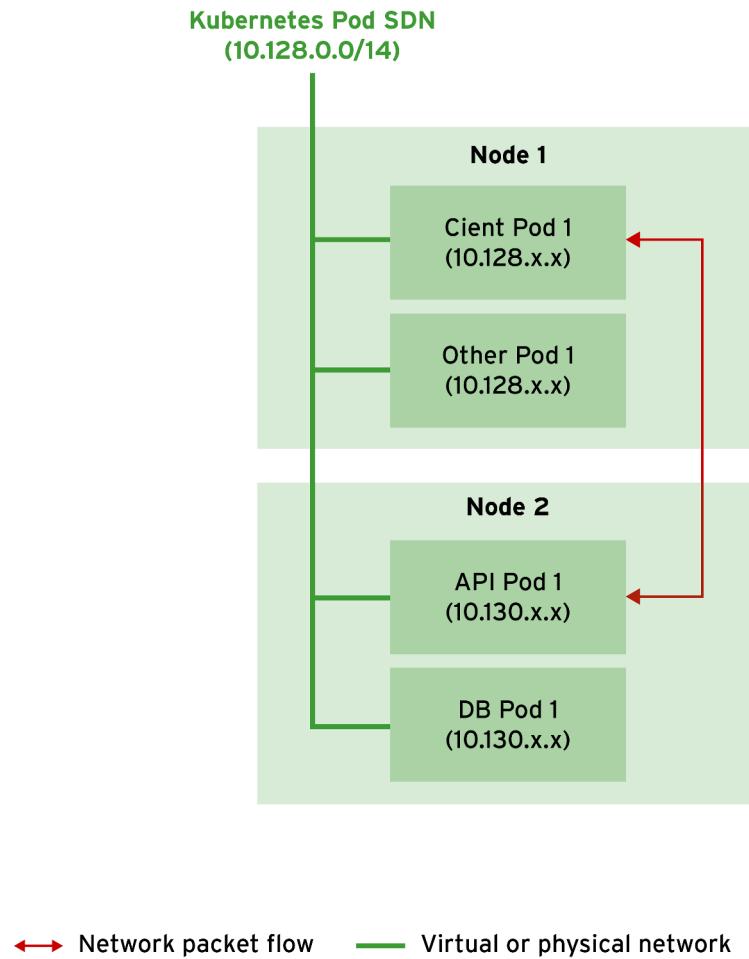


Figure 3.1: Kubernetes basic networking

In a default OpenShift Container Platform installation, each pod gets a unique IP address. All the containers within a pod behave as if they are on the same host. Giving each pod its own IP address means that pods can be treated like physical hosts or virtual machines in terms of port allocation, networking, DNS, load balancing, application configuration, and migration.

Kubernetes provides the concept of a *service*, which is an essential resource in any OpenShift application. A service acts as a load balancer in front of one or more pods. The service provides a stable IP address, and it allows communication with pods without having to keep track of individual pod IP addresses.

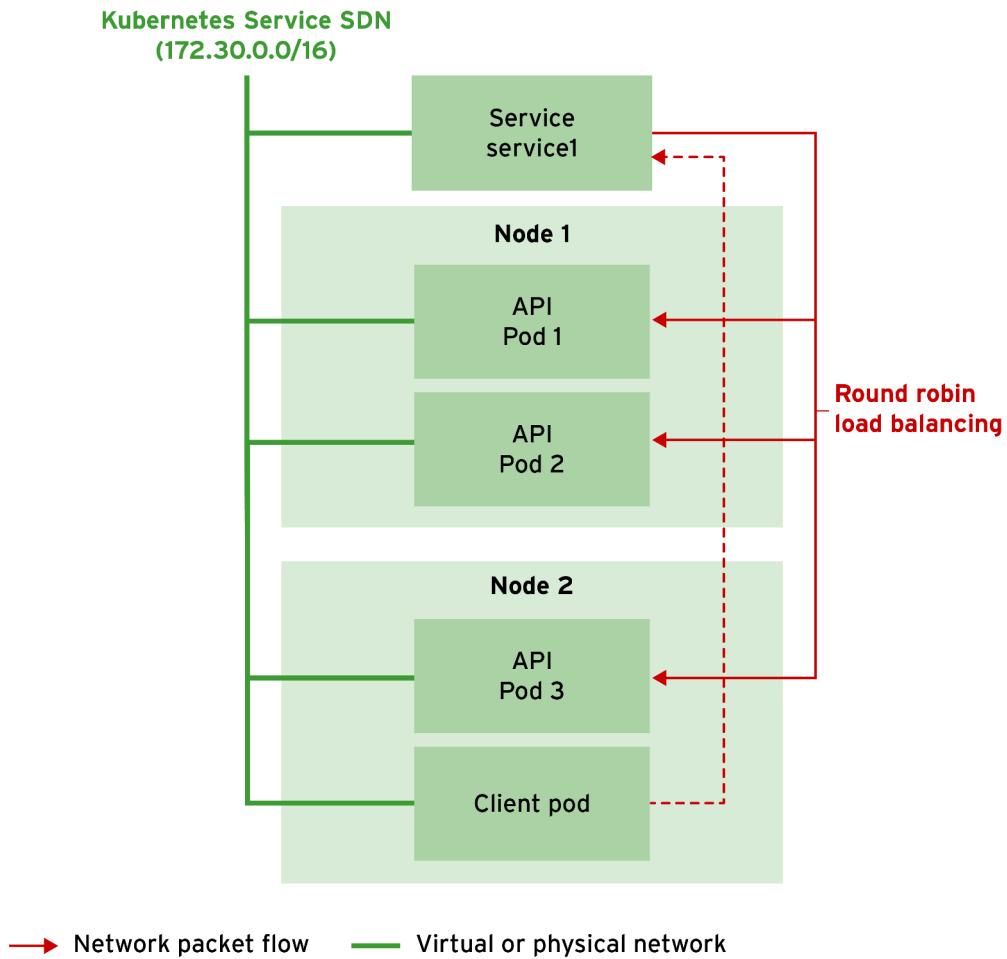


Figure 3.2: Kubernetes services networking

Most real-world applications do not run as a single pod. They need to scale horizontally, so an application could run on many pods to meet growing user demand. In an OpenShift cluster, pods are constantly created and destroyed across the nodes in the cluster. Pods get a different IP address each time they are created. Instead of a pod having to discover the IP address of another pod, a service provides a single, unique IP address for other pods to use, independent of where the pods are running. A service load-balances client requests among member pods.

OpenShift Network Topology

The set of pods running behind a service is managed automatically by OpenShift. Each service is assigned a unique IP address for clients to connect to. This IP address also comes from the OpenShift SDN and it is distinct from the pod's internal network, but visible only from within the cluster. Each pod matching the **selector** is added to the service resource as an endpoint. As pods are created and killed, the endpoints behind a service are automatically updated.

The following listing shows a minimal service definition in YAML syntax:

```
- apiVersion: v1
```

```

kind: Service 1
metadata:
  labels:
    app: hello.openshift
  name: hello.openshift 2
spec:
  ports: 3
  - name: 8080-tcp
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector: 4
    app: hello.openshift
    deploymentconfig: hello.openshift

```

- 1** The kind of Kubernetes resource. In this case, a *Service*.
- 2** A unique name for the service.
- 3** **ports** is an array of objects that describes network ports exposed by the service. The **targetPort** attribute has to match a **containerPort** attribute from a pod container definition. Clients connect to the service **port** and the service forwards packets to the **targetPort** defined in the pod specification.
- 4** The service uses the selector attribute to find pods to forward packets to. The target pods need to have matching labels in their metadata. If the service finds multiple pods with matching labels, it load balances network connections among them.

Getting Traffic into and out of the Cluster

By default, pod and service IP addresses are not reachable from outside the OpenShift cluster. For applications that need access to the service from outside the OpenShift cluster, three methods exist:

- **HostPort/HostNetwork:** In this approach, clients can reach application pods in the cluster directly via the network ports on the host. Ports in the application pod are bound to ports on the host where they are running. This approach requires escalated privileges to run, and there is a risk of port conflicts when there are a large number of pods running in the cluster. This approach is outside the scope of this course.
- **NodePort:** This is an older Kubernetes-based approach, where the service is exposed to external clients by binding to available ports on the node host, which then proxies connections to the service IP address. Use the **oc edit svc** command to edit service attributes, specify **NodePort** as the type, and provide a port value for the **nodePort** attribute. OpenShift then proxies connections to the service via the public IP address of the node host and the port value set in **nodePort**. This approach supports non-HTTP traffic.
- **OpenShift routes:** This is the preferred approach in OpenShift. It exposes services using a unique URL. Use the **oc expose** command to expose a service for external access, or expose a service from the OpenShift web console. OpenShift routes are discussed in more detail in the next section. In this approach, only HTTP, HTTPS, TLS with SNI, and WebSockets are currently supported.

The following figure shows how **NodePort** services allow external access to Kubernetes services.

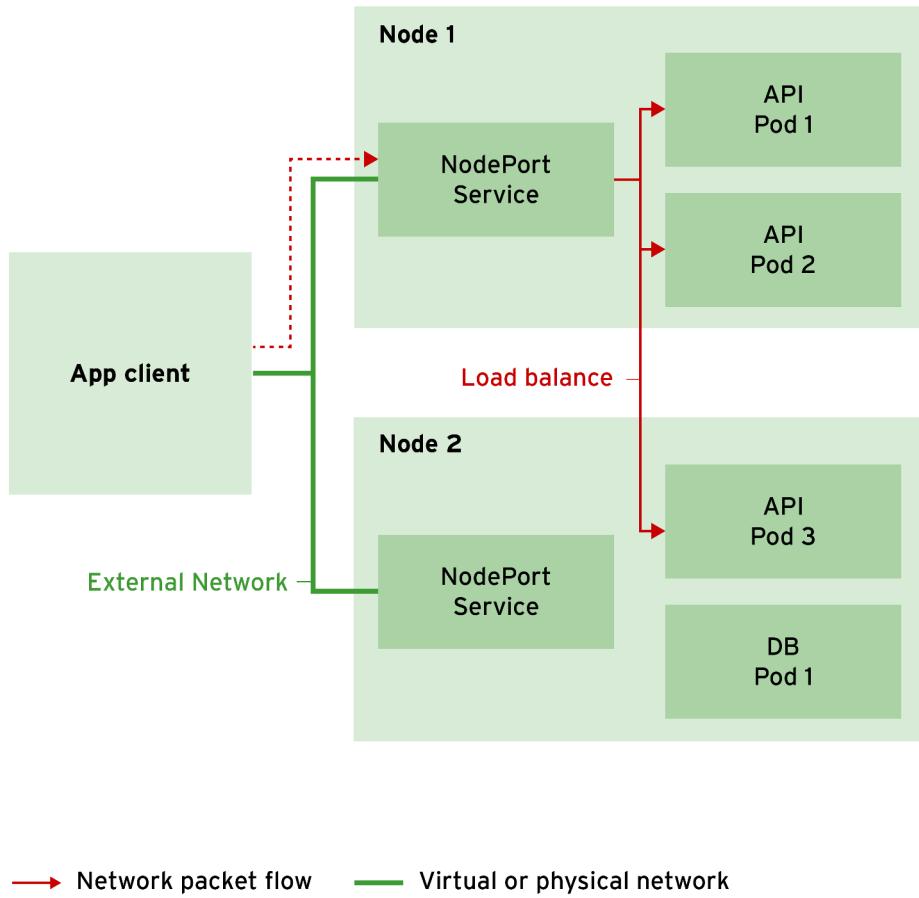


Figure 3.3: Kubernetes NodePort services

The following listing shows a NodePort definition in YAML syntax:

```

apiVersion: v1
kind: Service
metadata:
...
spec:
  ports:
    - name: 3306-tcp
      port: 3306
      protocol: TCP
      targetPort: 3306 1
      nodePort: 30306 2
  selector:
    app: mysqlDb
    deploymentconfig: mysqlDb
    sessionAffinity: None
  type: NodePort 3
...

```

- ➊ The port on which the pod listens for incoming requests. This matches the port number in the pod specification.
- ➋ The port on the host machines in the OpenShift cluster through which external clients communicate.
- ➌ The type of service. In this case, it is set to **NodePort**.

OpenShift binds the service to the value defined in the **nodePort** attribute of the service definition, and this port is opened for traffic on all nodes in the cluster. External clients can connect to any of the node's public IP addresses on the **nodePort** to access the service. The requests are round-robin load balanced among the pods behind the service. OpenShift routes are mostly restricted to HTTP and HTTPS traffic, but node ports can handle non-HTTP traffic, because the system administrator chooses the port to expose, and the clients can connect to this port using a protocol such as TCP or UDP.



Note

Port numbers for **NodePort** attributes are restricted to the range 30000-32767 by default. This range is configurable in the OpenShift master configuration file.



Note

The node port is open on *all* the nodes in the cluster, including the master. If the node port value is not provided, OpenShift assigns a random port in the configured range automatically.

Accessing External Networks

Pods can communicate with external networks using the host address it resides on. As long as the host can resolve the server that the pod wants to talk to, the pods can communicate with the target server using network address translation (NAT).



References

Additional information about services is available in the *OpenShift SDN* section of the *OpenShift Container Platform Architecture* document at

https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Exploring Software Defined Networking

In this exercise, you will deploy multiple pods of an application and understand OpenShift's software-defined networking (SDN).

Resources	
Files:	/home/student/D0280/labs/openshift-network/

Outcomes

You should be able to deploy multiple replicas of an application pod and access them:

- Directly via their pod IP addresses (from within the cluster)
- Using the OpenShift service IP address (from within the cluster)
- From external clients using node ports (from outside the cluster)

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands on the **workstation** host to ensure that the environment is set up correctly:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab openshift-network setup
```

Steps

1. Create a new project.

- 1.1. From the **workstation** VM, access the OpenShift master (**https://master.lab.example.com:8443**) with the OpenShift client.

Log in as **developer** and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443
```

- 1.2. Create the **network-test** project:

```
[student@workstation ~]$ oc new-project network-test
```

2. Deploy multiple pods of a test application.

2.1. Deploy the **scaling** application from the classroom private registry.

The application runs on port 8080 and displays the IP address of the host (in our environment, the pod IP address) running the application:

```
[student@workstation ~]$ oc new-app --name=hello -i php:7.0 \
http://workstation.lab.example.com/scaling
```

2.2. Run the following command to verify that the application pod is ready and running. It will take some time to build and deploy the pods.

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

The output from the command should be similar to the following:

NAME	READY	STATUS	RESTARTS	AGE
hello-1-build	0/1	Completed	0	30s
hello-1-nvfgd	1/1	Running	0	23s

2.3. Scale the application to two pods:

```
[student@workstation ~]$ oc scale --replicas=2 dc hello
deploymentconfig "hello" scaled
```

2.4. You should now see two pods running, typically one pod on each node:

NAME	..	STATUS	IP	NODE
hello-1-4bb1t	..	Running	10.129.0.27	node1.lab.example.com
hello-1-nvfgd	..	Running	10.130.0.13	node2.lab.example.com



Note

The IP addresses for the pods may be different on your system.

3. Verify that the application is accessible using the individual pod IP addresses.

3.1. Launch two new terminals on **workstation** and connect to **node1** and **node2** using **ssh**:

```
[student@workstation ~]$ ssh root@node1
```

```
[student@workstation ~]$ ssh root@node2
```

3.2. Verify that the application is *not* accessible from **workstation**, using the IP addresses listed in the previous step:

```
[student@workstation ~]$ curl http://10.129.0.27:8080
curl: (7) Failed connect to 10.129.0.27:8080; Network is unreachable

[student@workstation ~]$ curl http://10.130.0.13:8080
curl: (7) Failed connect to 10.130.0.13:8080; Network is unreachable
```

Pod IP addresses are not reachable from outside the cluster.

- 3.3. Verify that the application is accessible on **node1** and **node2**, using the respective IP addresses shown in the previous step:

```
[root@node1 ~]# curl http://10.129.0.27:8080
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<br/> Server IP: 10.129.0.27
</body>
</html>
```

```
[root@node2 ~]# curl http://10.130.0.13:8080
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<br/> Server IP: 10.130.0.13
</body>
</html>
```

4. Verify that the application is accessible using the service IP address (also known as the cluster IP):

- 4.1. Identify the service IP address using the **oc get svc** command:

```
[student@workstation ~]$ oc get svc hello
NAME      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
hello     172.30.105.51    <none>          8080/TCP    25m
```



Note

The cluster IP address may be different on your system.

- 4.2. Verify that the application is *not* accessible from **workstation** using the cluster IP address:

```
[student@workstation ~]$ curl http://172.30.105.51:8080
curl: (7) Failed connect to 172.30.105.51:8080; Network is unreachable
```

The cluster IP address is also not reachable from outside the cluster.

-
- 4.3. Verify that the application is accessible from either **master**, **node1**, or **node2** using the cluster IP address:

```
[root@node1 ~]# curl http://172.30.105.51:8080
...
Server IP: 10.129.0.27
...
```

- 4.4. Send more HTTP requests to the cluster IP URL, and observe how the requests are load balanced and routed to the two pods in a round-robin manner:

```
[root@node1 ~]# curl http://172.30.105.51:8080
...
<br/> Server IP: 10.130.0.13
...
```

- 4.5. Inspect the service from the application.

Describe the details of the **hello** service using the **oc describe svc** command:

```
[student@workstation ~]$ oc describe svc hello
Name: hello
Namespace: network-test
Labels: app=hello
Selector: app=hello,deploymentconfig=hello
Type: ClusterIP
IP: 172.30.105.51
Port: 8080-tcp 8080/TCP
Endpoints: 10.129.0.27:8080,10.130.0.13:8080
Session Affinity: None
No events.
```

The **Endpoints** attribute shows a list of pod IP addresses that the requests are routed to. These endpoints are automatically updated when pods are killed or when new pods are created.

OpenShift knows which pods it has to load balance for a given cluster IP address using *selectors* and *labels* defined for the pods. OpenShift routes requests for this service to all pods labeled **app=hello** and **deploymentconfig=hello**

Describe the details of one of the pods to verify:

```
[student@workstation ~]$ oc describe pod hello-1-4bb1t
...
Labels: app=hello
deployment=hello-1
deploymentconfig=hello
...
```

5. Enable access to the application from outside the cluster.

Edit the service configuration for the application and change the service type to *NodePort*.

- 5.1. Edit the service configuration for the application using the **oc edit svc** command:

```
[student@workstation ~]$ oc edit svc hello
```

This command opens a **vi** editor buffer which shows the service configuration in yaml format. Change the **type** of the service to **NodePort**, and add a new attribute called **nodePort** to the **ports** array with a value of 30800 for the attribute:

```
apiVersion: v1
kind: Service
metadata:
...
spec:
  clusterIP: 172.30.105.51
  ports:
    - name: 8080-tcp
      port: 8080
      protocol: TCP
      targetPort: 8080
      nodePort: 30800
  selector:
    app: hello
    deploymentconfig: hello
  sessionAffinity: None
  type: NodePort
status:
...
```

Type **:wq** to save the contents of the buffer and exit the editor.

5.2. Verify your changes by running the **oc describe svc** command again:

```
[student@workstation ~]$ oc describe svc hello
Name: hello
Namespace: network-test
Labels: app=hello
Selector: app=hello,deploymentconfig=hello
Type: NodePort
IP: 172.30.105.51
Port: 8080-tcp 8080/TCP
NodePort: 8080-tcp 30800/TCP
Endpoints: 10.129.0.27:8080,10.130.0.13:8080
Session Affinity: None
No events.
```

Access the application using the NodePort IP address from the **workstation** VM:

```
[student@workstation ~]$ curl http://node1.lab.example.com:30800
...
Server IP: 10.129.0.27
...

[student@workstation ~]$ curl http://node2.lab.example.com:30800
...
Server IP: 10.130.0.13
...
```

The application is accessible from outside the cluster and the requests are still load balanced between the pods.

6. You have seen how traffic is routed to pods from external clients. To verify the outgoing traffic, that is, from the pods to the outside world, you can use the **oc rsh** command to open a shell inside the pod as described below.

- 6.1. Use the **oc rsh** command to access the shell inside a pod:

```
[student@workstation ~]$ oc rsh hello-1-4bb1t
```

- 6.2. Access machines that are outside the OpenShift cluster from the pod. For example, access the classroom materials server and fetch the Yum repository file used in the classroom setup:

```
sh-4.2$ curl http://materials.example.com/training.repo
[rhel-7-server-optional-rpms]
baseurl = http://content.example.com/ocp3.5/x86_64/rhelopt
enabled = true
gpgcheck = false
name = Remote classroom copy of Optional RHEL RPMS
...
```

You can also access the Git repository hosted on the **workstation** VM to verify that pods can reach it:

```
sh-4.2$ curl http://workstation.lab.example.com
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US" lang="en-US">
<!-- git web interface version 1.8.3.1, (C) 2005-2006, Kay Sievers
  &lt;kay.sievers@vrfy.org&gt;, Christian Gierke --&gt;
<!-- git core binaries version 1.8.3.1 --&gt;
...</pre>
```

- 6.3. Type **exit** to exit the pod shell and return to the workstation prompt:

```
sh-4.2$ exit
[student@workstation ~]$
```

7. Clean up. Delete the **network-test** project.

```
[student@workstation ~]$ oc delete project network-test
project "network-test" deleted
```

This concludes the guided exercise.

Creating Routes

Objective

After completing this section, students should be able to describe how OpenShift routing works and create a route.

The OpenShift Router

While OpenShift services allow for network access between pods inside an OpenShift instance, OpenShift routes allow for network access to pods from outside the OpenShift instance.

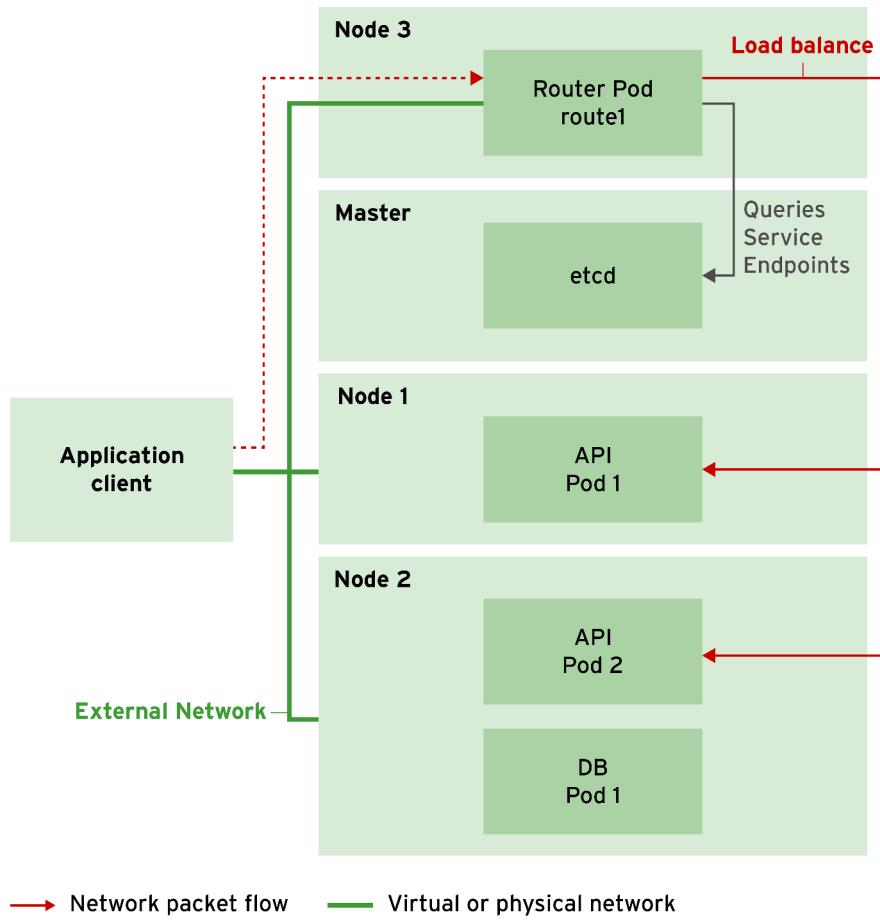


Figure 3.4: OpenShift routes

A route connects a public-facing IP address and DNS host name to an internal-facing service IP address. At least, this is the concept. In practice, to improve performance and reduce latency, the OpenShift router connects directly to the pods over the networks created by OpenShift Container Platform, using the service only to find the endpoints; that is, the pods exposed by the service.

OpenShift routes are implemented by a shared router service, which runs as a pod inside the OpenShift instance, and can be scaled and replicated like any other regular pod. This router service is based on the open source software *HAProxy*.

An important consideration for OpenShift administrators is that the public DNS host names configured for routes need to point to the public-facing IP addresses of the nodes running the router. Router pods, unlike regular application pods, bind to their nodes' public IP addresses, instead of to the internal pod network. This is typically configured using a DNS wildcard.

The following listing shows a minimal route defined using YAML syntax:

```
- apiVersion: v1
  kind: Route ①
  metadata:
    creationTimestamp: null
    labels:
      app: quoteapp
    name: quoteapp ②
  spec:
    host: quoteapp.cloudapps.lab.example.com ③
    port:
      targetPort: 8080-tcp
    to: ④
      kind: Service
      name: quoteapp
```

- ①** The kind of Kubernetes resource. In this case, a **Route**.
- ②** A unique name for the route.
- ③** The fully qualified domain name (FQDN) associated with the route. This must be preconfigured to resolve to the IP address of the node where the OpenShift router pod is running.
- ④** An object that states the **kind** of resource this route points to, which in this case is an OpenShift Service, and the **name** of that resource, which is **quoteapp**.



Note

The names of different resource types do not collide. It is perfectly legal to have a route named **quoteapp** that points to a service also named **quoteapp**.



Important

Unlike services, which use selectors to link to pod resources containing specific labels, routes link directly to the service resource name.

Creating Routes

The easiest and preferred way to create a route is to use the **oc expose** command, passing a service resource name as the input. The **--name** option can be used to control the name of the route resource, and the **--hostname** option can be used to provide a custom host name for the route. For example:

```
[user@demo ~]$ oc expose service quote --name quote --
  hostname=quoteapp.cloudapps.lab.example.com
```

Routes created from templates or by the **oc expose** command without an explicit **--hostname** option generate DNS names of the following form:

<route-name>-<project-name>.<default-domain>

Where:

- *route-name* is the name explicitly assigned to the route, or the name of the originating resource (template for **oc new-app** and service for **oc expose** or from the **--name** option).
- *project-name* is the name of the project containing the resource.
- *default-domain* is configured on the OpenShift master and corresponds to the wildcard DNS domain listed as a prerequisite for installing OpenShift.

For example, creating a route called **quote** in a project called **test** in an OpenShift cluster where the subdomain is **cloudapps.example.com** results in the FQDN **quote-test.cloudapps.example.com**.



Note

The DNS server that hosts the wildcard domain is unaware of any route host names; it only resolves any name to the configured IPs. Only the OpenShift router knows about route host names, treating each one as an HTTP virtual host. Invalid wildcard domain host names, that is, host names that do not correspond to any route, are blocked by the OpenShift router and result in an HTTP 404 error.

Route resources can also be created like any other OpenShift resource by providing **oc create** with a JSON or YAML resource definition file.

The **oc new-app** command does not create a route resource when building a pod from container images, Dockerfiles, or application source code. The **oc new-app** command does not know if the pod is intended to be accessible from outside the OpenShift instance or not. When the **oc new-app** command creates a group of pods from a template, nothing prevents the template from including a route resource as part of the application. The same is true for the web console.

Finding the Default Routing Subdomain

The default routing subdomain is defined in the **routingConfig** section of the OpenShift configuration file, **master-config.yaml**, with the keyword **subdomain**. For example:

```
routingConfig:  
  subdomain: cloudapps.example.com
```

The OpenShift HAProxy router binds to host ports 80 (HTTP) and 443 (HTTPS), by default. The router must be placed on nodes where these ports are not otherwise in use. Alternatively, a router can be configured to listen on other ports by setting the **ROUTER_SERVICE_HTTP_PORT** and **ROUTER_SERVICE_HTTPS_PORT** environment variables in the router deployment configuration. Routers support the following protocols:

- HTTP
- HTTPS with SNI

- WebSockets
- TLS with SNI

Routing Options and Types

Routes can be either secured or unsecured. Secure routes provide the ability to use several types of TLS termination to serve certificates to the client. Unsecured routes are the simplest to configure, because they require no key or certificates, but secured routes encrypt traffic to and from the pods.

A secured route specifies the TLS termination of the route. The available types of termination are listed below:

Edge Termination

With edge termination, TLS termination occurs at the router, before the traffic gets routed to the pods. TLS certificates are served by the router, so they must be configured into the route, otherwise the router's default certificate is used for TLS termination. Because TLS is terminated at the router, connections from the router to the endpoints over the internal network are not encrypted.

Pass-through Termination

With pass-through termination, encrypted traffic is sent straight to the destination pod without the router providing TLS termination. No key or certificate is required. The destination pod is responsible for serving certificates for the traffic at the endpoint. This is currently the only method that can support requiring client certificates (also known as two-way authentication).

Re-encryption Termination

Re-encryption is a variation on edge termination, where the router terminates TLS with a certificate, then re-encrypts its connection to the endpoint, which might have a different certificate. Therefore the full path of the connection is encrypted, even over the internal network. The router uses health checks to determine the authenticity of the host.

Creating Secure Routes

Before creating a secure route, you need to generate a TLS certificate. For creating a simple self-signed certificate for a route called **test.example.com**, perform the following steps:

1. Create a private key using the **openssl** command:

```
[user@demo ~]$ openssl genrsa -out example.key 2048
```

2. Create a certificate signing request (CSR) using the generated private key:

```
[user@demo ~]$ openssl req -new -key example.key -out example.csr \
-subj "/C=US/ST=CA/L=Los Angeles/O=Example/OU=IT/CN=test.example.com"
```

3. Generate a certificate using the key and CSR:

```
[user@demo ~]$ openssl x509 -req -days 366 -in example.csr \
-signkey example.key -out example.crt
```

- When the certificate is ready, create an edge-terminated route:

```
[user@demo ~]$ oc create route edge --service=test \
--hostname=test.example.com \
--key=example.key --cert=example.crt
```

You can now access the secured route using <https://test.example.com>.

Wildcard Routes for Subdomains

A *wildcard policy* allows a user to define a route that covers all hosts within a domain. A route can specify a wildcard policy as part of its configuration using the **wildcardPolicy** field. The OpenShift router has support for wildcard routes, which are enabled by setting the **ROUTER_ALLOW_WILDCARD_ROUTES** environment variable in the deployment configuration of the router to **true**. Any routes with the **wildcardPolicy** attribute set to **Subdomain** are serviced by the router. The router exposes the associated service (for the route) according the route's wildcard policy.

For example, for three different routes, **a.lab.example.com**, **b.lab.example.com**, and **c.lab.example.com**, that should be routed to an OpenShift service called **test**, you can configure a route with a wildcard policy as follows:

- Configure the router to handle wildcard routes as the cluster administrative user:

```
[user@demo ~]$ oc scale dc/router --replicas=0
[user@demo ~]$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
[user@demo ~]$ oc scale dc/router --replicas=1
```

- Create a new route with a wildcard policy:

```
[user@demo ~]$ oc expose svc test --wildcard-policy=Subdomain --
hostname='www.lab.example.com'
```

Monitoring Routes

The OpenShift HAProxy router provides a statistics page where router metrics and route information are displayed. The system administrator needs to perform some extra steps to make this statistics page visible to clients. The following steps describe how to access the router's statistics page:

- On the **master** host, ensure you are using the **default** project and then find the router name:

```
[root@master]# oc project default
[root@master]# oc get pods
NAME           READY   STATUS    RESTARTS   AGE
docker-registry-6-kwv2i   1/1     Running   4          7d
registry-console-1-zlrry  1/1     Running   4          7d
router-1-32toa        1/1     Running   4          7d
```

- On the **master** host, inspect the router environment variables to find connection parameters for the HAProxy process running inside the pod:

```
[root@master]# oc env pod router-1-32toa --list | tail -n 6
```

```
ROUTER_SERVICE_NAME=router
ROUTER_SERVICE_NAMESPACE=default
ROUTER_SUBDOMAIN=
STATS_PASSWORD=shRxnWSdn9
STATS_PORT=1936
STATS_USERNAME=admin
```



Note

The password in the **STATUS_PASSWORD** variable was randomly generated when you create the router. The **STATUS_USERNAME** and **STATUS_PORT** variables have fixed default values, but all of them can be changed at router creation time.

3. On the node where the router is running, configure **iptables** to open the port specified by the **STATUS_PORT** variable:

```
[root@node ~]# iptables -I OS_FIREWALL_ALLOW -p tcp -m tcp \
--dport 1936 -j ACCEPT
```



Note

This change is persistent only if the system administrator adds this firewall rule to **/etc/sysconfig/iptables**.

4. Open a web browser and access the HAProxy statistics URL
http://nodeIP:STATUS_PORT/.

Type the value from **STATUS_USERNAME** into the User Name field and from **STATUS_PASSWORD** into the Password field, and click OK. You should see the HAProxy metrics page displayed.



References

Additional information about the architecture of routes in OpenShift is available in the *Routes* section of the OpenShift Container Platform *Architecture* documentation:

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Additional developer information about routes is available in the *Routes* section of the OpenShift *Developer Guide*:

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Creating a Route

In this exercise, you will create a secure edge-terminated route for an application deployed on OpenShift.

Resources	
Files:	/home/student/D0280/labs/secure-route
Application URL:	https://hello.cloudapps.lab.example.com

Outcomes

You should be able to create a secure edge-terminated route for an application deployed on OpenShift.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run following commands from the **workstation** host to ensure the environment is set up correctly:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab secure-route setup
```

Steps

1. Create a new project.

- 1.1. From the **workstation** VM, access the OpenShift master (<https://master.lab.example.com:8443>) with the OpenShift client.

Log in as **developer** and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443
```

- 1.2. Create the **secure-route** project:

```
[student@workstation ~]$ oc new-project secure-route
```

2. Deploy a test application.

- 2.1. Deploy the **hello-openshift** application from the classroom private registry. The application runs on port 8080 and displays a simple text message. There should be no spaces after the **--docker-image=**\ option in the following command. The command

is available at **/home/student/D0280/labs/secure-route/commands.txt** to minimize typing errors.

```
[student@workstation ~]$ oc new-app --docker-image=\
workstation.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry --name=hello
```

- 2.2. Run the following command to verify that the application pod is ready and running. It will take some time to deploy the pods.

```
[student@workstation ~]$ oc get pods -o wide
NAME          READY   STATUS    ...   IP           NODE
hello-1-qmnbn 1/1     Running   ...  10.130.0.11  node1.lab.example.com
```

Make note of the IP address and the node FQDN for the **hello** pod. The name and IP address of the pod on your system might be different. You will need this IP to test the application later in the lab.

3. Create a self-signed TLS certificate for securing the route.
 - 3.1. Briefly review the commands in the **create-cert.sh** file in the **/home/student/D0280/labs/secure-route** folder:

```
[student@workstation ~]$ cat \
/home/student/D0280/labs/secure-route/create-cert.sh
echo "Generating a private key..."
openssl genrsa -out hello.cloudapps.lab.example.com.key 2048
...
echo "Generating a CSR..."
openssl req -new -key hello.cloudapps.lab.example.com.key \
-out hello.cloudapps.lab.example.com.csr \
-subj "/C=US/ST=NC/L=Raleigh/O=RedHat/OU=RHT/CN=hello.cloudapps.lab.example.com"
...
echo "Generating a certificate..."
openssl x509 -req -days 366 -in \
hello.cloudapps.lab.example.com.csr -signkey \
hello.cloudapps.lab.example.com.key \
-out hello.cloudapps.lab.example.com.crt
...
```

The script creates a self-signed TLS certificate that is valid for 366 days.

- 3.2. Run the **create-cert.sh** script:

```
[student@workstation ~]$ cd /home/student/D0280/labs/secure-route
[student@workstation secure-route]$ ./create-cert.sh
Generating a private key...
Generating RSA private key, 2048 bit long modulus
.....+
.....+
e is 65537 (0x10001)

Generating a CSR...
```

```
Generating a certificate...
Signature ok
subject=/C=US/ST=NC/L=Raleigh/O=RedHat/OU=RHT/CN=hello.cloudapps.lab.example.com
Getting Private key

DONE.
```

Verify that three files are created in the same folder:
hello.cloudapps.lab.example.com.crt,
hello.cloudapps.lab.example.com.csr, and
hello.cloudapps.lab.example.com.key.

4. Create a secure edge-terminated route using the generated TLS certificate and key.

- 4.1. Create a new secure edge-terminated route with the files generated in the previous step.

From the terminal window, run the following command. The command is available at **/home/student/D0280/labs/secure-route/commands.txt** file to minimize typing errors.

```
[student@workstation secure-route]$ oc create route edge \
--service=hello --hostname=hello.cloudapps.lab.example.com \
--key=hello.cloudapps.lab.example.com.key \
--cert=hello.cloudapps.lab.example.com.crt
route "hello" created
```

- 4.2. Verify that the route is created:

```
[student@workstation secure-route]$ oc get routes
NAME      HOST/PORT          SERVICES   PORT      TERMINATION ..
hello    hello.cloudapps.lab.example.com  hello     8080-tcp  edge ..
```

- 4.3. Inspect the route configuration in YAML format:

```
[student@workstation secure-route]$ oc get route/hello -o yaml
apiVersion: v1
kind: Route
metadata:
...
spec:
  host: hello.cloudapps.lab.example.com
  port:
    targetPort: 8080-tcp
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      MIIDZj...
      -----END CERTIFICATE-----
    key: |
      -----BEGIN RSA PRIVATE KEY-----
      MIIEpQ...
      -----END RSA PRIVATE KEY-----
  termination: edge
  to:
    kind: Service
    name: hello
```

```
    weight: 100
    wildcardPolicy: None
status:
...
...
```

5. Test the route.

5.1. Verify that the **hello** service is not accessible using the HTTP URL of the route:

```
[student@workstation secure-route]$ curl http://hello.cloudapps.lab.example.com
...
<h1>Application is not available</h1>
<p>The application is currently not serving requests at this endpoint. It
may not have been started or is still starting.</p>
...
```

The generic router home page is displayed, which indicates that the request has not been forwarded to any of the pods.

5.2. Verify that the **hello** service is accessible using the secure HTTPS URL of the route:

```
[student@workstation secure-route]$ curl -k -vvv \
  https://hello.cloudapps.lab.example.com
* About to connect() to hello.cloudapps.lab.example.com port 443 (#0)
*   Trying 172.25.250.11...
* Connected to hello.cloudapps.lab.example.com (172.25.250.11) port 443 (#0)
...
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*   subject:
CN=hello.cloudapps.lab.example.com,OU=RHT,O=RedHat,L=Raleigh,ST=NC,C=US
*   start date: Jun 29 07:02:24 2017 GMT
*   expire date: Jun 30 07:02:24 2018 GMT
*   common name: hello.cloudapps.lab.example.com
*   issuer:
CN=hello.cloudapps.lab.example.com,OU=RHT,O=RedHat,L=Raleigh,ST=NC,C=US
...
Hello OpenShift!
...
```

5.3. Because the encrypted traffic is terminated at the router, and the request is forwarded to the pods using unsecured HTTP, you can access the application over plain HTTP using the pod IP address. (Use the IP address you noted from the **oc get pods -o wide** command).

Open a new terminal on the **workstation** VM and log in to **node1** using SSH, and run the following command:

```
[student@workstation secure-route]$ ssh root@node1
[root@node1 ~]# curl -vvv http://10.130.0.11:8080
* About to connect() to 10.130.0.11 port 8080 (#0)
*   Trying 10.130.0.11...
* Connected to 10.130.0.11 (10.130.0.11) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 10.130.0.11:8080
> Accept: */*
```

```
...  
Hello OpenShift!  
...
```

6. Clean up.

Delete the **secure-route** project:

```
[student@workstation secure-route]$ oc delete project secure-route  
project "secure-route" deleted
```

This concludes the guided exercise.

Lab: Exploring OpenShift Networking Concepts

In this lab, you will troubleshoot and fix issues related to accessing an application using an OpenShift route.

Resources	
Application URL:	<code>http://hello.cloudapps.lab.example.com</code>

Outcomes

You should be able to troubleshoot and fix errors related to accessing an application using a route.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands from the **workstation** host to ensure that the environment is set up correctly:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab network-review setup
```

Steps

1. The lab setup script creates a new project called **network-review** using the **developer** user account, and deploys an application called **hello-openshift**. Log in to OpenShift as the **developer** user, and list the projects. Ensure that the **network-review** project is the default project for this user.
2. Inspect the resources in this project.
 - 2.1. Inspect the pods in this project and make a note of the pod IP address.
 - 2.2. Inspect the services in this project and make a note of the cluster IP address.
 - 2.3. Inspect the routes in this project and make a note of the route URL.
3. Access the application with the **curl** command to invoke the route URL. The default router home page is displayed and not the actual application output.
4. Investigate and troubleshoot why the route invocation failed.
 - 4.1. Use the **curl** command to directly invoke the pod IP address. Verify that you get a valid response from the application.

- 4.2. Use the **curl** command to invoke the cluster IP address. The application does not return a valid response. This means that there is something wrong in the service configuration.
 - 4.3. View details about the service using the **oc describe svc** command. Check the endpoints registered for this service.
 - 4.4. View details about the selector labels for the application pod using the **oc describe pod** command. Verify that the selector labels for the pod and the service match, to ensure that the pod is registered as an endpoint for the service.
 - 4.5. Edit the service configuration and fix the error.
 - 4.6. Verify that you can now see valid output from the application when you invoke the cluster IP address.
 - 4.7. Determine whether or not the route URL invocation from the **workstation** VM works. The route URL invocation still fails.
 - 4.8. View details about the route using the **oc describe route** command. Check the service name and the endpoints registered for this route.
 - 4.9. Edit the route configuration and fix the error.
 - 4.10. Verify that you can now see valid output from the application when you invoke the route URL.
5. Evaluation
Run the following command to grade your work:
- [student@workstation ~]\$ lab network-review grade
- If you do not get a PASS grade, review your work and run the grading command again.
6. Clean up
Delete the **network-review** project.
- This concludes the lab.

Solution

In this lab, you will troubleshoot and fix issues related to accessing an application using an OpenShift route.

Resources	
Application URL:	http://hello.cloudapps.lab.example.com

Outcomes

You should be able to troubleshoot and fix errors related to accessing an application using a route.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands from the **workstation** host to ensure that the environment is set up correctly:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab network-review setup
```

Steps

1. The lab setup script creates a new project called **network-review** using the **developer** user account, and deploys an application called **hello-openshift**. Log in to OpenShift as the **developer** user, and list the projects. Ensure that the **network-review** project is the default project for this user.

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

```
[student@workstation ~]$ oc projects
[student@workstation ~]$ oc project network-review
```

2. Inspect the resources in this project.

- 2.1. Inspect the pods in this project and make a note of the pod IP address.

```
[student@workstation ~]$ oc get pods -o wide
NAME          ..  IP           NODE
hello-openshift-1-2lxcg  ..  10.130.0.16  node1.lab.example.com
```

- 2.2. Inspect the services in this project and make a note of the cluster IP address.

```
[student@workstation ~]$ oc get svc
NAME      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
```

```
hello-openshift  172.30.162.216  <none>        8080/TCP, 8888/TCP  3m
```

- 2.3. Inspect the routes in this project and make a note of the route URL.

```
[student@workstation ~]$ oc get routes
NAME          HOST/PORT
hello-openshift  hello.cloudapps.lab.example.com  ..
```

3. Access the application with the **curl** command to invoke the route URL. The default router home page is displayed and not the actual application output.

```
[student@workstation ~]$ curl http://hello.cloudapps.lab.example.com
...
<h1>Application is not available</h1>
<p>The application is currently not serving requests at this endpoint...
...
```

4. Investigate and troubleshoot why the route invocation failed.

- 4.1. Use the **curl** command to directly invoke the pod IP address. Verify that you get a valid response from the application.

Open a new terminal and open an SSH session to the **master** VM as the **root** user before running the **curl** command.

```
[student@workstation ~]$ ssh root@master
[root@master ~]# curl http://10.130.0.16:8080
Hello OpenShift!
```

- 4.2. Use the **curl** command to invoke the cluster IP address. The application does not return a valid response. This means that there is something wrong in the service configuration.

```
[root@master ~]# curl http://172.30.162.216:8080
curl: (7) Failed connect to 172.30.162.216:8080; Connection refused
```

- 4.3. View details about the service using the **oc describe svc** command. Check the endpoints registered for this service.

```
[root@master ~]# oc describe svc hello-openshift \
-n network-review
Name: hello-openshift
Namespace: network-review
Labels: app=hello-openshift
Selector: app=hello.openshift,deploymentconfig=hello-openshift
Type: ClusterIP
IP: 172.30.162.216
Port: 8080-tcp 8080/TCP
Endpoints: <none>
Port: 8888-tcp 8888/TCP
Endpoints: <none>
Session Affinity: None
```

Observe that there are no endpoints for this service. This is the reason requests to the service IP returned a connection refused error. Recall that endpoints are registered based on the selector labels for the pods. Note the selector label for this service.

- 4.4. View details about the selector labels for the application pod using the **oc describe pod** command. Verify that the selector labels for the pod and the service match, to ensure that the pod is registered as an endpoint for the service.

```
[root@master ~]# oc describe pod hello-openshift-1-2lxcg \
-n network-review
Name: hello-openshift-1-2lxcg
Namespace: network-review
Security Policy: restricted
Node: node1.lab.example.com/172.25.250.11
...
Labels: app=hello-openshift
        deployment=hello-openshift-1
        deploymentconfig=hello-openshift
Status: Running
IP: 10.130.0.16
Controllers: ReplicationController/hello-openshift-1
Containers:
...
Events:
...
```

Observe that the selector label on the pod is **app=hello-openshift**, whereas the selector label on the service is **app=hello_openshift**. You must edit the service configuration and change the selector to match the pod selector.

- 4.5. Edit the service configuration and fix the error.

Switch to the terminal on **workstation**. Edit the service configuration using the **oc edit svc** command on the **workstation** VM:

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

The previous command displays the service configuration in a **vi** editor buffer. Observe that the **app** child attribute under the **selector** element in the service configuration is mistyped as **hello_openshift**. You are not getting a response from the application because there are no pods labeled **hello_openshift**. Change the **app** attribute to **hello-openshift**, which is a label on the **hello-openshift** pod. Save the file when you are finished. Run the **oc describe svc hello-openshift** command again. The selector and endpoints should now be displayed as follows:

```
...
Selector: app=hello-openshift,deploymentconfig=hello-openshift
...
Endpoints: 10.130.0.16:8080
```

- 4.6. Verify that you can now see valid output from the application when you invoke the cluster IP address.

```
[root@master ~]# curl http://172.30.162.216:8080
```

```
Hello OpenShift!
```

- 4.7. Determine whether or not the route URL invocation from the **workstation** VM works.
The route URL invocation still fails.

```
[student@workstation ~]$ curl http://hello.cloudapps.lab.example.com
...
<h1>Application is not available</h1>
<p>The application is currently not serving requests at this endpoint...
...
```

- 4.8. View details about the route using the **oc describe route** command. Check the service name and the endpoints registered for this route.

```
[root@master ~]# oc describe route hello-openshift
Name: hello-openshift
Namespace: network-review
Created: 41 minutes ago
Labels: app=hello-openshift
...
Service: hello-opensift
Weight: 100 (100%)
Endpoints: <error: endpoints "hello-opensift" not found>
```

Observe the error about no endpoints for this route. This is the reason requests to the route URL did not show the application home page. Recall that the router queries the service for endpoints and registers valid endpoints for load-balancing. Note that there is a typo in the service name. It should be **hello-openshift**, which is correct the name of the service for the application.

- 4.9. Edit the route configuration and fix the error.

Edit the route configuration using the **oc edit route** command on the **workstation** VM:

```
[student@workstation ~]$ oc edit route hello-openshift
```

The previous command displays the route configuration in a vi editor buffer. Observe that the **to** child attribute under the **spec** element in the route configuration is mistyped as **hello-opensift**. You are not getting a response from the route because there are no services called **hello-opensift** in the project. Change the **to** attribute to **hello-openshift**. Save the file when you are finished. Run the **oc describe route hello-openshift** command again. The service name and endpoints should now be displayed as follows:

```
...
Service: hello-openshift
...
Endpoints: 10.130.0.16:8080, 10.130.0.16:8888
```

- 4.10. Verify that you can now see valid output from the application when you invoke the route URL.

```
[student@workstation ~]$ curl http://hello.cloudapps.lab.example.com  
Hello OpenShift!
```

5. Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab network-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

6. Clean up

Delete the **network-review** project.

```
[student@workstation ~]$ oc delete project network-review  
project "network-review" deleted
```

This concludes the lab.

Summary

In this chapter, you learned:

- The OpenShift Software-defined Networking (SDN) implementation is based on Open vSwitch (OVS), and how it provides a unified cluster network that enables communication between pods across the OpenShift cluster.
- An OpenShift Service:
 - Has a unique IP address for clients to connect to access pods in the cluster
 - Has an IP address also comes from the OpenShift SDN and it is distinct from the pod's internal network, but visible only from within the cluster.
 - Ensures that pods matching the selector are added to the service resource as an endpoint. As pods are created and killed, the endpoints behind a service are automatically updated.
- For applications that need access to the service from outside the OpenShift cluster, there are two ways to achieve this objective:
 - *NodePort*: The service is exposed to external clients by binding to available ports on the node host, which then proxies connections to the service IP address. Port numbers for node ports are restricted to the range 30000-32767.
 - OpenShift *routes*: This approach exposes services using a unique URL. Use the **oc expose** command to expose a service for external access, or expose a service from the OpenShift web console.
- Pods can communicate with servers outside the OpenShift cluster using the host address by means of network address translation (NAT). NAT transfers network traffic via the host IP address.
- OpenShift routes are implemented by a shared router service, which runs as a pod inside the OpenShift instance and can be scaled and replicated like any other regular pod. This router service is based on the open source software *HAProxy*.
- Route resources can be created like any other OpenShift resource by providing **oc create** with a JSON or YAML resource definition file, or by using the **oc expose** command.
- Routes created from templates or by the **oc expose** command without an explicit **--hostname** option generate DNS names of the form **<route-name>-<project-name>. <default-domain>**.
- Routes support the following protocols:
 - HTTP
 - HTTPS with SNI
 - WebSockets
 - TLS with SNI
- You can create different types of routes:

- *Edge Termination*: TLS termination occurs at the router, before the traffic gets routed to the pods. TLS certificates are served by the router, so they must be configured into the route.
 - *Pass-through Termination*: Encrypted traffic is sent straight to the destination pod without the router providing TLS termination. No key or certificate is required. The destination pod is responsible for serving certificates for the traffic at the endpoint.
 - *Re-encryption Termination*: Re-encryption is a variation of edge termination where the router terminates TLS with a certificate, then re-encrypts its connection to the endpoint, which might have a different certificate.
- A *wildcard policy* allows a user to define a route that covers all hosts within a domain. A route can specify a wildcard policy as part of its configuration using the **wildcardPolicy** field. The OpenShift router has support for wildcard routes, which are enabled by setting the **ROUTER_ALLOW_WILDCARD_ROUTES** environment variable to **true**.



CHAPTER 4

EXECUTING COMMANDS

Overview	
Goal	Execute commands using the command-line interface.
Objectives	<ul style="list-style-type: none">Configure OpenShift resources using the command-line interface.Execute commands that assist in troubleshooting common problems.
Sections	<ul style="list-style-type: none">Configuring Resources with the CLI (and Guided Exercise)Executing Troubleshooting Commands (and Guided Exercise)
Lab	Executing Commands

Configuring Resources with the CLI

Objective

After completing this section, students should be able to configure resources using the OpenShift command-line interface.

Accessing Resources from the Managed OpenShift Instance

OpenShift Container Platform organizes entities in the OpenShift cluster as objects managed by the master node. These are collectively known as *resources*. You encountered some of these resources in earlier chapters:

- nodes
- services
- pods
- projects
- deploymentConfigs
- users

These are just some of the different resources available to OpenShift users. Regardless of the resource that the administrator is managing, the OpenShift command-line tools provide a unified and consistent way to update, modify, delete, and query these resources.

OpenShift ships with a command-line tool that enables system administrators and developers to work with an OpenShift cluster. The **oc** command-line tool provides the ability to modify and manage resources throughout the delivery life cycle of a software development project. Common operations with this tool include deploying applications, scaling applications, checking the status of projects, and similar tasks.

Installing the oc Command-line Tool

During the OpenShift installation process, the **oc** command-line tool is installed on all master and node machines. You can also install the **oc** client on systems that are not part of the OpenShift cluster, such as developer machines. When it is installed, you can issue commands after authenticating against any master node with a user name and password.

There are several different methods available for installing the **oc** command-line tool, depending on which platform is used:

- On Red Hat Enterprise Linux (RHEL) systems with valid subscriptions, the tool is available as an RPM file and installable using the **yum install** command.

```
[user@host ~]$ sudo yum install -y atomic-openshift-clients
```

- For other Linux distributions and other operating systems, such as Windows and macOS, native clients are available for download from the Red Hat Customer Portal. This also

requires an active OpenShift subscription. These downloads are statically compiled to reduce incompatibility issues.

Useful Commands to Manage OpenShift Resources

After the **oc** CLI tool has been installed, you can use the **oc help** command to display help information. There are **oc** subcommands for tasks such as:

- Logging in to and out of an OpenShift cluster.
- Creating, changing, and deleting projects.
- Creating applications inside a project.
- Creating a deployment configuration or a build configuration from a container image, and all associated resources.
- Creating, deleting, inspecting, editing, and exporting individual resources, such as pods, services, and routes inside a project.
- Scaling applications.
- Starting new deployments and builds.
- Checking logs from application pods, deployments, and build operations.



Note

When you install the OpenShift client package, bash completion for the **oc** command is not enabled by default. You can either open a new terminal window to run the **oc** command, or source the **/etc/bash_completion.d/oc** file in the terminal window where you installed the package.

You can use the **oc login** command to log in interactively, which prompts you for a server name, a user name, and a password, or you can include the required information on the command line.

```
[student@workstation ~]$ oc login https://master.lab.example.com:8443 \
-u developer -p openshift
```



Note

Note that the backslash character (\) in the previous command is a command continuation character and should only be used if you are not entering the command as a single line.

After successful authentication from a client, OpenShift saves an authorization token in the user's home folder. This token is used for subsequent requests, negating the need to re-enter credentials or the full master URL.

To check your current credentials, run the **oc whoami** command:

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

This command displays the user name that you used when logging in.

```
developer
```

To create a new project, use the **oc new-project** command:

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

Use run the **oc status** command to verify the status of the project:

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

Initially, the output from the status command reads:

```
In project working on server https://master.lab.example.com:8443  
You have no services, deployment configs, or build configs.  
Run 'oc new-app' to create an application.
```

The output of the above command changes as you create new projects and add resources to those projects.

To delete a project, use the **oc delete project** command:

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

To log out of the OpenShift cluster, use the **oc logout** command:

```
[student@workstation ~]$ oc logout  
Logged "developer" out on "https://master.lab.example.com:8443"
```

It is possible to log in as the OpenShift cluster administrator from any master node without a password by using the **system:admin** argument for the **-u** option.

```
[root@master ~]# oc login -u system:admin
```

This gives you full privileges over all operations and resources in the OpenShift instance, and should be used with care.

Typically, as an administrator, the **oc get** command is likely the tool that is used most frequently. This allows users to get information about resources in the cluster. Generally, this command displays only the most important characteristics of the resources and omits more detailed information.

If the **RESOURCE_NAME** parameter is omitted, then all resources of the specified **RESOURCE_TYPE** are summarized. The following output is a sample execution of **oc get pods**:

NAME	READY	STATUS	RESTARTS	AGE
docker-registry-1-5r583	1/1	Running	0	1h
trainingrouter-1-144m7	1/1	Running	0	1h

oc get all

If the administrator wants a summary of all of the most important components of the cluster, the **oc get all** command can be executed. This command iterates through the major resource types and prints out a summary of their information. For example:

NAME	DOCKER_REPO	TAGS	UPDATED	
is/registry-console	172.30.211.204:5000	3.3	2 days ago	
NAME	REVISION	DESIRED	CURRENT	TRIGGERED_BY
dc/docker-registry	4	1	1	config
dc/docker-console	1	1	1	config
dc/router	4	1	1	config
NAME	DESIRED	CURRENT	READY	AGE
rc/docker-registry	-1	0	0	2d
rc/docker-registry	-2	0	0	2d
rc/docker-registry	-3	0	0	2d
rc/docker-registry	-4	1	1	2d
rc/docker-console	-1	1	1	2d
rc/docker-router	-1	0	0	2d
NAME	HOST/PORT	PATH	SERVICES	
PORT	TERMINATION			
routes/docker-registry	docker-registry-default.cloudapps.lab.example.com		docker-	
5000-tcp	passthrough		registry	
routes/registry-console	registry-console-default.cloudapps.lab.example.com			
registry-console	registry-console passthrough			
NAME	CLUSTER_IP	EXTERNAL_IP	PORT(S)	AGE
svc/docker-registry	172.30.211.204	<none>	5000/TCP	2d
svc/kubernetes	172.30.0.1	<none>	443/TCP, 53/UDP, 53/TCP	2d
svc/registry-console	172.30.190.103	<none>	9000/TCP	2d
svc/router	172.230.63.165	<none>	80/TCP, 443/TCP, 1936/TCP	2d
NAME	READY	STATUS	RESTARTS	AGE
po/docker-registry-4-ku34r	1/1	Running	3	2d
po/registry-console-1-zxreg	1/1	Running	3	2d
po/router-1-yhunh	1/1	Running	5	2d

A useful option that you can pass to the **oc get** command is the **-w** option, which watches the resultant output in real-time. This is useful, for example, for monitoring the output of an **oc get pods** command continuously instead of running it multiple times from the shell.

oc describe RESOURCE_RESOURCE_NAME

If the summaries provided by **oc get** are insufficient, additional information about the resource can be retrieved by using the **oc describe** command. Unlike the **oc get** command, there is no way to iterate through all of the different resources by type. Although most major resources can be described, this functionality is not available across all resources. The following is an example output from describing a pod resource:

```
Name: docker-registry-4-ku34r
Namespace: default
Security Policy: restricted
Node: node.lab.example.com/172.25.250.11
Start Time: Mon, 23 Jan 2017 12:17:28 -0500
Labels: deployment=docker-registry-4
deploymentconfig=docker-registry
docker-registry=default
```

```
Status:          Running
...Output omitted...
No events
```

oc export

This command can be used to export a definition of a resource. Typical use cases include creating a backup, or to aid in modifying a definition. By default, the **export** command prints out the object representation in YAML format, but this can be changed by providing a **-o** option.

oc create

This command can be used to create resources from a resource definition. Typically, this is paired with the **oc export** command for editing definitions.

oc delete RESOURCE_TYPE name

This command can be used to remove a resource from the OpenShift cluster. Note that a fundamental understanding of the OpenShift architecture is needed here, because deleting managed resources such as pods results in newer instances of those resources being automatically re-created.

oc exec

This command can be used to execute commands inside a container. You can use this command to run interactive as well as non-interactive batch commands as part of a script.

OpenShift Resource Types

Applications in OpenShift Container Platform are composed of resources of different types. The supported types are listed below:

Container

A definition of how to run one or more processes inside a portable Linux environment. Containers are started from an image and are usually isolated from other containers on the same machine.

Image

A layered Linux file system that contains application code, dependencies, and any supporting operating system libraries. An image is identified by a name that can be local to the current cluster, or point to a remote Docker registry (a storage server for images).

Pod

A set of one or more containers that are deployed onto a node and share a unique IP address and volumes (persistent storage). Pods also define the security and runtime policy for each container.

Label

Labels are key-value pairs that can be assigned to any resource in the system for grouping and selection. Many resources use labels to identify sets of other resources.

Volume

Containers are not persistent by default; their contents are cleared when they are restarted. Volumes are mounted file systems available to pods and their containers, and which may be backed by a number of host-local or network-attached storage endpoints. The simplest volume type is **EmptyDir**, which is a temporary directory on a single machine. Administrators can also allow you to request a *Persistent Volume* that is automatically attached to your pods.

Node

Nodes are host systems set up in the cluster to run containers. Nodes are usually managed by administrators and not by end users.

Service

A service is a logical name representing a set of pods. The service is assigned an IP address and a DNS name, and can be exposed externally to the cluster via a port or a route. An environment variable with the name **SERVICE_HOST** is automatically injected into other pods.

Route

A route is a DNS entry that is created to point to a service so that it can be accessed from outside the cluster. Administrators can configure one or more routers to handle those routes, typically through a HAProxy load balancer.

Replication Controller

A replication controller maintains a specific number of pods based on a template that matches a set of labels. If pods are deleted (because the node they run on is taken out of service), the controller creates a new copy of that pod. A replication controller is most commonly used to represent a single deployment of part of an application based on a built image.

Deployment Configuration

A deployment configuration defines the template for a pod and manages deploying new images or configuration changes whenever the attributes are changed. A single deployment configuration is usually analogous to a single microservice. Deployment configurations can support many different deployment patterns, including full restart, customizable rolling updates, as well as pre and post lifecycle hooks. Each deployment is represented as a replication controller.

Build Configuration

A build configuration contains a description of how to build source code and a base image into a new image. Builds can be source-based, using builder images for common languages such as Java, PHP, Ruby, or Python, or Docker-based, which create builds from a Dockerfile. Each build configuration has webhooks and can be triggered automatically by changes to their base images.

Build

Builds create new images from source code, other images, Dockerfiles, or binary input. A build is run inside of a container and has the same restrictions that normal pods have. A build usually results in an image being pushed to a Docker registry, but you can also choose to run a post-build test that does not push an image.

Image Streams and Image Stream Tags

An image stream groups sets of related images using tag names. It is analogous to a branch in a source code repository. Each image stream can have one or more tags (the default tag is called "latest") and those tags might point to external Docker registries, to other tags in the same stream, or be controlled to directly point to known images. In addition, images can be pushed to an image stream tag directly via the integrated Docker registry.

Secret

The secret resource can hold text or binary secrets for delivery into your pods. By default, every container is given a single secret which contains a token for accessing the API (with limited privileges) at `/var/run/secrets/kubernetes.io/serviceaccount`. You can create new secrets and mount them in your own pods, as well as reference secrets from builds (for connecting to remote servers), or use them to import remote images into an image stream.

Project

All of the above resources (except nodes) exist inside of a project. Projects have a list of members and their roles, such as `view`, `edit`, or `admin`, as well as a set of security controls on the running pods, and limits on how many resources the project can use. Resource names are unique within a project. Developers may request that projects be created, but administrators control the resources allocated to projects.

Use the `oc types` command for a quick refresher on the concepts and types available.

Creating Applications Using `oc new-app`

Simple applications, complex multi-tier applications, and microservice applications can be described by a single resource definition file. This file contains many pod definitions, service definitions to connect the pods, replication controllers or deployment configs to horizontally scale the application pods, persistent volume claims to persist application data, and anything else needed that can be managed by OpenShift.

The `oc new-app` command can be used, with the `-o json` or `-o yaml` option, to create a skeleton resource definition file in JSON or YAML format, respectively. This file can be customized and used to create an application using the `oc create -f <filename>` command, or merged with other resource definition files to create a composite application.

The `oc new-app` command can create application pods to run on OpenShift in many different ways. It can create pods from existing docker images, from Dockerfiles, or from raw source code using the Source-to-Image (S2I) process.

Run the `oc new-app -h` command to understand all the different options available for creating new applications on OpenShift. The most common options are listed below:

To create an application based on an image from Docker Hub:

```
$ oc new-app mysql MYSQL_USER=user MYSQL_PASSWORD=pass MYSQL_DATABASE=testdb -l db=mysql
```

To create an application based on an image from a private registry:

```
$ oc new-app --docker-image=myregistry.com/mycompany/myapp --name=myapp
```

To create an application based on source code stored in a Git repository:

```
$ oc new-app https://github.com/openshift/ruby-hello-world --name=ruby-hello
```

To create an application based on source code stored in a Git repository and referring to an image stream:

```
$ oc new-app https://mygitrepo/php-hello -i php:7.0 --name=php-hello
```



References

Further information is available in the *CLI Reference* chapter of the OpenShift Container Platform documentation at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

OpenShift client downloads:

 | <https://access.redhat.com/downloads/content/290>

Guided Exercise: Managing an OpenShift Instance Using oc

In this exercise, you will manage an instance of OpenShift Container Platform using the **oc** command.

Resources	
Application URL	https://master.lab.example.com:8443

Outcomes

You should be able to log in to the OpenShift master and manage the cluster using the **oc** command-line tool.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands on the **workstation** host to ensure your environment is correctly configured:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

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

Steps

1. Get the current status of the OpenShift cluster.
 - 1.1. Open a new terminal on **workstation** and login to OpenShift as the **admin** user with a password of **redhat**:

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

- 1.2. Ensure that you are using the **default** project:

```
[student@workstation ~]$ oc project default
```

- 1.3. List the nodes that are part of the cluster and their status:

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

This command produces a tabulated list of nodes similar to the following. Take note of any nodes that have **SchedulingDisabled** as part of their status descriptions. Applications (pods) cannot be deployed on such nodes.

NAME	STATUS	AGE
master.lab.example.com	Ready, SchedulingDisabled	56m
node1.lab.example.com	Ready	56m
node2.lab.example.com	Ready	56m

- 1.4. Display more detailed information about the OpenShift master node using the **oc describe** command:

```
[student@workstation ~]$ oc describe node master.lab.example.com
Name:           master.lab.example.com
Role:
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/hostname=master.lab.example.com
Taints:        <none>
...
System Info:
...
Kernel Version: 3.10.0-514.el7.x86_64
OS Image:      Red Hat Enterprise Linux Server 7.3 (Maipo)
Operating System: linux
Architecture:   amd64
Container Runtime Version: docker://1.12.6
Kubelet Version: v1.5.2+43a9be4
Kube-Proxy Version: v1.5.2+43a9be4
ExternalID:    master.lab.example.com
...
Events:
...
Normal  Starting  Starting kubelet.
...
Normal  NodeReady  Node master.lab.example.com status is now: NodeReady
```

The **Events** section shows important life-cycle events that have occurred on the master node since the cluster was started. This information is very useful when troubleshooting issues on the master.

- 1.5. Similarly, examine the description of one of the OpenShift nodes:

```
[student@workstation ~]$ oc describe node node1.lab.example.com
Name:           node1.lab.example.com
Role:
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/hostname=node1.lab.example.com
                region=infra
...
System Info:
...
Kernel Version: 3.10.0-514.el7.x86_64
OS Image:      Red Hat Enterprise Linux Server 7.3 (Maipo)
Operating System: linux
Architecture:   amd64
Container Runtime Version: docker://1.12.6
Kubelet Version: v1.5.2+43a9be4
Kube-Proxy Version: v1.5.2+43a9be4
ExternalID:    node1.lab.example.com
...
```

```
Events:
...
Normal  Starting  Starting kubelet.
...
Normal  NodeReady  Node node1.lab.example.com status is now: NodeReady
```

- 1.6. Inspect the list of existing pods in the project by using the **oc get pods** command.

```
[student@workstation ~]$ oc get pods -o wide
NAME          READY   STATUS    ...   IP           NODE
docker-registry-1-n2019  1/1    Running  ..  10.129.0.4   node1.lab.example.com
registry-console-2-jk4t3  1/1    Running  ..  10.130.0.8   node2.lab.example.com
router-1-46g2j          1/1    Running  ..  172.25.250.11  node1.lab.example.com
router-1-xrc2h          1/1    Running  ..  172.25.250.12  node2.lab.example.com
```

The **NODE** column lists the node on which the pod is running.

- 1.7. Use the **oc describe** command to view detailed information about a pod.

```
[student@workstation ~]$ oc describe pod docker-registry-1-n2019
Name: docker-registry-1-n2019
Namespace: default
Security Policy: hostnetwork
Node: node1.lab.example.com/172.25.250.11
Start Time: Wed, 12 Jul 2017 09:35:11 +0530
Labels: deployment=docker-registry-1
        deploymentconfig=docker-registry
        docker-registry=default
Status: Running
IP: 10.129.0.4
...
Events:
...
Normal ... Successfully assigned docker-registry-1-n2019 to
node1.lab.example.com
Normal ... pulling image "openshift3/ose-docker-registry:v3.5.5.8"
Normal ... Successfully pulled image "openshift3/ose-docker-registry:v3.5.5.8"
Normal ... Created container with docker id ae512ac54c1c; Security:
[seccomp=unconfined]
Normal ... Started Started container with docker id ae512ac54c1c
```

Pay close attention to the **Events** section. It displays important life-cycle related event information about the pod, and is very useful when troubleshooting issues with pods and nodes.

2. Explore the pods.

- 2.1. One of the most useful commands available to the administrator is the **oc exec** command. This command allows the user to execute remote commands against a pod. Run the **hostname** command on the registry pod.

```
[student@workstation ~]$ oc exec docker-registry-1-n2019 hostname
docker-registry-1-n2019
```

Run the **ls** command on one of the router pods.

```
[student@workstation ~]$ oc exec router-1-46g2j ls /
```

```
bin  
boot  
dev  
etc  
exports  
home  
...
```

- 2.2. Arbitrary commands can be executed, provided they are available within the pods where you execute them. This ability can be useful for diagnosing files, contents, and processes from within the container itself. Inspect the **/etc/resolv.conf** file.

```
[student@workstation ~]$ oc exec docker-registry-1-n2019 cat /etc/resolv.conf  
search default.svc.cluster.local svc.cluster.local cluster.local lab.example.com  
example.com  
nameserver 172.25.250.11  
nameserver 172.25.250.11  
options ndots:5
```

- 2.3. 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. On the **master** node, launch a remote shell in the pod:

```
[student@workstation ~]$ oc exec docker-registry-1-n2019 -it bash  
bash-4.2$
```

- 2.4. Run the same **ls** command that was executed before without the interactive shell:

```
bash-4.2$ ls /  
bin config.yaml etc lib lost+found mnt proc root sbin sys usr  
boot dev home lib64 media opt registry run srv tmp var
```

- 2.5. Exit the remote shell:

```
bash-4.2$ exit  
exit
```



Note

You can also run the **oc rsh <pod-name>** command to get remote shell access to a running pod.

3. Explore the project status and cluster events.

- 3.1. Use the **oc status** command to get a high-level status of the current project:

```
[student@workstation ~]$ oc status -v  
In project default on server https://master.lab.example.com:8443  
  
https://docker-registry-default.cloudapps.lab.example.com (passthrough) (svc/  
docker-registry)
```

```

dc/docker-registry deploys docker.io/openshift3/ose-docker-registry:v3.5.5.8
deployment #1 deployed 2 hours ago - 1 pod

svc/kubernetes - 172.30.0.1 ports 443, 53->8053, 53->8053

https://registry-console-default.cloudapps.lab.example.com (passthrough) (svc/
registry-console)
dc/registry-console deploys workstation.lab.example.com:5000/openshift3/
registry-console:3.5
deployment #2 deployed 2 hours ago - 1 pod
deployment #1 failed 2 hours ago: newer deployment was found running

svc/router - 172.30.126.205 ports 80, 443, 1936
dc/router deploys docker.io/openshift3/ose-haproxy-router:v3.5.5.8
deployment #1 deployed 2 hours ago - 2 pods

```

The output on your master might be different from that shown above.

3.2. Use the **oc get events** command to view life-cycle events in the OpenShift cluster:

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

Information is presented in a tabular format, in the order in which the events occurred.

4. Importing and exporting resources.

4.1. Use the **oc get all** command to get a list of resources in the project:

```

[student@workstation ~]$ oc get all
NAME                      DOCKER REPO                                     TAGS
UPDATED
is/registry-console      172.30.149.44:5000/default/registry-console   3.5

NAME        REVISION  DESIRED  CURRENT  TRIGGERED BY
dc/docker-registry    1         1         1         config
dc/registry-console  2         1         1         config
dc/router          1         2         2         config

NAME        DESIRED  CURRENT  READY     AGE
rc/docker-registry-1  1         1         1         2h
rc/registry-console-1 0         0         0         2h
rc/registry-console-2 1         1         1         2h
rc/router-1          2         2         2         2h

NAME                      HOST/PORT
routes/docker-registry    docker-registry-default.cloudapps.lab.example.com ...
routes/registry-console   registry-console-default.cloudapps.lab.example.com ...

NAME                      CLUSTER-IP        EXTERNAL-IP      PORT(S)
AGE
svc/docker-registry      172.30.149.44    <none>           5000/TCP
2h
svc/kubernetes           172.30.0.1       <none>           443/TCP,53/UDP,53/TCP
2h
svc/registry-console    172.30.252.230   <none>           9000/TCP
2h
svc/router               172.30.126.205   <none>           80/TCP,443/TCP,1936/TCP
2h

NAME        READY   STATUS    RESTARTS  AGE
po/docker-registry-1-n2019 1/1     Running   0          2h

```

po/registry-console-2-jk4t3	1/1	Running	0	2h
po/router-1-46g2j	1/1	Running	0	2h
po/router-1-xrc2h	1/1	Running	0	2h

The output on your system might be different from that shown above.

- 4.2. The **oc export** command exports existing resources and converts them to configuration files (YAML or JSON) for backups, or for re-creating resources elsewhere in the cluster.

Export the *docker-registry-1-n2019* pod resource in the default YAML format:

```
[student@workstation ~]$ oc export pod docker-registry-1-n2019
apiVersion: v1
kind: Pod
metadata:
  annotations:
  ...
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
spec:
  containers:
  - env:
    - name: REGISTRY_HTTP_ADDR
      value: :5000
    - name: REGISTRY_HTTP_NET
      value: tcp
  ...
  image: openshift3/ose-docker-registry:v3.5.5.8
  imagePullPolicy: IfNotPresent
  ...
  name: registry
  ports:
  - containerPort: 5000
    protocol: TCP
  ...
  volumeMounts:
  - mountPath: /registry
    name: registry-storage
  ...
  nodeName: node1.lab.example.com
  nodeSelector:
    region: infra
  ...
```



Note

You can export the pod definition in JSON format by passing the **-o json** option to the **oc export** command.

- 4.3. You can export multiple resources simultaneously as an OpenShift *template* by passing the **--as-template** option to the **oc export** command.

For example, to export the service and deployment config definition as a single OpenShift template:

```
[student@workstation ~]$ oc export svc,dc docker-registry --as-template=docker-registry
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: docker-registry
objects:
- apiVersion: v1
  kind: Service
...
- apiVersion: v1
  kind: DeploymentConfig
...
  template:
...
    image: openshift3/ose-docker-registry:v3.5.5.8
...
    volumes:
- emptyDir: {}
  name: registry-storage
...
```

The above command exports both the service definition and the deployment config as a **Template** type. The output of this command can be sent as input to the **oc create** command to re-create the resource elsewhere in a cluster.

Run the **oc export --help** command to get a detailed list of options you can pass to the command.



Note

You can redirect the output from the **oc export** command to a file using the standard UNIX redirection symbol ">". For example:

```
oc export svc,dc docker-registry > docker-registry.yaml
```

This concludes the guided exercise.

Executing Troubleshooting Commands

Objective

After completing this section, students should be able to execute commands that assist in troubleshooting common problems.

General Environment Information

If you have installed OpenShift using the RPM installation method, the master and node components will run as native Red Hat Enterprise Linux services. A starting point for data collection from masters and nodes is to use the standard **sosreport** utility that gathers information about the environment along with docker and OpenShift-related information:

```
[root@master ~]# sosreport -k docker.all=on -k docker.logs=on
sosreport (version 3.3)

This command will collect diagnostic and configuration information from
this Red Hat Enterprise Linux system and installed applications.

...
Setting up archive ...
Setting up plugins ...
Running plugins. Please wait ...

Running 37/91: kubernetes...
...
Running 58/91: openshift...
Running 59/91: openssl...
Running 60/91: openvswitch...
Running 61/91: origin...
...

Creating compressed archive...

Your sosreport has been generated and saved in:
/var/tmp/sosreport-master.lab.example.com-20170723184404.tar.xz

The checksum is: 80180fc0d46fbcc633f900c7d413b4eb8

Please send this file to your support representative.
```

The **sosreport** command creates a compressed archive containing all the relevant information and saves it in a compressed archive in the **/var/tmp** directory. You can then send this archive file to RedHat support.

Another useful diagnostic tool for a cluster administrator is the **oc adm diagnostics** command, which gives you the possibility to run several diagnostic checks on the OpenShift cluster including networking, aggregated logging, the internal registry, master and node service checks and many more. Run the **oc adm diagnostics --help** command to get a detailed list of diagnostics that can be run.

OpenShift Troubleshooting Commands

The **oc** command-line client is the primary tool used by administrators to detect and troubleshoot issues in an OpenShift cluster. It has a number of options that enable you to detect, diagnose, and fix issues with masters and nodes, the services, and the resources managed by the

cluster. If you have the required permissions, you can directly edit the configuration for most of the managed resources in the cluster.

oc get events

Events allow OpenShift to record information about life-cycle events in a cluster. They allow developers and administrators to view information about OpenShift components in a unified way. The **oc get events** command provides information about events in an OpenShift namespace. Examples of events that are captured and reported are listed below:

- Pod creation and deletion
- Pod placement scheduling
- Master and node status

Events are useful during troubleshooting. Administrators can get high-level information about failures and issues in the cluster, and they can then proceed to investigate using log files and other **oc** subcommands.

You can get a list of events in a given project using the following command:

```
[student@workstation ~]$ oc get events -n <project>
```

You can also view events in your project from the web console in the **Monitoring > Events** page. Many other objects, such as pods and deployments, have their own **Events** tab as well, which shows events related to that object:

6:23:46 PM	Pod scaling-1-jb7zj	Created Created container with docker id 2d0b43a6b207; Security:[seccomp=unconfined]
6:23:42 PM	Pod scaling-1-dq36m	Failed ▲ Failed to pull image "172.30.53.104:5000/scaling /scaling@sha256:88a9beebab467735484c5405e1f241b65cb03c2de993ba9e1bbbbbd662 123985c0": Get http://172.30.53.104:5000/v2/: dial tcp 172.30.53.104:5000: getsockopt: connection refused
6:23:42 PM	Pod scaling-1-dq36m	Failed sync ▲ Error syncing pod, skipping: failed to "StartContainer" for "scaling" with ErrImagePull: "Get http://172.30.53.104:5000/v2/: dial tcp 172.30.53.104:5000: getsockopt: connection refused"
6:23:42 PM	Pod scaling-1-dq36m	Back off Back-off pulling image "172.30.53.104:5000/scaling /scaling@sha256:88a9beebab467735484c5405e1f241b65cb03c2de993ba9e1bbbbbd662 123985c0"
6:23:42 PM	Pod scaling-1-dq36m	Failed sync ▲ Error syncing pod, skipping: failed to "StartContainer" for "scaling" with ImagePullBackOff: "Back-off pulling image \"172.30.53.104:5000/scaling /scaling@sha256:88a9beebab467735484c5405e1f241b65cb03c2de993ba9e1bbbbbd662 123985c0\""

Figure 4.1: Viewing events in the web console.

A comprehensive list of events in the OpenShift Container Platform 3.5 is available at https://docs.openshift.com/container-platform/3.5/dev_guide/events.html.

oc logs

The **oc logs** command retrieves the log output for a specific build, deployment, or pod. This command works for builds, build configurations, deployment configurations, and pods.

To view the logs for a pod using the **oc** command-line tool:

```
[student@workstation ~]$ oc logs pod
```

To view the logs for a build:

```
[student@workstation ~]$ oc logs bc/build-name
```

You can use the **oc logs** command with the **-f** option to follow the log output in real time. This is useful, for example, for monitoring the progress of builds continuously and checking for errors.

You can also view log information about pods, builds, and deployments from the web console.

oc rsh

The **oc rsh** command opens a remote shell session to a container. This is useful for logging in and investigating issues in a running container.

To log in to a container shell remotely and execute commands:

```
[student@workstation ~]$ oc rsh <pod>
```

oc rsync

The **oc rsync** command copies the contents to or from a directory in a running pod. If a pod has multiple containers, you can specify the container ID using the **-c** option. Otherwise, it defaults to the first container in the pod. This is useful for transferring log files and configuration files from the container.

To copy contents from a directory in a pod to a local directory:

```
[student@workstation ~]$ oc rsync <pod>:<pod_dir> <local_dir> -c <container>
```

To copy contents from a local directory to a directory in a pod:

```
[student@workstation ~]$ oc rsync <local_dir> <pod>:<pod_dir> -c <container>
```

oc port-forward

You can use the **oc port-forward** command to forward one or more local ports to a pod. This allows you to listen on a given or random port locally, and have data forwarded to and from given ports in the pod.

The format of this command is as follows:

```
[student@workstation ~]$ oc port-forward <pod> [<local_port>:]<remote_port>
```

For example, to listen on port 3306 locally and forward to 3306 in the pod, run:

```
[student@workstation ~]$ oc port-forward <pod> 3306:3306
```

Troubleshooting Common Issues

Some of the most common errors and issues seen in OpenShift deployments, and the tools that can be used to troubleshoot them are discussed in the paragraphs below.

Resource Limits and Quota Issues

For projects that have resource limits and quotas set, the improper configuration of resources will cause deployment failures. Use the **oc get events** and **oc describe** commands to investigate the cause of the failure. For example, if you try to create more pods than is allowed in a project with quota restrictions on pod count, you will see the following output when you run the **oc get events** command:

```
14m
Warning FailedCreate {hello-1-deploy} Error creating: pods "hello-1" is forbidden: exceeded quota: project-quota, requested: cpu=250m, used: cpu=750m, limited: cpu=900m
```

Source-to-Image (S2I) Build Failures

Use the **oc logs** command to view S2I build failures. For example, to view logs for a build configuration named **hello**:

```
[student@workstation ~]$ oc logs bc/hello
```

You can adjust the verbosity of build logs by specifying a **BUILD_LOGLEVEL** environment variable in the build config strategy, for example:

```
{
  "sourceStrategy": {
    ...
    "env": [
      {
        "name": "BUILD_LOGLEVEL",
        "value": "5"
      }
    ]
  }
}
```

ErrImagePull and *ImgPullBackOff* Errors

These errors are caused by an incorrect deployment configuration, wrong or missing images being referenced during deployment, or improper docker configuration. For example:

```
Pod Warning FailedSync {kubelet node1.lab.example.com}
Error syncing pod, skipping: failed to "StartContainer" for "pod-diagnostics" with
  ErrImagePull: "image pull failed for registry.access.redhat.com/openshift3/ose-
deployer:v3.5.5.8..."
...
Pod       spec.containers{pod-diagnostics} Normal   BackOff {kubelet
  node1.lab.example.com}  Back-off pulling image "registry.access.redhat.com/openshift3/
ose-deployer:v3.5.5.8"
...
pod-diagnostic-test-27zqb Pod Warning FailedSync {kubelet node1.lab.example.com}
Error syncing pod, skipping: failed to "StartContainer" for "pod-diagnostics" with
  ImagePullBackOff: "Back-off pulling image \"registry.access.redhat.com/openshift3/ose-
deployer:v3.5.5.8\""
```

Use the **oc get events** and **oc describe** commands to check for details. Fix deployment configuration errors by editing the deployment config using the **oc edit dc/ <deploymentconfig>** command.

Incorrect Docker Configuration

Incorrect docker configuration on masters and nodes can cause many errors during deployment. Specifically, check the **ADD_REGISTRY**, **INSECURE_REGISTRY**, and **BLOCK_REGISTRY** settings and ensure that they are valid. Use the **systemctl status**, **oc logs**, **oc get events**, and **oc describe** commands to troubleshoot.

You can change the docker service log levels by adding the **--log-level** parameter for the **OPTIONS** variable in the docker configuration file located at **/etc/sysconfig/docker**. For example, to set the log level to debug:

```
OPTIONS='--insecure-registry=172.30.0.0/16 --selinux-enabled --log-level=debug'
```

Master and Node Service Failures

Use the **systemctl status** command for troubleshooting issues with the **atomic-openshift-master**, **atomic-openshift-node**, **etcd**, and **docker** services. Use the **journalctl -u <unit-name>** command to view the system log for issues related to the previously listed services.

You can increase the verbosity of logging from the **atomic-openshift-node** and the **atomic-openshift-master** services by editing the **--loglevel** variable in the respective configuration files, and then restarting the associated service.

For example, to set the OpenShift master log level to debug, add or edit this line in the **/etc/sysconfig/atomic-openshift-master** file:

```
OPTIONS='--loglevel=4'
```



Note

Openshift has five numbered log message severities. Messages with FATAL, ERROR, WARNING, and some INFO severities appear in the logs regardless of the log configuration. The severity levels are listed below:

- 0 - Errors and warnings only
- 2 - Normal information (Default)
- 4 - Debugging-level information
- 6 - API-level debugging information (request/response)
- 8 - API debugging information with full body of request

Similarly, the log level for OpenShift nodes can be changed in the **/etc/sysconfig/atomic-openshift-node** file.

Failures in Scheduling Pods

The OpenShift master schedules pods to run on nodes. Sometimes, pods cannot run due to issues with the nodes themselves not being in a *Ready* state, and also due to resource limits and quotas. Use the **oc get nodes** command to verify the status of nodes. During scheduling failures, pods will be in the *Pending* state, and you can check this using the **oc get pods -o wide** command, which also shows the node on which the pod was scheduled to run. Check details about the scheduling failure using the **oc get events** and **oc describe pod** commands.

A sample pod scheduling failure due to insufficient CPU is shown below, as output from the **oc describe** command:

```
{default-scheduler } Warning FailedScheduling pod (FIXEDhello-phb4j) failed to fit in  
any node  
fit failure on node (hello-wx0s): Insufficient cpu  
fit failure on node (hello-tgfm): Insufficient cpu  
fit failure on node (hello-qwds): Insufficient cpu
```

A sample pod scheduling failure due to a node not being in the *Ready* state is shown below, as output from the **oc describe** command:

```
{default-scheduler } Warning FailedScheduling pod (hello-phb4j): no nodes available to  
schedule pods
```

References

Troubleshooting OpenShift Container Platform
<https://access.redhat.com/solutions/1542293>

Configure log levels for OpenShift Container Platform
<https://access.redhat.com/solutions/2216951>

Common issues on OpenShift Container Platform
<https://access.redhat.com/solutions/1599603>

Guided Exercise: Troubleshooting Common Problems

In this exercise, you will troubleshoot a failing application deployment on OpenShift and fix the issues.

Resources	
S2I Application:	http://workstation.lab.example.com/php-helloworld

Outcomes

You should be able to troubleshoot a failing application deployment on OpenShift and fix the issues.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands on the **workstation** host to ensure your environment is correctly configured:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab common-troubleshoot setup
```

Steps

1. Create a new project.

- 1.1. On the **workstation** host, access the OpenShift master (<https://master.lab.example.com:8443>) with the OpenShift client.

Log in as **developer** and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 1.2. Create the **common-troubleshoot** project:

```
[student@workstation ~]$ oc new-project common-troubleshoot
Now using project "common-troubleshoot" on server "https://
master.lab.example.com:8443".
...
```

2. Deploy a Source-to-Image (S2I) application.

- 2.1. Create a new application in OpenShift using the source code from the **php-helloworld** application, available in the classroom git repository:

```
[student@workstation ~]$ oc new-app --name=hello -i php:5.6 \
http://workstation.lab.example.com/php-helloworld
error: multiple images or templates matched "php:5.6": 2

The argument "php:5.6" could apply to the following Docker images, OpenShift
image streams, or templates:

* Image stream "php" (tag "latest") in project "openshift"
  Use --image-stream="openshift/php:latest" to specify this image or template

* Image stream "php" (tag "7.0") in project "openshift"
  Use --image-stream="openshift/php:7.0" to specify this image or template
```

Observe the error that informs you about the wrong image stream tag.

- 2.2. List the valid tags in the **php** image stream using the **oc describe** command.

```
[student@workstation ~]$ oc describe is php -n openshift
7.0 (latest)
  tagged from workstation.lab.example.com:5000/rhscl/php-70-rhel7:latest
    will use insecure HTTPS or HTTP connections
...
  Tags: builder, php
  Supports: php:7.0, php
...
  * workstation.lab.example.com:5000/rhscl/php-70-rhel7@sha256:3ff8e5...
...
5.6
  tagged from workstation.lab.example.com:5000/rhscl/php-56-rhel7:latest
    will use insecure HTTPS or HTTP connections
...
! error: Import failed (NotFound): dockerimage
"workstation.lab.example.com:5000/rhscl/php-56-rhel7:latest" not found
```

The output of the command shows that **php:7.0** is a valid tag, whereas **php:5.6** is invalid, because the **php:5.6** image is not available.

- 2.3. Deploy the application with the correct image stream tag:

```
[student@workstation ~]$ oc new-app --name=hello -i php:7.0 \
http://workstation.lab.example.com/php-helloworld
--> Found image 9e23736 (8 weeks old) in image stream "openshift/php" under tag
"7.0" for "php:7.0"
...
--> Creating resources ...
  imagestream "hello" created
  buildconfig "hello" created
  deploymentconfig "hello" created
  service "hello" created
--> Success
  Build scheduled, use 'oc logs -f bc/hello' to track its progress.
  Run 'oc status' to view your app.
```

The **oc new-app** command should now succeed.

2.4. Verify that the application successfully built and deployed:

```
[student@workstation ~]$ oc get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE      IP          NODE
hello-1-build  0/1     Pending   0          1m      <none>
```

The **hello-1-build** pod is in the **Pending** state and the application pod was not started. Investigate why the deployment is in the **Pending** state and fix the issue.

3. Check the logs for the build pod:

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

This command will not produce any output. The logs show no useful information that can help you troubleshoot the issue.

4. Check the event log for the project. You can do this in two ways. One way is to use the **oc get events** command:

```
[student@workstation ~]$ oc get events
.. NAME           KIND .. REASON          .. MESSAGE
.. hello-1-build  Pod   .. FailedScheduling .. no nodes available to schedule
pods
```

You can also use the **oc describe** command to see if it gives some hints on why the pod is failing:

```
[student@workstation ~]$ oc describe pod hello-1-build
Name: hello-1-build
Namespace: troubleshoot-commands
Security Policy: privileged
Node: /
Labels: openshift.io/build.name=hello-1
Status: Pending
...
Events:
.. NAME           KIND .. REASON          .. MESSAGE
.. hello-1-build  Pod   .. FailedScheduling .. no nodes available to schedule
pods
```

This command also reports the same **FailedScheduling** warning in the **Events** section.

The event log shows that no nodes are available for scheduling pods to run.

5. Investigate the cause of this warning. Check the status of the nodes in the cluster to see if there are issues. Note that this command should be run as the **root** user on **master**.

```
[student@workstation ~]$ ssh root@master oc get nodes
NAME           STATUS    AGE
master.lab.example.com  Ready, SchedulingDisabled  4d
node1.lab.example.com   NotReady   4d
node2.lab.example.com   NotReady   4d
```

The **STATUS** column indicates that both **node1** and **node2** are in the **NotReady** state. OpenShift cannot schedule pods to run on nodes that are marked as **NotReady**.



Note

If you get an error in this step such as the following:

```
error: You must be logged in to the server (the server has asked for the client to provide credentials)
```

log in to the **master** host as **root** and run:

```
[root@master ~]# oc login -u system:admin
```

You can then proceed with running the **oc get nodes** command.

6. Investigate why the nodes are not in the **Ready** state. OpenShift nodes must be running the **atomic-openshift-node** service. This service is responsible for communicating with the master, and runs pods on demand when scheduled by the master.

Open two new terminals on **workstation** and log in to the **node1** and **node2** hosts as **root** using the **ssh** command:

```
[student@workstation ~]$ ssh root@node1  
[student@workstation ~]$ ssh root@node2
```

Check the status of the **atomic-openshift-node** service on both nodes:

```
[root@node1 ~]# systemctl status atomic-openshift-node.service -l  
[root@node2 ~]# systemctl status atomic-openshift-node.service -l
```

Although both nodes are reporting that the service is active and running, the service reports that something is wrong with the **docker** daemon on the nodes:

```
atomic-openshift-node.service - Atomic OpenShift Node  
...  
  Active: active (running) since Tue 2017-07-18 13:47:32 IST; 4h 53min ago  
...  
atomic-openshift-node[1349]...Unable to retrieve pods: Cannot connect to the Docker daemon. Is the docker daemon running on this host?
```

7. Check the status of the **docker** service on both nodes:

```
[root@node1 ~]# systemctl status docker.service -l  
[root@node2 ~]# systemctl status docker.service -l
```

The service is inactive on both nodes:

```
docker.service - Docker Application Container Engine
```

```
...  
Active: inactive (dead) since Tue 2017-07-18 17:33:21 IST; 2h 23min ago  
...  
Main PID: 1022 (code=exited, status=0/SUCCESS)...  
systemd[1]: Stopped Docker Application Container Engine.  
...
```

8. Start the **docker** service on both nodes:

```
[root@node1 ~]# systemctl start docker.service  
[root@node2 ~]# systemctl start docker.service
```

9. On the **workstation** host, check that the **oc get nodes** command shows both nodes in the **Ready** state:

```
[student@workstation ~]$ ssh root@master oc get nodes  
NAME          STATUS        AGE  
master.lab.example.com  Ready, SchedulingDisabled  4d  
node1.lab.example.com   Ready                    4d  
node2.lab.example.com   Ready                    4d
```



Note

It may take several minutes before the command reports that the nodes are in the **Ready** state.

10. On the **workstation** host, verify that the pod is now in the **Running** state:

```
[student@workstation ~]$ oc get pods  
NAME      READY     STATUS    RESTARTS   AGE  
...  
hello-1-pcbjt  1/1      Running   0          18m
```

You should see the application pod in the **Running** state.



Note

It might take some time for the application to build and deploy on OpenShift. Run the above command until you see the application pod in the **Running** state.



Note

While the cluster nodes register the docker restart event, if you check the application build logs, you might sometimes see failure messages:

```
Cloning "http://workstation.lab.example.com/php-helloworld" ...
Commit: 3674940fa32c1e27577d2c18dd00d03400a3e6ba (Initial commit)
Author: root <root@workstation.lab.example.com>
Date: Sat Jul 8 13:40:42 2017 +0530
---> Installing application source...
Pushing image 172.30.53.104:5000/common-troubleshoot/hello:latest ...
Warning: Push failed, retrying in 5s ...
Warning: Push failed, retrying in 5s ...
Warning: Push failed, retrying in 5s ...
```

This error can be safely ignored as long as the application pod is in the **Running** state. You can verify that the application built and was pushed to the OpenShift internal registry by running the **oc describe is** command:

```
[student@workstation ~]$ oc describe is
Name: hello
Namespace: common-troubleshoot
Created: 3 hours ago
Labels: app=hello
Annotations: openshift.io/generated-by=OpenShiftNewApp
Docker Pull Spec: 172.30.53.104:5000/common-troubleshoot/hello
Unique Images: 1
Tags: 1

latest
pushed image

* 172.30.53.104:5000/common-troubleshoot/hello@sha256:54594e.....
```

11. Clean up. Delete the **common-troubleshoot** project.

```
[student@workstation ~]$ oc delete project common-troubleshoot
project "common-troubleshoot" deleted
```

This concludes the guided exercise.

Lab: Executing Commands

In this lab, you will troubleshoot and fix issues related to an application deployed on OpenShift using the **oc** command line tool.

Resources	
Application URL:	http://hello.cloudapps.lab.example.com

Outcomes

You should be able to troubleshoot and fix errors related to an application deployed on OpenShift

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands from the **workstation** host to ensure that the environment is set up correctly:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab execute-review setup
```

Steps

1. The lab setup script creates a new project called **execute-review** using the **developer** user account, and deploys an application called **hello**. Log in to OpenShift as the **developer** user, and list the projects. Ensure that the **execute-review** project is the default project for this user.
2. Inspect the resources in this project. List all the resources in this project.

The **hello-1-deploy** pod will be stuck in the **ContainerCreating** state and the application pod will not be started. Investigate why the deployment is not in *Running* state and fix the issue.

3. Check the logs for the deploy pod. The logs do not provide any useful information that can help you troubleshoot the issue.
4. Check the event log for the project. The event log shows that OpenShift is unable to pull images defined in the application's deployment configuration.
5. Investigate the cause of this warning. Inspect the deployment configuration for the application and verify if the image settings are correct. The application uses the **workstation.lab.example.com:5000/node-hello** image from the classroom private registry on **workstation**.

The deploymentconfig should point to the **workstation.lab.example.com:5000/node-hello** image and should have a sha256 hash suffix.

6. Identify the node on which the pod is scheduled, and try to manually pull the docker image for the application (**workstation.lab.example.com:5000/node-hello**) on that node.

You should see a message saying that all endpoints are blocked. This type of error usually occurs in OpenShift due to incorrect deployment configuration or invalid docker settings.

7. Check the docker settings on both nodes and correct it. Verify if the settings for **ADD_REGISTRY**, **INSECURE_REGISTRY**, and **BLOCK_REGISTRY** variables in the docker configuration file are correct.
8. Restart the docker service on both nodes.
9. Switch back to the workstation VM and check the status of the application pod. It may take some time for the pod to be in *Running* state.

The pod must be in *Running* state.

10. Execute commands on the running pod and list the contents of the **/etc/resolv.conf** file inside the container. Note that your output maybe different depending on the node on which the pod is running.
11. Expose the **hello** service as a route. Use **hello.cloudapps.lab.example.com** as the hostname for the route.
12. Access the route URL **hello.cloudapps.lab.example.com** using the **curl** command, or a browser from the **workstation** VM. The host name of the pod on which the application is running should be displayed.
13. Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab execute-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

14. Clean up
Delete the **execute-review** project.

This concludes the lab.

Solution

In this lab, you will troubleshoot and fix issues related to an application deployed on OpenShift using the **oc** command line tool.

Resources	
Application URL:	http://hello.cloudapps.lab.example.com

Outcomes

You should be able to troubleshoot and fix errors related to an application deployed on OpenShift

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands from the **workstation** host to ensure that the environment is set up correctly:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab execute-review setup
```

Steps

1. The lab setup script creates a new project called **execute-review** using the **developer** user account, and deploys an application called **hello**. Log in to OpenShift as the **developer** user, and list the projects. Ensure that the **execute-review** project is the default project for this user.

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

```
[student@workstation ~]$ oc projects
[student@workstation ~]$ oc project execute-review
```

2. Inspect the resources in this project. List all the resources in this project.

```
[student@workstation ~]$ oc get all
NAME          DOCKER REPO                                     TAGS      UPDATED
is/hello      172.30.142.148:5000/execute-review/hello      latest    29 seconds ago

NAME        REVISION  DESIRED  CURRENT  TRIGGERED BY
dc/hello    1          1         0        config,image(hello:latest)

NAME        DESIRED  CURRENT  READY    AGE
rc/hello-1  0         0         0        29s

NAME          CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
svc/hello    172.30.175.252  <none>          3000/TCP, 8080/TCP  29s
```

NAME	READY	STATUS	RESTARTS	AGE
po/hello-1-deploy	0/1	ContainerCreating	0	29s

The **hello-1-deploy** pod will be stuck in the **ContainerCreating** state and the application pod will not be started. Investigate why the deployment is not in *Running* state and fix the issue.

- Check the logs for the deploy pod. The logs do not provide any useful information that can help you troubleshoot the issue.

```
[student@workstation ~]$ oc logs hello-1-deploy
Error from server (BadRequest): container "deployment" in pod "hello-1-deploy" is
waiting to start: ContainerCreating
```

- Check the event log for the project. The event log shows that OpenShift is unable to pull images defined in the application's deployment configuration.

```
[student@workstation ~]$ oc get events --sort-by='metadata.creationTimestamp'
... NAME          KIND   TYPE      REASON     MESSAGE
... hello-1-deploy Pod    Warning   FailedSync .. Error syncing pod, skipping:
failed to "StartContainer" for "POD" with ErrImagePull: "Error while pulling image:
Get https://index.docker.io/v1/repositories/openshift3/ose-pod/images: dial tcp:
lookup index.docker.io on 172.25.250.12:53: server misbehaving"
```

You can also use the **oc describe** command to see if it gives some hints on why the pod is failing:

```
[student@workstation ~]$ oc describe pod hello-1-deploy
Name: hello-1-deploy
Namespace: execute-review
...
Node: node2.lab.example.com/172.25.250.12
...
Status: Pending
...
Events:
...Error syncing pod, skipping: failed to "StartContainer" for "POD" with
ErrImagePull: "Error while pulling image: Get https://index.docker.io/v1/
repositories/openshift3/ose-pod/images: dial tcp: lookup index.docker.io on
172.25.250.12:53: server misbehaving"
```

This command also reports the same **FailedSync** warning in the **Events** section.

- Investigate the cause of this warning. Inspect the deployment configuration for the application and verify if the image settings are correct. The application uses the **workstation.lab.example.com:5000/node-hello** image from the classroom private registry on **workstation**.

```
[student@workstation ~]$ oc get dc hello -o yaml
apiVersion: v1
kind: DeploymentConfig
metadata:
...
labels:
  app: hello
```

```

name: hello
namespace: execute-review
...
spec:
  replicas: 1
  selector:
    app: hello
  deploymentconfig: hello
...
spec:
  containers:
  - image: workstation.lab.example.com:5000/node-hello@sha256:715df5...
    imagePullPolicy: Always
    name: hello

```

The deploymentconfig should point to the **workstation.lab.example.com:5000/node-hello** image and should have a sha256 hash suffix.

6. Identify the node on which the pod is scheduled, and try to manually pull the docker image for the application (**workstation.lab.example.com:5000/node-hello**) on that node.

```
[student@workstation ~]$ oc get pods -o wide
NAME           READY   STATUS          ..  NODE
hello-1-deploy 0/1     ContainerCreating   node2.lab.example.com
```

Open a new terminal on workstation and log in to the node where the pod is scheduled using SSH, and then pull the **node-hello** docker image:

```
[student@workstation ~]$ ssh root@node2
[root@node2 ~]# docker pull workstation.lab.example.com:5000/node-hello
Using default tag: latest
Trying to pull repository workstation.lab.example.com:5000/node-hello ...
All endpoints blocked.
```

You should see a message saying that all endpoints are blocked. This type of error usually occurs in OpenShift due to incorrect deployment configuration or invalid docker settings.

7. Check the docker settings on both nodes and correct it. Verify if the settings for **ADD_REGISTRY**, **INSECURE_REGISTRY**, and **BLOCK_REGISTRY** variables in the docker configuration file are correct.

From the **workstation** VM, log in to **node1** and **node2** using SSH:

```
[student@workstation ~]$ ssh root@node1
[student@workstation ~]$ ssh root@node2
```

Edit the docker configuration file at **/etc/sysconfig/docker** using a text editor. Observe the **BLOCK_REGISTRY** lines at the bottom:

```
# Commented out by the 'lab execute-review setup' script
#BLOCK_REGISTRY='--block-registry registry.access.redhat.com --block-registry
#docker.io'

# Added by the 'lab execute-review setup' script
BLOCK_REGISTRY='--block-registry workstation.lab.example.com:5000'
```

You are seeing errors about blocked endpoints because the classroom private registry is added to the **BLOCK_REGISTRY** variable. Delete the following two lines:

```
# Added by the 'lab execute-review setup' script
BLOCK_REGISTRY='--block-registry workstation.lab.example.com:5000'
```

Uncomment the line above the **BLOCK_REGISTRY** entry you just deleted. The final **BLOCK_REGISTRY** line should look like:

```
BLOCK_REGISTRY='--block-registry registry.access.redhat.com --block-registry
docker.io'
```

Repeat the previous steps and correct the docker configuration on the other node VM.

8. Restart the docker service on both nodes.

```
[root@node1 ~]# systemctl restart docker.service
[root@node2 ~]# systemctl restart docker.service
```

9. Switch back to the workstation VM and check the status of the application pod. It may take some time for the pod to be in *Running* state.

```
[student@workstation ~]$ oc get pods -o wide
NAME          READY   STATUS    IP           NODE
hello-1-j99vm 1/1     Running   10.129.0.41  node2.lab.example.com
```

The pod must be in *Running* state.

10. Execute commands on the running pod and list the contents of the **/etc/resolv.conf** file inside the container. Note that your output maybe different depending on the node on which the pod is running.

```
[student@workstation ~]$ oc exec hello-1-j99vm \
  cat /etc/resolv.conf
search execute-review.svc.cluster.local svc.cluster.local cluster.local
lab.example.com example.com
nameserver 172.25.250.12
nameserver 172.25.250.12
options ndots:5
```

11. Expose the **hello** service as a route. Use **hello.cloudapps.lab.example.com** as the hostname for the route.

```
[student@workstation ~]$ oc expose svc/hello \
  --hostname=hello.cloudapps.lab.example.com
route "hello" exposed
```

12. Access the route URL **hello.cloudapps.lab.example.com** using the **curl** command, or a browser from the **workstation** VM. The host name of the pod on which the application is running should be displayed.

```
[student@workstation ~]$ curl http://hello.cloudapps.lab.example.com  
Hi! I am running on host -> hello-1-j99vm
```

13. Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab execute-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

14. Clean up

Delete the **execute-review** project.

```
[student@workstation ~]$ oc delete project execute-review  
project "execute-review" deleted
```

This concludes the lab.

Summary

In this chapter, you learned:

- Red Hat OpenShift Container Platform provides the **oc** command-line client to view, edit and manage resources in an OpenShift cluster.
- On Red Hat Enterprise Linux (RHEL) systems with valid subscriptions, the tool is available as an RPM file and installable using the **yum install** command.
- For alternative Linux distributions and other operating systems, such as Windows and macOS, native clients are available for download from the Red Hat Customer Portal.
- Several essential commands are available to manage OpenShift resources, such as:
 - **oc get resourceType resourceName**: Outputs a summary with important information from *resourceName*.
 - **oc describe resourceType resourceName**: Outputs detailed information from *resourceName*.
 - **oc create**: Creates a resource from an input, such as a file or an input stream.
 - **oc delete resourceType resourceName**: Removes the resource from OpenShift.
- The **oc new-app** command can create application pods to run on OpenShift in many different ways. It can create pods from existing docker images, from Dockerfiles, and from raw source code using the Source-to-Image (S2I) process.
- The **oc get events** command provides information about events in an OpenShift namespace. Events are useful during troubleshooting. An administrator can get high-level information about failures and issues in the cluster.
- The **oc logs** command retrieves the log output for a specific build, deployment, or pod. This command works for builds, build configurations, deployment configurations, and pods.
- The **oc rsh** command opens a remote shell session to a container. This is useful for logging in and investigating issues in a running container.
- The **oc rsync** command copies the contents to or from a directory in a running pod. If a pod has multiple containers, you can specify the container ID using the **-c** option. Otherwise, it defaults to the first container in the pod. This is useful for transferring log files and configuration files from the container.
- You can use the **oc port-forward** command to forward one or more local ports to a pod. This allows you to listen on a given or random port locally, and have data forwarded to and from given ports in the pod.



CHAPTER 5

CONTROLLING ACCESS TO OPENSHIFT RESOURCES

Overview	
Goal	Control access to OpenShift resources.
Objectives	<ul style="list-style-type: none">Segregate resources and control access to them using OpenShift security features.Create and apply secrets to manage sensitive information.Manage security policies using the command-line interface.
Sections	<ul style="list-style-type: none">Securing Access to OpenShift Resources (and Guided Exercise)Managing Sensitive Information and Secrets (and Guided Exercise)Managing Security Policies (and Quiz)
Lab	Controlling Access to OpenShift Resources

Securing Access to OpenShift Resources

Objective

After completing this section, students should be able to segregate resources and control access to them using OpenShift security features.

Kubernetes Namespaces

A Kubernetes namespace provides a mechanism for grouping a set of related resources together. In OpenShift Container Platform, a *project* is a Kubernetes namespace with additional annotations.

Namespaces provide the following features:

- Named resources to avoid basic naming collisions
- Delegated management authority to trusted users
- The ability to limit user resource consumption
- User and group isolation

Projects

A project provides a mechanism through which access to resources by regular users is managed. A project allows a group of users to organize and manage their content in isolation from other groups. Users must be given access to projects by administrators. If allowed to create projects, users automatically have access to their own projects.

Projects can have a separate name, display name, and description:

- The mandatory *name* is a unique identifier for the project and is most visible when using the CLI tools or API. The maximum name length is 63 characters.
- The optional display name is how the project is displayed in the web console (defaults to name).
- The optional *description* can be a more detailed description of the project and is also visible in the web console.

The following components apply to projects:

- *Objects*: Pods, services, replication controllers, and more.
- *Policies*: Rules that determine which actions users can or cannot perform on objects.
- *Constraints*: Quotas for each kind of object that can be limited.

Cluster Administration

Cluster administrators can create projects and delegate administrative rights for the project to any user. In OpenShift Container Platform, projects are used to group and isolate related objects. Administrators can give users access to certain projects, allow them to create their own, and give them administrative rights within individual projects.

Administrators can apply roles to users and groups that allow or restrict their ability to create projects. Roles can be assigned prior to a user's initial login.

The following list shows how to restrict or grant the ability for users or groups to create new projects:

- **Restricting project creation:** Removing the **self-provisioner** cluster role from authenticated users and groups denies permissions for self-provisioning any new projects.

```
[root@master ~]$ oc adm policy remove-cluster-role-from-group \
    self-provisioner \
    system:authenticated \
    system:authenticated:oauth
```

- **Granting project creation:** Project creation is granted to users with the **self-provisioner** role and the **self-provisioner** cluster role binding. These roles are available to all authenticated users by default.

```
[root@master ~]$ oc adm policy add-cluster-role-to-group \
    self-provisioner \
    system:authenticated \
    system:authenticated:oauth
```

Creating a Project

If project creation permission is granted to users, they could, for example, create a project named **demoproject** using the following command:

```
[root@master ~]$ oc new-project demoproject \
--description="Demonstrate project creation" \
--display-name="demo_project"
```

User Types

Interaction with OpenShift Container Platform is associated with a user. An OpenShift Container Platform *user* object represents a user who may be granted permissions in the system by adding **roles** to that user or to a user's group.

- **Regular users:** This is the way most interactive OpenShift Container Platform users are represented. Regular users are represented with the **User** object. Examples of regular users include **user1** and **user2**.
- **System users:** Many of these are created automatically when the infrastructure is defined, mainly for the purpose of enabling the infrastructure to securely interact with the API. System users include a cluster administrator (with access to everything), a per-node user, users for use by routers and registries, and various others. An anonymous system user also exists that is used by default for unauthenticated requests. Examples of system users include: **system:admin**, **system:openshift-registry**, and **system:node:node1.example.com**.
- **Service accounts:** These are special system users associated with projects; some are created automatically when the project is first created, and project administrators can create more for the purpose of defining access to the contents of each project. Service accounts are represented with the **ServiceAccount** object. Examples of

service account users include **system:serviceaccount:default:deployer** and **system:serviceaccount:foo:builder**.

Every user must authenticate before they can access OpenShift Container Platform. API requests with no authentication or invalid authentication are authenticated as requests by the anonymous system user. After successful authentication, policy determines what the user is authorized to do.

Security Context Constraints (SCCs)

OpenShift provides *security context constraints* (SCCs) which control the actions a pod can perform and what resources it can access. By default, the execution of any container will be granted only the capabilities defined by the *restricted* SCC.

To list the available SCCs use the following command:

```
[user@demo ~]$ oc get scc
```

To display a detailed description of a selected SCC, use the following command syntax:

```
[user@demo ~]$ oc describe scc scc_name
```

To grant a user or group a specific SCC, use the following command syntax:

```
[user@demo ~]$ oc adm policy add-scc-to-user scc_name user_name  
[user@demo ~]$ oc adm policy add-scc-to-group scc_name group_name
```

To remove a user or group from a specific SCC, use the following command syntax:

```
[user@demo ~]$ oc adm policy remove-scc-from-user scc_name user_name  
[user@demo ~]$ oc adm policy remove-scc-from-group scc_name group_name
```

Use Case for a Service Account

Service accounts provide a flexible way to control API access without sharing a regular user's credentials. If you have an application that requires a capability not granted by the *restricted* SCC, create a new, specific service account and add it to the appropriate SCC.

For example, deploying an application that requires elevated privileges is not supported by OpenShift by default. However, if circumstances dictate the need to deploy this application in spite of its restrictions, one solution is to create a service account, modify the deployment configuration, and then add the service account to an SCC, such as **anyuid**, which meets the requirements to run as **root** user in the container.

- Create a new service account named **userroot**.

```
[user@demo ~]$ oc create serviceaccount userroot
```

- Modify the deployment configuration for the application.

```
[user@demo ~]$ oc patch dc/demo-app \
```

```
--patch '{"spec":{"template":{"spec":{"serviceAccountName": "userroot"}}}}'
```

- Add the **userroot** service account to the **anyuid** SCC to run as the **root** user in the container.

```
[user@demo ~]$ oc adm policy add-scc-to-user anyuid -z userroot
```

Managing User Membership

The default configuration for the Red Hat OpenShift Container Platform is to create new users automatically when they first log in. If the user credentials are accepted by the identity provider, the user object is created.

Membership Management Using the Web Console

To manage users allowed to access a project, log in to the web console as a project administrator or cluster administrator and select the project you want to manage. In the left pane, click **Resources > Membership** to enter the project membership page.

In the **Users** column, enter the user's name in the highlighted text box. In the **Add Another Role** column, select a role from the list in the same row as the user, and then click **Add**.

Name	Roles	Add Another Role
developer	admin	Select a role Add
demo-user	admin	Add

Figure 5.1: Adding users and roles

Membership Management Using the CLI

If automatic creation of users can is disabled, a cluster administrator uses the **oc create user** command to create new users.

```
[root@master ~]$ oc create user demo-user
```

Any user needs to be created also for the identity provider. For the **HTPasswdIdentityProvider** module, use the **htpasswd** command.

```
[root@master ~]$ htpasswd /etc/origin/openshift-passwd demo-user
```

To add a project role to a user, first enter the project using the **oc project** command and them use the **oc policy add-role-to-user** command.

```
[root@master ~]$ oc policy add-role-to-user edit demo-user
```

To remove a project role from a user, use the **oc policy remove-role-from-user** command.

```
[root@master ~]$ oc policy remove-role-from-user edit demo-user
```

Not all OpenShift roles are scoped by project. To assign these rules use the **oc adm policy command**. The following example given an user cluster administrator rights:

```
[root@master ~]$ oc adm policy add-cluster-role-to-user cluster-admin admin
```

Authentication and Authorization Layers

The authentication layer identifies the user associated with requests to the OpenShift Container Platform API. The authorization layer then uses information about the requesting user to determine if the request should be allowed.

Users and Groups

A user in OpenShift Container Platform is an entity that can make requests to the OpenShift Container Platform API. Typically, this represents the account of a developer or administrator that is interacting with OpenShift Container Platform.

A user can be assigned to one or more groups, each of which represent a certain set of roles (or permissions). Groups are useful when managing authorization policies to grant permissions to multiple users simultaneously, for example allowing access to objects within a project, versus granting them to users individually.

Authentication Tokens

API calls must be authenticated with an access token or an X.509 certificate. Session tokens represent a user and are short-lived, expiring within 24 hours by default.

Authenticated users can be verified by running the **oc whoami** command.

```
[root@master ~]$ oc login -u demo-user
```

```
[root@master ~]$ oc whoami
demo-user
```

Authentication Types

Throughout this course, authentication is provided by the **HTPasswdIdentityProvider** module, which validates user names and passwords against a flat file generated using the **htpasswd** command.

Other authentication types supported by OpenShift Container Platform include:

Basic Authentication (Remote)

A generic back-end integration mechanism that allows users to log in to OpenShift Container Platform with credentials validated against a remote identity provider. Users send their user name and password to OpenShift Container Platform, which then validates those credentials

against a remote server by making a server-to-server request, passing the credentials as a Basic Auth header. This requires users to enter their credentials to OpenShift Container Platform during the login process.

Request Header Authentication

Users log in to OpenShift Container Platform using request header values, such as *X-Remote-User*. It is typically used in combination with an authenticating proxy, which authenticates the user and then provides OpenShift Container Platform with the user's identity via a request header value.

Keystone Authentication

Keystone is an OpenStack project that provides identity, token, catalog, and policy services. OpenShift Container Platform integrates with Keystone to enable shared authentication with an OpenStack Keystone v3 server configured to store users in an internal database. This configuration allows users to log in to OpenShift Container Platform with their Keystone credentials.

LDAP Authentication

Users log in to OpenShift Container Platform with their LDAP credentials. During authentication, the LDAP directory is searched for an entry that matches the provided user name. If a match is found, a simple bind is attempted using the distinguished name (DN) of the entry plus the provided password.

GitHub Authentication

GitHub uses OAuth, which allows integration with OpenShift Container Platform to use that OAuth authentication to facilitate a token exchange flow. This allows users to log in to OpenShift Container Platform with their GitHub credentials. To prevent unauthorized users with GitHub user IDs from logging in to the OpenShift Container Platform cluster, access can be restricted to only those in specific GitHub organizations.



References

Further information is available in the *Core Concepts* chapter of the *Architecture Guide* for OpenShift Container Platform at
<https://access.redhat.com/documentation/en/openshift-container-platform>

Guided Exercise: Managing Projects and Accounts

In this exercise, you will disable auto-provisioning of projects by regular users, segregate project access, and enable service accounts for a specific project.

Resources	
Files:	/home/student/D0280/labs/secure-resources
Application URL:	http://nginx-proj-user1.cloudapps.lab.example.com

Outcomes

You should be able to disable project creation for users, enable individual access to a project, and create a service account to change security restrictions for a specific project.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, then, on the **workstation** VM, run the lab setup script followed by the Ansible playbook:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** VMs are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab secure-resources setup
```

Steps

1. Create the required users for this lab: **user1** is a project administrator, and **user2** is a project developer. Both users have **redhat** as their password.

The classroom setup uses the **htpasswd** tool to create users on the **master** VM.

- 1.1. Access the **master** VM from the **workstation** VM using SSH.

Open a terminal on the **workstation** VM and run the following command:

```
[student@workstation ~]$ ssh root@master  
[root@master ~]#
```

- 1.2. Create the project administrator user.

In the existing terminal window, run the following command:

```
[root@master ~]# htpasswd -b /etc/origin/openshift-passwd user1 redhat
```

1.3. Create the developer user.

In the terminal window, run the following command:

```
[root@master ~]# htpasswd -b /etc/origin/openshift-passwd user2 redhat
```

1.4. Log out of the **master** VM.

In the terminal window, run the following command:

```
[root@master ~]# exit
```

2. Disable project creation capabilities for all regular users.

2.1. Log in as **admin** on the **workstation** VM.

From the terminal window, run the following command:

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

Acknowledge that you accept insecure connections.

2.2. Remove the capability to create projects for all regular users.

The command in this step can be run or copied from the **configure-policy.sh** script in the **/home/student/D0280/labs/secure-resources** folder.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
self-provisioner system:authenticated system:authenticated:oauth
```

3. Verify that regular users cannot create projects in OpenShift.

3.1. Log in to OpenShift as **user1**.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u user1 -p redhat \
https://master.lab.example.com:8443
```

3.2. Try to create a new project.

Run the following command:

```
[student@workstation ~]$ oc new-project test
Error from server (Forbidden): You may not request a new project via this API.
```

Due to the change in the security policy, the user cannot create a new project. This task is delegated to the OpenShift cluster administrator.

4. Create two projects as a cluster administrator.

4.1. Log in to OpenShift as the **admin** user.

From the terminal window, run the following command:

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

4.2. Create a new project.

Run the following command:

```
[student@workstation ~]$ oc new-project proj-user1
```

Because you are a cluster administrator, a new project is created.

4.3. Create another project.

Run the following command:

```
[student@workstation ~]$ oc new-project proj-user2
```

5. Associate **user1** with **proj-user1** and **user2** with both **proj-user1** and **proj-user2**.

5.1. Add **user1** as an administrator for the **proj-user1** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc project proj-user1
Now using project "proj-user1" on server "https://master.lab.example.com:8443".
[student@workstation ~]$ oc policy add-role-to-user admin user1
role "admin" added: "user1"
```

5.2. Add **user2** as a developer for the **proj-user1** project.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc policy add-role-to-user edit user2
role "edit" added: "user2"
```

5.3. Add **user2** as a developer for the **proj-user2** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc project proj-user2
Now using project "proj-user2" on server "https://master.lab.example.com:8443".
[student@workstation ~]$ oc policy add-role-to-user edit user2
role "edit" added: "user2"
```

6. Test user access to each project.

6.1. Verify that **user1** can access only the **proj-user1** project.

Log in to OpenShift as **user1**.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u user1 -p redhat
```

6.2. Enter the **proj-user1** project.

From the terminal window, run the following command. It should work:

```
[student@workstation ~]$ oc project proj-user1
```

6.3. Enter the **proj-user2** project.

From the terminal window, run the following command. It should fail:

```
[student@workstation ~]$ oc project proj-user2  
error: You are not a member of project "proj-user2".
```

6.4. Verify that **user2** can access both **proj-user1** and **proj-user2** projects.

Log in to OpenShift as **user2**.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u user2 -p redhat
```

6.5. Enter the **proj-user1** project.

From the terminal window, run the following command. It should work:

```
[student@workstation ~]$ oc project proj-user1
```

6.6. Enter the **proj-user2** project

From the terminal window, run the following command. It should work:

```
[student@workstation ~]$ oc project proj-user2
```

7. As a developer user, deploy to **proj-user1** an application that requires elevated privileges.

7.1. Log in as **user2** and enter the **proj-user1** project:

```
[student@workstation ~]$ oc login -u user2 -p redhat  
[student@workstation ~]$ oc project proj-user1
```

7.2. Run the following command:

```
[student@workstation ~]$ oc new-app \
```

```
--name=nginx \
--docker-image=workstation.lab.example.com:5000/nginx:1.13.1 \
--insecure-registry=true
...
* WARNING: Image "workstation.lab.example.com:5000/nginx:1.13.1" runs as the
'root' user which may not be permitted by your cluster administrator
...
```

The command raises an alert that the **nginx** container image uses the **root** user. By default, OpenShift runs containers as a random operating system user.

Notice that **user2**, as a developer in the project, can create resources such as deployment configurations and services.

7.3. Verify the deployment.

From the command line, execute the following command:

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

As mentioned previously, deployment fails because the container image needs the root user. The pod ends in either the **CrashLoopBackOff** or **Error** states. Wait until you see one of these error states:

NAME	READY	STATUS	RESTARTS	AGE
nginx-2-fd68t	0/1	Error	0	1m

8. Decrease the security restrictions for the specific project.

To run the container with privileged access, create a service account that allows pods to run using any operating system user.

Some of these actions need to be done by a user with project administrator privileges, and some of these actions need to be done by a user with cluster administrator privileges.

All commands in this step can be run or copied from the **configure-sc.sh** script in the **/home/student/D0280/labs/secure-resources** folder.

8.1. Log in as the **user1** user and select the **proj-user1** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc login -u user1 -p redhat
[student@workstation ~]$ oc project proj-user1
```

8.2. Create a service account.

This action requires a project administrator user.

From the existing terminal window, run the following command:

```
[student@workstation ~]$ oc create serviceaccount userroot
```

8.3. Log in as the **admin** user and select the **proj-user1** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc login -u admin -p redhat  
[student@workstation ~]$ oc project proj-user1
```

8.4. Associate the new service account with the **anyuid** security context.

This action requires a cluster administrator user.

Run the following command:

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z useroot
```

8.5. Log in as the **user2** user and select the **proj-user1** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc login -u user2 -p redhat  
[student@workstation ~]$ oc project proj-user1
```

8.6. Update the deployment configuration resource responsible for managing the **nginx** deployment.

This action can be done by any developer user.

Run the following command. It can be copied from the **configure-sc.sh** script in the **/home/student/D0280/labs/secure-resources** folder.

```
[student@workstation ~]$ oc patch dc/nginx --patch \  
  '{"spec": {"template": {"spec": {"serviceAccountName": "userroot"}}}}'
```

After the deployment configuration resource is patched, the deployment will succeed.



Note

You can alternatively use the **oc edit** command to change the **serviceAccountName** attribute of the deployment configuration. The **oc patch** command has the advantage of being scriptable, but the **oc edit** command is easier for interactive use.

8.7. Verify a new the pod is created using the updated deployment configuration.

Wait until the new pod status as running:

```
[student@workstation ~]$ oc get pods  
NAME          READY   STATUS    RESTARTS   AGE  
nginx-2-fd68t  1/1     Running   0          8m
```

9. Test the container.

9.1. Expose the service to enable external access to the nginx pod.

Run the following command:

```
[student@workstation ~]$ oc expose svc nginx
NAME      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx     172.30.138.65    <none>          80/TCP       14m
```

9.2. Connect to the **master** VM.

From the terminal window, run the following command:

```
[student@workstation ~]$ curl -s \
  http://nginx-proj-user1.cloudapps.lab.example.com
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
...

```

The expected output is the nginx welcome page.

10. Clean up.

10.1. Re-enable project creation for all regular users.

The command in this step can be run or copied from the **restore-policy.sh** script in the **/home/student/D0280/labs/secure-resources** folder.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u admin -p redhat
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
  self-provisioner system:authenticated system:authenticated:oauth
```

10.2. Delete the projects.

Run the following commands:

```
[student@workstation ~]$ oc delete project proj-user1
[student@workstation ~]$ oc delete project proj-user2
```

10.3.Delete the users.

Run the following commands:

```
[student@workstation ~]$ ssh root@master  
[root@master ~]$ htpasswd -D /etc/origin/openshift-passwd user1  
[root@master ~]$ htpasswd -D /etc/origin/openshift-passwd user2  
[root@master ~]$ exit
```

This concludes the guided exercise.

Managing Sensitive Information with Secrets

Objective

After completing this section, students should be able to create and apply secrets to manage sensitive information.

Secrets

The *Secret* object type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, Docker configuration files, and private source repository credentials. Secrets decouple sensitive content from pods. You can mount secrets onto containers using a volume plug-in or the system can use secrets to perform actions on behalf of a pod.

Features of Secrets

The main features of secrets include:

- Secret data can be referenced independently from its definition.
- Secret data volumes are backed by temporary file storage.
- Secret data can be shared within a namespace.

Creating a Secret

A secret is created before the pods that depend on that secret.

When creating secrets:

- Create a secret object with secret data.

```
[user@demo ~]$ oc create secret generic secret_name \
--from-literal=key1=secret1 \
--from-literal=key2=secret2
```

- Update the pod's service account to allow the reference to the secret. For example, to allow a secret to be mounted by a pod running under a specific service account:

```
[user@demo ~]$ oc secrets add --for=mount serviceaccount/serviceaccount-name \
secret/secret_name
```

- Create a pod that consumes the secret as an environment variable or as a file using a data volume. This is usually done using templates.

How Secrets are Exposed to Pods

Secrets can be mounted as data volumes or exposed as environment variables to be used by a container in a pod. For example, to expose a secret to a pod, first create a secret and assign values such as a *username* and *password* to key/value pairs, then assign the key name to the pod's YAML file *env* definition.

Take a secret named **demo-secret**, that defines the key **username** and set the key's value to the user **demo-user**:

```
[user@demo ~]$ oc create secret generic demo-secret \
--from-literal=username=demo-user
```

To use the previous secret as the database administrator password for a MySQL database pod, define the environment variable with a reference to the secret name and the key:

```
env:
- name: MYSQL_ROOT_PASSWORD
  valueFrom:
    secretKeyRef:
      key: username
      name: demo-secret
```

Managing Secrets from the Web Console

To manage secrets from the web console:

1. Log in to the web console as an authorized user.
2. Create or select a project to host the secret.
3. Navigate to **Resources > Secrets**.

Name	Type	Created
demo-secret	kubernetes.io/basic-auth	2 minutes ago

Figure 5.2: Manage Secrets using the web console

Use Cases for Secrets

Passwords and User Names

Sensitive information, such as passwords and user names, can be stored in a secret that is mounted as a data volume in a container. The data appears as content in files located in the data volume directory of the container. An application, such as a database, can then use these secrets to authenticate users.

Transport Layer Security (TLS) and Key Pairs

Securing communication to a service can be accomplished by having the cluster generate a signed certificate and key pair into a secret within the project's namespace. The certificate and key pair are stored using PEM format, in files such as **tls.crt** and **tls.key**, located in the secret's data volume of the pod.

ConfigMap Objects

ConfigMaps are similar to secrets, but are designed to support the ability to work with strings that do not contain sensitive information. The ConfigMap object holds key-value pairs of

configuration data that can be consumed in pods or used to store configuration data for system components such as controllers.

The ConfigMap object provides mechanisms to inject configuration data into containers. ConfigMaps store granular information, such as individual properties, or detailed information, such as entire configuration files or JSON blobs.

Creating a ConfigMap from the CLI

A ConfigMap object can be created from literal values using the **--from-literal** option.

The following command creates a ConfigMap object that assigns the IP address **172.20.30.40** to the ConfigMap key named **serverAddress**.

```
[user@demo ~]$ oc create configmap special-config \
--from-literal=serverAddress=172.20.30.40
```

Use the following command to view the configMap:

```
[user@demo ~]$ oc get configmaps special-config -o yaml
apiVersion: v1
data:
  key1: serverAddress=172.20.30.40
kind: ConfigMap
metadata:
  creationTimestamp: 2017-07-10T17:13:31Z
  name: special-config
  namespace: secure-review
  resourceVersion: "93841"
  selfLink: /api/v1/namespaces/secure-review/configmaps/special-config
  uid: 19418d5f-6593-11e7-a221-52540000fa0a
```

Populate the environment variable *APISERVER* inside a pod definition from the config map:

```
env:
  - name: APISERVER
    valueFrom:
      configMapKeyRef:
        name: special-config
        key: serverAddress
```

Managing ConfigMaps from the Web Console

To manage ConfigMap objects from the web console:

1. Log in to the web console as an authorized user.
2. Create or select a project to host the ConfigMap.
3. Navigate to **Resources > Config Maps**.

The screenshot shows the OpenShift Container Platform web interface. At the top, there is a navigation bar with a home icon, the project name "demo_project", an "Add to project" button, and a user profile "developer". Below the navigation bar is a sidebar with links: "Overview", "Applications >", "Builds >", and "Resources >". The "Resources" link is highlighted with a blue underline. The main content area is titled "Config Maps" with a "Learn More" link. It features a search bar labeled "Filter by label" and an "Add" button. A table header with columns "Name", "Created", and "Labels" is shown, followed by the message "No config maps to show".

Figure 5.3: Managing ConfigMaps using the web console



References

Further information is available in the Developer Guide of the OpenShift Container Platform 3.5 at

|| https://access.redhat.com/documentation/en-us/openshift_container_platform/3.5

Further information is also available at

|| OpenShift Container Platform 3.5 Documentation

|| <https://docs.openshift.com/container-platform/3.5/>

Guided Exercise: Protecting a Database Password

In this exercise, you will use a secret to encrypt the credentials to access a database container.

Resources	
Files:	/home/student/D0280/labs/secure-secrets/
Application URL:	NA

Outcomes

You should be able to create a MySQL database container that uses an OpenShift **Secret** resource type for storing database authentication credentials.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, then, on the **workstation** VM, run the lab setup script followed by the Ansible playbook:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** VMs are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab secure-secrets setup
```

Steps

1. Create a new project.

- 1.1. From the **workstation** VM, access the OpenShift master (<https://master.lab.example.com:8443>) with the OpenShift client.

Log in as **developer** and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443
```

- 1.2. Create the **secure-secrets** project:

```
[student@workstation ~]$ oc new-project secure-secrets
```

- 1.3. Inspect the copy of the **mysql-ephemeral** template in this exercise **labs** folder.

Run the following command from the terminal window:

```
[student@workstation ~]$ cd ~/D0280/labs/secure-secrets  
[student@workstation secure-secrets]$ less mysql-ephemeral.yml
```

Briefly review the `mysql-ephemeral.yml` file. It contains the `mysql-ephemeral` template from the `openshift` project, without the secret definition. The secret required by the template will be created manually during this exercise.

- 1.4. Inside the template, the MySQL database pod specification initializes environment variables from values stored in the secret:

```
...  
spec:  
  containers:  
    - capabilities: {}  
    env:  
      - name: MYSQL_USER  
        valueFrom:  
          secretKeyRef:  
            key: database-user  
            name: ${DATABASE_SERVICE_NAME}  
      - name: MYSQL_PASSWORD  
        valueFrom:  
          secretKeyRef:  
            key: database-password  
            name: ${DATABASE_SERVICE_NAME}  
      - name: MYSQL_ROOT_PASSWORD  
        valueFrom:  
          secretKeyRef:  
            key: database-root-password  
            name: ${DATABASE_SERVICE_NAME}  
...  
...
```

- 1.5. Other environment variables required by the pod are initialized from template parameters and have default values.

Notice, close to the end of the file, the default value for the `DATABASE_SERVICE_NAME` parameter, which is `mysql`. This parameter was used as the name of the service definition when initializing environment variables, as seen in the previous step.

```
...  
  - description: The name of the OpenShift Service exposed for the database.  
    displayName: Database Service Name  
    name: DATABASE_SERVICE_NAME  
    required: true  
    value: mysql  
...  
...
```

2. Create a secret containing the credentials used by the MySQL container image, as requested by the template. Give to the secret the name `mysql`.
 - The database user name for application access is defined by the `database-user` key.
 - The password for the database user is defined by the `database-password` key.
 - The database administrator password is defined by the `database-root-password` key.

2.1. Create a new secret.

From the terminal window, run the following command:

```
[student@workstation secure-secrets]$ oc create secret generic mysql \
--from-literal='database-user='mysql' \
--from-literal='database-password='redhat' \
--from-literal='database-root-password='do280-admin'
```

- 2.2. Inspect the new secret to verify the database user and database administrator passwords are not stored in clear text.

From the terminal window, run the following command:

```
[student@workstation secure-secrets]$ oc get secret mysql -o yaml
apiVersion: v1
data:
  database-password: cmVkaGF0
  database-root-password: ZG8yODAtYWRTaW4=
  database-user: bXlzcWw=
kind: Secret
...
```

3. Create a MySQL database container using the template stored in the YAML file.

From the terminal window, run the following command:

```
[student@workstation secure-secrets]$ oc new-app --file=mysql-ephemeral.yml
```

4. Wait until the MySQL pod is ready and running.

From the terminal window, run the following command:

```
[student@workstation secure-secrets]$ oc get pods
NAME      READY   STATUS    RESTARTS   AGE
mysql-1-cl1zq  1/1     Running   0          1m
```

5. Create a port forwarding tunnel to the MySQL pod. Use the pod name from the previous step.

From the terminal window, run the following commands:

```
[student@workstation secure-secrets]$ cd ~
[student@workstation ~]$ oc port-forward mysql-1-cl1zq 3306:3306
```

Do not kill the **oc port-forward** command. Leave it running because this is required by the next step.

6. Connect to the database to verify access with the credentials defined in the Secret object.

Open another terminal window to run the following commands:

```
[student@workstation ~]$ mysql -uroot -pdo280-admin -h127.0.0.1
...
MySQL [(none)]> show databases;
+-----+
```

```
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| sampledb        |
| sys            |
+-----+
...
```

The MySQL container validates the credentials provided in the `mysql` command against the encrypted credentials stored in the Secret object. Accessing the `root` account using the **do280-admin** password works because the Secret object was passed as a parameter when that container was created and started.

7. Exit the MySQL client and terminate the tunnel.

On the terminal window running the MySQL client, run the following command:

```
MySQL [(none)]> exit
```

Go to the terminal running the `oc port-forward` command and type **Ctrl+C** to terminate the port forwarding tunnel.

8. Clean up.

Run the following commands.

```
[student@workstation ~]$ oc delete project secure-secrets
```

This concludes the guided exercise.

Managing Security Policies

Objective

After completing this section, students should be able to manage security policies using the command-line interface.

OpenShift Container Platform Authorization

OpenShift defines two major groups of operations that a user can execute: project-related (also known as *local policy*), and administration-related (also known as *cluster policy*) operations. Because of the number of operations available for both policies, some operations were grouped together and defined as *roles*.

Default Roles	Description
cluster-admin	All users in this role can manage the OpenShift cluster.
cluster-status	All users in this role are provided read-only access to information about the cluster.

To add the cluster **role** to a user, use the **add-cluster-role-to-user** subcommand:

```
$ oc adm policy add-cluster-role-to-user cluster-role username
```

For example, to change a regular user to a cluster administrator, use the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

To remove the cluster **role** from a user, use the **remove-cluster-role-from-user** subcommand:

```
$ oc adm policy remove-cluster-role-from-user cluster-role username
```

For example, to change a cluster administrator to a regular user, use the following command:

```
$ oc adm policy remove-cluster-role-from-user cluster-admin username
```

OpenShift organizes a set of rules as a *role*. Rules are defined by a verb and a resource. For example, **create user** is a rule from OpenShift and it is part of a role named **cluster-admin**.

You can use the **oc adm policy who-can** command to identify the users and roles that can execute a role. For example:

```
$ oc adm policy who-can delete user
```

To manage local policies, the following roles are available:

Default Roles	Description
edit	Users in the role can create, change and delete common application resources

Default Roles	Description
	from the project, such as services and deployment configurations. But cannot act on management resources such as limit ranges and quotas, and cannot manage access permissions to the project.
basic-user	Users in the role have read access to the project.
self-provisioner	Users in the role can create new projects. This is a cluster role, not a project role.
admin	Users in the role can manage all resources in a project, including granting access to other users to the project.

The **admin** role gives a user access to project resources such as quotas and limit ranges, besides the ability to and create new applications. The **edit** role gives a user sufficient access to act as a developer inside the project, but working under the restraints configured by a project administrator.

You can add a specified user to a role in a project with the **add-role-to-user** subcommand. For example:

```
$ oc adm policy add-role-to-user role-name username -n project
```

For example, to add the user **dev** to the role **basic-user** in the **wordpress** project:

```
$ oc adm policy add-role-to-user basic-user dev -n wordpress
```

Security Context Constraints (SCCs)

OpenShift provides a security mechanism named security context constraints, which restricts access to resources, but not to operations in OpenShift.

SCC limits the access from a running pod in OpenShift to the host environment. SCC controls:

- Running privileged containers
- Requesting extra capabilities to a container
- Using host directories as volumes
- Changing the SELinux context of the container
- Changing the user ID

Some containers developed by the community may require relaxed security context constraints because they might need access to resources that are forbidden by default, for example, file systems, sockets or to access an SELinux context.

The SCC defined by OpenShift can be listed using the following command as a cluster administrator:

```
$ oc get scc
```

OpenShift has seven SCCs:

- **anyuid**
- **hostaccess**
- **hostmount-anyuid**
- **nonroot**
- **privileged**
- **restricted**

To get additional information about an SCC, use the **describe** verb from the **oc** command line.

```
$ oc describe scc anyuid
Name:      anyuid
Priority:   10
Access:
  Users:    <none>
  Groups:   system:cluster-admins
Settings:
  Allow Privileged:  false
  Default Add Capabilities:  <none>
  Required Drop Capabilities: MKNOD, SYS_CHROOT
  Allowed Capabilities:  <none>
  Allowed Volume Types:  configMap, downwardAPI, emptyDir, persistentVolumeClaim, secret
  Allow Host Network:  false
  Allow Host Ports:  false
  Allow Host PID:  false
  Allow Host IPC:  false
  Read Only Root Filesystem:  false
  Run As User Strategy: RunAsAny
    UID:    <none>
    UID Range Min:  <none>
    UID Range Max:  <none>
  SELinux Context Strategy: MustRunAs
    User:    <none>
    Role:    <none>
    Type:    <none>
    Level:   <none>
  FSGroup Strategy: RunAsAny
    Ranges:  <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:  <none>
```

All containers created by OpenShift use the SCC named **restricted**, which provides limited access to resources external to OpenShift.

For the **anyuid** security context, the **run as user** strategy is defined as **RunAsAny**, which means that the pod can run as any user ID available in the container. This allows containers that require a specific user to run the commands using a specific user ID.

To change the container to run using a different SCC, you need to create a service account bound to a pod. To create a service account, use the following command:

```
$ oc create serviceaccount service-account-name
```

To associate the service account with an SCC, use the following command:

```
$ oc adm policy add-scc-to-user SCC -z service-account
```

To identify which account can create a pod that requires elevated security requirements, use the **scc-subject-review** subcommand which will return all the security constraint context limitations that can be used to overcome the limitation of a container. To review:

```
$ oc export pod pod-name > output.yaml
$ oc adm policy scc-subject-review -f output.yaml
```

OpenShift and SELinux

OpenShift requires SELinux to be enabled on each host to provide safe access to resources using mandatory access control. Similarly, Docker containers managed by OpenShift need to manage SELinux contexts to avoid compatibility problems. To minimize the risk of containers running without SELinux support, the SELinux context strategy can be created.

In order to update the SELinux context, a new SCC can be generated by using an existing SCC as a starting point:

```
$ oc export scc restricted > custom_selinux.yml
```

Edit the YAML file to change the SCC name and the SELinux context:

```
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups:
- system:authenticated
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: restricted denies access to all host features and
    requires
      pods to be run with a UID, and SELinux context that are allocated to the
      namespace. This
        is the most restrictive SCC.
  creationTimestamp: null
  name: restricted①
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT
```

```
- SETUID
- SETGID
runAsUser:
  type: MustRunAsRange
seLinuxContext:❶
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- secret
```

❶ SCC name

❷ SELinux context type: Must be changed to **RunAsAny** to disable SELinux.

To create an SCC, run the following command:

```
$ oc create -f yaml_file
```

Privileged Containers

Some containers might need to access the runtime environment of the host. The S2I builder containers require access to the host docker daemon to build and run containers. For example, the S2I builder containers are a class of privileged containers that require access beyond the limits of their own containers. These containers can pose security risks because they can use any resources on an OpenShift node. SCCs can be used to enable access for privileged containers by creating service accounts with privileged access.

Quiz: Managing Security Policies

Choose the correct answers to the following questions:

When you have completed the quiz, click **check**. If you want to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which command removes the **cluster-admin** role from a user named **student**?
 - a. oc adm policy delete-cluster-role-from-user cluster-admin student
 - b. oc adm policy rm-cluster-role-from-user cluster-admin student
 - c. oc adm policy remove-cluster-role-from-user cluster-admin student
 - d. oc adm policy del-cluster-role-from-user cluster-admin student

2. Which command adds the **admin** role to the user **student** in a project named **example**?
 - a. oc adm policy add-role-to-user owner student -p example
 - b. oc adm policy add-role-to-user cluster-admin student -n example
 - c. oc adm policy add-role-to-user admin student -p example
 - d. oc adm policy add-role-to-user admin student -n example

3. Which command provides users in the **developers** group with read-only access to the **example** project?
 - a. oc adm policy add-role-to-group view developers -n example
 - b. oc adm policy add-role-to-group view developers -p example
 - c. oc adm policy add-role-to-group display developers -p example
 - d. oc adm policy add-role-to-user display developers -n example

4. Which command obtains a list of all users who can perform a **get** action on node resources?
 - a. oc adm policy who-can get
 - b. oc adm policy roles all
 - c. oc adm policy who-can get nodes
 - d. oc adm policy get nodes users

Solution

Choose the correct answers to the following questions:

When you have completed the quiz, click **check**. If you want to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which command removes the **cluster-admin** role from a user named **student**?
 - a. oc adm policy delete-cluster-role-from-user cluster-admin student
 - b. oc adm policy rm-cluster-role-from-user cluster-admin student
 - c. **oc adm policy remove-cluster-role-from-user cluster-admin student**
 - d. oc adm policy del-cluster-role-from-user cluster-admin student

2. Which command adds the **admin** role to the user **student** in a project named **example**?
 - a. oc adm policy add-role-to-user owner student -p example
 - b. oc adm policy add-role-to-user cluster-admin student -n example
 - c. oc adm policy add-role-to-user admin student -p example
 - d. **oc adm policy add-role-to-user admin student -n example**

3. Which command provides users in the **developers** group with read-only access to the **example** project?
 - a. **oc adm policy add-role-to-group view developers -n example**
 - b. oc adm policy add-role-to-group view developers -p example
 - c. oc adm policy add-role-to-group display developers -p example
 - d. oc adm policy add-role-to-user display developers -n example

4. Which command obtains a list of all users who can perform a **get** action on node resources?
 - a. oc adm policy who-can get
 - b. oc adm policy roles all
 - c. **oc adm policy who-can get nodes**
 - d. oc adm policy get nodes users

Lab: Controlling Access to OpenShift Resources

In this lab, you will enable users to access a project and deploy a MySQL database container using secrets.

Resources	
Files:	/home/student/D0280/labs/secure-review
Application URL:	http://phpmyadmin.cloudapps.lab.example.com

Outcomes

You should be able to create a project that deploys a MySQL database container that uses a **Secret** object type for storing database authentication credentials. You will also enable users to create and access projects.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be complete and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, then, on the **workstation** VM, run the lab setup script followed by the Ansible playbook:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** VMs are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab secure-review setup
```

Steps

1. Create a user account with the following details:

- *login*: user-review
- *password*: redhat

Recall from previous labs that the classroom is configured with the OpenShift **HTPasswdIdentityProvider** module.

- 1.1. Access the **master** VM from the **workstation** VM using SSH.
- 1.2. Create the **user-review** user in OpenShift.
- 1.3. Log out of the **master** VM.
2. Disable project creation capabilities for all regular users.
- 2.1. On the **workstation** VM, log in to OpenShift as the **admin** user. The password for the OpenShift **admin** user is **redhat**.

- 2.2. Remove the capability to create projects for all regular users. The cluster role for auto provisioning is **self-provisioner**.
 3. Verify that regular users cannot create projects in OpenShift.
 - 3.1. Log in to OpenShift as the **user-review** user.
 - 3.2. Try to create a new project. It should fail.
 4. Create a project named **secure-review**.
 - 4.1. Log in to OpenShift as the **admin** user.
 - 4.2. Create a new project named **secure-review**.
 5. Associate the **user-review** user with the **secure-review** project.
 - 5.1. Add the **user-review** user as a developer user for the **secure-review** project.
 - 5.2. Test the access.
 6. A template is provided to deploy the database to be used by the **phpmyadmin**. Inspect the template inside the **mysql-ephemeral.yml** file to find the name of the secret to be created and the keys to define inside the secret.
 7. Use the **user-review** developer user to create a secret named **mysql**. The secret should store the user name **mysql**, the password **redhat**, and database administrator password **do280-admin**.
- The database user name is defined by the **database-user** key. The password for this user is defined by the **database-password** key.
- The database administrator password is defined by the **database-root-password** key.
- 7.1. Create the secret.
 - 7.2. Verify the secret was created.
 8. Create a MySQL database container using the template.
 - 8.1. Deploy the MySQL database server using the template in the YAML file.
 - 8.2. Wait until the MySQL server pod is ready and running
 9. Test access to the database server using the **mysql** database user.
 - 9.1. Create a port-forwarding tunnel to access the database.
 - 9.2. Access the container using the **mysql** command as the **mysql** user with the **redhat** password.
 - 9.3. Exit the MySQL database client using the **exit** command and terminate the port-forwarding tunnel using **Ctrl+C**.
 10. Deploy the **phpmyadmin:4.7** container. The container is available in the **workstation.lab.example.com** registry which is an insecure registry.

The **phpmyadmin:4.7** container requires the environment variable named **PMA_HOST** to provide the IP address of the MySQL Server. Use the service FQDN for the MySQL server pod created using the template, which is **mysql.secure-review.svc.cluster.local**.

- 10.1. Deploy the **phpmyadmin** application from the container image.
- 10.2. Verify that the deployment failed because of the default OpenShift security policy.
11. Decrease the security restrictions for the project.

To enable the container to run with root privileges, create a service account with root support.

 - 11.1. Log in as the **admin** user which is a cluster administrator.
 - 11.2. Create a service account named **phpmyadmin-account**.
 - 11.3. Associate the new service account with the **anyuid** security context.
 - 11.4. Update the deployment configuration resource responsible for managing the **phpmyadmin** deployment to use the newly created service account. You can use either the **oc patch** or the **oc edit** commands.

You can copy the **oc patch** command from the **patch-dc.sh** script in the **/home/student/D0280/labs/secure-review** folder.
 - 11.5. Log in back to the developer user created for this project.
 - 11.6. Wait for the new **phpmyadmin** pod to be ready and running.
12. Test the application using a web browser.
 - 12.1. Create a route for the **phpmyadmin** service. Use the following host name: **phpmyadmin.cloudapps.lab.example.com**.
 - 12.2. Access the welcome page for the **phpmyadmin** application with a web browser, using **mysql** as the login, **redhat** as the password.

The setup page from **phpmyadmin** is the expected output.
13. Run the grading script to verify that all the tasks were completed.

Run the **lab secure-review grade** command to verify that all the tasks were accomplished.

[student@workstation ~]\$ lab secure-review grade
14. Clean up.
 - 14.1. Re-enable the project creation role for all regular users.
 - 14.2. Delete the project.
 - 14.3. Delete the **user-review** user from the HTPasswd file:

This concludes the lab.

Solution

In this lab, you will enable users to access a project and deploy a MySQL database container using secrets.

Resources	
Files:	/home/student/D0280/labs/secure-review
Application URL:	http://phpmyadmin.cloudapps.lab.example.com

Outcomes

You should be able to create a project that deploys a MySQL database container that uses a **Secret** object type for storing database authentication credentials. You will also enable users to create and access projects.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be complete and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, then, on the **workstation** VM, run the lab setup script followed by the Ansible playbook:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** VMs are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab secure-review setup
```

Steps

1. Create a user account with the following details:

- *login*: user-review
- *password*: redhat

Recall from previous labs that the classroom is configured with the OpenShift **HTPasswdIdentityProvider** module.

- 1.1. Access the **master** VM from the **workstation** VM using SSH.

Open a terminal on the **workstation** VM, then run the following commands:

```
[student@workstation ~]$ ssh root@master
[root@master ~]#
```

- 1.2. Create the **user-review** user in OpenShift.

In the existing terminal window, run the following command:

```
[root@master ~]# htpasswd /etc/origin/openshift-passwd user-review
```

When prompted, enter **redhat** as the password.

1.3. Log out of the **master** VM.

In the terminal window, run the following command:

```
[root@master ~]# exit
```

2. Disable project creation capabilities for all regular users.

2.1. On the **workstation** VM, log in to OpenShift as the **admin** user. The password for the OpenShift **admin** user is **redhat**.

From the terminal window, run the following command:

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

Acknowledge that you accept insecure connections, if prompted.

2.2. Remove the capability to create projects for all regular users. The cluster role for auto provisioning is **self-provisioner**.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
self-provisioner system:authenticated system:authenticated:oauth
```

3. Verify that regular users cannot create projects in OpenShift.

3.1. Log in to OpenShift as the **user-review** user.

From the terminal window, run the following command:

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

3.2. Try to create a new project. It should fail.

Run the following command:

```
[student@workstation ~]$ oc new-project test
Error from server (Forbidden): You may not request a new project via this API.
```

Due to the change in the security policy, the user cannot create a new project. This task is delegated to the OpenShift cluster administrator.

4. Create a project named **secure-review**.

4.1. Log in to OpenShift as the **admin** user.

From the terminal window, run the following command:

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

4.2. Create a new project named **secure-review**.

Run the following command:

```
[student@workstation ~]$ oc new-project secure-review
```

5. Associate the **user-review** user with the **secure-review** project.

5.1. Add the **user-review** user as a developer user for the **secure-review** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc project secure-review
[student@workstation ~]$ oc policy add-role-to-user edit user-review
```

5.2. Test the access.

Log in as the **user-review** user and verify that this user can access the **secure-review** project.

Log in to OpenShift as the **user-review** user.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u user-review -p redhat
[student@workstation ~]$ oc project secure-review
```

6. A template is provided to deploy the database to be used by the **phpmyadmin**. Inspect the template inside the **mysql-ephemeral.yml** file to find the name of the secret to be created and the keys to define inside the secret.

Run the following command from the terminal window:

```
[student@workstation ~]$ cd ~/DO280/labs/secure-review/
[student@workstation secure-review]$ less mysql-ephemeral.yml
```

Notice a few environment variables are initialized from a secret named from the **DATABASE_SERVICE_NAME** parameter. The default value for the parameter is **mysql**.

```
...
spec:
  containers:
    - capabilities: {}
      env:
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              key: database-user
              name: ${DATABASE_SERVICE_NAME}
    ...
    - description: The name of the OpenShift Service exposed for the database.
      displayName: Database Service Name
      name: DATABASE_SERVICE_NAME
```

```
required: true
value: mysql
...
```

The following secret keys are required to initialize environment variables: **database-password**, **database-root-password**, and **database-user**.

7. Use the **user-review** developer user to create a secret named **mysql**. The secret should store the user name **mysql**, the password **redhat**, and database administrator password **do280-admin**.

The database user name is defined by the **database-user** key. The password for this user is defined by the **database-password** key.

The database administrator password is defined by the **database-root-password** key.

7.1. Create the secret.

From the terminal window, run the following command:

```
[student@workstation secure-review]$ oc create secret generic mysql \
--from-literal='database-user='mysql' \
--from-literal='database-password='redhat' \
--from-literal='database-root-password='do280-admin'
```

7.2. Verify the secret was created.

From the terminal window, run the following command:

```
[student@workstation secure-review]$ oc get secret mysql -o yaml
```

8. Create a MySQL database container using the template.

8.1. Deploy the MySQL database server using the template in the YAML file.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc new-app --file=mysql-ephemeral.yaml
```

8.2. Wait until the MySQL server pod is ready and running

From the terminal window, run the following command:

```
[student@workstation secure-secret]$ oc get pods
NAME          READY     STATUS    RESTARTS   AGE
mysql-1-s6gxf  1/1      Running   0          1m
```

9. Test access to the database server using the **mysql** database user.

9.1. Create a port-forwarding tunnel to access the database.

From the terminal window, run the following command:

```
[student@workstation secure-secret]$ cd ~
[student@workstation ~]$ oc port-forward mysql-1-s6gxf 3306:3306
```

- 9.2. Access the container using the **mysql** command as the **mysql** user with the **redhat** password.

Open another command line window to execute the following command, that connect to the database:

```
[student@workstation ~]$ mysql -umysql -predhat -h127.0.0.1
...
MySQL [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| sampledb       |
+-----+
...
```

- 9.3. Exit the MySQL database client using the **exit** command and terminate the port-forwarding tunnel using **Ctrl+C**.
10. Deploy the **phpmyadmin:4.7** container. The container is available in the **workstation.lab.example.com** registry which is an insecure registry.

The **phpmyadmin:4.7** container requires the environment variable named **PMA_HOST** to provide the IP address of the MySQL Server. Use the service FQDN for the MySQL server pod created using the template, which is **mysql.secure-review.svc.cluster.local**.

- 10.1. Deploy the **phpmyadmin** application from the container image.

Run the following command:

```
[student@workstation ~]$ oc new-app --name=phpmyadmin \
--docker-image=workstation.lab.example.com:5000/phpmyadmin:4.7 \
--insecure-registry=true \
-e PMA_HOST=mysql.secure-review.svc.cluster.local
```

The command raises an alert that it requires root privileges. By default, OpenShift does not support running containers as the **root** operating system user.

- 10.2. Verify that the deployment failed because of the default OpenShift security policy.

From the command line, execute the following command:

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

As mentioned previously, without root privileges the deployment process will fail. The expected output shows the **phpmyadmin** application with status **Error** or **CrashLoopBackOff**:

NAME	READY	STATUS	RESTARTS	AGE
mysql-1-s6gxf	1/1	Running	0	23m
phpmyadmin-1-ttgp5	0/1	CrashLoopBackOff	5	5m

11. Decrease the security restrictions for the project.

To enable the container to run with root privileges, create a service account with root support.

11.1. Log in as the **admin** user which is a cluster administrator.

From the terminal window, run the following command:

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

11.2. Create a service account named **phpmyadmin-account**.

From the existing terminal window, run the following command:

```
[student@workstation ~]$ oc create serviceaccount phpmyadmin-account
```

11.3. Associate the new service account with the **anyuid** security context.

Run the following command:

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid \
-z phpmyadmin-account
```

11.4. Update the deployment configuration resource responsible for managing the **phpmyadmin** deployment to use the newly created service account. You can use either the **oc patch** or the **oc edit** commands.

You can copy the **oc patch** command from the **patch-dc.sh** script in the **/home/student/D0280/labs/secure-review** folder.

Run the following command:

```
[student@workstation ~]$ oc patch dc/phpmyadmin --patch \
'{"spec":{"template":{"spec":{"serviceAccountName": "phpmyadmin-account"}}}}'
"phpmyadmin" patched
```

After the change, a new deployment executes automatically.

11.5. Log in back to the developer user created for this project.

From the terminal window, run the following commands:

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

11.6. Wait for the new **phpmyadmin** pod to be ready and running.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
mysql-1-s6gxf  1/1     Running   0          25m
phpmyadmin-2-g4wb4 1/1     Running   0          46s
```

12. Test the application using a web browser.

- 12.1. Create a route for the **phpmyadmin** service. Use the following host name: **phpmyadmin.cloudapps.lab.example.com**.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc expose svc/phpmyadmin \
--hostname=phpmyadmin.cloudapps.lab.example.com
route "phpmyadmin" exposed
```

- 12.2. Access the welcome page for the **phpmyadmin** application with a web browser, using **mysql** as the login, **redhat** as the password.

On the workstation VM, open a web browser and navigate to the URL:

http://phpmyadmin.cloudapps.lab.example.com

The setup page from **phpmyadmin** is the expected output.

13. Run the grading script to verify that all the tasks were completed.

Run the **lab secure-review grade** command to verify that all the tasks were accomplished.

```
[student@workstation ~]$ lab secure-review grade
```

14. Clean up.

- 14.1. Re-enable the project creation role for all regular users.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u admin -p redhat
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
self-provisioner system:authenticated system:authenticated:oauth
```

- 14.2. Delete the project.

Run the following command:

```
[student@workstation ~]$ oc delete project secure-review
```

- 14.3. Delete the **user-review** user from the HTPasswd file:

Run the following command:

```
[student@workstation ~]$ ssh root@master htpasswd -D \
/etc/origin/openshift-passwd user-review
```

This concludes the lab.

Summary

In this chapter, you learned:

- A Kubernetes namespace provides a way of grouping a set of related resources together in a cluster. A project is a Kubernetes namespace that allows a group of authorized users to organize and manage project resources in isolation from other groups.
- Cluster administrators can create projects and delegate administrative rights for the project to any user. Administrators can give users access to certain projects, allow them to create their own projects, and give them administrative rights within individual projects.
- The authentication layer identifies the user associated with requests to the OpenShift Container Platform API. The authorization layer then uses information about the requesting user to determine if the request should be allowed.
- OpenShift provides security context constraints (SCCs) that control the actions a pod can perform and what resources it can access. By default, when a container is created it has only the capabilities defined by the restricted SCC.

The **oc get scc** command lists the available SCCs.

The **oc describe scc** command displays a detailed description of a security context constraint.

- The **Secret** object type provides a mechanism for holding sensitive information, such as passwords, OpenShift Container Platform client configuration files, dockercfg files, and private source repository credentials. Secrets decouple sensitive content from pods. You can mount secrets onto containers using a volume plug-in or the system can use secrets to perform actions on behalf of a pod.
- **ConfigMaps** are similar to secrets, but are designed to support working with strings that do not contain sensitive information.
- OpenShift defines two major groups of operations that users can execute: project-related (also known as **local policy**) and administration-related (also known as **cluster policy**) operations.
- OpenShift requires that SELinux be enabled on each host to provide safe access to resources using mandatory access control. Similarly, Docker containers managed by OpenShift need to manage SELinux contexts to avoid compatibility problems.



CHAPTER 6

ALLOCATING PERSISTENT STORAGE

Overview	
Goal	Implement persistent storage.
Objectives	<ul style="list-style-type: none">Provision persistent storage for use by applications.Configure the internal container registry for persistence.
Sections	<ul style="list-style-type: none">Provisioning Persistent Storage (and Guided Exercise)Configuring the OCP Internal Registry for Persistence (and Guided Exercise)
Lab	Allocating Persistent Storage

Provisioning Persistent Storage

Objective

After completing this section, students should be able to provision persistent storage for use by applications.

Persistent Storage

By default, running containers use *ephemeral* storage within the container. Pods consist of one or more containers that are deployed together, share the same storage and other resources, and can be created, started, stopped, or destroyed at any time. Using ephemeral storage means that data written to the file system within the container is lost when the container is stopped.

When deploying applications that require persistent data when the container is stopped, OpenShift uses Kubernetes persistent volumes (PVs) to provision persistent storage for pods.

Use Case for Persistent Storage

Consider a database container that uses the default ephemeral storage provided when the pod is started. If the database pod is destroyed and recreated, the ephemeral storage is destroyed and the data lost. If persistent storage is used, the database stores data to a persistent volume that is external to the pod. If the pod is destroyed and recreated, the database application continues to access the same external storage where the data was stored.

Providing Persistent Storage for an Application

Persistent volumes are OpenShift resources that are created and destroyed only by an OpenShift administrator. A persistent volume resource represents network-attached storage accessible to all OpenShift nodes.

Persistent Storage Components

OpenShift Container Platform uses the Kubernetes *persistent volume (PV)* framework to allow administrators to provision persistent storage for a cluster. A *persistent volume claim (PVC)* is used by developers to request PV resources without having specific knowledge of the underlying storage infrastructure.

Persistent Volume

A PV is a resource in the OpenShift cluster, defined by a **PersistentVolume** API object, which represents a piece of existing networked storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs have a life cycle independent of any individual pod that uses the PV.

Persistent Volume Claim

PVCs are defined by a **PersistentVolumeClaim** API object, which represents a request for storage by a developer. It is similar to a pod in that pods consume node resources and PVCs consume PV resources.

OpenShift-supported Plug-ins for Persistent Storage

Volumes are mounted file systems that are available to pods and their containers, and can be backed by a number of local or network-attached storage endpoints. OpenShift uses plug-ins to support the following different back ends for persistent storage:

- NFS

- GlusterFS
- OpenStack Cinder
- Ceph RBD
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Fibre Channel

Persistent Volume Access Modes

A PV can be mounted on a host in any way supported by the resource provider. Providers have different capabilities and each PV's access modes are set to the specific modes supported by that particular volume. For example, NFS can support multiple read/write clients, but a specific NFS PV might be exported on the server as read-only. Each PV receives its own set of access modes describing that specific PV's capabilities.

The following table describes the access modes:

Access Mode	CLI Abbreviation	Description
ReadWriteOnce	RWO	The volume can be mounted as read/write by a single node.
ReadOnlyMany	ROX	The volume can be mounted read-only by many nodes.
ReadWriteMany	RWX	The volume can be mounted as read/write by many nodes.

PV claims are matched to volumes with similar access modes. The only two matching criteria are access modes and size. A claim's access modes represent a request. Therefore, users can be granted greater, but never lesser access. For example, if a claim requests RWO, but the only volume available was an NFS PV (RWO+ROX+RWX), the claim would match NFS because it supports RWO.

All volumes with the same modes are grouped, and then sorted by size (smallest to largest). The service on the master responsible for binding the PV to the PVC receives the group with matching modes and iterates over each (in size order) until one size matches, then binds the PV to the PVC.

Persistent Volume Storage Classes

PV Claims can optionally request a specific storage class by specifying its name in the **storageClassName** attribute. Only PVs of the requested class with the same storage class name as the PVC, can be bound to the PVC. The cluster administrator can set a default storage class for all PVCs or configure dynamic provisioners to service one or more storage classes that will match the specifications in an available PVC.

Creating PVs and PVC Resources

PVs are resources in the cluster. PVCs are requests for those resources and also act as claim checks to the resource. The interaction between PVs and PVCs have the following life cycle:

Create the Persistent Volume

A cluster administrator creates any number of PVs, which represent the details of the actual storage that is available for use by cluster users through the OpenShift API.

Define a Persistent Volume Claim

A user creates a PVC with a specific amount of storage and with certain access modes and optional storage classes. The master watches for new PVCs and either finds a matching PV or waits for a provisioner for the storage class to create one, then binds them together.

Use Persistent Storage

Pods use claims as volumes. The cluster inspects the claim to find the bound volume and mounts that volume for a pod. For those volumes that support multiple access modes, the user specifies which mode is desired when using their claim as a volume in a pod.

Once a user has a claim and that claim is bound, the bound PV belongs to the user for as long as they need it. Users schedule pods and access their claimed PVs by including a persistent volume claim in their pod's **volumes** block.

Using NFS for Persistent Volumes

OpenShift runs containers using random UIDs, therefore mapping Linux users from OpenShift nodes to users on the NFS server does not work as intended. NFS shares used as OpenShift PVs must be configured as follows:

- Owned by the **nfsnobody** user and group.
- Having **rwx-----** permissions (expressed as 0700 using octal).
- Exported using the **all_squash** option.



Note

Using the **all_squash** option is not required when using supplemental groups as described later in this section.

For example, the following is an **/etc(exports** entry:

```
/var/export/vol *(rw,async,all_squash)
```

Other NFS export options, for example **sync** and **async**, do not matter to OpenShift; OpenShift works if either option is used. In high-latency environments, however, adding the **async** option facilitates faster write operations to the NFS share (for example, image pushes to the registry). Using the **async** option is faster because the NFS server replies to the client as soon as the request is processed, without waiting for the data to be written to disk. When using the **sync** option the behavior is the opposite; the NFS server replies to the client only after the data has been written to disk.



Important

The NFS share file system size and user quotas have no effect on OpenShift. A PV size is specified in the PV resource definition. If the actual file system is smaller, the PV is created and bound anyway. If the PV is larger, OpenShift does not limit used space to the specified PV size, and allows containers to use all free space on the file system. OpenShift offers storage quotas and storage placement restrictions that can be used to control resource allocation in projects.

Default SELinux policies do not allow containers to access NFS shares. The policy must be changed in every OpenShift instance node by setting the `virt_use_nfs` and `virt_sandbox_use_nfs` variables to `true`. These flags are automatically configured by the OpenShift installer:

```
# setsebool -P virt_use_nfs=true
# setsebool -P virt_sandbox_use_nfs=true
```

Reclamation Policies: Recycling

NFS implements the OpenShift Container Platform *Recyclable* plug-in interface. Automatic processes handle reclamation tasks based on policies set on each persistent volume.

By default, persistent volumes are set to *Retain*. The *Retain* reclaim policy allows for manual reclamation of the resource. When the persistent volume claim is deleted, the persistent volume still exists and the volume is considered *released*. But it is not yet available for another claim because the data from the previous claim remains on the volume. However, an administrator can manually reclaim the volume.

NFS volumes with their reclamation policy set to *Recycle* are scrubbed after being released from their claim. For example, when the reclamation policy is set to *Recycle* on an NFS volume, the command `rm -rf` is ran on the volume after the user's persistent volume claim bound to the volume is deleted. After it has been recycled, the NFS volume can be bound to a new claim.

Using Supplemental Groups for File-based Volumes

Supplemental groups are regular Linux groups. When a process runs in Linux, it has a UID, a GID, and one or more supplemental groups. These attributes can be set for a container's main process. The supplemental group IDs are typically used for controlling access to shared storage, such as NFS and GlusterFS, whereas `fsGroup` is used for controlling access to block storage, such as Ceph RBD and iSCSI.

OpenShift shared storage plug-ins mount volumes such that the POSIX permissions on the mount match the permissions on the target storage. For example, if the target storage's owner ID is 1234 and its group ID is 5678, then the mount on the host node and in the container will have those same IDs. Therefore, the container's main process must match one or both of those IDs in order to access the volume.

For example, on **node1** VM:

```
[root@node1 ~]# showmount -e
Export list for master.lab.example.com:
```

```
/var/export/nfs-demo *
```

From the NFS server on the **master** VM:

```
[root@master ~]# cat /etc/exports.d/nfs-demo.conf
/var/export/nfs-demo
...
[root@master ~]# ls -lZ /var/export -d
drwx-----. 10000000 650000 unconfined_u:object_r:usr_t:s0 /var/export/nfs-demo
```

In the above example, the **/var/export/nfs-demo** export is accessible by UID 10000000 and the group 650000. In general, containers should not run as root. In this NFS example, containers that are not run as UID 10000000 and are not members of the group 650000 do not have access to the NFS export.

Using FS Groups for Block Storage-based Volumes

For file-system groups, an *fsGroup* defines a pod's "file-system group" ID, which is added to the container's supplemental groups. The supplemental group ID applies to shared storage, whereas the *fsGroup* ID is used for block storage.

Block storage, such as Ceph RBD, iSCSI, and various types of cloud storage, is typically dedicated to a single pod. Unlike shared storage, block storage is taken over by a pod, meaning that user and group IDs supplied in the pod definition (or image) are applied to the actual, physical block device. Block storage is normally not shared.

SELinux and Volume Security

All predefined security context constraints, except for the privileged security context constraint, set the **seLinuxContext** to **MustRunAs**. The security context constraints most likely to match a pod's requirements force the pod to use an SELinux policy. The SELinux policy used by the pod can be defined in the pod itself, in the image, in the security context constraint, or in the project (which provides the default).

SELinux labels can be defined in a pod's **securityContext.seLinuxOptions** section, and supports **user**, **role**, **type**, and **level** labels.

SELinuxContext Options

MustRunAs

Requires **seLinuxOptions** to be configured if not using pre-allocated values. Uses **seLinuxOptions** as the default. Validates against **seLinuxOptions**.

RunAsAny

No default provided. Allows any **seLinuxOptions** to be specified.



References

Additional information about configuring persistent storage is available in the *Installation and Configuration* document which can be found at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Additional information about resource quotas is available in the *Cluster Administration* document which can be found at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Additional information about persistent volumes is available in the *OpenShift Container Platform Developer Guide* which can be found at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Additional information about persistent storage concepts is available in the *OpenShift Container Platform Architecture Guide* which can be found at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Implementing Persistent Database Storage

In this exercise, you will configure persistent storage for a MySQL database server pod.

Resources	
Files:	/root/D0280/labs/deploy-volume/ (master)
	/home/student/D0280/labs/deploy-volume/ (workstation)

Outcomes

You should be able to configure an NFS share on the OpenShift **master** VM to provide storage for OpenShift nodes, and OpenShift persistent volumes bound to a database pod.

Before you begin

All labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you must have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, and on the **workstation** VM, run the lab setup script followed by the Ansible playbook:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab deploy-volume setup
```

Steps

- Configure an NFS share on the **master** VM. This share is used as an OpenShift Container Platform persistent volume.

- Log in to the **master** VM as the **root** user:

```
[student@workstation ~]$ ssh root@master
```

- The **config-nfs.sh** script is available to automate the configuration of the NFS share for the persistent volume. Prior to running the script it can be viewed at **/root/D0280/labs/deploy-volume/config-nfs.sh** to become familiar with the process for configuring the NFS share:

```
[root@master ~]$ less -Fxi /root/D0280/labs/deploy-volume/config-nfs.sh
... output omitted ...
```

- Run the script:

```
[root@master ~]$ /root/D0280/labs/deploy-volume/config-nfs.sh
```

-
- 1.4. Verify that the **/var/export/dbvol** share is included in the export list on the OpenShift **master** VM.

```
[root@master ~]# showmount -e  
Export list for master.lab.example.com:  
... output omitted ...  
/var/export/dbvol *
```

- 1.5. Log out of the **master** VM:

```
[root@master ~]# exit  
[student@workstation ~]$
```

2. Verify that both the **node1** and **node2** hosts can access the NFS exported volume from the **master** VM.

- 2.1. Log in to the **node1** host as the **root** user:

```
[student@workstation ~]$ ssh root@node1
```

- 2.2. Confirm that the **node1** host can access the NFS share on the OpenShift master VM:

```
[root@node1 ~]# mount -t nfs master.lab.example.com:/var/export/dbvol /mnt
```

- 2.3. Verify that the file system has the correct permissions from **node1**:

```
[root@node1 ~]# ls -la /mnt ; mount | grep /mnt  
total 0  
drwx----- 2 nfsnobody nfsnobody 6 Jul 18 02:07 .  
dr-xr-xr-x. 17 root root 224 Jun 5 08:54 ..  
master.lab.example.com:/var/export/dbvol on /mnt type nfs4  
(rw,relatime,vers=4.0,rsize=262144,wsize=262144,namlen=255,hard,  
proto=tcp,port=0,timeo=600,retrans=2,sec=sys,  
clientaddr=172.25.250.11,local_lock=none,addr=172.25.250.10)
```

- 2.4. Unmount the NFS share:

```
[root@node1 ~]# umount /mnt
```

- 2.5. Log out of the **node1** host:

```
[root@node1 ~]# exit  
[student@workstation ~]$
```

- 2.6. Log in to the **node2** host as **root** user:

```
[student@workstation ~]$ ssh root@node2
```

- 2.7. Confirm that the **node2** host can access the NFS share on the OpenShift master VM:

```
[root@node2 ~]# mount -t nfs master.lab.example.com:/var/export/dbvol /mnt
```

2.8. Verify that the file system has the correct permissions from the node2:

```
[root@node2 ~]# ls -la /mnt ; mount | grep /mnt
total 0
drwx----- 2 nfsnobody nfsnobody 6 Jul 18 02:07 .
dr-xr-xr-x. 17 root      root    224 Jun  5 08:54 ..
master.lab.example.com:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.0,rsize=262144,wsize=262144,namlen=255,hard,
proto=tcp,port=0,timeo=600,retrans=2,sec=sys,
clientaddr=172.25.250.12,local_lock=none,addr=172.25.250.10)
```

2.9. Unmount the NFS share:

```
[root@node2 ~]# umount /mnt
```

2.10. Log out of the **node2** host:

```
[root@node2 ~]# exit
[student@workstation ~]$
```



Note

The NFS share is automatically mounted by OpenShift when needed.

3. On the **workstation** host, log in to OpenShift as the **admin** user, and create a persistent volume to be used by the MySQL database pod.

3.1. Log in as the **admin** user:

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

3.2. Inspect the persistent volume definition:

```
[student@workstation ~]$ less -FiX \
~/DO280/labs/deploy-volume/mysqlDb-volume.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysqlDb-volume
spec:
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /var/export/dbvol
    server: master.lab.example.com
```

-
- 3.3. Create the persistent volume using the provided YAML resource definition file:

```
[student@workstation ~]$ oc create -f \
~/DO280/labs/deploy-volume/mysqlDb-volume.yml
persistentvolume "mysqlDb-volume" created
```

- 3.4. Verify that the persistent volume is available to be claimed by projects:

```
[student@workstation ~]$ oc get pv
NAME          CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS      ...
mysqlDb-volume  3Gi       RWX         Retain        Available   ...
```

4. On the **workstation** host, log in to OpenShift as the **developer** user, and create a new project named **persistent-storage**.
- 4.1. Log in as the **developer** user, using **openshift** as the password. If prompted, accept the insecure connections.

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 4.2. Create a new project named **persistent-storage**:

```
[student@workstation ~]$ oc new-project persistent-storage
```

5. Use the **oc new-app** command to create a new application named **mysqlDb**:

```
[student@workstation ~]$ oc new-app --name=mysqlDb \
--docker-image=workstation.lab.example.com:5000/rhscl/mysql-57-rhel7 \
--insecure-registry \
-e MYSQL_USER=ose \
-e MYSQL_PASSWORD=openshift \
-e MYSQL_DATABASE=quotes
... output omitted ...
--> Creating resources ...
imagestream "mysqlDb" created
deploymentconfig "mysqlDb" created
service "mysqlDb" created
--> Success
Run 'oc status' to view your app.
```

6. Verify the successful deployment of the **mysqlDb** application, and then modify the deployment configuration to use a persistent volume by creating a persistent volume claim.

- 6.1. Verify the successful deployment of the **mysqlDb** application:

```
[student@workstation ~]$ oc status
In project persistent-storage on server https://master.lab.example.com:8443

svc/mysqlDb - 172.30.144.48:3306
dc/mysqlDb deploys istag/mysqlDb:latest
    deployment #1 deployed 34 seconds ago - 1 pod
```

```
View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.
```

- 6.2. Use the **oc describe pod** command to confirm that the name of the volume is **mysqldb-volume-1** and that its type is currently set to **EmptyDir**:

```
[student@workstation ~]$ oc describe pod mysqldb | grep -A 2 'Volumes'
Volumes:
  mysqldb-volume-1:
    Type: EmptyDir (a temporary directory that shares a pod's lifetime)
```

- 6.3. Use the **oc set volume** command to modify the deployment configuration and create a persistent volume claim:

```
[student@workstation ~]$ oc set volume dc/mysqldb \
  --add --overwrite --name=mysqldb-volume-1 -t pvc \
  --claim-name=mysqldb-pvclaim \
  --claim-size=3Gi \
  --claim-mode='ReadWriteMany'
persistentvolumeclaims/mysqldb-pvclaim
deploymentconfig "mysqldb" updated
```

- 6.4. Use the **oc describe pod** command to confirm that the pod is now using a persistent volume. The command output should display *Volumes* as **mysqldb-volume-1**, *Type* as **PersistentVolumeClaim**, and *ClaimName* as **mysqldb-pvclaim**.

```
[student@workstation ~]$ oc describe pod mysqldb | \
  grep -E -A 2 'Volumes|ClaimName'
Volumes:
  mysqldb-volume-1:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the
         same namespace)
    ClaimName: mysqldb-pvclaim
    ReadOnly: false
    default-token-vldqb:
```

7. On the **workstation** host, verify that the persistent volume claim **mysqldb-pvclaim** is bound to the persistent volume named **mysqldb-volume**:

```
[student@workstation ~]$ oc get pvc
NAME           STATUS  VOLUME      CAPACITY  ACCESSMODES  AGE
mysqldb-pvclaim Bound   mysqldb-volume  3Gi       RWX          15m
```

8. From the **workstation** host, populate the database using the SQL file available at **/home/student/D0280/labs/deploy-volume/quote.sql**. Use the **oc port-forward** command to forward local port 3306 to pod port 3306. Use the **mysql** command to populate the database.
- 8.1. Open two terminals. From the first one, run the **oc get pods** command to retrieve the status of the pods. Ensure that the **mysqldb** pod is ready and running. Run the **oc port-forward** command to forward the local port 3306 to the pod port 3306. The command keeps the port open until the connection is manually terminated.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql-2-k2vlh 1/1     Running   0          1h
```

```
[student@workstation ~]$ oc port-forward mysql-2-k2vlh 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
```

- 8.2. From the second terminal, populate the **quotes** database by using the provided **quote.sql** file.

```
[student@workstation ~]$ mysql -h127.0.0.1 -uose -popenshift \
quotes < /home/student/D0280/labs/deploy-volume/quote.sql
```

- 8.3. Connect to the database to verify that the quote table contains data:

```
[student@workstation ~]$ mysql -h127.0.0.1 -uose -popenshift \
quotes -e "select count(*) from quote;"
```

The expected output is:

```
+-----+
| count(*) |
+-----+
|      3 |
+-----+
```

- 8.4. Verify that the MySQL server has initialized a database in the exported folder:

```
[student@workstation ~]$ ssh root@master ls -la /var/export/dbvol/
total 41032
-rw-r----. 1 nfsnobody nfsnobody      56 Jul 16 18:53 auto.cnf
-rw-----. 1 nfsnobody nfsnobody    1676 Jul 16 18:53 ca-key.pem
-rw-r--r--. 1 nfsnobody nfsnobody    1075 Jul 16 18:53 ca.pem
-rw-r--r--. 1 nfsnobody nfsnobody    1079 Jul 16 18:53 client-cert.pem
-rw-----. 1 nfsnobody nfsnobody    1676 Jul 16 18:53 client-key.pem
-rw-r----. 1 nfsnobody nfsnobody     291 Jul 16 22:16 ib_buffer_pool
-rw-r----. 1 nfsnobody nfsnobody 12582912 Jul 16 22:16 ibdata1
-rw-r----. 1 nfsnobody nfsnobody  8388608 Jul 16 22:16 ib_logfile0
-rw-r----. 1 nfsnobody nfsnobody  8388608 Jul 16 18:53 ib_logfile1
-rw-r----. 1 nfsnobody nfsnobody 12582912 Jul 16 22:17 ibtmp1
drwxr-x--. 2 nfsnobody nfsnobody    4096 Jul 16 18:53 mysql
-rw-r----. 1 nfsnobody nfsnobody         2 Jul 16 22:16 mysql-2-k2vlh.pid
drwxr-x--. 2 nfsnobody nfsnobody    8192 Jul 16 18:53 performance_schema
-rw-----. 1 nfsnobody nfsnobody    1680 Jul 16 18:53 private_key.pem
-rw-r--r--. 1 nfsnobody nfsnobody     452 Jul 16 18:53 public_key.pem
drwxr-x--. 2 nfsnobody nfsnobody        20 Jul 16 18:53 quotes
-rw-r--r--. 1 nfsnobody nfsnobody    1079 Jul 16 18:53 server-cert.pem
-rw-----. 1 nfsnobody nfsnobody    1680 Jul 16 18:53 server-key.pem
drwxr-x--. 2 nfsnobody nfsnobody    8192 Jul 16 18:53 sys
```



Important

Ensure that the **quotes** directory exists. This matches the name of the database in the MySQL pod resource file.

- 8.5. Verify that the MySQL server created table metadata in the **quotes** directory:

```
[student@workstation ~]$ ssh root@master ls -la /var/export/dbvol/quotes
total 116
drwxr-x---. 2 nfsnobody nfsnobody 54 Jul 16 23:12 .
drwx-----. 6 nfsnobody nfsnobody 4096 Jul 16 22:16 ..
-rw-r-----. 1 nfsnobody nfsnobody 65 Jul 16 18:53 db.opt
-rw-r-----. 1 nfsnobody nfsnobody 8584 Jul 16 23:12 quote.frm
-rw-r-----. 1 nfsnobody nfsnobody 98304 Jul 16 23:12 quote.ibd
```



Important

There should be a file named **quote.frm** corresponding to the name of the table created in the database.

- 8.6. From the first terminal, finish the **oc port-forward** command by pressing **Ctrl+C**

9. Clean up

- 9.1. On the **workstation** host, delete the **persistent-storage** project, which also deletes all PVCs and pods created during this lab.

```
[student@workstation ~]$ oc delete project persistent-storage
project "persistent-storage" deleted
```

- 9.2. On the **workstation** VM, ensure that you are logged in to OpenShift as the **admin** user, and delete the PV so that it can be recreated (and the NFS share reused) in subsequent labs.

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

```
[student@workstation ~]$ oc delete pv mysqldb-volume
persistentvolume "mysqldb-volume" deleted
```

- 9.3. Verify that the database files are still present on the NFS shared directory located on the **master** VM. This shows that the files remain even after the PV has been deleted.

```
[student@workstation ~]$ ssh root@master ls -la /var/export/dbvol/
total 41032
-rw-r-----. 1 nfsnobody nfsnobody 56 Jul 16 18:53 auto.cnf
-rw-----. 1 nfsnobody nfsnobody 1676 Jul 16 18:53 ca-key.pem
-rw-r--r--. 1 nfsnobody nfsnobody 1075 Jul 16 18:53 ca.pem
```

```
-rw-r--r--. 1 nfsnobody nfsnobody      1079 Jul 16 18:53 client-cert.pem
-rw-----. 1 nfsnobody nfsnobody      1676 Jul 16 18:53 client-key.pem
-rw-r-----. 1 nfsnobody nfsnobody      291 Jul 16 22:16 ib_buffer_pool
-rw-r-----. 1 nfsnobody nfsnobody 12582912 Jul 16 22:16 ibdata1
-rw-r-----. 1 nfsnobody nfsnobody   8388608 Jul 16 22:16 ib_logfile0
-rw-r-----. 1 nfsnobody nfsnobody   8388608 Jul 16 18:53 ib_logfile1
-rw-r-----. 1 nfsnobody nfsnobody 12582912 Jul 16 22:17 ibtmp1
drwxr-x---. 2 nfsnobody nfsnobody     4096 Jul 16 18:53 mysql
-rw-r-----. 1 nfsnobody nfsnobody          2 Jul 16 22:16 mysqldb-2-k2vlh.pid
drwxr-x---. 2 nfsnobody nfsnobody     8192 Jul 16 18:53 performance_schema
-rw-r-----. 1 nfsnobody nfsnobody     1680 Jul 16 18:53 private_key.pem
-rw-r--r--. 1 nfsnobody nfsnobody      452 Jul 16 18:53 public_key.pem
drwxr-x---. 2 nfsnobody nfsnobody        20 Jul 16 18:53 quotes
-rw-r--r--. 1 nfsnobody nfsnobody     1079 Jul 16 18:53 server-cert.pem
-rw-----. 1 nfsnobody nfsnobody     1680 Jul 16 18:53 server-key.pem
drwxr-x---. 2 nfsnobody nfsnobody     8192 Jul 16 18:53 sys
```

- 9.4. Use SSH to remotely access the OpenShift master VM, and then delete the contents of the NFS shared **/var/export/dbvol/** directory:

```
[student@workstation ~]$ ssh root@master rm -rf /var/export/dbvol/*
```

Verify that the files have been deleted:

```
[student@workstation ~]$ ssh root@master ls -la /var/export/dbvol/
total 0
drwx-----. 2 nfsnobody nfsnobody  6 Jul 16 19:00 .
drwxr-xr-x. 4 root      root    39 Jul 16 19:00 ..
```

This concludes the guided exercise.

Configuring the OpenShift Internal Registry for Persistence

Objective

After completing this section, students should be able to configure the OpenShift internal container registry for persistence.

Making the OpenShift Internal Image Registry Persistent

The OpenShift Container Platform internal image registry is a vital component of the Source-to-Image (S2I) process used to create pods from application source code. The final output from the S2I process is a container image that is pushed to the OpenShift internal registry, from where it can be used for deployments.

While it is possible to run OpenShift using only ephemeral storage for small test beds and proofs of concepts (POCs), configuring persistent storage for the registry is a better proposition for a production setup. Otherwise, pods created by S2I may fail to start after the registry pod is recreated; for example, after a host node reboot.

The OpenShift installer configures and starts a default registry, which uses NFS shares exported from the OpenShift master. In a production environment, Red Hat recommends that persistent storage be provided by an external server that is configured for resilience and high availability.

The following steps change the default OpenShift internal registry to use persistent storage on an external NFS server:

- 1 Provision a persistent volume (PV) with sufficient capacity for the expected size and number of container images generated by S2I on the OpenShift instance.

The following listing illustrates a **registry-pv.yaml** definition file used by the **oc create** command to create a persistent volume:

```
apiVersion: v1
kind: PersistentVolume 1
metadata:
  name: registry-volume 2
spec:
  capacity:
    storage: 15Gi 3
  accessModes:
    - ReadWriteOnce 4
  nfs: 5
    path: /var/export/registryvol 6
    server: master.lab.example.com 7
  persistentVolumeReclaimPolicy: Recycle 8
  claimRef: 9
    name: registry-pvclaim
```

```
namespace: default
```

- ❶ The kind of resource object that is created.
- ❷ The persistent volume name. This value can be used to identify the persistent volume when using commands such as **oc get pv *volume-name***.
- ❸ The amount of storage requested.
- ❹ Used to match the PV to a PVC.
- ❺ Identifies the nfs plug-in as the volume type used.
- ❻ The path to the NFS share exported by the NFS server.
- ❼ The host name or IP address of the NFS server.
- ❽ NFS volumes that are set to Recycle are scrubbed after the user's persistent volume claim bound to the volume is deleted. After it has been recycled, the NFS volume can be bound to a new claim.
- ❾ When a PV has its claimRef set to some PVC name and namespace, and is reclaimed according to a Retain or Recycle recycling policy, its claimRef will remain set to the same PVC name and namespace.

The following command provisions a persistent volume using a YAML definition file:

```
[user@demo ~] oc create -f registry-pv.yml
persistentvolume "registry-volume" created
```

2. Claiming the **registry-volume** persistent volume and modifying the **docker-registry** deployment configuration to use the claim can be done at the same time using the **oc set volume** command.

Confirm that the OpenShift internal registry is using the volume **registry-storage** and the default **EmptyDir** volume type:

```
[user@demo ~] oc describe dc/docker-registry | grep -A2 Volumes \
Volumes:
registry-storage:
Type: EmptyDir (a temporary directory that shares a pod's lifetime)
```

The following command claims persistent storage for the OpenShift internal registry:

```
[user@demo ~] oc set volume dc/docker-registry ❶ -n default ❷ \
--add --overwrite --name=registry-storage ❸ -t pvc ❹ \
--claim-name=registry-pvclaim❺ --claim-size=15Gi❻ \
--claim-mode='ReadWriteOnce'❻
persistentvolumeclaims/registry-pvclaim
deploymentconfig "docker-registry" updated
```

- ❶ The name of the deployment configuration for the docker registry pod.
- ❷ The namespace where this resource is defined.
- ❸ The name of the current volume in the deployment configuration that is updated.
- ❹ The type of resource object added to this namespace.
- ❺ The persistent volume claim name that can be used in commands such as **oc get pvc *claim-name***.

- ⑥ The size of the actual persistent volume matching this persistent volume claim.
- ⑦ The access mode defined in the persistent volume that matches this persistent volume claim.

Confirm that the OpenShift internal registry is now configured to use the volume **registry-storage** with the default **PersistentVolumeClaim** volume type:

```
[user@demo ~] oc describe dc/docker-registry | grep -A3 Volumes \
Volumes:
  registry-storage:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same
    namespace)
      ClaimName: registry-pvclaim
```

3. The OpenShift internal image registry stores images and metadata as plain files and folders, which means that the PV source storage can be inspected to see if the registry has written files to it.

In a production environment this is done by accessing the external NFS server. However, in our classroom environment the NFS share is configured on the **master** VM, therefore SSH can be used to remotely access the NFS share to verify that the OpenShift internal registry is storing images to the persistent storage.

Assuming there is an application named **hello** running in the default namespace, the following command verifies that images are now stored in persistent storage:

```
[user@demo ~] ssh root@master ls -l \
/var/export/registryvol/docker/registry/v2/repositories/default/
total 0
drwxr-xr-x. 4 nfsnobody nfsnobody 37 Feb 8 19:26 hello
```



Note

A recommended practice in the OpenShift Container Platform Administrator Guide describes how to configure the internal registry to use local host storage (using the **--mount-host** option). This might be sufficient for a proof of concept, but is not ideal for a production environment, where the registry may need to be scaled to multiple nodes or migrated between nodes. Using a PV solves these issues.



References

Additional information about storage for the docker registry is available in the *OpenShift Container Platform Administrator Guide* which can be found at
https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Creating a Persistent Registry

In this exercise, you will configure a **PersistentVolume** and attach it to the already deployed OpenShift Container Platform internal registry.

Resources	
Files:	/home/student/D0280/labs/deploy-registry/ (workstation)

Outcomes

You should be able to set up the OpenShift internal registry with a **PersistentVolume** so that images created by the Source-to-Image (S2I) build process can be stored persistently in the registry.

Before you begin

You must have a working OpenShift Container Platform installation.

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you must have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, and then run the following commands on the **workstation** VM:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab deploy-registry setup
```

Steps

1. Log in to the **master** VM as the **root** user and confirm that the NFS share configured by the OpenShift quick installer was created.

```
[student@workstation ~]$ ssh root@master
[root@master ~]#
```

Verify that the export list contains the **/exports/registry** NFS share:

```
[root@master ~]# showmount -e
Export list for master.lab.example.com:
... output omitted ...
/exports/registry      *
```

Log out of the **master** VM.

```
[root@master ~]# exit
```

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

2. Verify that the OpenShift internal registry is running.

- 2.1. On the **workstation** host, log in to OpenShift as the **admin** user, using **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://master.lab.example.com:8443
Logged into "https://master.lab.example.com:8443" as "admin" using existing
credentials.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

* default
  kube-system
  logging
  management-infra
  openshift
  openshift-infra

Using project "default".
```

- 2.2. Ensure that you are in the **default** project:

```
[student@workstation ~]$ oc project default
```

- 2.3. Confirm that the **docker-registry** pod is running.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
docker-registry-1-bcxmlv   1/1     Running   0          8m
registry-console-2-wt9sz   1/1     Running   0          8m
router-1-hdljf           1/1     Running   0          9m
router-1-ld94q           1/1     Running   0          9m
```

- 2.4. Use the **oc set volume pod** command to confirm that the docker-registry pod identified in Step 2.3 has an empty directory named **registry-storage** mounted at **/registry**:

```
[student@workstation ~]$ oc set volume pod docker-registry-1-bcxmlv
pods/docker-registry-1-bcxmlv
empty directory as registry-storage
  mounted at /registry
secret/registry-certificates as registry-certificates
  mounted at /etc/secrets
secret/registry-token-vxrml as registry-token-vxrml
  mounted at /var/run/secrets/kubernetes.io/serviceaccount
```

- 2.5. Use the **oc get dc** command to identify the deployment configuration name for the registry:

NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY
------	----------	---------	---------	--------------

docker-registry	1	1	1	config
registry-console	2	1	1	config
router	1	2	2	config

3. From **workstation**, create a new persistent volume (PV) and a persistent volume claim (PVC) for the registry:

- 3.1. Review the PV resource definition file, **/home/student/D0280/labs/deploy-registry/registry-volume.yml**:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: registry-volume
spec:
  capacity:
    storage: 15Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /exports/registry
    server: master.lab.example.com
  persistentVolumeReclaimPolicy: Recycle
  claimRef:
    name: registry-pvclaim
  namespace: default
```

- 3.2. Use the **oc create** command to create the persistent volume using the provided YAML resource definition file:

```
[student@workstation ~]$ oc create -f \
~/D0280/labs/deploy-registry/registry-volume.yml
persistentvolume "registry-volume" created
```

4. Claim the persistent volume:

```
[student@workstation ~]$ oc set volume dc/docker-registry -n default \
--add --overwrite --name=registry-storage -t pvc \
--claim-name=registry-pvclaim --claim-size=15Gi \
--claim-mode='ReadWriteOnce'
persistentvolumeclaims/registry-pvclaim
deploymentconfig "docker-registry" updated
```

5. Verify that the persistent volume claim is bound to the persistent volume:

```
[student@workstation ~]$ oc get pvc
NAME           STATUS VOLUME      CAPACITY ACCESSMODES AGE
registry-pvclaim   Bound  registry-volume  15Gi      RWO        8s
```

6. Verify that the registry is using the NFS share to save container images. Create a new application named **hello** in a new project named **registry-storage**, and then list the contents of the NFS share mounted on **/exports/registry** to confirm that the image is saved to the share.

- 6.1. On the **workstation** host, log in to OpenShift as the **developer** user, using **openshift** as the password:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 6.2. Create a new project named **registry-storage**:

```
[student@workstation ~]$ oc new-project registry-storage
```

- 6.3. Use the **oc new-app** command to create a pod using the following Git repository: <http://workstation.lab.example.com/php-helloworld>. Name the pod **hello**.

```
[student@workstation ~]$ oc new-app --name=hello -i php:7.0 \
http://workstation.lab.example.com/php-helloworld
...
--> Creating resources ...
imagestream "hello" created
buildconfig "hello" created
deploymentconfig "hello" created
service "hello" created
--> Success
Build scheduled, use 'oc logs -f bc/hello' to track its progress.
Run 'oc status' to view your app.
```

- 6.4. View the build logs for the build configuration of the **hello** application:

```
[student@workstation ~]$ oc logs -f bc/hello
Cloning "http://workstation.lab.example.com/php-helloworld" ...
Commit: de02d79a7eedd4603184d6b964450f990526b0de (Initial commit)
Author: root <root@workstation.lab.example.com>
Date: Mon Feb 6 11:23:18 2017 +0530
---> Installing application source...
Pushing image 172.30.0.103:5000/default/hello:latest ...
Pushed 0/5 layers, 7% complete
Pushed 1/5 layers, 23% complete
Pushed 2/5 layers, 42% complete
Pushed 2/5 layers, 85% complete
Pushed 3/5 layers, 87% complete
Pushed 4/5 layers, 94% complete
Pushed 5/5 layers, 100% complete
Push successful
```

- 6.5. The application is built and pushed to the internal OpenShift registry. Wait until the pod called **hello** is running.

Repeat the following commands until the output shows that all the pods are **running** and the **READY** column indicates that the pods are ready:

```
[student@workstation ~]$ oc status ; oc get pods
In project registry-storage on server https://master.lab.example.com:8443
svc/hello - 172.30.164.107:8080
```

```
dc/hello deploys istag/hello:latest <
bc/hello source builds http://workstation.lab.example.com/php-helloworld on
openshift/php:7.0
deployment #1 deployed about a minute ago - 1 pod

View details with 'oc describe <resource>/<name>' or list everything with 'oc
get all'.
NAME      READY     STATUS    RESTARTS   AGE
hello-1-build  0/1     Completed  0          1m
hello-1-rnvf6  1/1     Running   0          1m
```

- 6.6. Use the **ssh** command to connect to the **master** server and verify that the new image layers are stored on the NFS share mounted on **/exports/registry**:

```
[student@workstation ~]$ ssh root@master ls -la \
/exports/registry/docker/registry/v2/repositories/registry-storage
total 0
drwxr-xr-x. 3 nfsnobody nfsnobody 19 Feb  8 19:26 .
drwxr-xr-x. 3 nfsnobody nfsnobody 21 Feb  8 19:26 ..
drwxr-xr-x. 4 nfsnobody nfsnobody 37 Feb  8 19:26 hello
```

7. Clean up.

Deleted the **registry-storage** project:

```
[student@workstation ~]$ oc delete project registry-storage
```

Do not make any change to the **default** project. Leave the internal registry configured to use persistent storage.

This concludes the guided exercise.

Lab: Allocating Persistent Storage

In this lab, you will use a template to deploy an application that integrates with a database, which stores data in an NFS backed persistent volume.

Resources	
Files:	/home/student/D0280/labs/storage-review (workstation VM) /root/D0280/labs/storage-review (master VM)
Application URL:	http://instructor.cloudapps.lab.example.com

Outcomes

You should be able to deploy an application that integrates with a database that uses persistent storage for storing data.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the Ansible playbook using the following commands from the **workstation** host:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on the **workstation** VM and run the following command:

```
[student@workstation ~]$ lab storage-review setup
```

Steps

- On the **master** VM, run the script located at **/root/D0280/labs/storage-review/config-review-nfs.sh** to configure the NFS share **/var/export/review-dbvol** that will be used in this lab for the persistent volume.
- On the **workstation** VM, log in to OpenShift as the **admin** user, using **redhat** as the password. Create a persistent volume named **review-pv** using the provided **/home/student/D0280/labs/storage-review/review-volume-pv.yaml** file.
- Make sure you are logged in as the OpenShift **admin** user in the openshift namespace on the workstation VM. Import the instructor application template from **/home/student/D0280/labs/storage-review/instructor-template.yaml**.



Note

It is important to apply the **-n openshift** namespace parameter to ensure that the template is visible in the web console.

-
4. On the **workstation** host, log in to OpenShift as the **developer** user, using **openshift** as the password, then create a new project named **instructor**.
 5. From **workstation**, access the OpenShift web console at **https://master.lab.example.com:8443**. Log in to OpenShift as the **developer** user with **openshift** as the password. In the **instructor** project, click **Add to Project**. Select the PHP framework. Select the **instructor** application template, then add **instructor.cloudapps.lab.example.com** in the **Application Host** field to expose the host name that will route to the PHP service. Create the **instructor** application.
 6. The template created a database server. From **workstation**, use the **oc port-forward** command to forward the local port 3306 to the pod port 3306. Populate the database using the SQL file available at **/home/student/D0280/labs/storage-review/instructor.sql**.

Use the following **mysql** command to populate the database:

```
[student@workstation ~]$ mysql -h127.0.0.1 -u instructor -ppassword \
instructor < /home/student/D0280/labs/storage-review/instructor.sql
```

7. Access the application, available at **http://instructor.cloudapps.lab.example.com**. Use the application to add a new record, according to the following table.

Add Instructor	
Name	InstructorUser4
Email address	iuser4@workstation.example.com
City	Raleigh
Country	United States

8. Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab storage-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

9. Clean up

On the **workstation** host, remove the **instructor** project and the **review-pv** persistent volume. On the **master** host, delete the database files located in the **/var/export/review-dbvol** directory.

This concludes the lab.

Solution

In this lab, you will use a template to deploy an application that integrates with a database, which stores data in an NFS backed persistent volume.

Resources	
Files:	/home/student/D0280/labs/storage-review (workstation VM) /root/D0280/labs/storage-review (master VM)
Application URL:	http://instructor.cloudapps.lab.example.com

Outcomes

You should be able to deploy an application that integrates with a database that uses persistent storage for storing data.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the Ansible playbook using the following commands from the **workstation** host:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on the **workstation** VM and run the following command:

```
[student@workstation ~]$ lab storage-review setup
```

Steps

- On the **master** VM, run the script located at **/root/D0280/labs/storage-review/config-review-nfs.sh** to configure the NFS share **/var/export/review-dbvol** that will be used in this lab for the persistent volume.
 - Log in to the **master** VM as the **root** user:

```
[student@workstation ~]$ ssh root@master
[root@master ~]#
```

- View the script at **/root/D0280/labs/storage-review/config-review-nfs.sh** to become familiar with the process of configuring the NFS share before running the script:

```
[root@master ~]# less -FiX /root/D0280/labs/storage-review/config-review-nfs.sh
... output omitted ...
```

- Run the script to configure the NFS share:

```
[root@master ~]# /root/D0280/labs/storage-review/config-review-nfs.sh
```

```
Export directory /var/export/review-dbvol created.
```

- 1.4. Run the **showmount -e** command to verify that the export list includes the **/var/export/review-dbvol** NFS share.

```
[root@master ~]# showmount -e
Export list for master.lab.example.com:
/var/export/review-dbvol *
... output omitted ...
```

- 1.5. Log out of the **master** VM.

```
[root@master ~]# exit
[student@workstation ~]$
```

2. On the **workstation** VM, log in to OpenShift as the **admin** user, using **redhat** as the password. Create a persistent volume named **review-pv** using the provided **/home/student/D0280/labs/storage-review/review-volume-pv.yaml** file.

- 2.1. Log in as the OpenShift **admin** user:

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

- 2.2. Inspect the contents of the **/home/student/D0280/labs/storage-review/review-volume-pv.yaml** for correct entries of PV attributes and the NFS share:

```
[student@workstation ~]$ less -FiX \
/home/student/D0280/labs/storage-review/review-volume-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: review-pv
spec:
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /var/export/review-dbvol
    server: master.lab.example.com
```

- 2.3. Use the following command to create a persistent volume named **review-pv**:

```
[student@workstation ~]$ oc create -f \
/home/student/D0280/labs/storage-review/review-volume-pv.yaml
persistentvolume "review-pv" created
```

3. Make sure you are logged in as the OpenShift **admin** user in the openshift namespace on the workstation VM. Import the instructor application template from **/home/student/D0280/labs/storage-review/instructor-template.yaml**.



Note

It is important to apply the **-n openshift** namespace parameter to ensure that the template is visible in the web console.

- 3.1. Review the various parameters in the template file. Notice the **php:7.0** image used to create the PHP application and the **mysql:5.7** image used to create the database. The template creates a persistent volume claim which is bound to the persistent volume **review-pv** created earlier in this lab.

```
[student@workstation ~]$ less -FiX \
/home/student/D0280/labs/storage-review/instructor-template.yaml
apiVersion: v1
kind: Template
labels:
  template: instructor
...
... output omitted ...
from:
  kind: ImageStreamTag
  name: php:7.0
...
... output omitted ...
from:
  kind: ImageStreamTag
  name: mysql:5.7
...
... output omitted ...
```

- 3.2. Import the **/home/student/D0280/labs/storage-review/instructor-template.yaml** instructor template file.

```
[student@workstation ~]$ oc create -n openshift -f \
/home/student/D0280/labs/storage-review/instructor-template.yaml
template "instructor" created
```

4. On the **workstation** host, log in to OpenShift as the **developer** user, using **openshift** as the password, then create a new project named **instructor**.

- 4.1. Log in to OpenShift as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 4.2. Create a new project named **instructor**:

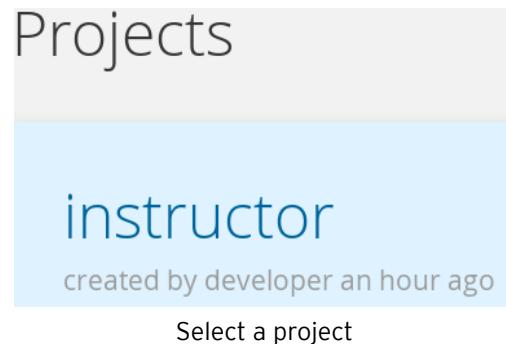
```
[student@workstation ~]$ oc new-project instructor
```

5. From **workstation**, access the OpenShift web console at **https://master.lab.example.com:8443**. Log in to OpenShift as the **developer** user

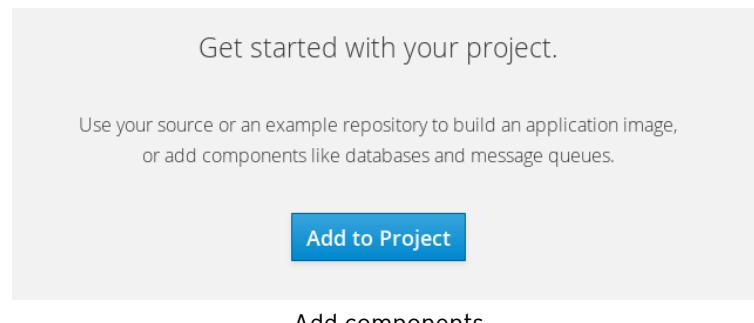
with **openshift** as the password. In the **instructor** project, click **Add to Project**. Select the PHP framework. Select the **instructor** application template, then add **instructor.cloudapps.lab.example.com** in the Application Host field to expose the host name that will route to the PHP service. Create the **instructor** application.

- 5.1. Open Firefox and navigate to **https://master.lab.example.com:8443** (if prompted, accept the security certificate) and log in as the **developer** user, using **openshift** as the password.

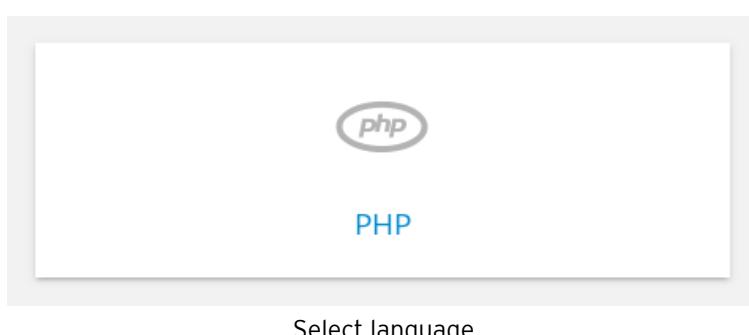
Click the **instructor** project to access the project.



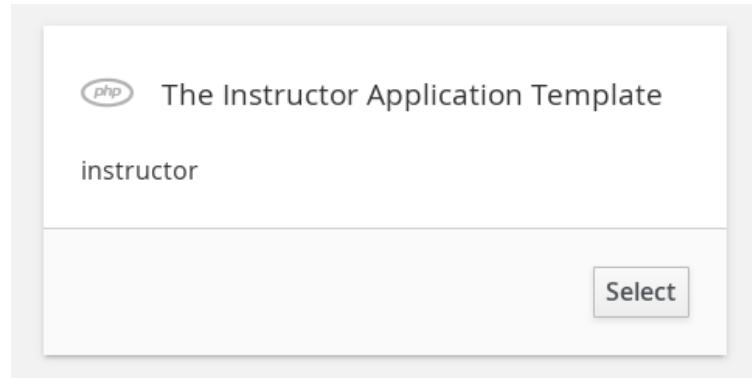
- 5.2. Click **Add to Project** to view the available components.



- 5.3. On the **Browser Catalog** page, in the **Languages** section, select **PHP** as the framework.



- 5.4. Locate the **The Instructor Application Template** entry, and click **Select** to edit the template properties.



Select application template

- 5.5. All the values populated from the template will be used as is. However, the **Application Hostname** field must be updated. Enter **instructor.cloudapps.lab.example.com** as the value, and then click **Create** to create the application.

Application Hostname
instructor.cloudapps.lab.example.com
The exposed hostname that will route to the PHP service, if left blank a value will be defaulted.

Enter application hostname

- 5.6. Click **Continue to overview** to monitor the application as it builds. When the build completes, a green check mark next to **Complete** should appear, and the application route, **<http://instructor.cloudapps.lab.example.com>**, displays at the upper right of the screen.

Project overview

6. The template created a database server. From **workstation**, use the **oc port-forward** command to forward the local port 3306 to the pod port 3306. Populate the database using the SQL file available at **/home/student/D0280/labs/storage-review/instructor.sql**.

Use the following **mysql** command to populate the database:

```
[student@workstation ~]$ mysql -h127.0.0.1 -u instructor -ppassword \
instructor < /home/student/D0280/labs/storage-review/instructor.sql
```

- 6.1. Make sure you are logged in to OpenShift as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 6.2. Open two terminals. From the first one, run the **oc get pods** command to retrieve the status of the pods. Ensure that the **mysql** pod is marked as **running**. Run the **oc port-forward** command to forward the local port 3306 to the pod port 3306. The command keeps the port open until the connection is manually terminated.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
instructor-1-9n6cx  1/1     Running   0          42m
instructor-1-build  0/1     Completed  0          42m
mysql-1-z95g1      1/1     Running   0          42m
```

```
[student@workstation ~]$ oc port-forward mysql-1-z95g1 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
```

- 6.3. From the second terminal, populate the **instructor** database by using the provided **instructor.sql** file.

```
[student@workstation ~]$ mysql -h127.0.0.1 -u instructor -ppassword \
instructor < /home/student/D0280/labs/storage-review/instructor.sql
```

- 6.4. Ensure that the database is now populated with new records by running a MySQL command:

```
[student@workstation ~]$ mysql -h127.0.0.1 -u instructor -ppassword \
instructor -e "select * from instructors;"
```

instructorName	email	city
DemoUser1	duser1@workstation.example.com	Raleigh
InstructorUser1	iuser1@workstation.example.com	Rio de Janeiro
InstructorUser2	iuser2@workstation.example.com	Raleigh
InstructorUser3	iuser3@workstation.example.com	Sao Paulo

- 6.5. When finished, press **Ctrl+C** to close the port forward connection in the first terminal.

7. Access the application, available at **http://instructor.cloudapps.lab.example.com**. Use the application to add a new record, according to the following table.

Add Instructor	
Name	InstructorUser4
Email address	iuser4@workstation.example.com
City	Raleigh
Country	United States

- 7.1. Click **Add new Instructor** to add a new instructor, and then complete the form using the information provided in the table.

Click **Add New Instructor** to update the database. The instructor list updates automatically.

8. Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab storage-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

9. Clean up

On the **workstation** host, remove the **instructor** project and the **review-pv** persistent volume. On the **master** host, delete the database files located in the **/var/export/review-dbvol** directory.

- 9.1. On the **workstation** host, log in as the OpenShift **admin** user, using **redhat** as the password:

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

- 9.2. Delete the **instructor** project:

```
[student@workstation ~]$ oc delete project instructor
project "instructor" deleted
```

- 9.3. Delete the **review-pv** persistent volume:

```
[student@workstation ~]$ oc delete pv review-pv
persistentvolume "review-pv" deleted
```

- 9.4. Log in to the **master** VM to delete the database files.

From **workstation**, log in to the **master** VM as the **root** user:

```
[student@workstation ~]$ ssh root@master
[root@master ~]#
```

Delete the database files, and then verify that the files have been removed:

```
[root@master ~]# rm -rf /var/export/review-dbvol/*
[root@master ~]# ls -la /var/export/review-dbvol/
total 0
drwx----- 2 nfsnobody nfsnobody 6 Aug  1 05:33 .
drwxr-xr-x. 3 root      root      26 Aug  1 04:25 ..
```

This concludes the lab.

Summary

In this chapter, you learned:

- OpenShift uses *PersistentVolumes (PVs)* to provision persistent storage for pods.
- An OpenShift project uses *PersistentVolumeClaim (PVC)* resources to request that a PV be assigned to the project.
- The OpenShift installer configures and starts a default registry, which uses NFS shares exported from the OpenShift master.
- The OpenShift default registry can be changed to use a PVC and PV provided by an external NFS server by running the **oc set volume** command and passing the name of the PVC as an option to the command.



CHAPTER 7

MANAGING APPLICATION DEPLOYMENTS

Overview	
Goal	Manipulate resources to manage deployed applications.
Objectives	<ul style="list-style-type: none">Control the number of replications of a pod.Describe and control how pods are scheduled on the cluster.Manage the images, image streams, and templates used in application builds.
Sections	<ul style="list-style-type: none">Scaling an Application (and Guided Exercise)Controlling Pod Scheduling (and Guided Exercise)Managing Images, Image Streams, and Templates (and Guided Exercise)
Lab	Managing Application Deployments

Scaling an Application

Objective

After completing this section, you should be able to control the number of replicas of a pod.

Replication Controllers

A *replication controller* guarantees that the specified number of replicas of a pod are running at all times. The replication controller instantiates more pods if pods are killed, or are deleted explicitly by an administrator. Similarly, it deletes pods as necessary to match the specified replica count, if there are more pods running than the desired count.

The definition of a replication controller consists mainly of:

- The desired number of replicas
- A pod definition for creating a replicated pod
- A selector for identifying managed pods

The selector is a set of labels that all of the pods managed by the replication controller must match. The same set of labels must be included in the pod definition that the replication controller instantiates. This selector is used by the replication controller to determine how many instances of the pod are already running in order to adjust as needed.



Note

The replication controller does not perform autoscaling, because it does not track load or traffic. The *horizontal pod autoscaler* resource, presented later in this section, manages autoscaling.

Although Kubernetes administrators usually manage replication controllers directly, the recommended approach for OpenShift users is to manage a deployment configuration that creates or changes replication controllers on demand.

Creating Replication Controllers from a Deployment Configuration

The most common way to create applications in OpenShift is by using either the `oc new-app` command or the web console. Applications created this way use **DeploymentConfig** resources that create replication controllers at runtime to create application pods.

A **DeploymentConfig** resource definition defines the number of replicas of the pod to create, as well as a template for the pods to be created.



Important

Do not mistake the **template** attribute from a **DeploymentConfig** or **ReplicationController** resource with the OpenShift **template** resource type, which is used for building applications based on some commonly used language runtimes and frameworks.

The following listing illustrates a **DeploymentConfig** resource created by the **oc new-app** command for a MySQL database container image:

```
{
  "kind": "DeploymentConfig",
  "apiVersion": "v1",
  "metadata": {
    "name": "mydb",
  },
  "spec": {

    ... "strategy" and "triggers" attributes omitted ...

    "replicas": 1, ❶
    "selector": {
      "deploymentconfig": "mydb" ❷
    },
    "template": { ❸
      "metadata": {
        "labels": {
          "deploymentconfig": "mydb" ❹
        }
      },
      "spec": {
        "containers": [
          {
            "name": "mysql-56-rhel7",
            "image": "registry.access.redhat.com/rhscl/mysql-56-
rhel7:latest",
            "ports": [
              {
                "name": "mysql-56-rhel7-tcp-3306",
                "containerPort": 3306,
                "protocol": "TCP"
              }
            ],
            ... "env" and "volumeMount" attributes omitted ...
          }
        ],
        ... "volumes" attributes omitted ...
      }
    }
  }
}
```

- ❶** Specifies the number of copies (or replicas) of the pod to run.
- ❷** A replication controller uses a selector to count the number of running pods, in the same way that a service uses a selector to find the pods to load balance.

- ③ A template for pods that the controller creates.
- ④ Labels on pods created by the replication controller must match those from the selector.

Changing the Number of Replicas for an Application

The number of replicas in a **DeploymentConfig** or **ReplicationController** resource can be changed dynamically using the **oc scale** command:

```
$ oc get dc
NAME      REVISION  DESIRED  CURRENT  TRIGGERED BY
myapp     1          3         3         config,image(scaling:latest)
```

```
$ oc scale --replicas=5 dc myapp
```

The **DeploymentConfig** resource propagates the change to the **ReplicationController**, which reacts to the change by creating new pods (replicas) or deleting existing ones.

Although it is possible to manipulate the **ReplicationController** resource directly, the recommended practice is to manipulate the **DeploymentConfig** resource instead. Changes made directly to a **ReplicationController** resource may be lost when a deployment is triggered, for example to recreate pods using a new release of the container image.

Autoscaling Pods

OpenShift can autoscale a deployment configuration, based on current load on the application pods, by means of a **HorizontalPodAutoscaler** resource type.

A **HorizontalPodAutoscaler (HPA)** resource uses performance metrics collected by the OpenShift Metrics subsystem, which is presented later in this book. Without the Metrics subsystem, more specifically the *Heapster* component, autoscaling is not possible.

The recommended way to create a **HorizontalPodAutoscaler** resource is using the **oc autoscale** command, for example:

```
$ oc autoscale dc/myapp --min 1 --max 10 --cpu-percent=80
```

The previous command creates a **HorizontalPodAutoscaler** resource that changes the number of replicas on the **myapp** deployment configuration to keep its pods under 80% of their total requested CPU usage.

The **oc autoscale** command creates a **HorizontalPodAutoscaler** resource using the name of the deployment configuration as an argument (**myapp** in the previous example).

The maximum and minimum values for the **HorizontalPodAutoscaler** resource serve to accommodate bursts of load and avoid overloading the OpenShift cluster. If the load on the application changes too quickly, it might be advisable to keep a number of spare pods to cope with sudden bursts of user requests. Conversely, too high a number of pods can use up all cluster capacity and impact other applications sharing the same OpenShift cluster.

To get information about **HorizontalPodAutoscaler** resources in the current project, use the **oc get** and **oc describe** commands. For example:

```
$ oc get hpa/frontend
```

NAME	REFERENCE	TARGET	CURRENT	MINPODS	MAXPODS
AGE					
<code>frontend DeploymentConfig/myapp/frontend/scale 8d</code>					
\$ oc describe hpa/frontend					
Name:	frontend				
Namespace:	myapp				
Labels:	<none>				
CreationTimestamp:	Mon, 26 Jul 2017 21:13:47 -0400				
Reference:	DeploymentConfig/myapp/frontend/scale				
Target CPU utilization:	80%				
Current CPU utilization:	59%				
Min pods:	1				
Max pods:	10				

Notice that a **HorizontalPodAutoscaler** resource only works for pods that define *resource requests* for the reference performance metric. Pod resource requests are explained later in this chapter.

In Red Hat OpenShift Container Platform 3.5 only CPU usage is supported as a reference performance metric. Future OpenShift releases might support additional reference performance metrics.

Most of the pods created by the `oc new-app` command define no resource requests. Using the OpenShift autoscaler may therefore require either creating custom YAML or JSON resource files for your application, or adding resource range resources to your project. See the section about quotas in the *Managing and Monitoring OpenShift Container Platform* chapter in this book for information about resource ranges.



References

Additional information about replication controllers is available in the *Architecture* chapter of the Red Hat OpenShift Container Platform documentation at

|| https://access.redhat.com/documentation/en-us/openshift_container_platform

Additional information about autoscaling pods is available in the *Developer Guide* chapter of the Red Hat OpenShift Container Platform documentation at

|| https://access.redhat.com/documentation/en-us/openshift_container_platform

Guided Exercise: Scaling An Application

In this lab, you will scale an application by increasing the number of running pods.

Resources	
Files	http://workstation.lab.example.com/scaling
Application URL	http://scaling.cloudapps.lab.example.com

Outcome

You should be able to scale an application by using a deployment configuration to deploy multiple pods.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts, and then run the Ansible Playbook using the following commands on the **workstation** host:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

Steps

1. Create a new project.

1.1. Log in to OpenShift as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443
```

1.2. Create a project called **scaling**:

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

2. Create an application to test scaling.

2.1. Create a new application and export its definition to a YAML file. Use the **php:7.0** image stream:

```
[student@workstation ~]$ oc new-app -o yaml -i php:7.0 \  
http://workstation.lab.example.com/scaling \  
> ~/scaling.yaml
```

2.2. Open the YAML resource definition file in a text editor:

```
[student@workstation ~]$ vi ~/scaling.yaml
```

Locate the **DeploymentConfig** resource. Change the **replicas** attribute from 1 to 3.

```

...
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    creationTimestamp: null
    labels:
      app: scaling
      name: scaling
  spec:
    replicas: 3
    selector:
...

```

With this change, when the application is created, three pods are created. Save the file and exit from the editor.

2.3. Create the application using the **oc create** command:

```
[student@workstation ~]$ oc create -f ~/scaling.yml
imagestream "scaling" created
buildconfig "scaling" created
deploymentconfig "scaling" created
service "scaling" created
```

2.4. Review the status of the build:

```
[student@workstation ~]$ watch -n 3 oc get builds
NAME      TYPE      FROM      STATUS      STARTED      DURATION
scaling-1  Source    Git@1ba7b7d  Complete   48 seconds ago  16s
```



Note

It might take a few moments for the build to finish. Wait until the build status transitions to *Complete*, and then press Ctrl+C to exit the **watch** command.

2.5. List the available pods:

```
[student@workstation ~]$ oc get pods
NAME        READY      STATUS      RESTARTS      AGE
scaling-1-018wr  1/1      Running     0            23s
scaling-1-build  0/1      Completed   0            52s
scaling-1-dq36m  1/1      Running     0            23s
scaling-1-jb7zj  1/1      Running     0            23s
```

Repeat the previous command until you see the three pods of the **scaling** application. It might take a few moments until all three pods are ready and running.

3. Create a route for the application in order to balance requests for each pod:

```
[student@workstation ~]$ oc expose service scaling \
```

```
--hostname=scaling.cloudapps.lab.example.com
route "scaling" exposed
```

4. Retrieve the application pod IP addresses using the web console. Compare them to the IP addresses reported by the scaling application.
 - 4.1. Open a web browser (**Applications > Internet > Firefox**) from **workstation** and access the following URL: **https://master.lab.example.com:8443**. Use **developer** as the user name and **openshift** as the password.
 - 4.2. On the **Projects** page, click **scaling**.

The screenshot shows the OpenShift Container Platform interface. At the top, it says "OPENSHIFT CONTAINER PLATFORM". Below that is a navigation bar with "Projects", a search bar, and sorting options. A "New Project" button is also present. In the main area, there is a card for the "scaling" project, which was created by "developer" 5 minutes ago. The card includes icons for edit, delete, and more. Below the card, the text "The scaling project" is displayed.

- 4.3. Click the **Overview** tab to access the three application pods.

The screenshot shows the "Project scaling" overview page. The left sidebar has links for Overview, Applications, Builds, Resources, Storage, and Monitoring. The main area shows a summary for the "SCALING" deployment config, indicating 3 pods. It also shows a note: "scaling has containers without health checks, which ensure your application is running correctly. Add Health Checks". A deployment log entry for "Build scaling, #1" is shown, stating it is complete 6 minutes ago. The right side of the screen shows a message: "No grouped services. No services are grouped with scaling". Below the main area, the text "Project overview page" is displayed.

- 4.4. Navigate to **Applications > Pods** in the left navigation pane to view the pods.

Name	Status	Containers Ready	Container Restarts	Age
scaling-1-0l8wr	Running	1/1	0	6 minutes
scaling-1-dq36m	Running	1/1	0	6 minutes
scaling-1-jb7zj	Running	1/1	0	6 minutes
scaling-1-build	Completed	0/1	0	7 minutes

Application pods

Click one of the pod names to display details about the pod, including its internal IP address.

Status:	Running
Deployment:	scaling_#1
IP:	10.128.0.27
Node:	node2.lab.example.com (172.25.250.12)
Restart Policy:	Always

State:	Running since Jul 20, 2017 1:48:19 PM
Ready:	true
Restart Count:	0

Pod details

- On **workstation**, run the **oc get pods** command with the **-o wide** option to view the internal IP address of each running pod:

```
[student@workstation ~]$ oc get pods -o wide
NAME        READY   STATUS    ...   IP          NODE
scaling-1-0l8wr  1/1    Running  ...  10.128.0.27  node2.lab.example.com
scaling-1-build  0/1    Completed ...  10.128.0.25  node2.lab.example.com
scaling-1-dq36m  1/1    Running  ...  10.128.0.26  node2.lab.example.com
scaling-1-jb7zj  1/1    Running  ...  10.130.0.31  node1.lab.example.com
```

- Ensure that the OpenShift router is balancing requests to the application. To do so, use a **for** loop to run the **curl** command.

Each request should return a different IP address, because each request is served by a different pod. As you make more requests than there are pods available to serve them, you will see duplicate IP addresses.

```
[student@workstation ~]$ for i in {1..5}; do curl -s \
  http://scaling.cloudapps.lab.example.com | grep IP; done
<br/> Server IP: 10.128.0.26
<br/> Server IP: 10.128.0.27
<br/> Server IP: 10.130.0.31
<br/> Server IP: 10.128.0.26
<br/> Server IP: 10.128.0.27
```



Note

You cannot check load balancing using a web browser because the OpenShift routers implement session affinity (also known as *sticky sessions*). All requests from the same web browser go to the same pod. Opening a new web browser tab or window will not avoid session affinity; you need to use a different web browser application, or to open a web browser from a different computer.

6. Scale the application to run more pods.

6.1. View the number of replicas specified in the current **DeploymentConfig**:

```
[student@workstation ~]$ oc describe dc scaling | grep Replicas
Replicas: 3
Replicas: 3 current / 3 desired
```

6.2. Use the **oc scale** command to increase the number of pods (replicas) to five:

```
[student@workstation ~]$ oc scale --replicas=5 dc scaling
deploymentconfig "scaling" scaled
```

6.3. You can also change the number of pods from the **Overview** tab of the OpenShift web console. Click the **up arrow** or **down arrow** next to the blue donut to increase or decrease the number of pods, respectively.

Scaling pods

6.4. As the number of pods scales up or down, you should see the blue donut in the **Overview** page change accordingly.



Note

The blue donut will be a complete circle when all five requested pods are running.

Pod count after scaling up

6.5. Ensure that five pods are now running for this application:

```
[student@workstation ~]$ oc get pods -o wide
NAME        READY   STATUS    ...   IP          NODE
scaling-1-0l8wr  1/1    Running   ...  10.128.0.27  node2.lab.example.com
scaling-1-build  0/1    Completed  ...  10.128.0.25  node2.lab.example.com
scaling-1-dq36m  1/1    Running   ...  10.128.0.26  node2.lab.example.com
scaling-1-g2jmz  1/1    Running   ...  10.130.0.35  node1.lab.example.com
scaling-1-jb7zj  1/1    Running   ...  10.130.0.31  node1.lab.example.com
scaling-1-rv2pv  1/1    Running   ...  10.128.0.31  node2.lab.example.com
```

6.6. Ensure that the router is balancing requests to the new pods using the same URL:

```
[student@workstation ~]$ for i in {1..5}; do curl -s \
  http://scaling.cloudapps.lab.example.com | grep IP; done
<br/> Server IP: 10.128.0.26
<br/> Server IP: 10.128.0.27
<br/> Server IP: 10.128.0.31
<br/> Server IP: 10.130.0.31
<br/> Server IP: 10.130.0.35
```

Compare the output with the output in Step 5. You will see that the router load balances the request between the five application pods in a round-robin manner.

7. Clean up. On **workstation**, run the following command to delete the **scaling** project:

```
[student@workstation ~]$ oc delete project scaling
project "scaling" deleted
```

This concludes the guided exercise.

Controlling Pod Scheduling

Objective

After completing this section, students should be able to describe and control how pods are scheduled on the cluster.

Introduction to the OpenShift Scheduler Algorithm

The pod scheduler determines placement of new pods onto nodes in the OpenShift cluster. It is designed to be highly configurable and adaptable to different clusters. The default configuration shipped with Red Hat OpenShift Container Platform 3.5 supports the common data center concepts of *zones* and *regions* by using node labels, affinity rules, and anti-affinity rules.

The OpenShift pod scheduler algorithm follows a three-step process:

1. Filtering nodes.

The scheduler filters the list of running nodes by the availability of node resources, such as host ports. Filtering continues considering node selectors and resource requests from the pod. The end result is a shorter list of candidates to run the pod.

A pod can define a node selector that match the labels in the cluster nodes. Nodes whose labels do not match are not eligible.

A pod can also define resource requests for compute resources such as CPU, memory, and storage. Nodes that have insufficient free computer resources are not eligible.

2. Prioritizing the filtered list of nodes.

The list of candidate nodes is evaluated using multiple priority criteria, which add up to a weighted score. Nodes with higher values are better candidates to run the pod.

Among the criteria are **affinity** and **anti-affinity** rules. Nodes with higher affinity for the pod have a higher score, and nodes with higher anti-affinity have a lower score.

A common use for **affinity** rules is to schedule related pods to be close to each other, for performance reasons. An example is to use the same network backbone for pods that need to stay synchronized with each other.

A common use for **anti-affinity** rules is to schedule related pods not too close to each other, for high availability. One example is to avoid scheduling all pods from the same application to the same node.

3. Selecting the best fit node.

The candidate list is sorted based on the scores and the node with the highest score is selected to host the pod. If multiple nodes have the same high score, then one is selected at random.

The scheduler configuration file at `/etc/origin/master/scheduler.json` defines a set of *predicates* that are used as either filter or priority functions. This way the scheduler can be configured to support different cluster topologies.

Scheduling and Topology

A common topology for large data centers, such as cloud providers, is to organize hosts into **regions** and **zones**:

- A **region** is a set of hosts in a close geographic area, which guarantees high-speed connectivity between them.
- A **zone**, also called an **availability zone**, is a set of hosts that might fail together because they share common critical infrastructure components, such as a network, storage, or power.

The standard configuration of the OpenShift pod scheduler supports this kind of cluster topology by defining predicates based on the **region** and **zone** labels. The predicates are defined in such a way that:

- Replica pods, created from the same replication controller, or from the same deployment configuration, are scheduled to run in nodes having the same value for the **region** label.
- Replica pods are scheduled to run in nodes having different values for the **zone** label.

The figure below shows a sample topology that consists of multiple regions, each with multiple zones, and each zone with multiple nodes.

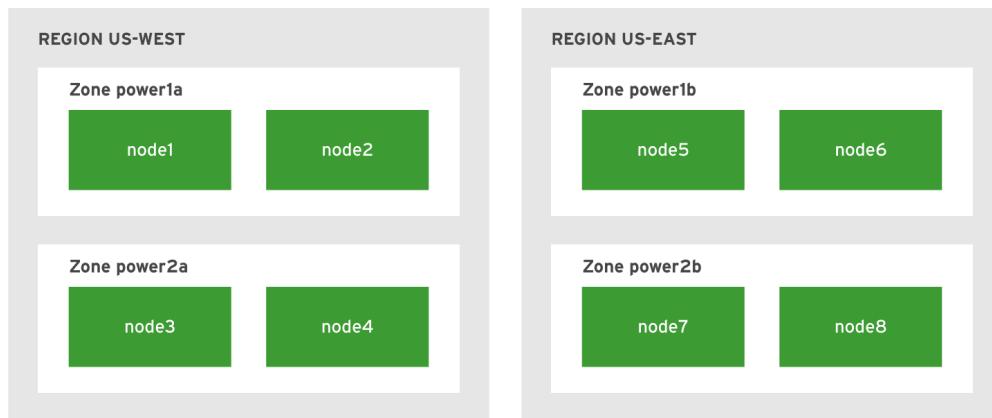


Figure 7.7: Sample cluster topology using regions and zones

To implement the sample topology from the previous figure, use the **oc label** command as a cluster administrator. For example:

```

$ oc label node1 region=us-west zone=power1a --overwrite
$ oc label node2 region=us-west zone=power1a --overwrite
$ oc label node node3 region=us-west zone=power2a --overwrite
$ oc label node node4 region=us-west zone=power2a --overwrite
$ oc label node node5 region=us-east zone=power1b --overwrite
$ oc label node node6 region=us-east zone=power1b --overwrite
$ oc label node node7 region=us-east zone=power2b --overwrite
$ oc label node node8 region=us-east zone=power2b --overwrite
  
```



Important

Each node must be identified by its fully qualified name (FQDN). The example commands use short names for brevity.

Notice that changes to the **region** label require the **--overwrite** option, because the Red Hat OpenShift Container Platform 3.5 quick installer configures all nodes with the **region=infra** label by default.

To inspect the labels assigned to a node, use the **oc get node** command with the **--show-labels** option, for example:

```
$ oc get node node1.lab.example.com --show-labels
NAME STATUS AGE
LABELS
node1.lab.example.com Ready 1d
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,
kubernetes.io/hostname=node1.lab.example.com,region=infra
```

Notice that a node might have a few default labels assigned by OpenShift. Labels whose keys include the **kubernetes.io** suffix should not be changed by a cluster administrator because they are used internally by the scheduler.

Cluster administrators can also use the **-L** option to determine the value of a single label. For example:

```
$ oc get node node1.lab.example.com -L region
NAME STATUS AGE REGION
node1.lab.example.com Ready 1d infra
```

Multiple **-L** options in the same **oc get** command are supported, for example:

```
$ oc get node node1.lab.example.com -L region -L zone
NAME STATUS AGE REGION ZONE
node1.lab.example.com Ready 1d infra <none>
```

Unschedulable Nodes

Sometimes a cluster administrator needs to take a node down for maintenance. The node might need a hardware upgrade or a kernel security update. To take the node down with minimum impact on the OpenShift cluster users, the administrator should follow a two-step process:

1. Mark the node as unschedulable. This prevents the scheduler from assigning new pods to the node.

To mark a node as unschedulable, use the **oc adm manage-node** command:

```
$ oc adm manage-node --schedulable=false node2.lab.example.com
```

2. Drain the node. This destroys all pods running in the pod, and assumes these pods will be recreated in the remaining available nodes by a deployment configuration.

To drain a node, use the **oc adm drain** command:

```
$ oc adm drain node2.lab.example.com
```

When the maintenance operation is complete, use **oc adm manage-node** command to mark the node to schedulable:

```
$ oc adm manage-node --schedulable=true node2.lab.example.com
```

Controlling Pod Placement

Some applications might require running on a specific set of nodes. For example, certain nodes provide hardware acceleration for certain types of workloads, or the cluster administrator does not want to mix production applications with development applications. Whatever the need, node labels and node selectors are used to implement these kinds of scenarios.

A *node selector* is part of a pod definition, but it is recommended to change the deployment configuration, and not the pod definition. To add a node selector, change the pod definition using either the **oc edit** command or the **oc patch** command. For example, to configure the **myapp** deployment configuration so that its pods only run on nodes that have the **env=qa** label, use the following command:

```
$ oc patch dc myapp --patch '{"spec":{"template":{"nodeSelector":{"env":"qa"}}}}'
```

This change triggers a new deployment, and the new pods are scheduled according to the new node selector.

If the cluster administrator does not want to let developers control the node selector for their pods, a default node selector should be configured in the project resource.

Managing the default Project

A common practice for production setups is to dedicate a set of nodes to run OpenShift infrastructure pods, such as the router and the internal registry. Those pods are defined in the **default** project.

The standard implementation of this practice consists of two steps:

1. Label the dedicated nodes with the **region=infra** label.
2. Configure a default node selector for the **default** namespace.

To configure a default node selector for a project, add an *annotation* to the namespace resource with the **openshift.io/node-selector** key. You can use either the **oc edit** or the **oc annotate** command. The following example uses the **oc annotate** command:

```
$ oc annotate --overwrite namespace default \
  openshift.io/node-selector='region=infra'
```

The Ansible Playbooks for the Red Hat OpenShift Container Platform 3.5 quick installer and advanced installer support Ansible variables that control labels assigned to nodes during

installation, and also variables that control node selectors assigned to each infrastructure pod. Playbooks that install subsystems such as the metrics subsystem also support variables for these subsystem node selectors. See the product documentation for details.



References

Further information about the scheduler configuration is available in the *Scheduler* chapter of the *OpenShift Container Platform Cluster Administration Guide* at
| https://access.redhat.com/documentation/en-us/openshift_container_platform

Further information about the Ansible variables related to node selectors is available in the *Advanced Installation* chapter of the *OpenShift Container Platform Cluster Installation and Configuration Guide* at
| https://access.redhat.com/documentation/en-us/openshift_container_platform

Guided Exercise: Controlling Pod Scheduling

In this lab, you will deploy an application configured to run in a specific subset of the cluster nodes. Then you will quiesce a node for maintenance with minimal impact to application users.

Outcomes

You should be able to configure a node selector to restrict pods from an application to run in a subset of cluster nodes. Later you will prepare a node for maintenance by deleting all pods in the node and recreating these pods in another node.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** VMs, and then run the following commands on the **workstation** host:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** VMs are started, and to download the files needed by this guided exercise, open a terminal and run the following command:

```
[student@workstation ~]$ lab schedule-control setup
```

Steps

1. Review the labels for both **node1** and **node2** hosts. Both are in the same region, and pods from the same application are scheduled to be deployed on these nodes.

- 1.1. Log in to OpenShift as the **admin** user:

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

You need to use a cluster administrator user to inspect and later change node labels. In a real-world scenario the administrator would not create the projects and scale the applications, but to make this exercise shorter you will not switch from the administrator to a developer user and back again.

- 1.2. Review the **region** labels on all cluster nodes. They should be as configured by the OpenShift quick installer:

```
[student@workstation ~]$ oc get nodes -L region  
NAME           STATUS     ...   REGION  
master.lab.example.com Ready, SchedulingDisabled ... <none>  
node1.lab.example.com  Ready    ...   infra  
node2.lab.example.com  Ready    ...   infra
```

- 1.3. Create a new project and a new application:

```
[student@workstation ~]$ oc new-project schedule-control
```

```
Now using project "schedule-control" on server "https://
master.lab.example.com:8443".
[student@workstation ~]$ oc new-app --name=hello --docker-image=\
workstation.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry
```

Note that there should be no space after the backslash character in the first line of the **oc new-app** command.

- 1.4. Scale the application to five nodes:

```
[student@workstation ~]$ oc scale dc/hello --replicas=5
```

- 1.5. Verify that the application pods are spread across multiple nodes. Wait until all five pods are ready and running:

```
[student@workstation ~]$ oc get pod -o wide
NAME      READY   STATUS    ...   IP          NODE
hello-1-fkrrd  1/1   Running  ...  10.128.0.91  node2.lab.example.com
hello-1-gmpmt  1/1   Running  ...  10.129.0.67  node1.lab.example.com
hello-1-k7sr3   1/1   Running  ...  10.128.0.93  node2.lab.example.com
hello-1-rd661   1/1   Running  ...  10.128.0.92  node2.lab.example.com
hello-1-wfx33   1/1   Running  ...  10.129.0.66  node1.lab.example.com
```

At this point, it does not matter which node each pod was scheduled to run on. All that matters is there are pods in both **node1** and **node2**.

2. Change the **region** label on **node2** to **apps**.

- 2.1. Run the following command to change the label:

```
[student@workstation ~]$ oc label node node2.lab.example.com \
region=apps --overwrite=true
```

- 2.2. Verify that the label was changed on **node2** only:

```
[student@workstation ~]$ oc get nodes -L region
NAME           STATUS     ...   REGION
master.lab.example.com Ready, SchedulingDisabled  ...  <none>
node1.lab.example.com Ready     ...  infra
node2.lab.example.com Ready     ...  apps
```

3. Configure the deployment configuration to request the nodes to be scheduled only to run on nodes in the **apps** region.

- 3.1. Use the **oc get** command to export the deployment configuration created by the **oc new-app** command to a YAML file:

```
[student@workstation ~]$ oc get dc/hello -o yaml > dc.yaml
```

- 3.2. Add a node selector to the pod template inside the deployment configuration.

Open the **dc.yml** file with a text editor. Notice there are two **spec** attributes in the deployment configuration. Add the following two lines to the second one, which is the **specs** section of the **template** group:

```
...  
    spec:  
      nodeSelector:  
        region: apps  
      containers:  
...
```

3.3. Apply the changes to the deployment configuration:

```
[student@workstation ~]$ oc apply -f dc.yml  
deploymentconfig "hello" configured
```

3.4. Verify that a new deployment was triggered, and wait for all the new application pods to be ready and running. All five pods should be scheduled to **node2**:

```
[student@workstation ~]$ oc get pod -o wide  
NAME      READY   STATUS    ...   IP          NODE  
hello-2-265bh  1/1    Running   ...  10.128.0.115  node2.lab.example.com  
hello-2-dj136  1/1    Running   ...  10.128.0.117  node2.lab.example.com  
hello-2-g91kb  1/1    Running   ...  10.128.0.118  node2.lab.example.com  
hello-2-mdjbg  1/1    Running   ...  10.128.0.116  node2.lab.example.com  
hello-2-v22hv  1/1    Running   ...  10.128.0.114  node2.lab.example.com
```

4. Add **node1** to the **apps** region.

At this point, the OpenShift cluster is configured in a way that **node2** is the only node in the **apps** region, and if it is quiesced, there are no other nodes eligible to run the **hello** application pods.

4.1. Change the **region** label on **node1** to **apps**:

```
[student@workstation ~]$ oc label node node1.lab.example.com \  
region=apps --overwrite=true  
node "node1.lab.example.com" labeled
```

4.2. Verify that the label was changed on **node1** only, and that both nodes are in the **apps** region:

```
[student@workstation ~]$ oc get nodes -L region  
NAME           STATUS     ...   REGION  
master.lab.example.com  Ready, SchedulingDisabled  ...  <none>  
node1.lab.example.com   Ready     ...  apps  
node2.lab.example.com   Ready     ...  apps
```

5. Quiesce the **node2** host.

5.1. Run the following command to disable scheduling on **node2**:

```
[student@workstation ~]$ oc adm manage-node --schedulable=false \
node2.lab.example.com
NAME STATUS AGE
node2.lab.example.com Ready, SchedulingDisabled 1h
```

5.2. Delete all the pods on **node2** and create replacement pods on **node1**.

Run the following command:

```
[student@workstation ~]$ oc adm drain node2.lab.example.com
node "node2.lab.example.com" already cordoned
pod "router-1-v3rgv" evicted
...
hello-2-265bh evicted
hello-2-dj136 evicted
...
node "node2.lab.example.com" drained
```

5.3. Ensure that all the application pods were re-created on **node1**:

```
[student@workstation ~]$ oc get pods -o wide
NAME READY STATUS ... IP NODE
hello-2-dtbp9 1/1 Running ... 10.129.0.88 node1.lab.example.com
hello-2-g502k 1/1 Running ... 10.129.0.87 node1.lab.example.com
hello-2-tr4cz 1/1 Running ... 10.129.0.85 node1.lab.example.com
hello-2-x3nh5 1/1 Running ... 10.129.0.86 node1.lab.example.com
hello-2-z3w7w 1/1 Running ... 10.129.0.84 node1.lab.example.com
```

6. Clean up the lab environment.

6.1. Revert the status on the **node2** host to be schedulable:

```
[student@workstation ~]$ oc adm manage-node --schedulable=true \
node2.lab.example.com
NAME STATUS AGE
node2.lab.example.com Ready 1h
```

6.2. Change the **region** label on both **node1** and **node2** to **infra**:

```
[student@workstation ~]$ oc label node node1.lab.example.com \
region=infra --overwrite=true
node "node1.lab.example.com" labeled
[student@workstation ~]$ oc label node node2.lab.example.com \
region=infra --overwrite=true
node "node2.lab.example.com" labeled
```

6.3. Verify that the label was changed on both nodes, and that both are schedulable:

```
[student@workstation ~]$ oc get nodes -L region
NAME STATUS ... REGION
master.lab.example.com Ready, SchedulingDisabled ... <none>
node1.lab.example.com Ready ... infra
node2.lab.example.com Ready ... infra
```

6.4. Delete the **schedule-control** project by running the following command:

```
[student@workstation ~]$ oc delete project schedule-control  
project "schedule-control" deleted
```

This concludes this exercise.

Managing Images, Image Streams, and Templates

Objective

After completing this section, you should be able to manage images, image streams, and templates.

Introduction to Images

In OpenShift terminology, an image is a deployable runtime template that includes all of the requirements for running a single container, and which includes metadata that describes the image needs and capabilities. Images can be administered in multiple ways; they can be tagged, imported, pulled, and updated. Images can be deployed in multiple containers across multiple hosts. Developers can either use Docker to build images or use OpenShift builder tools.

OpenShift implements a flexible image management mechanism; a single image name can actually refer to many different versions of the same image. A unique image is referenced by its sha256 hash. Docker does not use version numbers; rather, it uses *tags* to manage images, such as **v1**, **v2**, or the default **latest** tag.

Image Streams

An image stream comprises any number of container images identified by tags. It is a consolidated virtual view of related images, similar to a Docker image repository. Developers have many ways of interacting with images and image streams. For example, builds and deployments can receive notifications when new images are added or modified and react accordingly by running a new build or a new deployment.

The following example illustrates an image stream definition:

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2016-01-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample
  tags:
    - items:
        - created: 2016-01-29T13:40:11Z
          dockerImageReference: 172.30.56.218:5000/test/origin-ruby-sample@sha256:
[...]f7dd13d
        generation: 1
        image: sha256:4[...]f7dd13d
```

```
tag: latest
```

Tagging Images

OpenShift Container Platform provides the **oc tag** command, which is similar to the **docker tag** command, however, it operates on image streams instead of images.

You can add *tags* to images to make it easier to determine what they contain. A tag is an identifier that specifies the version of the image. For example, if developers have an image that contains the **2.4** version of the Apache web server, they can tag their image as follows:

```
apache:2.4
```

If the repository contains the latest version of the Apache web server, they can use the **latest** tag to indicate that this is the latest image available in the repository:

```
apache:latest
```

The **oc tag** command is used for tagging images:

```
[user@demo ~]$ oc tag source① destination②
```

- ① The existing tag or image from an image stream.
- ② The most recent image for a tag in one or more image streams.

For example, to configure a **ruby** image's **latest** tag to always refer to the current image for the tag **2.0**, run the following command:

```
[user@demo ~]$ oc tag ruby:latest ruby:2.0
```

To remove a tag from an image, use the **-d** parameter:

```
[user@demo ~]$ oc tag -d ruby:latest
```

Different types of tags are available. The default behavior uses a *permanent* tag, which points to a specific image in time even when the source changes; it is not reflected in the destination tag. A **tracking** tag instructs the destination tag's metadata to be imported during the import of the image. To ensure the destination tag is updated whenever the source tag changes, use the **--alias=true** flag:

```
[user@demo ~]$ oc tag --alias=true source destination
```

To reimport the tag, use the **--scheduled=true** flag.

```
[user@demo ~]$ oc tag --scheduled=true source destination
```

To instruct Docker to always fetch the tagged image from the integrated registry, use the **--reference-policy=local** flag. By default, image blobs are mirrored locally by the registry. As

a result, they can be pulled more quickly the next time they are needed. The flag also allows for pulling from insecure registries without a need to supply the `--insecure-registry` option to the Docker daemon if the image stream has an insecure annotation, or the tag has an insecure import policy.

```
[user@demo ~]$ oc tag --reference-policy=local source destination
```

Recommended Tagging Conventions

Developers should take into consideration the life cycle of an image when managing tags. If there is too much information embedded in a tag name, such as `v2.0.1-may-2017`, the tag will point to just one revision of an image and will never be updated. The default image pruning options mean that such an image will never be removed. In very large clusters, the practice of creating new tags for every revised image could eventually fill up the data store with tag metadata for outdated images. The following table describes possible tag naming conventions that developers can use to manage their images:

Description	Example
Revision	<code>myimage:v2.0.1</code>
Architecture	<code>myimage:v2.0-x86_64</code>
Base Image	<code>myimage:v1.2-rhel7</code>
Latest Image	<code>myimage:latest</code>
Latest Stable Image	<code>myimage:stable</code>

Introduction to Templates

A template describes a set of objects with parameters that are processed to produce a list of objects. A template can be processed to create anything that developers have permission to create within a project, such as services, builds, configurations, and deployment configurations. A template can also define a set of labels to apply to every object that it defines. Developers can create a list of objects from a template using the command-line interface or the web console.

Managing Templates

Developers can write their templates in JSON or YAML format, and import them using the command-line interface or the web console. Templates are saved to the project for repeated use by any user with appropriate access to that specific project. The following command shows how to import a template using the command-line interface.

```
[user@demo ~]$ oc create -f filename
```

Labels can also be assigned while importing the template. This means that all objects defined by the template will be labeled.

```
[user@demo ~]$ oc create -f filename -l name=mylabel
```

The following listing shows a basic template definition:

```
apiVersion: v1
kind: Template①
```

```

metadata:
  name: redis-template2
  annotations:
    description: "Description"
    tags: "database,nosql"3
objects:
- apiVersion: v1
  kind: Pod4
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
        - name: REDIS_PASSWORD5
          value: ${REDIS_PASSWORD}
        image: dockerfile/redis
        name: master
        ports:
        - containerPort: 6379
          protocol: TCP
parameters:6
- description: Password used for Redis authentication
  from: '[A-Z0-9]{8}'
  generate: expression
  name: REDIS_PASSWORD
labels:
  redis: master

```

- ¹ Defines the file as a template.
- ² Specifies the name of the template.
- ³ Applies tags to the template. Tags can be used for searching and grouping.
- ⁴ Declares a pod as a resource for this template.
- ⁵ Defines environment variables for the pod defined in the template.
- ⁶ Sets parameters for the template. Parameters allow a value to be supplied by the user or generated when the template is instantiated.

Instant App and QuickStart Templates

OpenShift Container Platform provides a number of default *Instant App* and *QuickStart* templates that allow developers to quickly create new applications for different languages. Templates are provided for Rails (Ruby), Django (Python), Node.js, CakePHP (PHP), and Dancer (Perl).

To list the available templates in the cluster, run the **oc get templates** command. The **-n** parameter specifies the project to use.

```
[user@demo ~]$ oc get templates -n openshift
```

Developers can also use the web console to browse templates. When you select a template, the available parameters can be adjusted to customize the resources defined by the template.

The screenshot shows the OpenShift web console's Catalog page. At the top, there are three navigation links: "Browse Catalog", "Deploy Image", and "Import YAML / JSON". Below these, a search bar allows filtering by name or description. The main content area is divided into sections:

- Languages:** Java (with icon), JS (JavaScript) (with icon), .NET (with icon), Perl (with icon).
- Technologies:**
 - Continuous Integration & Deployment:** Automate the build, test, and deployment of your application with each new code revision.
 - Data Stores:** Store and manage collections of data.
 - Single Sign-On:** A centralized authentication server for users to log in, log out, register, and manage user accounts for applications and RESTful web services.

Figure 7.8: Templates in the web console

The screenshot shows the OpenShift web console's Catalog page for the "Rails + PostgreSQL (Persistent)" template. The template icon is a stylized globe with red and blue segments. The template name is displayed above the configuration sections.

Images:

- openshift/ruby:2.3 from parameter Namespace
- rails-pgsql-persistent:latest from parameter Name
- postgres:9.5

Parameters:

*** Name:** rails-pgsql-persistent
The name assigned to all of the frontend objects defined in this template.

*** Namespace:** openshift
The OpenShift Namespace where the ImageStream resides.

Figure 7.9: Template parameters



References

Additional information about deployments is available in the *Templates* section of the *OpenShift Container Platform Developer Guide* which can be found at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Additional information about deployments is available in the *Managing Images* section of the *OpenShift Container Platform Developer Guide* which can be found at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Managing Image Streams

In this lab, you will update an existing image stream to deploy images that were recently updated in the OpenShift internal registry.

Resources	
Files	/home/student/D0280/labs/schedule-is (workstation VM) /root/D0280/labs/schedule-is/phpmyadmin-latest.tar (master VM)

Outcomes

You should be able to automatically update application pods after a new image is pushed to the OpenShift internal registry.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts, and then run the following commands from the **workstation** host:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts have started, to download the files needed by this guided exercise, and to deploy a phpMyAdmin application, open a terminal and run the following command:

```
[student@workstation ~]$ lab schedule-is setup
```

Steps

- Evaluate the pods deployed on the project.

In the previous command, a **phpmyadmin** image is pulled from the external image registry (**workstation.lab.example.com:5000**) and pushed to the OpenShift internal registry, using the **oc new-app** command.

- On the **workstation** host, log in as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- Connect to the **schedule-is** project:

```
[student@workstation ~]$ oc project schedule-is
```

- In this project, there is a single pod running. The number "2" in its name indicates that this is the second deployment of the application. Inspect the pods deployed on this project and wait for the **phpmyadmin** pod to be ready and running.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
phpmyadmin-2-r47m3   1/1     Running   0          27m
```

2. Customize the image stream to capture new updates identified in the internal docker registry.

The image stream does not contain any configuration that specifies when a new image deployment should occur . In this step, configure the image stream to monitor any change made to the **phpmyadmin** image in the internal OpenShift registry. The image stream should not monitor the external registry (**workstation.lab.example.com:5000**) because it does not generate events that notify OpenShift about image changes.

- 2.1. Log in as an administrative user. The next step requires cluster administrator privileges.

From the terminal window, run the following command:

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

- 2.2. OpenShift provides access to the internal image registry using an IP address that is visible internally to the OpenShift cluster. The registry is available in the **default** project. To get the IP address, run the following command:

```
[student@workstation ~]$ oc get svc docker-registry -n default
NAME           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
docker-registry   172.30.213.201  <none>       5000/TCP     3h
```

- 2.3. Log in as the **developer** user to continue working on the **schedule-is** project.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 2.4. Export the image stream from the phpMyAdmin application to customize its configuration.

To identify changes in an image, the image stream must be customized to monitor an image registry. Run the following command from the terminal window:

```
[student@workstation ~]$ oc export is phpmyadmin > is.yaml
```

- 2.5. Customize the image stream to point to the internal registry, and add a new image change trigger.

Edit the **is.yaml** file to include the following lines, between the **specs:** and the **tags:** lines. The lines can be copied from the **is-trigger.txt** file in the **~/D0280/labs/schedule-is** folder:

```

spec:
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "REGISTRYIP:5000/phpmyadmin:4.7"
    type: "imagechange"
    imageChange:
      from:
        kind: "ImageStreamTag"
        name: "REGISTRYIP:5000/phpmyadmin:4.7"
      tags:
        ...

```

Notice that the **specs:** and the **tags:** lines already exist in the file. You should only add the content between them.

Replace *REGISTRYIP* with the OpenShift internal registry IP address (Cluster IP) you got in *Step 2.2*.

2.6. Update the image stream configuration to support image changes.

To update the image stream from the YAML file, run the following command:

```
[student@workstation ~]$ oc replace -f is.yaml
imagestream "phpmyadmin" replaced
```

2.7. This change triggers a new build, which causes the application to have two new pods: a **phpmyadmin-deploy** pod and a **phpmyadmin-#-xxx** pod.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
phpmyadmin-2-r47m3  1/1     Terminating   0          27m
phpmyadmin-3-deploy 1/1     Running     0          27m
phpmyadmin-3-xpt01  1/1     Creating    0          27m
```

If you take too long to run this command, the deployment pod may be terminated and the output will already look as it does in the next step.

2.8. Wait until the new pod is ready and running:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
phpmyadmin-3-xpt01  1/1     Running     0          27s
```

Make a note of the sequence number in the pod name, so you can refer to it later to ensure that a new deployment was performed. The sequence number is 3 in the sample output.

3. Update the image in the internal image registry to trigger a new deploy process. You will use the local Docker daemon on the **master** host to load a new container image and pull the image into the OpenShift internal registry.

3.1. Connect to the **master** host to update the image in the internal docker registry.

From the terminal window, run the following command:

```
[student@workstation ~]$ ssh root@master  
[root@master ~]#
```

- 3.2. The service IP address of the internal registry is used for multiple commands in this exercise. To avoid mistakes, save the address to a shell variable using the following command. Copy the command from the **registry-ip.sh** file in the **/root/D0280/labs/schedule-is** folder.

```
[root@master ~]# REGISTRYIP=$(oc get svc docker-registry -n default -o \  
custom-columns='IP:.spec.clusterIP' --no-headers)
```

- 3.3. On the master host, a new docker image for **phpmyadmin** is available in the **/root/D0280/labs/schedule-is** folder. Load it to the local docker daemon.

Run the following command:

```
[root@master ~]# docker load -i \  
/root/D0280/labs/schedule-is/phpmyadmin-latest.tar
```

- 3.4. Retrieve the image ID from the image loaded to the local docker daemon cache.

Run the following command:

```
[root@master ~]# docker images  
REPOSITORY          TAG      IMAGE ID ...  
phpmyadmin/phpmyadmin    latest   200931982ab6 ...
```

- 3.5. Tag the image with the URL of the internal registry.

Because the image was imported from a file, the image ID should be exactly as seen in the previous command output.

From the terminal window, run the following command:

```
[root@master ~]# docker tag 200931982ab6 \  
${REGISTRYIP}:5000/schedule-is/phpmyadmin:4.7
```

- 3.6. Verify that the image tag is set:

```
[root@master ~]# docker images  
REPOSITORY          TAG      IMAGE ID ...  
172.30.213.201:5000/schedule-is/phpmyadmin    4.7    200931982ab6 ...  
phpmyadmin/phpmyadmin           latest   200931982ab6 ...
```

- 3.7. Log in to OpenShift as the cluster administrator.

From the terminal window, run the following command:

```
[root@master ~]# oc login -u admin -p redhat \
https://master.lab.example.com:8443
```

3.8. Get the authentication token to access the OpenShift APIs.

From the terminal window, run the following command:

```
[root@master ~]# TOKEN=$(oc whoami -t)
```

3.9. Log in to the internal image registry using the token.

From the terminal window, run the following command:

```
[root@master ~]# docker login -u admin -p ${TOKEN} ${REGISTRYIP}:5000
Login Succeeded
```

3.10. Update the image by pushing the image from the local docker daemon to the internal docker registry.

From the terminal window, run the following command:

```
[root@master ~]# docker push ${REGISTRYIP}:5000/schedule-is/phpmyadmin:4.7
f33dd3bcd071: Pushed
1b337c46652a: Pushed
2b17eb777748: Pushed
3e8fd20cdf2c: Pushed
6e75928f8d95: Pushed
e7fd82f2e8dd: Pushed
040fd7841192: Pushed
4.7: digest:
sha256:50f622a84d0208e0a7947903442e37d2046490e00c5255f310cee13e5d0c5598 size:
9754
```

4. Verify that the new image triggered a new deploy process.

This step is performed on the **workstation** host.

List the available pods to verify that the build has completed and a new **phpmyadmin** pod has a status of *Running*:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
phpmyadmin-4-deploy  0/1     Completed   0          3m
phpmyadmin-4-tbt4g  1/1     Running    0          2m
```

Because a new image was pushed to the docker registry, a new pod is created.

If you take too long to type this command, the deployer pod may not be shown. Inspect the application pod sequence number: it should be higher than the one you got from *Step 2.8*.

5. Clean up. Run the following command to delete the project:

```
[student@workstation ~]$ oc delete project schedule-is  
project "schedule-is" deleted
```

This concludes the guided exercise.

Lab: Managing Application Deployments

In this lab, you will manage pods to run ordinary maintenance tasks on an OpenShift cluster.

Resources	
Application URL:	http://version.cloudapps.lab.example.com

Outcomes

You should be able to improve scalability by increasing the number of running pods, restrict them to run on a single node, and roll back the deployment to a prior version.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the Ansible Playbook using the following commands on the **workstation** host:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

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

Steps

1. Update the **region** label on the **node1** node to **services**, and on the **node2** node to **applications**.
2. As the OpenShift **admin** user, create a new project named **schedule-review**.
3. Deploy the new **version** application scaled to three pods. The application source code is available at <http://workstation.lab.example.com/version>.
4. Configure the deployment configuration to request pods to be scheduled only to the **applications** region.
5. Verify that a new deployment was started and a new set of version pods are running on the **node2** node. Wait for all three new application pods to be ready and running.
6. Change the **region** label on the **node1** node to **applications**, in preparation for maintenance of the **node2** node.
7. Prepare the **node2** node for maintenance, by setting it to *unschedulable* and then draining the node. Delete all of its pods and recreate them on the **node1** node.
8. Create a route to allow external communication with the **version** application. The route must be accessible using the host name **version.cloudapps.lab.example.com**.

9. Test the application using the **curl** command.

The exact version string depends on previous exercises that use the same Git repository.

10. Grade your work.

Run the following command to grade your work:

```
[student@workstation ~]$ lab manage-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

11. Clean up. Revert the **node1** and **node2** hosts to use the **region: infra** label and have both hosts configured as schedulable.

This concludes the lab.

Solution

In this lab, you will manage pods to run ordinary maintenance tasks on an OpenShift cluster.

Resources	
Application URL:	http://version.cloudapps.lab.example.com

Outcomes

You should be able to improve scalability by increasing the number of running pods, restrict them to run on a single node, and roll back the deployment to a prior version.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the Ansible Playbook using the following commands on the **workstation** host:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started and to download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

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

Steps

1. Update the **region** label on the **node1** node to **services**, and on the **node2** node to **applications**.

- 1.1. Log in to OpenShift as the **admin** user:

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

- 1.2. Verify the current node labels. They should be as configured by the OpenShift quick installer:

```
[student@workstation ~]$ oc get nodes -L region
```

The expected output is:

NAME	STATUS	... REGION
master.lab.example.com	Ready, SchedulingDisabled	... <none>
node1.lab.example.com	Ready	... infra
node2.lab.example.com	Ready	... infra

- 1.3. Update the **region** label on **node1** to **services**:

```
[student@workstation ~]$ oc label node node1.lab.example.com \
region=services --overwrite=true
```

1.4. Update the **region** label on **node2** to **applications**:

```
[student@workstation ~]$ oc label node node2.lab.example.com \
    region=applications --overwrite=true
```

1.5. Verify that the labels were changed on both nodes:

```
[student@workstation ~]$ oc get nodes -L region
```

The expected output is as follows:

NAME	STATUS	...	REGION
master.lab.example.com	Ready, SchedulingDisabled	...	<none>
node1.lab.example.com	Ready	...	services
node2.lab.example.com	Ready	...	applications

2. As the OpenShift **admin** user, create a new project named **schedule-review**.

Run the following command to create a new project:

```
[student@workstation ~]$ oc new-project manage-review
Now using project "schedule-review" on server "https://master.lab.example.com:8443".
```

3. Deploy the new **version** application scaled to three pods. The application source code is available at <http://workstation.lab.example.com/version>.

3.1. Run the following command in the terminal window:

```
[student@workstation ~]$ oc new-app http://workstation.lab.example.com/version
```

3.2. Increase the number of pods for the version application to three.

```
[student@workstation ~]$ oc scale dc/version --replicas=3
```

3.3. Wait for the build to finish and the three application pods to be ready and running. Verify that they were scheduled to run on the same node:

```
[student@workstation ~]$ oc get pod -o wide
NAME      READY   STATUS    ...   NODE
version-1-build  0/1     Completed  ...   node1.lab.example.com
version-1-f39t7  1/1     Running   ...   node2.lab.example.com
version-1-j9b41  1/1     Running   ...   node2.lab.example.com
version-1-rq6q4  1/1     Running   ...   node2.lab.example.com
```

Notice that the application pods were not scattered between both cluster nodes because each node belongs to a different region, and the default OpenShift scheduler configuration has region affinity turned on. It does not matter which node the scheduler selects because you will add a node selector to force the pods to run on the **node2** node.

4. Configure the deployment configuration to request pods to be scheduled only to the **applications** region.

- 4.1. Export the deployment configuration definition to change the label:

```
[student@workstation ~]$ oc get dc/version -o yaml > version-dc.yml
```

- 4.2. Update the YAML file to include a node selector.

Open the **version-dc.yml** file with a text editor and add the following two lines:

```
...
  template:
    metadata:
      ...
    spec:
      nodeSelector:
        region: applications
      containers:
        - image: ...
...
...
```

These lines should be added after the second **spec** attribute in the file. Make sure the indentation is correct, according to the previous listing.

- 4.3. Apply changes from the YAML file to the deployment configuration:

```
[student@workstation ~]$ oc apply -f version-dc.yml
deploymentconfig "configured" replaced
```

5. Verify that a new deployment was started and a new set of version pods are running on the **node2** node. Wait for all three new application pods to be ready and running.

```
[student@workstation ~]$ oc get pod -o wide
NAME          READY   STATUS    ...   NODE
version-1-build  0/1     Completed  ...   node1.lab.example.com
version-2-g36mp  1/1     Running   ...   node2.lab.example.com
version-2-k1rs8  1/1     Running   ...   node2.lab.example.com
version-2-w0sn2  1/1     Running   ...   node2.lab.example.com
```

6. Change the **region** label on the **node1** node to **applications**, in preparation for maintenance of the **node2** node.

- 6.1. Update the **region** label on **node1** to **applications**:

```
[student@workstation ~]$ oc label node node1.lab.example.com \
region=applications --overwrite=true
```

- 6.2. Verify that the label was changed only on the **node1** node:

```
[student@workstation ~]$ oc get nodes -L region
```

The expected output is as follows:

NAME	STATUS	... REGION
master.lab.example.com	Ready, SchedulingDisabled	... <none>
node1.lab.example.com	Ready	... applications
node2.lab.example.com	Ready	... applications

7. Prepare the **node2** node for maintenance, by setting it to *unschedulable* and then draining the node. Delete all of its pods and recreate them on the **node1** node.

- 7.1. Run the following command to disable scheduling on **node2**:

```
[student@workstation ~]$ oc adm manage-node --schedulable=false \
node2.lab.example.com
NAME STATUS AGE
node2.lab.example.com Ready, SchedulingDisabled 1h
```

- 7.2. Delete all the pods from **node2** and recreate them on **node1**:

```
[student@workstation ~]$ oc adm drain node2.lab.example.com
node "node2.lab.example.com" already cordoned
...
pod version-2-g36mp evicted
...
node "node2.lab.example.com" drained
```

- 7.3. Ensure that all three application pods are now scheduled to run on **node1**. Wait for all three pods to be ready and running:

```
[student@workstation ~]$ oc get pods -o wide
NAME READY STATUS ... NODE
version-1-build 0/1 Completed ... node1.lab.example.com
version-2-d5q9s 1/1 Running ... node1.lab.example.com
version-2-sd478 1/1 Running ... node1.lab.example.com
version-2-xmgwt 1/1 Running ... node1.lab.example.com
```

You might not have a builder pod in the output, because it may have been deleted. The builder pod might have been scheduled on **node2**.

8. Create a route to allow external communication with the **version** application. The route must be accessible using the host name **version.cloudapps.lab.example.com**.

```
[student@workstation ~]$ oc expose service version \
--hostname=version.cloudapps.lab.example.com
route "version" exposed
```

9. Test the application using the **curl** command.

```
[student@workstation ~]$ curl http://version.cloudapps.lab.example.com
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<p>Version vX</p>
```

```
</body>
</html>
```

The exact version string depends on previous exercises that use the same Git repository.

10. Grade your work.

Run the following command to grade your work:

```
[student@workstation ~]$ lab manage-review grade
```

If you do not get a PASS grade, review your work and run the grading command again.

11. Clean up. Revert the **node1** and **node2** hosts to use the **region: infra** label and have both hosts configured as schedulable.

11.1. Revert the label on the **node2** host to be schedulable.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc adm manage-node --schedulable=true \
    node2.lab.example.com
NAME          STATUS     AGE
node2.lab.example.com   Ready      1h
```

11.2. Change the **region** label on both nodes back to **infra**:

```
[student@workstation ~]$ oc label node node1.lab.example.com \
    region=infra --overwrite=true
node "node2.lab.example.com" labeled
[student@workstation ~]$ oc label node node2.lab.example.com \
    region=infra --overwrite=true
node "node2.lab.example.com" labeled
```

11.3. Verify that both nodes had their labels changed:

```
[student@workstation ~]$ oc get nodes -L region
```

The expected output is as follows:

NAME	STATUS	... REGION
master.lab.example.com	Ready, SchedulingDisabled	... <none>
node1.lab.example.com	Ready	... infra
node2.lab.example.com	Ready	... infra

11.4. Delete the **manage-review** project.

```
[student@workstation ~]$ oc delete project manage-review
project "manage-review" deleted
```

This concludes the lab.

Summary

In this chapter, you learned:

- A replication controller guarantees that the specified number of pod replicas are running at all times.
- The OpenShift **HorizontalPodAutoscaler** automatically scales based on the current load.
- The scheduler determines placement of new pods onto nodes in the OpenShift cluster. To restrict the set of nodes where a pod can run, cluster administrators label nodes, and a developer defines node selectors.
- Triggers drive the creation of new deployments based on events either inside or outside of OpenShift. Image streams present a single virtual view of related images, similar to a Docker image repository.
- An image stream comprises any number of container images identified by tags. Image streams present a single virtual view of related images, similar to a Docker image repository.



CHAPTER 8

INSTALLING AND CONFIGURING THE METRICS SUBSYSTEM

Overview	
Goal	Install and configure the metrics gathering subsystem.
Objectives	<ul style="list-style-type: none">Describe the architecture and operation of the metrics subsystem.Install the metrics subsystem.
Sections	<ul style="list-style-type: none">Describing the Architecture of the Metrics Subsystem (and Quiz)Installing the Metrics Subsystem (and Guided Exercise)

Describing the Architecture of the Metrics Subsystem

Objective

After completing this section, students should be able to describe the architecture and operation of the Metrics subsystem.

Metrics Subsystem Components

The OpenShift Metrics subsystem enables the capture and long-term storage of performance metrics for an OpenShift cluster. Metrics are collected for nodes and for all containers running in each node.

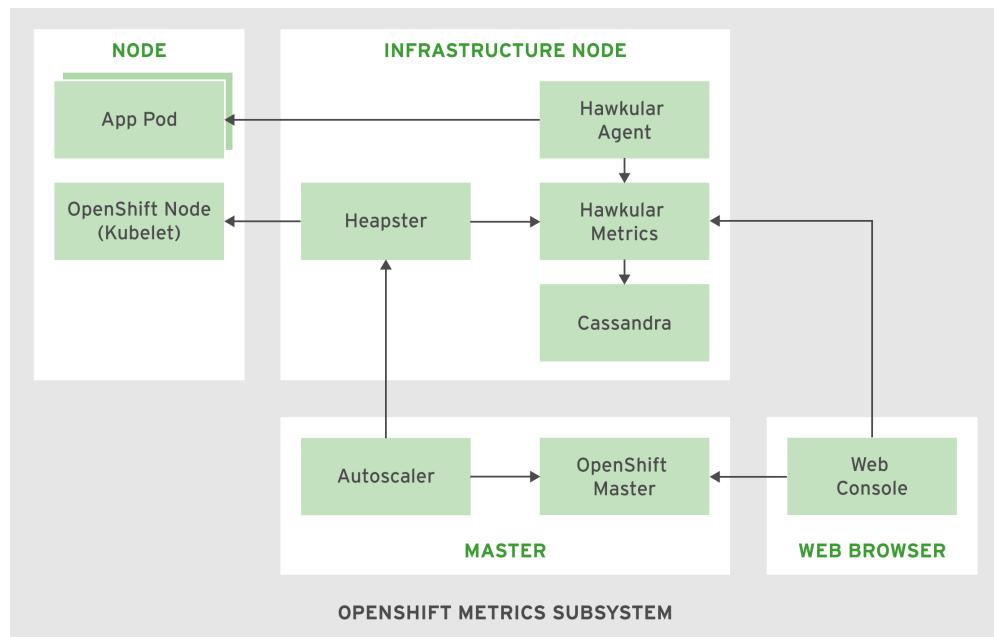


Figure 8.1: OpenShift Metrics subsystem architecture

The Metrics subsystem is deployed as a set of containers based on the following open source projects:

Heapster

Collects metrics from all nodes in a Kubernetes cluster and forwards them to a storage engine for long-term storage. OpenShift uses Hawkular as the storage engine for Heapster.

The Heapster project was started by the Kubernetes community to provide a way for third-party applications to capture performance data from a Kubernetes cluster.

Hawkular Metrics

Provides a REST API for storing and querying time-series data. The Hawkular Metrics component is part of the larger Hawkular project.

Hawkular Metrics uses Cassandra as its data store.

Hawkular was created as the successor to the RHQ Project (the upstream to Red Hat JBoss Operations Network product) and is a key piece of the middleware management capabilities of the Red Hat CloudForms product.

Hawkular Agent

Collects custom performance metrics from applications and forwards them to Hawkular Metrics for storage. The application has to provide metrics in a way that the Hawkular Agent can collect.

The Hawkular Agent is a *Technology Preview* feature at the time of the Red Hat OpenShift Container Platform 3.5 release and is not used during this course.

Cassandra

Stores time-series data in a non-relational, distributed database.

The OpenShift Metrics subsystem works independently of other OpenShift components. Only three parts of OpenShift need the Metrics subsystem in order to provide some optional feature:

- The web console calls the Hawkular Metrics API to fetch data to render performance graphics about pods in a project. If the Metrics subsystem is not deployed, these graphics are not displayed. Notice that the calls are made from the user web browser, not from the OpenShift master.
- The **oc adm top** command uses the Heapster API to fetch data about the current state of all pods and nodes in the cluster.
- The autoscaler controller from Kubernetes calls the Heapster API to fetch data about the current state of all pods from a deployment and make decisions about scaling the deployment controller.

OpenShift does not force an organization to use the full Metrics subsystem. If an organization already has a monitoring system and wants to use it to manage an OpenShift cluster, there is the option of deploying only the Heapster component and to delegate long-term storage of metrics to the external monitoring system.

If the existing monitoring system provides only alerting and health capabilities, then the monitoring system can use the Hawkular API to capture metrics to generate alerts.

Heapster collects metrics for a node and its containers, then aggregates the metrics for pods, namespaces, and the entire cluster. Among the metrics that Heapster collects for a node are:

- *Working set*: the memory effectively used by all processes running in the node, measured in bytes.
- *CPU usage*: the amount of CPU used by all processes running in the node, measured in millicores. Ten millicores is equivalent to one CPU busy 1% of the time.

A complete reference about Heapster Metrics and their meaning can be found in the Heapster storage schema page in the **References** section at the end of this section.

Heapster also supports simple queries against the metrics retained in memory. These queries allows fetching metrics collected and aggregated during a specific time range.

Accessing Heapster and Hawkular

An OpenShift user needs to distinguish between declared resource requests (and limits) versus actual resource usage. The resource requests declared by a pod are used for scheduling. The declared resource requests are subtracted from the node capacity and the difference is the remaining available capacity for a node. The available capacity for a node does not reflect the actual memory and CPU in use by containers and other applications that are running inside a node.

The **oc describe node** command, as of Red Hat OpenShift Container Platform 3.5, only shows information related to resource requests declared by a pod. If a pod does not declare any resource requests, the pod's actual resource usage is not considered, and the node may appear to have more capacity available than it actually does.

The web console shows the same information as the **oc describe node** command, and can also show actual resource usage from Hawkular Metrics. But the web console, as of Red Hat OpenShift Container Platform 3.5, shows metrics only for pods and projects. The web console shows no metrics about a node.

To get actual resource usage for a node, and to determine whether a node is close to its full hardware or virtual capacity, system administrators need to use the **oc adm top** command. If more detailed information is required, system administrators can use standard Linux commands, such as **vmstat** and **ps**. A better option for some use cases is accessing the Heapster API.

OpenShift does not expose the Heapster component to outside the cluster. External applications that need to access Heapster have to use the OpenShift master API proxy feature. The master API proxy ensures accesses to internal component APIs are subject to OpenShift cluster authentication and access control policies.

The following listing shows an example of accessing the Heapster API using the **curl** command.

```
# Assumes MASTERURL, NODE and START env vars are defined in the user environment

APIPROXY=${MASTERURL}/api/v1/proxy/namespaces/openshift-infra/services①
HEAPSTERAPI=https://heapster:8080/api/v1/model②

TOKEN=$(oc whoami -t)③

curl -k -H "Authorization: Bearer $TOKEN" \④
    -X GET $APIPROXY/$HEAPSTERAPI/$NODE/memory/working_set?start=$START⑤
```

- ① Set the URL for the master proxy service. Notice it proxies to services in the **openshift-infra** namespace.
- ② Set the Heapster API URL, without a host name. The service name (**heapster**) replaces the host name in the URL.
- ③ Get the authentication token for the current OpenShift user. It needs at least cluster read privileges.
- ④ Accept insecure SSL certificates (-k) and set the authentication token as an HTTP request header (-H).
- ⑤ Get the working set metric for the node since the specified time stamp. All measurements since the **START** time stamp are returned. The time stamp follows the yyyy-mm-

ddThh:mm:ssZ format, in the UTC timezone. For example: 2017-07-27T17:27:37Z. For the **date** command syntax, the mask is '**+%FT%TZ**'.

Exposing Hawkular to external access involves some security considerations. More information is available in the Red Hat OpenShift Container Platform product documentation.

If a system administrator considers using the Heapster and Hawkular APIs to be too complicated, the upstream communities on the Origin and Kubernetes open source projects also provide integration to popular open source monitoring tools such as Nagios and Zabbix. Red Hat provides integration with the Red Hat CloudForms product, which might be included as part of the Red Hat OpenShift Container Platform subscription depending on its size.

Sizing the Metrics Subsystem

This topic provides general information about sizing the OpenShift Metrics subsystem. The Red Hat OpenShift Container Platform product documentation, specifically the *Installation and Configuration* document and the *Scaling and Performance Guide*, provide detailed information about sizing the Metrics subsystem for an OpenShift cluster, based on the expected number of nodes and pods.

Each component of the OpenShift Metrics subsystem is deployed using its own deployment controller and is scaled independently of the others. They can be scheduled to run anywhere in the OpenShift cluster, but system administrators will probably reserve a few nodes for the Metrics subsystem pods in a production environment.

Cassandra and Hawkular are Java applications. Hawkular runs inside the JBoss EAP7 application server. Both Hawkular and Cassandra take advantage of large heaps and the defaults are sized for a small to medium OpenShift cluster. A test environment might require changing the defaults to request less memory and CPU resources.

Heapster and Hawkular deployments are sized, scaled, and scheduled using standard OpenShift tools. A small number of Heapster and Hawkular pods can manage metrics for hundreds of OpenShift nodes and thousands of projects.

System administrators can use **oc** commands to configure Heapster and Hawkular deployments; for example: to increase the number of replicas or the amount of resources requested by each pod, but the recommended way to configure these parameters is by changing Ansible variables for the Metrics installation playbook. The next section, about installing the Metrics subsystem, provides more information about configuring these Ansible variables.

Cassandra cannot be scaled and configured using standard **oc** commands, because Cassandra (as is the case for most databases) is not a stateless cloud application. Cassandra has strict storage requirements and each Cassandra pod gets a different deployment configuration. The Metrics installation playbook has to be used to scale and configure the Cassandra deployments.

Providing Persistent Storage for Cassandra

Cassandra can be deployed as a single pod, using a single persistent volume. At least three Cassandra pods are required to achieve high availability (HA) for the Metrics subsystem. Each pod requires an exclusive volume: Cassandra uses a shared-nothing storage architecture.

Although Cassandra can be deployed using ephemeral storage, this means there is a risk of permanent data loss. Using ephemeral storage, that is, an **emptyDir** volume type, is not recommended except for a short-lived test-bed environment.

The amount of storage to use for each Cassandra volume depends not only on the expected cluster size (number of nodes and pods) but also on the resolution and duration of the time series for metrics. The Red Hat OpenShift Container Platform product documentation, specifically the *Installation Guide* and the *Scaling and Performance Guide*, provide detailed information about sizing the persistent volumes used by Cassandra for the Metrics subsystem.

The Metrics installation playbook supports using either statically provisioned persistent volumes or dynamic volumes. Whatever the choice, the playbook creates persistent volume claims based on a prefix, to which a sequential number is appended. Be sure to use the same naming convention for statically provisioned persistent volumes.

References

Further information about installing the metrics subsystem is available in the *Installation Guide* for Red Hat OpenShift Container Platform at
https://access.redhat.com/documentation/en-us/openshift_container_platform

Further information about sizing and configuration for the metrics subsystem is available in the *Scaling and Performance Guide* for Red Hat OpenShift Container Platform at
https://access.redhat.com/documentation/en-us/openshift_container_platform

Upstream open source project documentation:

 | Heapster Project on GitHub
 <https://github.com/kubernetes/heapster>

 | Heapster Storage Schema documentation
 <https://github.com/kubernetes/heapster/blob/master/docs/storage-schema.md>

 | Hawkular Project website
 <http://www.hawkular.org/>

 | Apache Cassandra web site
 <http://cassandra.apache.org/>

 | OpenShift Origin on GitHub
 <https://github.com/openshift/origin>

Quiz: The Metrics Subsystem

Choose the correct answers to the following questions:

1. Which of the OpenShift Metrics subsystem components collects performance metrics from the cluster nodes and its running containers?
 - a. Heapster
 - b. Hawkular Agent
 - c. Hawkular Metrics
 - d. Cassandra

2. Which of the OpenShift Metrics subsystem components uses a persistent volume for long-term storage of metrics?
 - a. Heapster
 - b. Hawkular Agent
 - c. Hawkular Metrics
 - d. Cassandra

3. Which of the OpenShift Metrics subsystem provides the REST API used by the web console to display performance graphics for pods inside a project?
 - a. Heapster
 - b. Hawkular Agent
 - c. Hawkular Metrics
 - d. Cassandra

4. Which two of the following OpenShift features can be used to get current CPU usage information for a node? (Choose two.)
 - a. Add extra columns to the **oc get node** command output using the **-o** option.
 - b. Use the Master API proxy to call the Heapster API.
 - c. Filter the output from **oc describe node** to get the **Allocated resources:** table.
 - d. Open the **Cluster Admin** menu of the web console.
 - e. Use the **oc adm top** command to call the Heapster API.

5. Which four of the following factors need to be considered to size the persistent volumes used by the OpenShift Metrics subsystem? (Choose four.)
 - a. The retention period of the metrics (duration).
 - b. The frequency of the metrics collection (resolution).
 - c. The number of nodes in the cluster.
 - d. The expected total number of pods in the cluster.
 - e. The number of Hawkular pod replicas.
 - f. The number of master nodes in the cluster.

6. Which is the recommended way of changing the OpenShift Metrics subsystem configuration, such as the number of replicas of each pod, or the duration of the storage of metrics?
 - a. Change environment variables in each of the Metrics subsystem deployment configurations.
 - b. Create custom container images for the Metrics subsystem components.
 - c. Run the Metrics installation playbook with new values for its Ansible variables.
 - d. Override the configuration volumes for each of the Metrics subsystem pods in their deployment configurations.

Solution

Choose the correct answers to the following questions:

1. Which of the OpenShift Metrics subsystem components collects performance metrics from the cluster nodes and its running containers?
 - a. **Heapster**
 - b. Hawkular Agent
 - c. Hawkular Metrics
 - d. Cassandra
2. Which of the OpenShift Metrics subsystem components uses a persistent volume for long-term storage of metrics?
 - a. Heapster
 - b. Hawkular Agent
 - c. Hawkular Metrics
 - d. **Cassandra**
3. Which of the OpenShift Metrics subsystem provides the REST API used by the web console to display performance graphics for pods inside a project?
 - a. Heapster
 - b. Hawkular Agent
 - c. **Hawkular Metrics**
 - d. Cassandra
4. Which two of the following OpenShift features can be used to get current CPU usage information for a node? (Choose two.)
 - a. Add extra columns to the `oc get node` command output using the `-o` option.
 - b. **Use the Master API proxy to call the Heapster API.**
 - c. Filter the output from `oc describe node` to get the `Allocated resources:` table.
 - d. Open the Cluster Admin menu of the web console.
 - e. **Use the `oc adm top` command to call the Heapster API.**
5. Which four of the following factors need to be considered to size the persistent volumes used by the OpenShift Metrics subsystem? (Choose four.)
 - a. **The retention period of the metrics (duration).**
 - b. **The frequency of the metrics collection (resolution).**
 - c. **The number of nodes in the cluster.**
 - d. **The expected total number of pods in the cluster.**
 - e. The number of Hawkular pod replicas.
 - f. The number of master nodes in the cluster.
6. Which is the recommended way of changing the OpenShift Metrics subsystem configuration, such as the number of replicas of each pod, or the duration of the storage of metrics?

- a. Change environment variables in each of the Metrics subsystem deployment configurations.
- b. Create custom container images for the Metrics subsystem components.
- c. **Run the Metrics installation playbook with new values for its Ansible variables.**
- d. Override the configuration volumes for each of the Metrics subsystem pods in their deployment configurations.

Installing the Metrics Subsystem

Objective

After completing this section, students should be able to Install the Metrics subsystem.

Deploying the Metrics Subsystem

The OpenShift Metrics subsystem is deployed by an Ansible playbook, which can be executed as part of either the quick or advanced installation paths.

Most of the Metrics subsystem configuration is performed using Ansible variables. These variables are passed either using the command line, for the quick installation method, or using the Ansible inventory file, for the advanced installation method.

The Metrics subsystem requires no configuration in many production environments, and can be installed with default settings by just running the Metrics installation playbook, as in the following example:

```
# ansible-playbook -i OPENSIFT_ANSIBLE_INVENTORY① \
    OPENSIFT_ANSIBLE_DIR/byo/openshift-cluster/openshift-metrics.yml② \
    -e openshift_metrics_install_metrics=True③
```

- ① The Ansible inventory file originally used for installing the OpenShift cluster.
- ② The OpenShift Metrics installation playbook. It usually comes from the *openshift-ansible-playbooks* package which is installed as a dependency of the *atomic-openshift-utils* package. Remember that the *atomic-openshift-utils* package provides the OpenShift quick installer. In this case the *OPENSIFT_ANSIBLE_DIR* path is **/usr/share/ansible/openshift-ansible/playbooks/**.
- ③ The **openshift_metrics_install_metrics** Ansible variable configures the playbook to deploy the Metrics subsystem. The playbook creates deployment configurations, services and other supporting Kubernetes resources for the Metrics subsystem.

The Ansible inventory file can come from the following sources:

1. Written manually for the advanced installation path, usually as **/etc/ansible/hosts**.
2. Created by the quick installer in interactive mode, and saved as **/root/.config/openshift/hosts**.
3. Created by the quick installer in unattended mode, as **/root/hosts**.

To understand the difference between the two quick installer methods, consider that sometimes the quick installer configuration file can use syntax that is invalid for an Ansible inventory file. One example is the configuration of a set of private, insecure registries for container images.

If the quick installer configuration file is not a valid Ansible inventory file, the quick installer creates a new inventory file that should be used for the Metrics installation playbook and a few other OpenShift cluster administration tasks. If not, the quick installer configuration file should be used as the inventory file.

The following example deploys the OpenShift Metrics subsystem using the Ansible inventory created by the quick installer, in unattended mode:

```
# ansible-playbook -i /root/hosts \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster\
/openshift-metrics.yml \
-e openshift_metrics_install_metrics=True
```

The Metrics subsystem installer playbook creates all Kubernetes resources in the **openshift-infra** project. The installer playbook does not configure any node selector to restrict where the Metrics pods are scheduled to run.

Uninstalling the Metrics Subsystem

One way to uninstall the OpenShift Metrics subsystem is to manually delete all of its Kubernetes resources in the **openshift-infra** project. This method requires lots of **oc** commands, and is prone to errors because other OpenShift subsystems are deployed to the same project.

The recommended way to uninstall the Metrics subsystem is to run the installation playbook, but setting the **openshift_metrics_install_metrics** Ansible variable to **False**, as in the following example:

```
# ansible-playbook -i /root/hosts \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster\
/openshift-metrics.yml \
-e openshift_metrics_install_metrics=False
```

Usually, it is not required to uninstall the Metrics subsystem to change the parameters configured using Ansible. It should be sufficient to run the installation playbook with changes to the Metrics subsystem Ansible variables. See the Red Hat OpenShift Container Platform documentation for exceptions to this rule.

Verifying the Deployment of the Metrics Subsystem

After the OpenShift Metrics subsystem playbook finishes, all Cassandra, Hawkular, and Heapster pods should be created and might take some time to initialize. Sometimes the Hawkular and Heapster pods are restarted because the Cassandra pods took too long to initialize, but this creates no issues.

Unless configured otherwise, the installer playbook creates, for each component, a deployment configuration with a single pod, and the **oc get pod** output for the **openshift-infra** project will be similar to the following:

NAME	READY	STATUS	RESTARTS	AGE
hawkular-cassandra-1-kahmr	1/1	Running	0	8m
hawkular-metrics-0jnkb	1/1	Running	0	8m
heapster-43po7	1/1	Running	0	8m

If the Metrics pods take too long to initialize, inspect the logs for each pod for errors, and use the **oc describe** command on each pod and deployment and look for error messages. The common cause for deployment errors are:

- Missing container images. Verify the Metrics subsystem container image tags because they do not follow the same release string pattern that the main OpenShift container images follow.

- Resource requests from the Metrics subsystem pods are too high for the available nodes in the OpenShift cluster.
- The persistent volumes for the Cassandra pods are not provisioned or have non-matching capacity and access modes.

The Ansible variables used to configure resource requests, storage, and other parameters for the metrics subsystem are discussed later in this section.

Post-installation Steps

After all pods are ready and running, a single post-installation step needs to be performed. If this step is skipped, the OpenShift web console cannot display graphics for project metrics, although the underlying metrics subsystem is working properly.

The OpenShift web console is a JavaScript application that accesses the Hawkular API directly, without going through the OpenShift master service. The API is secured using TLS, and by default the TLS certificate is not signed by a trusted certification authority. The end result is the web browser refuses to connect to the Hawkular API endpoints.

A similar issue occurs with the web console itself, after the OpenShift installation, and the solution is the same: have the browser accept the TLS certificate as an exception. To do this, open the Hawkular API welcome page in the web browser, and accept the untrusted TLS certificate.

The Hawkular API welcome page URL is:

`https://hawkular-metrics.<master-wildcard-domain>`

The **master-wildcard-domain** DNS suffix should be the same that is configured in the OpenShift master service and used as default domain for new routes at the time the Metrics installation playbook was executed.

The playbook gets the **master-wildcard-domain** value from the Ansible hosts file, and if the OpenShift master service configuration was changed then they will not match. In this case, provide the new value for the wild card domain as an Ansible variable when executing the Metrics installation playbook.

Ansible Variables for the Metrics Subsystem

The Red Hat OpenShift Container Platform *Install and Configuration* document provides a list of all Ansible variables used by the Metrics installation playbook.

These variables follow an intuitive naming convention and they control various configuration parameters, such as:

- Scale for pods from each component:
 - **openshift_metrics_cassandra_replicas**
 - **openshift_metrics_hawkular_replicas**
 - **openshift_metrics_heapster_replicas**
- Resource requests and limits for pods from each component:
 - **openshift_metrics_cassandra_requests_memory**

- **openshift_metrics_cassandra_limits_memory**
- **openshift_metrics_cassandra_requests_cpu**
- **openshift_metrics_cassandra_limits_cpu**
- And similarly for Hawkular and Heapster, for example:
 openshift_metrics_hawkular_requests_memory and
 openshift_metrics_heapster_requests_memory.
- Resolution and retention parameters for collecting metrics:
 - **openshift_metrics_duration**
 - **openshift_metrics_resolution**
- Persistent volume claim attributes for the Cassandra pods:
 - **openshift_metrics_cassandra_storage_type**
 - **openshift_metrics_cassandra_pvc_prefix**
 - **openshift_metrics_cassandra_pvc_size**
- Registry to use to pull the Metrics subsystem container images:
 - **openshift_metrics_image_prefix**
 - **openshift_metrics_image_version**
- Other configuration parameters:
 - **openshift_metrics_heapster_standalone**
 - **openshift_metrics_hawkular_hostname**

See the Red Hat OpenShift Container Platform *Install and Configuration* document for the definition, default values, and syntax of each of these Ansible variables.

The following example installs the metric subsystem with a custom configuration:

```
ansible-playbook -i /root/hosts \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster\
/openshift-metrics.yml \
    -e openshift_metrics_image_prefix=registry.example.com:5000/openshift3/ose-① \
    -e openshift_metrics_image_version=v3.5-10② \
    -e openshift_metrics_heapster_requests_memory=2Gi③ \
    -e openshift_metrics_hawkular_requests_memory=2Gi④ \
    -e openshift_metrics_cassandra_replicas=3⑤ \
    -e openshift_metrics_cassandra_requests_memory=4Gi⑥ \
    -e openshift_metrics_cassandra_storage_type=pv⑦ \
    -e openshift_metrics_cassandra_pvc_size=256i⑧ \
    -e openshift_metrics_cassandra_pvc_prefix=cassandra⑨ \
```

```
-e openshift_metrics_duration=1510 \
-e openshift_metrics_install_metrics=True
```

- ① ② Container image names for the Metrics subsystem will follow the pattern **registry.example.com:5000/openshift3/ose-<container>:v3.5-10**
- ③ ④ The Heapster and Hawkular pods will request 2 GiB of memory each. A single pod will be created for each component, because no scale was specified.
- ⑤ ⑥ Three Cassandra pods will be created, each requesting 4 GiB of memory.
- ⑦ Cassandra will use persistent volumes for storage.
- ⑧ ⑨ Each Cassandra pod will get a persistent volume claim of 25 GiB, and the name for each claim will follow the pattern **cassandra-<number>**. The system administrator needs to provision matching persistent volumes before running the playbook.
- ⑩ Metrics older than 15 days are purged from Cassandra.

Most of these parameters can be changed using OpenShift **oc** commands, but the recommended way is to run the Metrics installation playbook with updated variable values.



References

Further information about installing the metrics subsystem is available in the *Installation Guide* for Red Hat OpenShift Container Platform at

- | https://access.redhat.com/documentation/en-us/openshift_container_platform

Further information about sizing and configuration for the metrics subsystem is available in the *Scaling and Performance Guide* for Red Hat OpenShift Container Platform at

- | https://access.redhat.com/documentation/en-us/openshift_container_platform

Guided Exercise: Installing the Metrics Subsystem

In this exercise, you will install and configure the OpenShift Metrics subsystem and verify that it is working properly.

Resources	
Files:	/home/student/D0280/labs/install-metrics (workstation VM) /root/D0280/labs/install-metrics (master VM)
Application URL:	https://master.lab.example.com:8443 https://hawkular-metrics.cloudapps.lab.example.com http://hello-load.cloudapps.lab.example.com/

Outcomes

You should be able to use the OpenShift installer playbooks to install the Metrics subsystem into an existing Red Hat OpenShift Container Platform cluster, which was created using the quick installer.

Before you begin

This exercise requires a working OpenShift installation. If you do not have one, run the following commands:

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

Open a terminal window on the workstation VM and run the following command to download the files used during this exercise:

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

Steps

- Verify that the container images required by the Metrics subsystem are in the private registry. The following images are required:

- metrics-cassandra
- metrics-hawkular-metrics
- metrics-heapster

Use the `docker-registry-cli` utility to search for these images:

```
[student@workstation ~]$ docker-registry-cli workstation.lab.example.com:5000 \
    search metrics-cassandra
...
1) Name: openshift3/ose-metrics-cassandra
Tags: v3.5
...
```

```
[student@workstation ~]$ docker-registry-cli workstation.lab.example.com:5000 \
    search metrics-hawkular-metrics
...
1) Name: openshift3/ose-metrics-hawkular-metrics
Tags: v3.5
...
[student@workstation ~]$ docker-registry-cli workstation.lab.example.com:5000 \
    search metrics-heapster
...
1) Name: openshift3/ose-metrics-heapster
Tags: v3.5
...
```

Notice from the search output that all image names have the prefix **openshift3/ose-** and the **v3.5** tag. This information is required by the Metrics subsystem installation playbook.

2. Create a persistent volume for the Metrics subsystem data storage.
- 2.1. Open an SSH session to the **master** host, and perform the Metrics subsystem installation from there.

```
[student@workstation ~]$ ssh root@master
```

- 2.2. The quick installer creates an NFS share for use by the Metrics subsystem. Review the permissions for the **/exports/metrics** folder and the **/etc(exports.d/openshift-ansible.exports** configuration file.

```
[root@master ~]# ls -alZ /exports/metrics/
drwxrwxrwx. nfsnobody nfsnobody unconfined_u:object_r:default_t:s0 .
drwxr-xr-x. root      root      unconfined_u:object_r:default_t:s0 ..
[root@master ~]# cat /etc(exports.d/openshift-ansible.exports
/exports/registry *(rw,root_squash)
/exports/metrics *(rw,root_squash)
/exports/logging-es *(rw,root_squash)
/exports/logging-es-ops *(rw,root_squash)
```

- 2.3. Create a persistent volume (PV) definition file for the NFS share.

Use the following parameters so that the PV uses the NFS share created by the quick installer. These parameter also make sure that only the Cassandra pod can bind to it. The Metrics playbook creates the claim as **metrics-1**.

```
capacity.storage
  5GiB

accessModes
  ReadWriteOnce

nfs.path
  /exports/metrics

nfs.server
  master.lab.example.com
```

persistentVolumeClaimPolicy

Recycle (for a production cluster, many system administrators will prefer **Retain**)

claimRef.name
metrics-1

claimRef.namespace
openshift-infra

The PV definition is ready to use in the **metrics-pv.yml** file in the **/root/D0280/labs/install-metrics** folder. The complete file content is shown below:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: metrics
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /exports/metrics
    server: master.lab.example.com
  persistentVolumeReclaimPolicy: Recycle
  claimRef:
    name: metrics-1
    namespace: openshift-infra
```

- 2.4. Log in as a cluster administrator and create the persistent volume using the YAML file provided in this exercise **labs** folder:

```
[root@master ~]# oc login -u admin -p redhat \
https://master.lab.example.com:8443
...
[root@master ~]# oc create -f \
/root/D0280/labs/install-metrics/metrics-pv.yml
...
```

- 2.5. Verify that the persistent volume was created and is available to be claimed:

```
[root@master ~]# oc get pv
NAME          CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM          REASON      AGE
...
metrics        5Gi        RWO          Recycle         Available
  openshift-infra/metrics-1
...
...
```

3. Install the Metrics subsystem using the playbook provided by the Red Hat OpenShift Container Platform installer.

- 3.1. Determine the Ansible variables to pass to the Metrics playbook.

The following Ansible variables need to be customized to fit the cluster installed in the classroom environment:

openshift_metrics_image_prefix

Points to the private registry on the **workstation** VM, and also adds **openshift3/ose-** as the image name prefix.

openshift_metrics_image_version

The container image tag to use. The private registry provides only the **v3.5** tag.

openshift_metrics_heapster_requests_memory

300 MB is sufficient this course.

openshift_metrics_hawkular_requests_memory

750 MB is sufficient this course.

openshift_metrics_cassandra_requests_memory

750 MB is sufficient this course.

openshift_metrics_cassandra_storage_type

Use **pv** to select a persistent volume as the storage type.

openshift_metrics_cassandra_pvc_size

5 GiB is sufficient this course.

openshift_metrics_cassandra_pvc_prefix

Use **metrics** as the prefix for the PVC name.

Notice that it is necessary to change the values for memory resource requests for each of the Metrics subsystem pods because their default values are too large for the classroom. Without changes, all pods would fail to start.

More information about these variables is available in the *Metrics Ansible Role* section of the Red Hat OpenShift Container Platform 3.5 *Installation and Configuration* document. Scroll down to review the variable definitions provided by the *Ansible Variables* table.

3.2. Inspect the script that runs the Metrics playbook.

A ready-to-use script is available in the **install-metrics.sh** file in the **/root/D0280/labs/install-metrics** folder. Follows the script contents:

```
ansible-playbook -i ~/hosts \
    /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster\
    /openshift-metrics.yml \
    -e openshift_metrics_image_prefix=workstation.lab.example.com:5000\
    /openshift3/ose- \
    -e openshift_metrics_image_version=v3.5 \
    -e openshift_metrics_heapster_requests_memory=300M \
    -e openshift_metrics_hawkular_requests_memory=750M \
    -e openshift_metrics_cassandra_requests_memory=750M \
    -e openshift_metrics_cassandra_storage_type=pv \
    -e openshift_metrics_cassandra_pvc_size=5Gi \
    -e openshift_metrics_cassandra_pvc_prefix=metrics \
    -e openshift_metrics_install_metrics=True
```

Notice that the backslash (\) character is not preceded nor followed by spaces in lines two and four because they are in the middle of values that are too long for the printed page.

The `~/hosts` file is the Ansible inventory file created by the OpenShift quick installer. Notice the long file path to the Metrics playbook (`openshift-metrics.yml`).

- 3.3. Run the script that invokes the Metrics playbook with the variables gathered during the previous step:

```
[root@master ~]# /root/D0280/labs/install-metrics/install-metrics.sh
...
PLAY RECAP ****
localhost          : ok=10    changed=0     unreachable=0    failed=0
master.lab.example.com   : ok=196   changed=48    unreachable=0    failed=0
node1.lab.example.com    : ok=1     changed=0     unreachable=0    failed=0
node2.lab.example.com    : ok=1     changed=0     unreachable=0    failed=0
```

Watch the Ansible output carefully. Incorrect variable names or values might cause some playbook tasks to fail, or cause pod deployment errors later.

A few **FAILED - RETRYING** messages can be safely ignored as long as an **ok** message is displayed immediately afterwards. For example:

```
...
RUNNING HANDLER [openshift_metrics : Verify API Server] ****
FAILED - RETRYING: HANDLER: openshift_metrics : Verify API Server (120 retries left).
FAILED - RETRYING: HANDLER: openshift_metrics : Verify API Server (119 retries left).
ok: [master.lab.example.com]
...
```

Notice these failed messages should not increase the failed count at the end of the playbook execution, after the **PLAY RECAP** message. Verify that the failed count is zero for all hosts before continuing with the next step.



Important

If something goes wrong during this or the following steps, run the `/root/D0280/labs/install-metrics/uninstall-metrics.sh` script to uninstall the Metrics subsystem.

4. Verify that the Metrics subsystem was deployed successfully.

- 4.1. Verify that the persistent volume claim generated by the Metrics playbook is bound:

```
[root@master ~]# oc get pvc -n openshift-infra
NAME      STATUS    VOLUME   CAPACITY  ACCESSMODES  AGE
metrics-1  Bound     metrics   5Gi       RWO          10m
```

Notice that having a PVC bound does not mean the PV definition is correct. The PV attributes are actually used only when a pod tries to mount the volume, so any mistakes in the PV definition will cause errors in the Cassandra startup.

- 4.2. Wait until all Metrics subsystem pods are ready and running. This may take a few minutes.

```
[root@master ~]# oc get pod -n openshift-infra
NAME          READY   STATUS    ...
hawkular-cassandra-1-kdhgr  1/1     Running   ...
hawkular-metrics-0jmkm      1/1     Running   ...
heapster-43p13              1/1     Running   ...
```

Notice that your cluster might schedule each pod to run on different nodes, compared to the previous output.

If something goes wrong, you can obtain diagnostic information using **oc describe** in each of the deployment configurations and pods. You can also use **oc logs** in the Metrics subsystem pods and their respective deployer pods. The fix is probably to correct the Ansible variables in the **~/install-metrics.sh** file and then follow the instructions in the *Important* box from the previous step.

5. Open the Hawkular home page in a web browser to accept its self-signed certificate.

- 5.1. Find the route host name for the Hawkular pod:

```
[root@master ~]# oc get route -n openshift-infra
NAME          HOST/PORT ...
hawkular-metrics  hawkular-metrics.cloudapps.lab.example.com ...
```

- 5.2. Open a web browser on the **workstation** VM and open the Hawkular URL:

https://hawkular-metrics.cloudapps.lab.example.com

Accept the SSL certificate as trusted. The web page shows only the Hawkular mascot and release number.

You need to do this for all web browsers that you use to access the OpenShift web console.

6. Deploy a test application to generate some load in the OpenShift cluster.

- 6.1. Open another terminal on the **workstation** VM. Log in as the **developer** user, create a new project, and a "hello, world" application from the **hello-openshift** container image:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
...
[student@workstation ~]$ oc new-project load
...
[student@workstation ~]$ oc new-app --name=hello --docker-image=\
workstation.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry
```

```
...
```

Notice there are no spaces before or after the backslash (\) character in the first line of the **oc new-app** command.

- 6.2. Scale the application and wait until all nine pods are ready and running:

```
[student@workstation ~]$ oc scale --replicas=9 dc/hello
...
[student@workstation ~]$ oc get pod -o wide
NAME      READY   STATUS    ...   IP          NODE
hello-1-0h950  1/1     Running   ...  10.130.0.17  node1.lab.example.com
...
hello-1-ds6kh  1/1     Running   ...  10.129.0.12  node2.lab.example.com
```

- 6.3. Expose the application to outside access:

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

- 6.4. Install the *httpd-tools* package on the **workstation** VM:

```
[student@workstation ~]$ sudo yum -y install httpd-tools
```

- 6.5. Generate some load using the Apache Bench utility. Note that the trailing forward slash is mandatory at the end of the URL:

```
[student@workstation ~]$ ab -n 300000 -c 20 \
  http://hello-load.cloudapps.lab.example.com/
...
Benchmarking hello-load.cloudapps.lab.example.com (be patient)
Completed 5000 requests
...
```

Continue to the next step while Apache Bench is running.

7. Open another terminal on the **workstation** VM and log in as a cluster administrator user to fetch current metrics from the Heapster pod using the **oc adm top** command:

```
[student@workstation ~]$ oc login -u admin -p redhat
[student@workstation ~]$ oc adm top node --heapster-namespace=openshift-infra \
  --heapster-scheme=https
NAME           CPU(cores)   CPU%    MEMORY(bytes)  MEMORY%
master.lab.example.com  37m       1%    677Mi        36%
node1.lab.example.com    80m      4%    1773Mi       46%
node2.lab.example.com    216m     10%   1756Mi       61%
```

Your numbers might be different.

8. Fetch current metrics from the Heapster pod using the **curl** command.

- 8.1. Open another terminal on the **workstation** VM and inspect the **node-metrics.sh** script. It connects to the Heapster API to fetch current memory and CPU usage metrics for a node.

The script contains the following:

```
[student@workstation ~]$ cat ~/DO280/labs/install-metrics/node-metrics.sh
#!/bin/bash

oc login -u admin -p redhat

TOKEN=$(oc whoami -t)
APIPROXY=https://master.lab.example.com:8443/api/v1/proxy/namespaces/openshift-
infra/services
HEAPSTER=https://heapster:8086/api/v1/model
NODE=nodes/node1.lab.example.com
START=$(date -d '1 minute ago' -u '+%FT%TZ')

curl -kH "Authorization: Bearer $TOKEN" \
-X GET $APIPROXY/$HEAPSTER/$NODE/metrics/memory/working_set?start=$START

curl -kH "Authorization: Bearer $TOKEN" \
-X GET $APIPROXY/$HEAPSTER/$NODE/metrics/cpu/usage_rate?start=$START
```

The script retrieves the **working_set** memory metric, measured in bytes, and the CPU **usage_rate** metric, measured in millicores. The script displays measurements from the last minute.

8.2. Run the script to see the current node metrics:

```
[student@workstation ~]$ ./DO280/labs/install-metrics/node-metrics.sh
...
{
  "metrics": [
    {
      "timestamp": "2017-06-21T21:45:00Z",
      "value": 1856569344 ①
    },
    {
      "timestamp": "2017-06-21T21:45:30Z",
      "value": 1860599808 ②
    }
  ],
  "latestTimestamp": "2017-06-21T21:45:30Z"
}
{
  "metrics": [
    {
      "timestamp": "2017-06-21T21:45:00Z",
      "value": 499 ③
    },
    {
      "timestamp": "2017-06-21T21:45:30Z",
      "value": 1213 ④
    }
  ],
  "latestTimestamp": "2017-06-21T21:45:30Z"
}
```

The sample output shows:

①② Effective memory usage (**working_set**) of about 1.7 GiB.

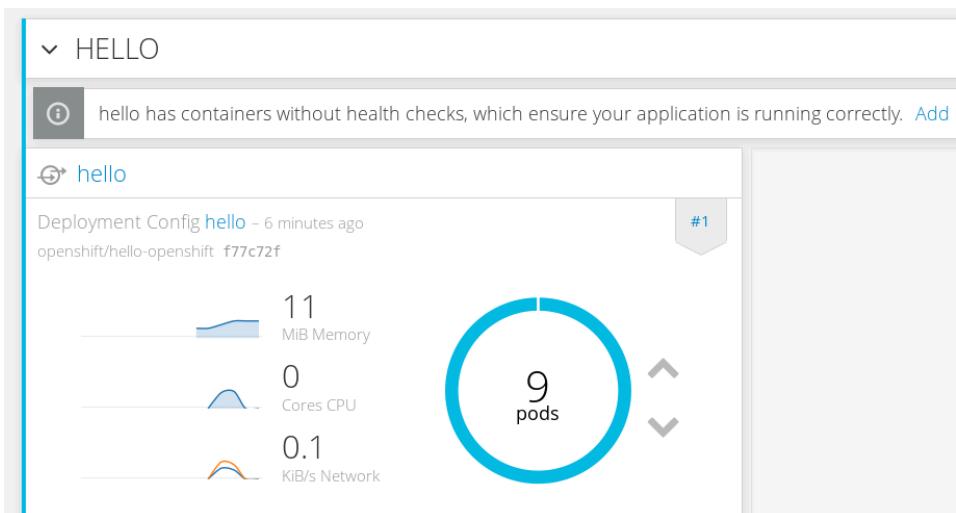
- ③ Approximately half a CPU busy (**usage_rate**).
- ④ Approximately one full CPU busy plus 20% of another CPU busy (**usage_rate**).

If the script does not work, the most probable cause is the Heapster pod not ready and running. You might need to uninstall the metrics subsystem, fix your installation script, and reinstall.

9. View the project metrics from Hawkular, using the OpenShift web console.
 - 9.1. Open a web browser and log in to the OpenShift web console as the **developer** user, using **openshift** as the password.

The web console URL is **https://master.lab.example.com:8443**.

 - 9.2. Enter the **load** project and visit its Overview page. The following image displays several pod metrics:



If you do not see the graphics in the web browser, the most probable causes are:

- The browser window is too narrow. Resize the window and the graphics should be visible.
- You did not visit the Hawkular home page before logging in to the web console. Do this as explained in *Step 5* and refresh the web console page.
- The Hawkular pod is not ready and running. You may need to uninstall the Metrics subsystem, fix your installation script, and reinstall.

- 9.3. Clean up. Delete the test application project:

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

This concludes the guided exercise.

Summary

In this chapter, you learned:

- The Red Hat OpenShift Container Platform provides the optional Metrics subsystem that performs the collection and long-term storage of performance metrics about cluster nodes and containers.
- The Metrics subsystem is composed of three main components that run as containers in the OpenShift cluster:
 - Heapster collects metrics from OpenShift nodes and containers running on each node. The Kubernetes autoscaler needs Heapster to work.
 - Hawkular Metrics stores the metrics and provides querying capabilities. The OpenShift web console needs Hawkular to display performance graphics for a project.
 - Cassandra is the database used by Hawkular to store metrics.
- Heapster and Hawkular Metrics provide REST APIs to integrate with external monitoring systems.
- It is required to use the OpenShift master API proxy to access the Heapster API and retrieve information about a node's current memory usage, CPU usage, and other metrics.
- The recommended way to configure the Metrics subsystem is to run the installer playbook with changed Ansible variables.
- Sizing the Metrics subsystem involves a number of parameters: CPU and memory requests for each pod, capacity of each persistent volume, number of replicas for each pod, and so on. They depend on the number of nodes in the OpenShift cluster, the expected number of pods, the duration of the metrics storage, and resolution of the collection of metrics.
- The Metrics subsystem installation playbook requires the Ansible inventory file used by either the quick or advanced OpenShift installation paths. The same playbook is also used to uninstall and reconfigure the Metrics subsystem.
- After running the installation playbook and verifying that all Metrics subsystem pods are ready and running, all OpenShift users need to visit the Hawkular welcome page to trust its TLS certificate. If this is not done, the web console will not be able to display performance graphics.



CHAPTER 9

MANAGING AND MONITORING OPENSHIFT CONTAINER PLATFORM

Overview	
Goal	Manage and monitor OpenShift resources and software.
Objectives	<ul style="list-style-type: none">• Limit the amount of resources consumed by an application.• Upgrade an instance of OpenShift.• Configure probes to monitor application health.• Monitor OpenShift resources using data obtained from the web console.
Sections	<ul style="list-style-type: none">• Limiting Resource Usage (and Guided Exercise)• Upgrading the OpenShift Container Platform (and Quiz)• Monitoring Applications with Probes (and Guided Exercise)• Monitoring Resources with the Web Console
Lab	Lab: Managing and Monitoring OpenShift

Limiting Resource Usage

Objective

After completing this section, students should be able to limit the resources consumed by an application.

Resource Requests and Limits for Pods

A pod definition can include both resource requests and resource limits:

resource requests

Used for scheduling, and indicate that a pod is not able to run with less than the specified amount of compute resources. The scheduler tries to find a node with sufficient compute resources to satisfy the pod requests.

resource limits

Used to prevent a pod from using up all compute resources from a node. The node that runs a pod configures the Linux kernel cgroups feature to enforce the resource limits for the pod.

Although resource requests and resource limits are part of a pod definition, they are usually set up in a deployment configuration. OpenShift recommended practices prescribe that a pod should not be created stand alone, but should instead be created by a deployment configuration.

Applying Quotas

OpenShift Container Platform can enforce quotas that track and limit the use of two kinds of resources:

Object counts

The number of Kubernetes resources, such as pods, services, and routes.

Compute resources

The number of physical or virtual hardware resources, such as CPU, memory, and storage capacity.

Imposing a quota on the number of Kubernetes resources helps with the stability of the OpenShift master, by avoiding unbounded growth of the master data store (the **Etcdb** database). Having quotas on Kubernetes resources also avoids exhausting other limited software resources, such as IP addresses for services.

In a similar way, imposing a quota on the amount of compute resources avoids exhausting the compute capacity of a single node in an OpenShift cluster. It also avoids having one application using all the cluster capacity, starving other applications that share the cluster.

OpenShift manages quotas for the use of objects and compute resources in a cluster by using a **ResourceQuota** object, or simply a **quota**. The **ResourceQuota** object specifies hard resource usage limits for a project. All attributes of a quota are optional, meaning that any resource that is not restricted by a quota can be consumed without bounds.



Note

A project can contain multiple **ResourceQuota** objects. Their effect is cumulative, but it is expected that two different **ResourceQuota** objects for the same project do not try to limit the use of the same type of Kubernetes or compute resource.

The following table describes some object count quotas that may be enforced by a **ResourceQuota**

Object Count Name	Description
pods	Total number of pods
replicationcontrollers	Total number of replication controllers
services	Total number of services
secrets	Total number of secrets
persistentvolumeclaims	Total number of persistent volume claims

The following table describes some of the compute resource quotas that can be enforced by a **ResourceQuota**.

Compute Resource Name	Description
cpu	Total CPU use across all containers
memory	Total memory use across all containers
storage	Total disk use across all containers



Note

See the Red Hat OpenShift Container Platform 3.5 *Cluster Administration and Developer Guide* documentation for a complete list and the meaning of each resource quota attribute.

Quota attributes can track either the resource requests or the resource limits for all pods in the project. By default, quota attributes tracks resource requests. To track resource limits instead, prefix the compute resource name with **limits**, for example, **limits.cpu**.

The following listing show a **ResourceQuota** resource defined using YAML syntax, and which specifies quotas for both object counts and compute resources:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dev-quota
spec:
  hard:
    services: "10"
    cpu: "1300m"
    memory: "1.5Gi"
```

Resource units are the same for pod resource requests and resource limits, for example: **Gi** means GiB, and **m** means millicores.

Resource quotas can be created the same way as any other OpenShift Container Platform resource; that is, by passing a JSON or YAML resource definition file to the **oc create** command:

```
$ oc create -f dev-quota.yml
```

Another way to create a resource quota is by using the **oc create quota** command, for example:

```
$ oc create quota dev-quota \
--hard=services=10 \
--hard=cpu=1300m \
--hard=memory=1.5Gi
```

Use the **oc get resourcequota** command to list available quotas, and use the **oc describe resourcequota NAME** command to view usage statistics related to any hard limits defined in the quota, for example:

```
$ oc get resourcequota
object-quota
compute-quota
```

```
$ oc describe resourcequota object-quota
Name:          object-quota
Resource           Used     Hard
-----
pods              1        3
replicationcontrollers   1        5
services           1        2
```

```
$ oc describe resourcequota compute-quota
Name:          compute-quota
Resource           Used     Hard
-----
cpu               500m    10
memory            300Mi   1Gi
```

The **oc describe resourcequota** command, without arguments, shows the cumulative limits set for all **ResourceQuota** objects in the project, without displaying which object defines which limit.

```
$ oc describe quota
Resource           Used     Hard
-----
cpu               500m    10
memory            300Mi   1Gi
pods              1        3
replicationcontrollers   1        5
services           1        2
```

An active quota can be deleted by name with the **oc delete resourcequota NAME** command:

```
$ oc delete resourcequota compute-quota
```

When a quota is first created in a project, the project restricts the ability to create any new resources that might violate a quota constraint until it has calculated updated usage statistics. After a quota has been created and usage statistics are up-to-date, the project accepts the creation of new content. When a new resource is created, the quota usage is incremented immediately. When a resource is deleted, the quota use is decremented during the next full recalculation of quota statistics for the project.



Important

ResourceQuota constraints are applied for the project as a whole, but many OpenShift processes, such as builds and deployments, create pods inside the project, and might fail because starting them would exceed the project quota.

If a modification to a project exceeds the quota for an object count, the action is denied by the server, and an appropriate error message is returned to the user. If the modification exceeds the quota for a compute resource, however, the operation does not fail immediately; OpenShift retries the operation several times, giving the administrator an opportunity to increase the quota or to perform another corrective action, such as bringing a new node online.



Important

If a quota that restricts usage of compute resources for a project is set, OpenShift will refuse to create pods that do not specify resource requests or resource limits for that compute resource. Most of the standard S2I builder images and templates provided with OpenShift do not specify these. To use most templates and builders with a project restricted by quotas, the project needs to also contain a limit range object that specifies default values for container resource requests.

Applying Limit Ranges

A **LimitRange** resource, also called a **limit**, defines the default, minimum, and maximum values for compute resource requests and limits for a single pod or for a single container defined inside the project. A resource request or limit for a pod is the sum of its containers.

To understand the difference between a limit range and a resource quota resource, consider that a limit range defines valid ranges and default values for a single pod, while a resource quota defines only top values for the sum of all pods in a project. A cluster administrator concerned about resource usage in an OpenShift cluster usually defines both limits and quotas for a project.

A **LimitRange** resource can also define default, minimum, and maximum values for the storage capacity requested by an image, image stream, or persistent volume claim. If a resource added to a project does not provide a compute resource request, it takes the default value provided by the project's limit ranges. If a new resource provides compute resource requests or limits that are smaller than the minimum specified by the project's limit ranges, the resource is not created. In a similar way, if a new resource provides compute resource requests or limits that are higher than the maximum specified by the project's limit ranges, the resource is not created.

The following table describes some of the compute resources that can be specified by a **LimitRange**. See the Red Hat OpenShift Platform 3.5 *Cluster Administration and Developer Guide* documentation for a complete list and the meaning of each compute resource limit range attribute.

Type	Resource Name	Description
Container	cpu	Minimum and maximum CPU allowed per container
Container	memory	Minimum and maximum memory allowed per container
Pod	cpu	Minimum and maximum CPU allowed across all containers in a pod
Pod	memory	Minimum and maximum memory allowed across all containers in a pod
Image	storage	Maximum size of an image that can be pushed to the internal registry
PVC	storage	Minimum and maximum capacity of the volume that can be requested by one claim

The following listing shows a limit range defined using YAML syntax:

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "dev-limits"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
    - type: "Container"
      default:
        cpu: "1"
        memory: "512Mi"
```

Users can create **LimitRange** resources the same way as any other OpenShift resource; that is, by passing a JSON or YAML resource definition file to the **oc create** command:

```
$ oc create -f dev-limits.yml
```

Red Hat OpenShift Container Platform 3.5 does not provide an **oc create** command specifically for limit ranges like it does for resource quotas. The only alternative is to use YAML files.

Use the **oc describe limitranges NAME** command to view the limit constraints enforced in a project:

```
$ oc get limitranges
compute-limits
storage-limits

$ oc describe limitranges compute-limits

Name:          compute-limits
Type           Resource   Min     Max     Default
-----         -----     ---     ---     ---

```

Pod	cpu	10m	500m	-
Pod	memory	5Mi	750Mi	-
Container	memory	5Mi	750Mi	100Mi
Container	cpu	10m	500m	100m

An active limit range can be deleted by name with the `oc delete limitranges NAME` command:

```
$ oc delete limitranges dev-limits
```

After a limit range has been created in a project, all resource create requests are evaluated against each **LimitRange** resource in the project. If the new resource violates the minimum or maximum constraint enumerated by any **LimitRange**, then the resource is rejected. If the new resource does not set an explicit value, and the constraint supports a default value, then the default value is applied to the new resource as its usage value.

All resource update requests are also evaluated against each **LimitRange** resource in the project. If the updated resource violates any constraint, the update is rejected.



Important

Avoid setting **LimitRange** constraints that are too high, or **ResourceQuota** constraints that are too low. Violation of **LimitRange** constraints prevents pod from being created, showing clear error messages. Violation of **ResourceQuota** constraints prevents a pod from being scheduled to any node. The pod might be created but remain in the pending state.

Applying Quotas to Multiple Projects

The **ClusterResourceQuota** resource is created at cluster level, in a similar way to a persistent volume, and specifies resource constraints that apply to multiple projects.

Developers can specify which projects are subject to cluster resource quotas in either of two ways:

- Using the **openshift.io/requester** annotation to specify the project owner. All projects with the specified owner are subject to the quota.
- Using a selector. All projects whose labels match the selector are subject to the quota.

The following is an example of creating a cluster resource quota for all projects owned by the **qa** user:

```
$ oc create clusterquota user-qa \
--project-annotation-selector openshift.io/requester=qa \
--hard pods=12 \
--hard secrets=20
```

The following is an example of creating a cluster resource quota for all projects that have the **environment=qa** label:

```
$ oc create clusterquota env-qa \
```

```
--project-label-selector environment=qa \  
--hard pods=10 \  
--hard services=5
```

Project users can use the **oc describe QUOTA** command to view cluster resource quotas that applies to the current project, if any.

Use the **oc delete** command to delete a cluster resource quota:

```
$ oc delete clusterquota QUOTA
```

It is not recommended to have a single cluster resource quota that matches over a hundred projects. This is to avoid large locking overheads. When resources in a project are created or updated, the project is locked while searching for all applicable resource quotas.

References

Further information is available in the *Installation Guide* of the Red Hat OpenShift Container Platform 3.5 documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform

Further information is available in the *Quotas and Limit Ranges* chapter of the *Developer Guide* available in the Red Hat OpenShift Container Platform 3.5 documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform

Guided Exercise: Limiting Resource Usage

Outcomes

You should be able to define a resource quota and a resource limit for a project, and troubleshoot quota violation errors. You should also be able to troubleshoot scheduling issues caused by resource requests not limited by a quota.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts, and run the following commands on the **workstation** host to ensure that your environment is correctly configured for this lab.

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

Open a terminal window on the **workstation** VM and run the following command to download files used in this exercise:

```
[student@workstation ~]$ lab monitor-limit setup
```

Steps

- As a cluster administrator, create a project to verify that new pods are created with no default resource requests.

To save time, all commands from this step are in the **create-project.sh** script in the **/home/student/D0280/labs/monitor-limit** folder. You can copy and paste the commands from that script.

- On the **workstation** host, log in to OpenShift as the **admin** user:

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

- Display the allocated resources for the two nodes in the OpenShift cluster:

```
[student@workstation ~]$ oc describe node node1.lab.example.com \
| grep -A 4 Allocated
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.
CPU Requests      CPU Limits      Memory Requests      Memory Limits
-----          -----          -----          -----
100m (5%)        0 (0%)        1018435456 (25%)    2500M (62%)
```

```
[student@workstation ~]$ oc describe node node2.lab.example.com \
| grep -A 4 Allocated
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.
CPU Requests      CPU Limits      Memory Requests      Memory Limits
-----          -----          -----          -----
200m (10%)       0 (0%)        1586870912 (53%)    5750M (192%)
```

The values you see might be different. Keep in mind that this exercise is about how the values change, not about the initial values. Make a note of the values from the CPU Requests column for each node.

- 1.3. Create a new project called **resources**:

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

- 1.4. Create pods from a container image:

```
[student@workstation ~]$ oc new-app --name=hello \
--docker-image=workstation.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry
```

- 1.5. Wait for the **hello** pod to be ready and running, and then check which node the pod was scheduled to run in:

```
[student@workstation ~]$ oc get pod -o wide
NAME          READY   STATUS    ..   IP           NODE
hello-1-k9h8k  1/1     Running   ..  10.130.0.40  node1.lab.example.com
```

You might need to repeat the previous command a few times until the pod is ready and running.

Make a note of the node name reported by the command. You will need this information for the next step.

- 1.6. Ensure that the allocated resources from the node have not changed. Only check only the node that is running the hello pod, using the output from the previous step:

```
[student@workstation ~]$ oc describe node node1.lab.example.com \
| grep -A 4 Allocated
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.
CPU Requests  CPU Limits  Memory Requests  Memory Limits
-----  -----  -----  -----
100m (5%)  0 (0%)  1018435456 (25%)  2500M (62%)
```



Note

You can also use the **oc describe** command against the pod and its deployment configuration to check that they do not specify any resource requests.

- 1.7. Delete all resources from the **resources** project before moving to the next step:

```
[student@workstation ~]$ oc delete all -l app=hello
```

-
2. As a cluster administrator, add a quota and a limit range to the project to provide default resource requests for pods in the project.

To save time, all commands from this step are in the **add-quota.sh** script in the **/home/student/D0280/labs/monitor-limit** folder.

- 2.1. Inspect the limit range definition. It specifies default resource requests for CPU only.

```
[student@workstation ~]$ cat ~/D0280/labs/monitor-limit/limits.yml
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "project-limits"
spec:
  limits:
    - type: "Container"
      default:
        cpu: "250m"
```

The Red Hat OpenShift Container Platform 3.5 documentation provides a sample file if you want to create this file from scratch. Consult the *Setting Quotas* chapter of the *Cluster Administration* documentation for details.

- 2.2. Add the limit range to the project:

```
[student@workstation ~]$ oc create -f ~/D0280/labs/monitor-limit/limits.yml
limitrange "project-limits" created
```

- 2.3. Ensure that a default CPU resource request is set for the project:

```
[student@workstation ~]$ oc describe limits
Name:          project-limits
Namespace:     resources
Type           Resource   Min     Max     Default Request   Default Limit
----           -----   ---     ---     -----              -----
Container     cpu        -       -       250m               250m
```

Notice that setting a default resource request also sets a default resource limit for a compute resource.

- 2.4. Inspect the provided quota definition. It sets a maximum CPU usage.

```
[student@workstation ~]$ cat ~/D0280/labs/monitor-limit/quota.yml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: project-quota
spec:
  hard:
    cpu: "900m"
```

The Red Hat OpenShift Container Platform 3.5 documentation provides a sample if you want to create this file from scratch. The sample is available in the *Setting Limit Range* section of the *Cluster Administration* documentation.

2.5. Add the quota to the project:

```
[student@workstation ~]$ oc create -f ~/D0280/labs/monitor-limit/quota.yml  
resourcequota "project-quota" created
```

2.6. Ensure that a CPU quota is set for the project:

```
[student@workstation ~]$ oc describe quota  
Name:          project-quota  
Namespace:     resources  
Resource       Used      Hard  
-----  -----  
cpu        0        900m
```

2.7. Give the **developer** user permission to create deployments in the project before moving to the next step:

```
[student@workstation ~]$ oc adm policy add-role-to-user edit developer  
role "edit" added: "developer"
```

3. As the **developer** user, create pods in the project, and verify that the pods consume resources from the project's quota.

To save time, all commands from this step are in the **add-pod.sh** script in the **/home/student/D0280/labs/monitor-limit** folder.

3.1. Log in to OpenShift as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443
```

3.2. Switch to the resources project:

```
[student@workstation ~]$ oc project resources
```

3.3. Verify that the **developer** user can inspect the resource limits and quotas set for the project, but cannot make changes to them:

```
[student@workstation ~]$ oc get limits  
NAME      AGE  
project-limits  29m  
[student@workstation ~]$ oc delete limits project-limits  
Error from server (Forbidden): User "developer" cannot delete limitranges in  
project "resources"
```

```
[student@workstation ~]$ oc get quota  
NAME      AGE  
project-quota  18m  
[student@workstation ~]$ oc delete quota project-quota  
Error from server (Forbidden): User "developer" cannot delete resourcequotas in  
project "resources"
```

If there was no error in either of the **oc delete** commands, you are still logged in as the **admin** user and you need to go back to *Step 2* to recreate the limit range and the resource quota inside the project.

3.4. Create pods from a container image:

```
[student@workstation ~]$ oc new-app --name=hello  
--docker-image=workstation.lab.example.com:5000/openshift/hello-openshift \  
--insecure-registry
```

If you get errors, ensure that you deleted the pod and its associated resources, which were created during the beginning of this exercise.

3.5. Wait for the **hello** pod to be ready and running:

```
[student@workstation ~]$ oc get pod  
NAME READY STATUS RESTARTS AGE  
hello-1-k9h8k 1/1 Running 1 1d
```

You might need to repeat the previous command a few times until the pod is ready and running.

3.6. Verify that the **hello** pod consumed part of the project's quota:

```
[student@workstation ~]$ oc describe quota  
Name: project-quota  
Namespace: resources  
Resource Used Hard  
-----  
cpu 250m 900m
```

Compare this output with the output from *Step 2.6*.

Even though the pod is idle and not serving any HTTP requests, and thus not actually using any CPU, its resource request makes it consume from the project's resource quotas.

4. **Optional:** Check that the node has fewer resources available.

To save time typing, all commands from this step are in the **check-nodes.sh** script in the **/home/student/D0280/labs/monitor-limit** folder.

4.1. Log in to OpenShift as the **admin** user:

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

4.2. Check which node the **hello** pod was scheduled to run in.

```
[student@workstation ~]$ oc get pod -o wide -n resources  
NAME READY STATUS IP NODE  
hello-1-k9h8k 1/1 Running 10.130.0.40 node1.lab.example.com
```

Make note of the node name because you will use the name in the next step.

- 4.3. Ensure that the allocated CPU requests for the node increased by the same amount as the pod resource request. Only check the node that is running the hello pod, using the output from previous step.

```
[student@workstation ~]$ oc describe node node1.lab.example.com \
| grep -A 4 Allocated
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.
CPU Requests    CPU Limits    Memory Requests    Memory Limits
-----          -----          -----          -----
350m (17%)     250m (12%)   1018435456 (25%)  2500M (62%)
```

- 4.4. Use the **oc describe** command in the pod to check that the pod specifies a CPU resource request.

```
[student@workstation ~]$ oc describe pod hello-1-k9h8k | grep -A 2 Requests
Requests:
  cpu:           250m
  State:        Running
```

- 4.5. Return to the **developer** user before moving to the next step:

```
[student@workstation ~]$ oc login -u developer -p openshift\
https://master.lab.example.com:8443
```

5. Scale the hello deployment configuration to increase the resource usage from the project's pods, and check this consumes from the project's quota.

To save time typing, all commands from this step are in the **increase-bounded.sh** script in the **/home/student/D0280/labs/monitor-limit** folder.

- 5.1. Scale the hello deployment configuration to two replicas.

```
[student@workstation ~]$ oc scale dc hello --replicas=2
```

- 5.2. Wait for the new pod to be ready and running:

```
[student@workstation ~]$ oc get pod
NAME      READY   STATUS    RESTARTS   AGE
hello-1-hwwft  1/1     Running   0          3m
hello-1-k9h8k  1/1     Running   0          8m
```

You might need to repeat the previous command a few times until the pod is ready and running.

- 5.3. Check that the new pod fits the project quota:

```
[student@workstation ~]$ oc describe quota
Name:          project-quota
Namespace:     resources
```

Resource	Used	Hard
cpu	500m	900m

- 5.4. Scale the hello deployment configuration to four replicas, which would be above the project quota:

```
[student@workstation ~]$ oc scale dc hello --replicas=4
```

- 5.5. Check that, after a few moments, a third hello pod is created, but a fourth hello pod is never created. The fourth pod would exceed the project's quota: the sum of the CPU resource requests would be 1000 millicores, but the quota limits this to 900 millicores:

```
[student@workstation ~]$ oc get pod
NAME      READY   STATUS    RESTARTS   AGE
hello-1-4bd69  1/1     Running   0          1m
hello-1-hwwft  1/1     Running   0          9m
hello-1-k9h8k  1/1     Running   0          13m
```

No error is shown. The deployment configuration waits for a long time for configuration changes that would allow it to create the fourth pod.

```
[student@workstation ~]$ oc describe dc hello | grep Replicas
Replicas:        4
Replicas:        3 current / 4 desired
```

- 5.6. Get the event list for the project. It shows that the replication controller was not able to create the fourth pod because of the quota violation:

```
[student@workstation ~]$ oc get events | grep -i error
14m      14m      1      hello DeploymentConfig Warning  FailedCreate
{hello-1-deploy} Error creating: pods "hello-1-" is forbidden: exceeded quota:
project-quota, requested: cpu=250m, used: cpu=750m, limited: cpu=900m
```

- 5.7. Scale the hello deployment configuration back to one replica before moving to the next step:

```
[student@workstation ~]$ oc scale dc hello --replicas=1
```

- 5.8. Check that, after a few moments, only one hello pod is running.

```
[student@workstation ~]$ oc get pod
NAME      READY   STATUS    RESTARTS   AGE
hello-1-k9h8k  1/1     Running   0          17m
```

6. Add a resource request to the hello pods that is not restricted by the project's quota.

To save time typing, all commands from this step are in the **increase-unbounded.sh** script in the **/home/student/D0280/labs/monitor-limit** folder.

- 6.1. Add a resource request for 256 MiB of memory to the **hello** deployment configuration.

```
[student@workstation ~]$ oc set resources dc hello --requests=memory=256Mi
deploymentconfig "hello" resource requirements updated
```

- 6.2. Wait until the new pod is ready and running, and the old pod has been deleted:

```
[student@workstation ~]$ oc get pod
NAME      READY   STATUS    RESTARTS   AGE
hello-2-d1grk  1/1     Running   0          1m
```

You might need to repeat the previous command a few times until the new pod is ready and running and the old pod disappears.

- 6.3. Check that the new pod has a memory request in addition to a CPU request:

```
[student@workstation ~]$ oc describe pod hello-2-d1grk | grep -A 3 Requests
Requests:
  cpu:           250m
  memory:        256Mi
  State:         Running
```

- 6.4. Check that the memory request is not counted against the project's quota:

```
[student@workstation ~]$ oc describe quota
Name:          project-quota
Namespace:     resources
Resource       Used      Hard
-----
cpu            250m     900m
```

From the project's quota point of view, nothing has changed.

7. Increase the memory resource request to a value that is over the capacity of any node in the cluster.

To save time typing, all commands from this step are in the **increase-toomuch.sh** script in the **/home/student/D0280/labs/monitor-limit** folder.

- 7.1. Change the memory request of the hello deployment configuration to 8 GiB. The node with more capacity has only 4 GiB of memory.

```
[student@workstation ~]$ oc set resources dc hello --requests=memory=8Gi
deploymentconfig "hello" resource requirements updated
```

- 7.2. Check that a new deployer pod is created, but it cannot create a new pod. The new pod remains in the pending status.

```
[student@workstation ~]$ oc get pod
NAME      READY   STATUS    RESTARTS   AGE
hello-2-d1grk  1/1     Running   0          13m
hello-3-deploy 1/1     Running   0          18s
hello-3-f3b2l  0/1     Pending   0          11s
```

The deployer pod keeps running for a long time, waiting for a configuration change that would allow the new pod to be ready and running. In the mean time, the hello pod from the previous deployment is not deleted, to avoid an interruption of service to the application users.

Do not wait for the following errors to be displayed. Continue to finish this exercise:

```
[student@workstation ~]$ oc get pod
NAME        READY   STATUS    RESTARTS   AGE
hello-2-d1grk  1/1     Running   0          14m
hello-3-deploy 0/1     Error     0          11m
```

```
[student@workstation ~]$ oc logs hello-3-deploy
--> Scaling up hello-3 from 0 to 1, scaling down hello-2 from 1 to 0 (keep 1
pods available, don't exceed 2 pods)
      Scaling hello-3 up to 1
error: timed out waiting for any update progress to be made
```

```
[student@workstation ~]$ oc status
In project resources on server https://master.lab.example.com:8443

svc/hello - 172.30.49.19 ports 8080, 8888
dc/hello deploys istag/hello:latest
      deployment #3 failed 12 minutes ago: config change
      deployment #2 deployed 15 minutes ago - 1 pod
      deployment #1 deployed 20 minutes ago
```

- 7.3. Get the events list for the project. It shows a warning that states that it was not possible to schedule the pod to any node because of insufficient memory:

```
[student@workstation ~]$ oc get events | grep -A 1 failed
17s      1m      8      hello-3-f3b21   Pod Warning
      FailedScheduling {default-scheduler} pod (hello-3-45r4x) failed to fit in
any node fit failure summary on nodes : Insufficient memory (2)
...
```

8. **Clean up:** Log in as the **admin** user to delete the project created for this exercise.

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://master.lab.example.com:8443
[student@workstation ~]$ oc delete project resources
```

This concludes the guided exercise.

Upgrading OpenShift Container Platform

Objective

After completing this section, students should be able to upgrade OpenShift Container Platform.

Upgrading OpenShift

When new versions of OpenShift Container Platform are released, administrators can upgrade their existing clusters to apply the latest enhancements and bug fixes. This includes upgrading from previous minor versions, such as an upgrade from 3.4 to 3.5, and applying updates to a minor version (3.5.z releases).



Note

Because of the core architectural changes between the major versions, OpenShift Enterprise 2 environments cannot be upgraded to OpenShift Container Platform 3; a fresh installation is required.

Unless noted otherwise, nodes and masters within a major version are forward and backward compatible across one minor version. However, administrators should not run mismatched versions longer than is necessary to upgrade the entire cluster.

Upgrade Methods

There are two methods for performing OpenShift Container Platform cluster upgrades. Administrators can either do in-place upgrades (automated or manual), or upgrade using a *blue-green deployment* method.

In-place Upgrades

With this process, the cluster upgrade is performed on all hosts in a single, running cluster. Masters are upgraded first, and then the nodes. Pods are migrated to other nodes in the cluster before a node upgrade begins. This helps reduce downtime of user applications.

If the cluster was installed with the quick or the advanced installation method, and the installation configuration file, `~/.config/openshift/installer.cfg.yml`, is available, administrators can perform either automated or manual in-place upgrades.

Blue-green Deployments

A blue-green deployment is an approach aimed at reducing downtime while upgrading an environment. In a blue-green deployment, identical environments are run with one active environment while the other environment is updated and thoroughly tested. When upgrading OpenShift nodes, administrators can update their nodes to prevent them from running any pods. This is achieved by running the `oc edit node node` command and setting the **unschedulable** property to **True**. For more information on pod scheduling, refer to *the section called "Controlling Pod Scheduling"*.

Performing an Automated Cluster Upgrade

If the cluster was installed with the advanced installation method and the inventory file is available, administrators can use Ansible Playbooks to automate the OpenShift cluster upgrade process. Playbooks for upgrade are available in `/usr/share/ansible/openshift-ansible/`

playbooks/byo/openshift-cluster/upgrades/. The automated upgrade performs the following tasks:

- Applies the latest configuration changes
- Upgrades the master node, the **etcd** components, and restarts services
- Applies the latest cluster policies
- If present, updates the default router
- If present, updates the default registry
- Updates default image streams and templates



Important

Ensure that you have met all prerequisites before proceeding with an upgrade. Failure to do so can result in a failed upgrade. Consult the reference linked at the end for the prerequisites.



Important

Before upgrading the cluster to OpenShift Container Platform 3.5, the cluster must already be upgraded to the latest asynchronous release of version 3.4. Asynchronous packages are identified via installed RPMs mapped to specific errata, and can be queried using the **yum** and **rpm** commands. Cluster upgrades cannot span more than one minor version at a time, so if the cluster is at a version earlier than 3.4, administrators must first upgrade incrementally, for example, 3.2 to 3.3, then 3.3 to 3.4.



Note

Before attempting the upgrade, administrators should verify the cluster's health with the **oc adm diagnostics** command. This confirms that nodes are in the **Ready** state, running the expected starting version, and that there are no diagnostic errors or warnings. For offline installations, use the **--network-pod-image='REGISTRY URL/IMAGE'** parameter to specify the image to use.

Automated Upgrades

The following procedure shows how to perform an automated upgrade:

1. If this is an upgrade from OpenShift Container Platform 3.4 to 3.5, manually disable the 3.4 channel and enable the 3.5 channel on each master and node host:

```
[root@demo ~]$ subscription-manager repos --disable="rhel-7-server-ose-3.4-rpms" \
--enable="rhel-7-server-ose-3.5-rpms" \
--enable="rhel-7-server-extras-rpms" \
--enable="rhel-7-fast-datapath-rpms"
yum clean all
```

2. Ensure that you have the latest version of the *atomic-openshift-utils* package on each Red Hat Enterprise Linux 7 system, which also updates the *openshift-ansible-** packages:

```
[root@demo ~]$ yum update atomic-openshift-utils
```

3. Install or update the *atomic-openshift-excluder* and *atomic-openshift-docker-excluder* packages. These packages configure */etc/yum.conf* to not update the *docker*, *kubernetes*, and *openshift* packages.

```
[root@demo ~]$ yum update atomic-openshift-excluder atomic-openshift-docker-excluder
```



Note

These packages add entries to the **exclude** directive in the host's */etc/yum.conf* file.

4. Log in as a cluster administrative user on the master host for the upgrade to finish:

```
[root@demo ~]$ oc login -u system:admin
```

Upgrading OpenShift Container Platform Using the Installer

If OpenShift Container Platform was installed using the quick installation method, administrators should have an installation configuration file located at *~/.config/openshift/installer.cfg.yml*. The installer requires this file to start an upgrade and supports upgrading OpenShift Container Platform one minor version at a time, for example, 3.4 to 3.5, and between asynchronous errata updates within a minor version (for example, 3.5.z). If the installation configuration file in *~/.config/openshift/installer.cfg.yml* is in an older format, the installer attempts to upgrade the file to the new supported format. Administrators can also create that file manually.

The following procedure shows how to upgrade an OpenShift cluster installed with the quick installation method:

1. Ensure that you have completed all of the steps in the *Automated Upgrades* section above. The steps ensure that the latest upgrade playbooks are used.
2. Run the installer with the **upgrade** subcommand:

```
[root@demo ~]$ atomic-openshift-installer upgrade
```

3. Follow the on-screen instructions to upgrade to the latest release.
4. After all masters and nodes have been upgraded, administrators will be prompted to reboot all hosts. After rebooting, if there are no additional features enabled, verify the upgrade. Otherwise, the next step depends on what additional features were enabled; for example, if metrics were enabled, the deployment must also be upgraded.

Verifying the Upgrade

After the upgrade finishes, there are steps that administrators should perform to ensure the success of the upgrade. The following procedure describes some of the steps to ensure that the upgrade successfully completed.

1. Ensure that all nodes are marked as **Ready**:

```
[root@demo ~]$ oc get nodes
NAME           STATUS        AGE
master.example.com   Ready,SchedulingDisabled 165d
node1.example.com    Ready         165d
node2.example.com    Ready         165d
```

2. Verify the versions of the **docker-registry** and **router** images:

```
[root@demo ~]$ oc get -n default dc/docker-registry -o json | grep '\"image\"'
'image": "openshift3/ose-docker-registry:tag"

[root@demo ~]$ oc get -n default dc/router -o json | grep '\"image\"'
'image": "openshift3/ose-haproxy-router:tag",
```

3. Use the diagnostics tool on the master node to look for common issues:

```
[root@demo ~]$ oc adm diagnostics
...
[Note] Summary of diagnostics execution:
[Note] Completed with no errors or warnings seen.
```

Performing Manual In-place Cluster Upgrades

As an alternative to performing an automated upgrade, administrators can manually upgrade their OpenShift cluster. To manually upgrade without disruption, it is important to upgrade each component separately. Specific releases may require additional steps to be performed at key points before or during the standard upgrade process.

Preparing for a Manual Upgrade

The following procedure describes how to perform a manual upgrade.

1. If this is an upgrade from OpenShift Container Platform 3.4 to 3.5, disable the 3.4 channel and enable the 3.5 channel on each host:

```
[root@demo ~]$ subscription-manager repos --disable="rhel-7-server-ose-3.4-rpms" \
--enable="rhel-7-server-ose-3.5-rpms" \
--enable="rhel-7-server-extras-rpms" \
--enable="rhel-7-fast-datapath-rpms"
[root@demo ~]$ yum clean all
```

2. Install or update to the latest available version of the *atomic-openshift-utils* package on each RHEL 7 system, which provides files that are used in later sections:

```
[root@demo ~]$ yum update atomic-openshift-utils
```

3. Install or update to the following latest available **-excluder* packages on each RHEL 7 system. These packages configure **/etc/yum.conf** to not update the *docker*, *kubernetes*, and *openshift* packages.

```
[root@demo ~]$ yum install atomic-openshift-excluder atomic-openshift-docker-excluder
```

4. Create an **etcd** backup on each master node. The *etcd* package is required even if using the embedded *etcd* packages, because this *etcd* package gives access to the **etcdctl** command.

```
[root@demo ~]$ yum install etcd
```

5. Create a backup:

```
[root@demo ~]$ ETCD_DATA_DIR=/var/lib/origin/openshift.local.etcd  
[root@demo ~]$ etcdctl backup \  
--data-dir $ETCD_DATA_DIR \  
--backup-dir $ETCD_DATA_DIR.bak.date
```

6. Ensure that the latest kernel on each RHEL 7 system is installed:

```
[root@demo ~]$ yum update kernel
```

Upgrading Master Nodes

Before upgrading any stand-alone nodes, upgrade the master components. Master nodes provide the control plane for the cluster. The following procedure describes the required steps for upgrading master nodes.

1. Remove the *atomic-openshift* packages from the list of Yum excludes on the host:

```
[root@demo ~]$ atomic-openshift-excluder unexclude
```

2. Upgrade *etcd* on all master hosts and any external **etcd** hosts.

- For RHEL 7 systems using the RPM-based method, update the *etcd* package and restart the service:

```
[root@demo ~]$ yum update etcd && \  
systemctl restart etcd
```

- For RHEL Atomic Host 7 systems and RHEL 7 systems using the containerized method, pull the image and restart the **etcd_container** service.

```
[root@demo ~]$ docker pull registry.access.redhat.com/rhel7/etcd && \  
systemctl restart etcd_container
```

3. On each master host, upgrade the *atomic-openshift* packages and related images.

- For masters using the RPM-based method on a RHEL 7 system, upgrade all installed **atomic-openshift** packages and the **openvswitch** package:

```
[root@demo ~]$ yum upgrade atomic-openshift\* openvswitch
```

- For masters using the containerized method on a RHEL 7 or RHEL Atomic Host 7 system, set the **IMAGE_VERSION** parameter to the version that matches the upgrade in the following files:
 - /etc/sysconfig/atomic-openshift-master** (single master clusters only)
 - /etc/sysconfig/atomic-openshift-master-controllers** (multi-master clusters only)
 - /etc/sysconfig/atomic-openshift-master-api** (multi-master clusters only)
 - /etc/sysconfig/atomic-openshift-node**
 - /etc/sysconfig/atomic-openshift-openvswitch**

- Restart the master service on each master and review the log files to ensure they started successfully.
- For single master clusters, run the following commands:

```
[root@demo ~]$ systemctl restart atomic-openshift-master && \
journalctl -r -u atomic-openshift-master
```

- For multi-master clusters, run the following commands:

```
[root@demo ~]$ systemctl restart atomic-openshift-master-controllers && \
systemctl restart atomic-openshift-master-api
[root@demo ~]$ journalctl -r -u atomic-openshift-master-controllers && \
journalctl -r -u atomic-openshift-master-api
```

- Masters have node components running on them so that they can be configured as part of the OpenShift SDN. This means that the **atomic-openshift-node** and **openvswitch** services must be restarted:

```
[root@demo ~]$ systemctl restart openvswitch && \
systemctl restart atomic-openshift-node
[root@demo ~]$ journalctl -r -u openvswitch && \
journalctl -r -u atomic-openshift-node
```

- On each master, add the *atomic-openshift* package back to the list of Yum excludes on the host:

```
[root@demo ~]$ atomic-openshift-excluder exclude
```



Important

If you are performing a cluster upgrade that requires updating Docker to version 1.12, extra steps are required. Refer to the links provided at the end for the detailed procedure.

Updating Policy Definitions

After a cluster upgrade, the default cluster policy must be updated if the roles are outdated. The following procedure describes the required steps for updating policy definitions.



Important

If the environment uses customized cluster roles, and administrators want to ensure a role reconciliation, the roles can be annotated with **openshift.io/reconcile-protect** set to **true**. Administrators are then responsible for manually updating those roles with any new or required permissions during upgrades.

Run the following command to determine if an update is recommended for the environment by running the following command. The output lists the roles that are out of date and their new proposed values.

```
[root@demo ~]$ oc adm policy reconcile-cluster-roles
apiVersion: v1
items:
- apiVersion: v1
  kind: ClusterRole
  metadata:
    creationTimestamp: null
    name: admin
  rules:
  - attributeRestrictions: null
    resources:
    - builds/custom
...
...
```

Administrators can either modify the output to re-apply any local policy changes they have made, or automatically apply the new policy using the following procedure:

1. Reconcile the cluster roles:

```
[root@demo ~]$ oc adm policy reconcile-cluster-roles \
--additive-only=true \
--confirm
```

2. Reconcile the cluster role bindings:

```
[root@demo ~]$ oc adm policy reconcile-cluster-role-bindings \
--exclude-groups=system:authenticated \
--exclude-groups=system:authenticated:oauth \
--exclude-groups=system:unauthenticated \
--exclude-users=system:anonymous \
```

```
--additive-only=true \
--confirm
[root@demo ~]$ oc adm policy reconcile-cluster-role-bindings \
system:build-strategy-jenkinspipeline \
--confirm -o name
```

3. Reconcile security context constraints:

```
[root@demo ~]$ oc adm policy reconcile-sccs \
--additive-only=true \
--confirm
```

Upgrading Nodes

After upgrading masters, administrators can upgrade the nodes in their OpenShift cluster. When restarting the **atomic-openshift-node** service, there will be a brief disruption of outbound network connectivity from running pods to services while the service proxy is restarted. The disruption should be brief, but scales on the number of services in the entire cluster. For each node that is not a master, administrators must disable scheduling and evacuate the running pods to other nodes, and then upgrade the packages before restarting the necessary services. The following procedure describes the required steps for upgrading nodes.



Note

Repeat this procedure for each node until all nodes have been upgraded.

1. Remove the *atomic-openshift* package from the list of yum excludes on the host:

```
[root@demo ~]$ atomic-openshift-excluder unexclude
```

2. As a user with **cluster-admin** privileges, disable scheduling for the node:

```
[root@demo ~]$ oc adm manage-node node --schedulable=false
```

3. Evacuate pods on the node to other nodes:



Important

The **--force** option deletes any pods that are not managed by a replication controller.

```
[root@demo ~]$ oc adm drain node --force --delete-local-data --ignore-daemonsets
```

4. Upgrade the node component packages or related images.

- For nodes using the RPM-based method on a RHEL 7 system, upgrade all installed *atomic-openshift* packages and the *openvswitch* package:

```
[root@demo ~]$ yum upgrade atomic-openshift\* openvswitch
```

- For nodes using the containerized method on a RHEL 7 or RHEL Atomic Host 7 system, set the **IMAGE_VERSION** parameter to the version that matches the upgrade in the **/etc/sysconfig/atomic-openshift-node** and **/etc/sysconfig/openvswitch** configuration files.
5. Restart the **atomic-openshift-node** and **openvswitch** services and review the logs to ensure they restart successfully:

```
[root@demo ~]$ systemctl restart openvswitch && systemctl restart atomic-openshift-node
[root@demo ~]$ journalctl -r -u atomic-openshift-node && journalctl -r -u openvswitch
```

6. Re-enable scheduling for the node:

```
[root@demo ~]$ oc adm manage-node node --schedulable
```

7. Add the *atomic-openshift* packages to the list of Yum excludes on the host:

```
[root@demo ~]$ atomic-openshift-excluder exclude
```



Important

If you are performing a cluster upgrade that requires updating Docker to version 1.12, extra steps are required. Refer to the link provided at the end for the detailed procedure.

After all nodes have been upgraded, as a user with **cluster-admin** privileges, verify that all nodes are showing as **Ready**:

```
[root@demo ~]$ oc get nodes
NAME           STATUS            AGE
master.example.com  Ready, SchedulingDisabled  165d
node1.example.com   Ready           165d
node2.example.com   Ready           165d
```

Upgrading the Registry

The registry must also be upgraded for changes to take effect in the registry image. If a **PersistentVolumeClaim** or a host mount point is used, administrators can restart the registry without losing the contents of their registry. The registry upgrade procedure is described below.

- Edit the registry's deployment configuration:

```
[root@demo ~]$ oc edit dc/docker-registry
```

2. Adjust the **tag** to match the version of the upgrade, for example, **v3.5.5.15** for the latest version.

```
...
spec:
  template:
    spec:
      containers:
        - env:
          ...
          image: registry.access.redhat.com/openshift3/ose-docker- registry:tag
          imagePullPolicy: IfNotPresent
        ...
...
```



Important

Images that are being pushed or pulled from the internal registry at the time of upgrade will fail and should be restarted automatically. This does not disrupt pods that are already running.

Updating the Default Image Streams and Templates

By default, the quick and advanced installation methods automatically create default image streams, **InstantApp** templates, and database service templates in the **openshift** project, which is a default project to which all users have view access. These objects are created during installation from the JSON files located in the **/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/** directory. The following procedure describes the required steps for upgrading the default image streams and templates.



Note

RHEL Atomic Host 7 cannot use **yum** to update packages; you need to run the following steps on a RHEL 7 system.

1. Update the packages that provide the example JSON files. On a subscribed Red Hat Enterprise Linux 7 system, install or update to the latest version of the *atomic-openshift-utils* package. By updating this package, the *openshift-ansible-packages* is also updated.

```
[root@demo ~]$ yum update atomic-openshift-utils
```



Note

The *openshift-ansible-roles* package provides the latest example JSON files.

2. Get the latest templates from the *openshift-ansible-roles* package:

```
[root@demo ~]$ rpm -ql openshift-ansible-roles | grep examples | grep v1.5
```

In this example, `/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v1.5/image-streams/image-streams-rhel7.json` is the latest file that is required in the latest `openshift-ansible-roles` package. The `/usr/share/openshift/examples/image-streams/image-streams-rhel7.json` file is not owned by a package, but is updated by Ansible. If administrators are upgrading outside of Ansible, they need to get the latest JSON files on the system where they are running `oc`. Administrators can run this command from any server that has access to the master.

3. Install the `atomic-openshift-utils` package and its dependencies to get the new content into `/usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v1.5/`:

```
[root@demo ~]$ cd /usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v1.5/image-streams/image-streams-rhel7.json
[root@demo image-streams]$ oc create -n openshift -f image-streams-rhel7.json
[root@demo image-streams]$ oc create -n openshift -f dotnet_imagestreams.json
[root@demo image-streams]$ oc replace -n openshift -f image-streams-rhel7.json
[root@demo image-streams]$ oc replace -n openshift -f dotnet_imagestreams.json
```

4. Update the templates. Errors are generated for items that already exist, which is expected behavior:

```
[root@demo ~]$ cd /usr/share/ansible/openshift-ansible/roles/openshift_examples/files/examples/v1.5
[root@demo ~]$ oc create -n openshift -f quickstart-templates/
[root@demo v1.5]$ oc create -n openshift -f db-templates/
[root@demo v1.5]$ oc create -n openshift -f infrastructure-templates/
[root@demo v1.5]$ oc create -n openshift -f xpaas-templates/
[root@demo v1.5]$ oc create -n openshift -f xpaas-streams/
[root@demo v1.5]$ oc replace -n openshift -f quickstart-templates
[root@demo v1.5]$ oc replace -n openshift -f db-templates/
[root@demo v1.5]$ oc replace -n openshift -f infrastructure-templates/
[root@demo v1.5]$ oc replace -n openshift -f xpaas-templates/
[root@demo v1.5]$ oc replace -n openshift -f xpaas-streams/
```

Importing the Latest Images

After updating the default image streams, administrators may want to ensure that the images within the streams are updated. The following procedure describes the required steps for importing the latest images.

1. Run the `oc import-image` command for each image stream in the default `openshift` project:

```
[root@demo ~]$ oc import-image -n openshift imagestream
```

2. Update each image stream one at a time. For example, to update the `nodejs` image:

```
[root@demo ~]$ oc import-image -n openshift nodejs
The import completed successfully.
Name: nodejs

Created: 10 seconds ago
```

```
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2016-07- 05T19:20:30Z
Docker Pull Spec: 172.30.204.22:5000/openshift/nodejs
...
```



Note

Administrators can run a **for** loop to import all images.



Important

To update S2I-based applications, administrators must manually trigger a new build of those applications after importing the new images, using the **oc start-build app** command. The command does not apply if an image-change trigger is defined in the deployment template.

Testing the Upgrade

After an upgrade, there are some steps that administrators should run in order to ensure that the environment is properly running with the newer version. The following procedure shows how to test an upgrade.

1. Ensure that all nodes are marked as **Ready**:

```
[root@demo ~]$ oc get nodes
NAME           STATUS      AGE
master.example.com  Ready, SchedulingDisabled  165d
node1.example.com   Ready      165d
node2.example.com   Ready      165d
```

2. Verify that the expected versions of the **docker-registry** and **router** images is running. Replace **tag** with **v3.5.5.15** for the latest version.

```
[root@demo ~]$ oc get -n default dc/docker-registry -o json | grep \"image\"
"image": "openshift3/ose-docker-registry:tag",
[root@demo ~]$ oc get -n default dc/router -o json | grep \"image\"
"image": "openshift3/ose-haproxy-router:tag",
```

3. Use the diagnostics tool, **oc adm diagnostics**, on the master to look for common issues:

```
[root@demo ~]$ oc adm diagnostics
...
[Note] Summary of diagnostics execution:
[Note] Completed with no errors or warnings seen.
```



References

Further information is available in the *Upgrading a Cluster* chapter of the *OpenShift Container Platform Installation Guide* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Further information is available in the Installation and Configuration chapter of the *OpenShift Container Platform* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Quiz: Upgrading OpenShift

The steps to manually upgrade the master node in an OpenShift cluster are shown below. Indicate the order in which the steps should be run.

- a. Install or update to the latest available version of the *atomic-openshift-utils* package.
- b. Upgrade the *etcd* package.
- c. Update the *kernel* package.
- d. Restart the **atomic-openshift-master** service and review the logs.
- e. Back up the **etcd** data.
- f. Add the *atomic-openshift* package to the list of Yum excludes.
- g. Disable the current OpenShift yum repository, and enable the yum repository for the OpenShift version that you are upgrading to.
- h. Upgrade the *atomic-openshift* packages and related images.
- i. Remove the *atomic-openshift* packages from the list of Yum excludes.
- j. Restart the **atomic-openshift-node** and **openvswitch** services.

Solution

The steps to manually upgrade the master node in an OpenShift cluster are shown below. Indicate the order in which the steps should be run.

- 2 a. Install or update to the latest available version of the *atomic-openshift-utils* package.
- 6 b. Upgrade the *etcd* package.
- 4 c. Update the *kernel* package.
- 8 d. Restart the **atomic-openshift-master** service and review the logs.
- 3 e. Back up the **etcd** data.
- 10 f. Add the *atomic-openshift* package to the list of Yum excludes.
- 1 g. Disable the current OpenShift yum repository, and enable the yum repository for the OpenShift version that you are upgrading to.
- 7 h. Upgrade the *atomic-openshift* packages and related images.
- 5 i. Remove the *atomic-openshift* packages from the list of Yum excludes.
- 9 j. Restart the **atomic-openshift-node** and **openvswitch** services.

Monitoring Applications with Probes

Objective

After completing this section, students should be able to configure probes to monitor the health of applications deployed on OpenShift.

Introduction to OpenShift Probes

OpenShift applications can become unhealthy due to issues such as temporary connectivity loss, configuration errors, or application errors. Developers can use *probes* to monitor their applications. A probe is a Kubernetes action that periodically performs diagnostics on a running container. Probes can be configured using either the **oc** command-line client or the OpenShift web console. There are currently two types of probes that administrators can use:

Liveness Probe

A liveness probe determines whether or not an application running in a container is in a **healthy** state. If the liveness probe returns detects an unhealthy state, OpenShift kills the pod and tries to re-deploy it again. Developers can set a liveness probe by configuring the **template.spec.containers.livenessprobe** stanza of a pod configuration.

Readiness Probe

A readiness probe determines whether or not a container is ready to serve requests. If the readiness probe returns a failed state, OpenShift removes the container's IP address from the endpoints of all services. Developers can use readiness probes to signal to OpenShift that even though a container is running, it should not receive any traffic from a proxy. Developers can set a readiness probe by configuring the **template.spec.containers.readinessprobe** stanza of a pod configuration.

OpenShift provides a number of timeout options for probes. There are five options that control these two probes:

- **initialDelaySeconds**: Mandatory. Determines how long to wait after the container starts before beginning the probe.
- **timeoutSeconds**: Mandatory. Determines how long to wait for the probe to finish. If this time is exceeded, OpenShift Container Platform considers that the probe failed.
- **periodSeconds**: Optional. Specifies the frequency of the checks.
- **successThreshold**: Optional. Specifies the minimum consecutive successes for the probe to be considered successful after it has failed.
- **failureThreshold**: Optional. Specifies the minimum consecutive failures for the probe to be considered failed after it has succeeded. This field is optional.

Methods of Checking Application Health

Readiness and liveness probes can check the health of applications in three ways:

HTTP Checks

When using HTTP checks, OpenShift uses a webhook to determine the health of a container. The check is deemed successful if the HTTP response code is between 200 and 399. The following example demonstrates how to implement a readiness probe with the HTTP check method.

```
...  
readinessProbe:  
  httpGet:  
    path: /health①  
    port: 8080  
  initialDelaySeconds: 15②  
  timeoutSeconds: 1③  
...
```

- ① The URL to query.
- ② How long to wait after the container starts before checking its health.
- ③ How long to wait for the probe to finish.



Note

An HTTP check is ideal for applications that return HTTP status codes.

Container Execution Checks

When using container execution checks, the **kubelet** agent executes a command inside the container. Exiting the check with a status of **0** is considered a success. The following example demonstrates how to implement a container execution check.

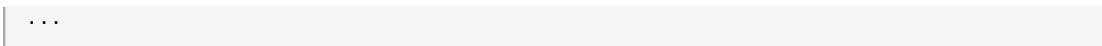
```
...  
livenessProbe:  
  exec:  
    command:  
      - cat  
      - /tmp/health①  
  initialDelaySeconds: 15  
  timeoutSeconds: 1  
...
```

- ① The command to run.

TCP Socket Checks

When using TCP socket checks, the **kubelet** agent attempts to open a socket to the container. The container is considered healthy if the check can establish a connection. The following example demonstrates how to implement a liveness probe using the TCP socket check method.

```
...  
livenessProbe:  
  tcpSocket:  
    port: 8080①  
  initialDelaySeconds: 15  
  timeoutSeconds: 1
```



- ① The TCP port to check.

Using the Web Console to Manage Probes

Developers can use the OpenShift web console to manage both readiness and liveness probes. For each deployment, probe management is available in the **Actions** drop-down list.

Deployment	Status	Created	Trigger
#5 (latest)	Active, 1 replica	3 hours ago	Config change
#4	✓ Complete	3 hours ago	Config change
#3	✓ Complete	3 hours ago	Config change
#2	✓ Complete	3 hours ago	Config change
#1	✓ Complete	4 hours ago	Image change

Figure 9.1: Managing probes using the web console

For each probe type, developers can select the type, such as **HTTP GET**, **TCP Socket**, or **Container Command**, and specify the parameters for each type. The web console also provides the option to delete the probe. *Figure 9.2: Managing readiness probes using the web console* shows the management of readiness and liveness probes.

Health Checks: probes

Container health is periodically checked using readiness and liveness probes. [Learn More](#)

Readiness Probe

A readiness probe checks if the container is ready to handle requests. A failed readiness probe means that a container should not receive any traffic from a proxy, even if it's running.

*Type:

Use HTTPS

Path:

*Port:

Initial Delay: seconds

How long to wait after the container starts before checking its health.

Timeout: seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.

Figure 9.2: Managing readiness probes using the web console

The screenshot shows the 'Liveness Probe' configuration page. It includes fields for Type (HTTP GET), Path (/health), Port (3000), Initial Delay (3 seconds), and Timeout (3 seconds). A 'Remove Liveness Probe' button is also present.

Figure 9.3: Managing liveness probes using the web console



Note

periodSeconds, **successThreshold**, and **failureThreshold** cannot be set via the web console.

The web console can also be used to edit the YAML file that defines the deployment configuration. Upon creation of a probe, a new entry is added to the configuration file for the deployment configuration. Using the live editor, administrators can review or edit a probe. The live editor allows developers to edit the **periodSeconds**, **successThreshold**, and **failureThreshold** options. The following example shows the live editor for a deployment configuration.

```

63    livenessProbe:
64      httpGet:
65        path: /health
66        port: 3000
67        scheme: HTTP
68      initialDelaySeconds: 3
69      timeoutSeconds: 3
70      periodSeconds: 10
71      successThreshold: 1
72      failureThreshold: 3
73    readinessProbe:
74      httpGet:
75        path: /ready
76        port: 3000
77        scheme: HTTP
78      initialDelaySeconds: 3
79      timeoutSeconds: 2
80      periodSeconds: 10
81      successThreshold: 1
82      failureThreshold: 3

```

Figure 9.4: Managing liveness probes using the web console



References

Further information is available in the *Application Health* chapter of the *OpenShift Container Platform Developer Guide* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Further information is available in the *Configure Liveness and Readiness Probes* page of the *Kubernetes* website at

 | <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/>

Guided Exercise: Monitoring Applications with Probes

In this exercise, you will configure readiness and liveness probes for an application deployed on OpenShift.

Resources	
Application URL:	https://master.lab.example.com:8443 http://probe.cloudapps.lab.example.com

Outcomes

You should be able to:

- Create a new project.
- Create a new application.
- Create *readiness* and *liveness* probes for an application.
- View the event log for the project.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform* should be completed and you should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts and run the following commands on the **workstation** host to ensure your environment is correctly configured for this exercise.

```
[student@workstation ~]$ lab install-post setup
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab probes setup
```

Steps

1. On **workstation**, log in to the OpenShift cluster as the **developer** user, and create the **probes** project.
 - 1.1. On **workstation**, open a new terminal and use the **oc login** command to log in to the OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p openshift \
  https://master.lab.example.com:8443
Login successful.
```

```
You don't have any projects. You can try to create a new project, by running
  oc new-project <projectname>
```

-
- 1.2. Create a new project called **probes**.

```
[student@workstation ~]$ oc new-project probes
Now using project "probes" on server "https://master.lab.example.com:8443".
...
```

2. Create a new application using the **node-hello** image that is available in the classroom private registry. Name the application **probes** and review the deployment.

- 2.1. Use the **oc new-app** command to create a new application. Use the **node-hello** image.

```
[student@workstation ~]$ oc new-app --name=probes \
--docker-image=workstation.lab.example.com:5000/node-hello \
--insecure-registry

--> Found Docker image f73799b (4 days old) from
workstation.lab.example.com:5000 for "workstation.lab.example.com:5000/node-
hello"

Node.js 6
-----
Platform for building and running Node.js 6 applications

Tags: builder, nodejs, nodejs6

* An image stream will be created as "probes:latest" that will track this
image
* This image will be deployed in deployment config "probes"
* Ports 3000/tcp, 8080/tcp will be load balanced by service "probes"
* Other containers can access this service through the hostname "probes"

--> Creating resources ...
imagestream "probes" created
deploymentconfig "probes" created
service "probes" created
--> Success
Run 'oc status' to view your app.
```

- 2.2. Run the **oc status** command to review the deployment of the application.

```
[student@workstation ~]$ oc status
In project probes on server https://master.lab.example.com:8443

svc/probes - 172.30.237.30 ports 3000, 8080
dc/probes deploys istag/probes:latest
  deployment #1 deployed about a minute ago - 1 pod
...
```

- 2.3. OpenShift has created one pod. Run the **oc get pods** command until there is one pod in the *Running* state.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
probes-1-6w9rp 1/1     Running   0          4m
```

3. Expose a route for the service that allows external clients to access the application.

```
[student@workstation ~]$ oc expose svc probes \
--hostname=probe.cloudapps.lab.example.com
route "probes" exposed
```

4. Use the **curl** command to access the application.

```
[student@workstation ~]$ curl http://probe.cloudapps.lab.example.com
Hi! I am running on host -> probes-1-6w9rp
```

5. The application exposes two HTTP **GET** URLs at **/health** and **/ready**. The **/health** URL is used by the liveness probe, and the **/ready** URL is used by the readiness probe. Use the **curl** command to test these URLs.

```
[student@workstation ~]$ curl http://probe.cloudapps.lab.example.com/health
OK
[student@workstation ~]$ curl http://probe.cloudapps.lab.example.com/ready
READY
```

6. Connect to the OpenShift web console as the **developer** user and create a readiness probe.

- 6.1. From **workstation**, open Firefox and navigate to the OpenShift web console available at **https://master.lab.example.com:8443**. Use **developer** as the user name, and **openshift** as the password. Click **probes** to access the project.

- 6.2. In the **Overview** section, notice the following banner:

probes has containers without health checks, which ensure your application is running correctly. Add Health Checks

To add a probe, click **Add Health Checks**.

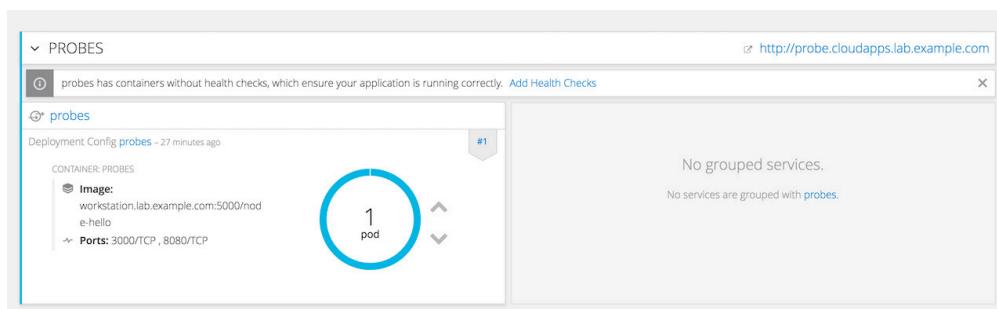


Figure 9.5: Adding a health check

- 6.3. On the next page, click **Add Readiness Probe** to create a new readiness probe. Use the information provided in the following table:

Readiness Probe

Field	Value
Type	HTTP GET

Field	Value
Path	/ready
Port	3000
Initial Delay	3
Timeout	2

Health Checks: probes

Container health is periodically checked using readiness and liveness probes.

[Learn More](#)

Readiness Probe

A readiness probe checks if the container is ready to handle requests. A failed readiness probe means that a container should not receive any traffic from a proxy, even if it's running.

* Type:

Use HTTPS

Path:

* Port:

Initial Delay: seconds

How long to wait after the container starts before checking its health.

Timeout: seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.

Figure 9.6: Adding a readiness probe

Click **Save** to create the readiness probe.

- 6.4. On the next page, click the latest deployment from the table to access the latest deployment configuration.
7. Create a liveness probe by navigating to **Actions > Edit Health Checks**.

Deployments > probes

probes created a day ago

[app](#) [probes](#)

History Configuration Environment Events

⌚ Deployment #5 is active. [View Log](#)

[Filter by label](#) [Add](#)

Deployment	Status	Created	Trigger
------------	--------	---------	---------

Actions

- Deploy
- Edit
- Pause Rollouts
- Add Storage
- Add Autoscaler
- Edit Resource Limits
- Edit Health Checks**
- Edit YAML
- Delete

Figure 9.7: Editing health checks

Scroll down and click **Add Liveness Probe** to create a liveness probe. Use the values provided in the following table. Notice the typo, **healtz**, instead of **health**. This error will cause OpenShift to consider the pods as unhealthy, which will trigger the redeployment of the pod.

Liveness Probe

Field	Value
Type	HTTP GET
Path	/healtz

Field	Value
Port	3000
Initial Delay	3
Timeout	3

Click **Save** to create the liveness probe.

- Review the implementation of the probes by clicking **Monitoring** on the sidebar. Watch as the **Events** panel updates in real time. Notice the entries marked as **Unhealthy**, which indicates that the liveness probe failed to access the `/healtz` resource.

Figure 9.8: Viewing monitoring events

- Click the **View Details** link on the upper-right corner to access the monitoring events. Review the events.

Figure 9.9: Viewing events

- Use the command-line interface to review the events in the project. On **workstation**, use the terminal to run the `oc get events` command. Locate the event type **Warning** with a reason of **Unhealthy**.

```
[student@workstation ~]$ oc get events
...
Normal   Created          {kubelet node2.lab.example.com}  Created container
with docker id a9b0459e0b5e; Security:[seccomp=unconfined]
Normal   Started          {kubelet node2.lab.example.com}  Started container
with docker id a9b0459e0b5e
Warning  Unhealthy        {kubelet node2.lab.example.com}  Liveness probe
failed: HTTP probe failed with statuscode: 404
...
```

- Edit the liveness probe.

-
- 11.1. Update the liveness probe to query the correct URL. Navigate to **Applications > Deployments**. Click the **probes** entry to access the deployment configuration.
 - 11.2. Click **Action** to edit the liveness probe, and then select the **Edit Health Checks** entry. Scroll down to the **Liveness Probe** section. In the **Path** directory, replace **/healtz** with **/health**.
Click **Save** to save your changes.
 12. From the terminal, rerun the **oc get events** command. Ensure that you do not see any more entries about the pods being unhealthy.

```
[student@workstation ~]$ oc get events
...
13m      13m      1      probes          DeploymentConfig      Normal
DeploymentCreated  {deploymentconfig-controller }  Created new replication
controller "probes-3" for version 3
12m      12m      1      probes          DeploymentConfig      Normal
DeploymentCreated  {deploymentconfig-controller }  Created new replication
controller "probes-4" for version 4
8m       8m      1      probes          DeploymentConfig      Normal
DeploymentCreated  {deploymentconfig-controller }  Created new replication
controller "probes-5" for version 5
```

13. Clean up.
- 13.1. On the **workstation** host, delete the **probes** project, which also deletes all the pods created during this lab.

```
[student@workstation ~]$ oc delete project probes
project "probes" deleted
```

This concludes the guided exercise.

Monitoring Resources with the Web Console

Objective

After completing this section, students should be able to monitor OpenShift Container Platform resources using data obtained from the web console.

Introduction to the Web Console

The OpenShift web console is a user interface accessible from a web browser. It is a convenient way to manage and monitor applications. Although the command-line interface can be used for managing the life cycle of applications, the web console presents benefits, such as the state of a deployment, pod, service, and other resources, as well as providing information about system-wide events.

Cluster administrators can use the web console to monitor critical properties in their infrastructure, which include:

- The readiness or the status of a pod.
- The availability of a volume.
- The availability of an application via the use of probes.

After logging in, the web console provides users with an overview of their project:

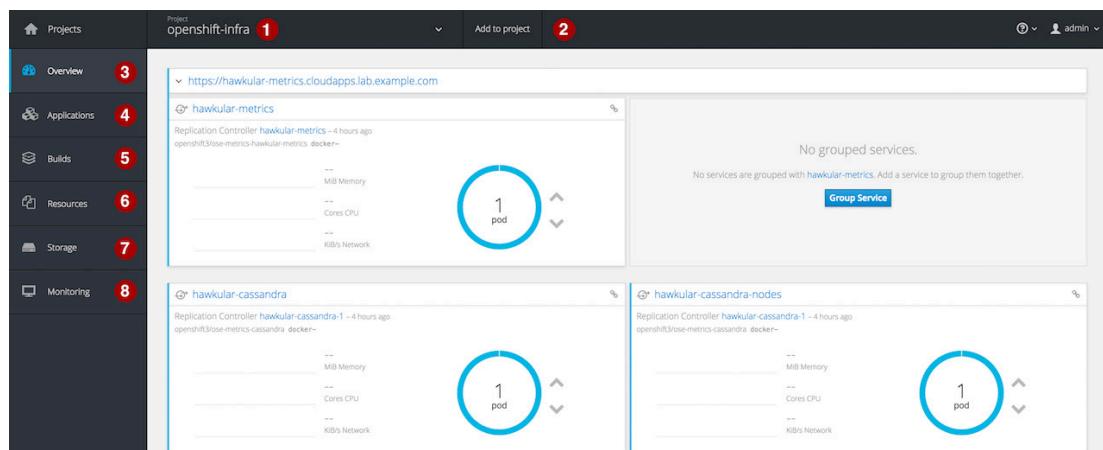


Figure 9.10: The web console overview

1. The project selector allows users to switch between projects they have access to.
2. The **Add to project** link allows users to add new resources and applications to the project.
3. The **Overview** tab provides a high-level view of the current project. It displays the name of the services and their associated pods running in the project.
4. The **Applications** tab provides access to deployments, pods, services, and routes. It also gives access to *Stateful Sets*, a technology preview that provides a unique identity to pods for managing the ordering of deployments.
5. The **Builds** tab provides access to builds and image streams.

6. The **Resources** tab provides access to quota management and various resources such as roles and endpoints.
7. The **Storage** tab provides access to persistent volumes and storage requests.
8. The **Monitoring** tab provides access to build, deployment, and pod logs. It also provides access to event notifications for the various objects in the project.

The web console provides a consolidated view of an application by linking the various elements that it contains. For example, the following figure shows a running pod, the template it belongs to, the volumes it uses, and the monitoring probes.

The screenshot shows the Hawkular Metrics web interface for a pod named "hawkular-metrics-wh0z1".

- 1.** The pod name "hawkular-metrics-wh0z1" is displayed at the top left, along with a timestamp "created 5 hours ago".
- 2.** Application labels "metrics-infra" and "hawkular-metrics" are shown below the pod name.
- 3.** The "Status" section shows the pod is "Running" with IP 10.130.0.13 and Node node1.lab.example.com (172.25.250.11). It also shows the restart policy is "Always".
- 4.** The "Template" section shows the container name is HAWKULAR-METRICS. It lists the image (openshift3/ose-metrics-hawkular-metrics), command ("/opt/hawkular/scripts/hawkular-metrics-wrapping.sh -b 0.0.0.0 -Dhawkular.metrics..."), ports (8080/TCP, 8443/TCP, 8888/TCP), and mounts (hawkular-metrics-secrets, hawkular-metrics-client-secrets, hawkular-token-04312, /var/run/secrets/kubernetes.io/serviceaccount). Monitoring probes for Readiness and Liveness are also listed.
- 5.** The "Volumes" section shows a single volume "hawkular-metrics-secrets" of type "secret".
- 6.** The "Actions" dropdown menu is located at the top right of the pod card.

Figure 9.11: Pods overview

1. The name of the pod.
2. Application labels.
3. Overall status of the pod.
4. Template specifications.
5. Monitoring probes declared for the application.
6. Available actions for the pod. For example, developers can attach storage to their pod.

Managing Metrics with Hawkular

Hawkular is a set of open source projects developed for monitoring environments. It is composed of various components, such as Hawkular services, Hawkular *Application Performance Management (APM)*, and Hawkular metrics. Hawkular can collect application metrics within an OpenShift cluster via the Hawkular OpenShift Agent. By deploying Hawkular in their OpenShift cluster, administrators have access to various metrics, such as the memory used by the pods, the number of CPUs, and the network usage.

After you have deployed Hawkular agents, graphs are available in the web console. The following two images show the various charts that administrators can access after the deployment of Hawkular agents:

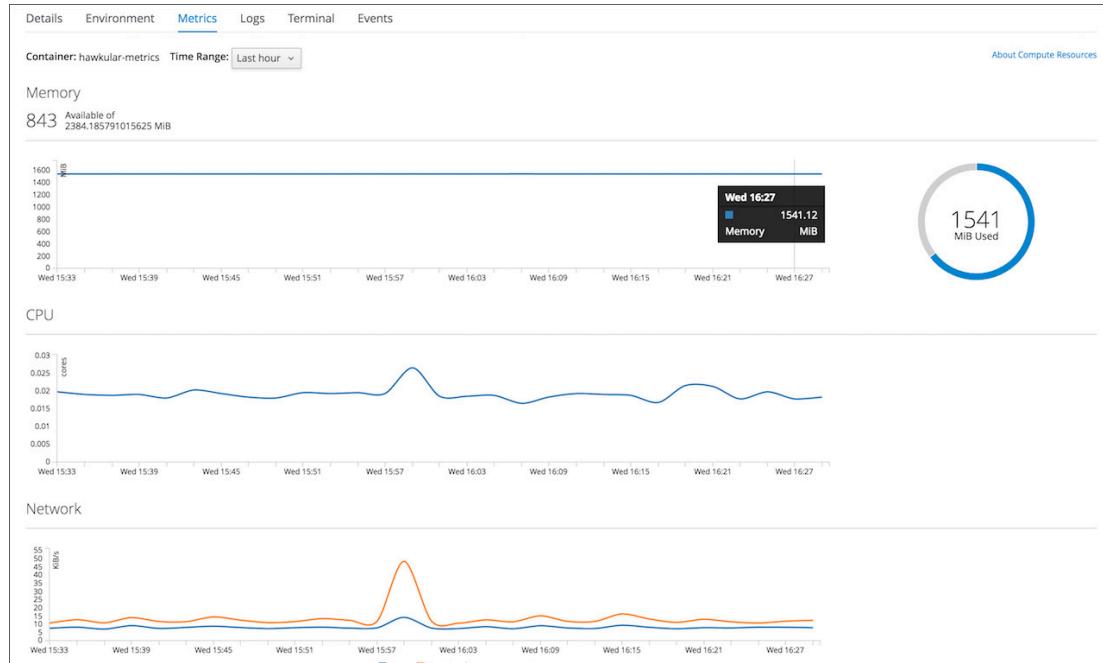


Figure 9.12: Monitoring pods with graphs

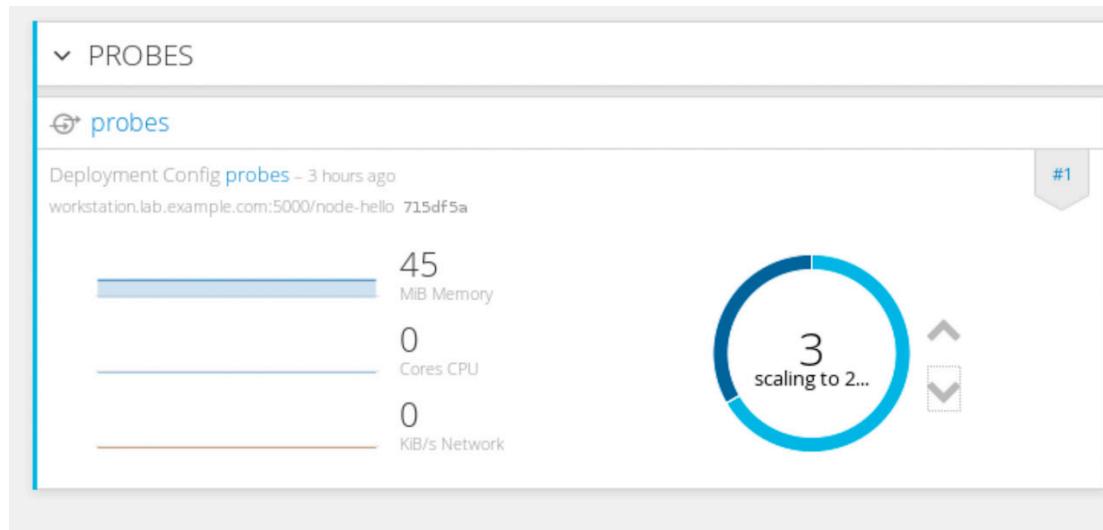


Figure 9.13: Overview of pods

Managing Deployments and Pods

The **Actions** button, available for pods and deployment, allows users to manage various settings. For example, they can add storage or a health check to a deployment. The button also gives access to a YAML editor for updating the configuration through the web console.

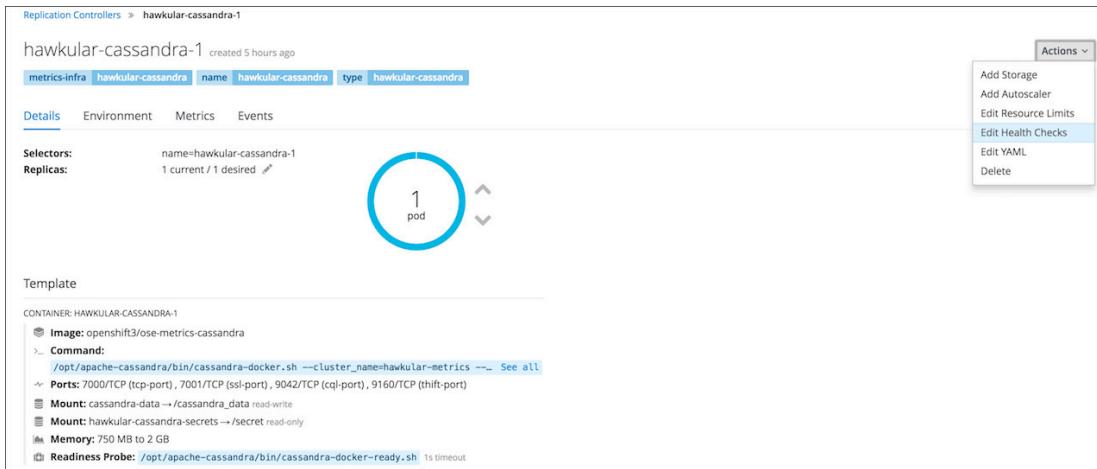


Figure 9.14: Managing deployments

Managing Storage

The web console gives developers access to storage management. Users can use the interface to create volume claims. Note that the interface cannot be used to create persistent volumes, because only administrators can perform this task. After a persistent volume has been created by the administrator, developers can use the web console to create a claim. The interface supports the use of selectors and labels attributes.

The following image shows how users can manage storage via the web console.

The screenshot shows the 'Create Storage' form. The top navigation bar includes 'load > Storage > Create Storage'. The form fields are as follows:

- * Name:** database (highlighted in yellow)
- * Access Mode:** Single User (RWO) (radio button selected)
- * Size:** 2 GiB
- Label Selector:**
 - datacenter: east
 - rack: 2
- Create** and **Cancel** buttons

Figure 9.15: Creating a persistent volume

After a volume claim has been defined, the console displays the persistent volume it uses, as defined by the administrator. The following image shows a volume claim bound to the **demo-storage** volume.

The screenshot shows the 'Storage' interface with a table of Persistent Volume Claims. There is one entry named 'database' which is bound to a volume named 'demo-storage'. The table includes columns for Name, Status, Capacity, Access Modes, and Age.

Name	Status	Capacity	Access Modes	Age
database	Bound to volume demo-storage	2 GiB	RWX (Read-Write-Many)	15 minutes

Figure 9.16: Listing available persistent volume claims

Use the **Add Storage** entry in the **Actions** menu to add storage to the deployment configuration.

The screenshot shows the 'Deployments' interface for a deployment named 'load'. The 'Actions' menu is open, and the 'Add Storage' option is highlighted. The deployment status is shown as active with 1 replica.

Figure 9.17: Adding storage for a deployment

This option allows developers to add an existing persistent volume claim to the template of a deployment configuration. After selecting this option, developers can specify the mount path, which is the path for the volume inside the container. If the path is not specified, the volume is not mounted. Developers can also specify the volume name, or let the console generate a name.

The screenshot shows the 'Add Storage' dialog. It includes fields for selecting a storage claim ('database', 2 GiB, Read-Write-Many, Bound to volume 'demo-storage'), specifying a mount path ('/data'), defining a subpath ('example: application/resources'), naming the volume ('database'), and setting optional flags ('Read only', 'Pause rollouts for this deployment config').

Figure 9.18: Storage definition

Adding storage triggers a new deployment for the deployment configuration. The new deployment instructs the pods to mount the volume.

The screenshot shows the Hawkular Metrics interface with the following details:

- CONTAINER: LOAD**
- Image:** workstation.lab.example.com:5000/node-hello 715df5a 164.8 MiB
- Ports:** 3000/TCP , 8080/TCP
- Mount:** database → /data read-write
- Volumes**
- database**
- Type:** persistent volume claim (reference to a persistent volume claim)
- Claim name:** database
- Mode:** read-write

Buttons at the bottom: [Add Storage to load](#) | [Add Config Files to load](#)

Figure 9.19: Overview of volumes attached to a pod

The screenshot shows the OpenShift web console for a pod named `load-18-fxtcq`, created 19 minutes ago. The pod has the following labels: `app`, `load`, `deployment`, `load-18`, `deploymentconfig`, and `load`. The `Terminal` tab is selected. The terminal output shows the following command:

```
dsh-4.2$ df -h
Filesystem
/dev/mapper/docker-253:1-75499049-cc6d12017eba5862e46eae2fab9ab0365a3f82322b9cfe7a0bd
tmpfs
tmpfs
master.lab.example.com:/var/export/storage-demo
/dev/vda1
shm
tmpfs
sh-4.2$
```

Figure 9.20: Using the terminal to review storage



References

Further information is available in the *Infrastructure Components* chapter of the *OpenShift Container Platform Architecture Guide* at

https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Exploring Metrics with the Web Console

In this exercise, you will explore metrics and storage with the OpenShift web console.

Outcomes

You should be able to:

- Create a new project.
- Deploy and scale an application.
- Read graphs from the web console.
- Create a volume claim.
- Add storage to a deployment configuration.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform*, and the section called “*Guided Exercise: Installing the Metrics Subsystem*” should be completed.

You should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts, and then run the following commands on the **workstation** host to ensure your environment is correctly configured for this exercise.

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab web-console setup
```

Steps

1. Create the **load** project.

1.1. On **workstation**, log in to the OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443  
Login successful.  
  
You have access to the following projects and can switch between them with 'oc  
project <projectname>':
```

1.2. Create the **load** project.

```
[student@workstation ~]$ oc new-project load  
Now using project "load" on server "https://master.lab.example.com:8443".
```

You can add applications to this project with the 'new-app' command. For example, try:

```
oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git
```

to build a new example application in Ruby.

2. Browse the Docker registry to ensure that the **hello-openshift** image is present. Create the **load** application using the **hello-openshift** image.
 - 2.1. Run the **docker-registry-cli** command to browse the Docker registry at **workstation.lab.example.com**. Use the **search** option to locate the image.

```
[student@workstation ~]$ docker-registry-cli workstation.lab.example.com:5000 \
  search node-hello
-----
1) Name: node-hello
Tags: latest

1 images found !
```

2.2. Create the **load** application:

```
[student@workstation ~]$ oc new-app --name=load \
  --docker-image=workstation.lab.example.com:5000/node-hello \
  --insecure-registry
--> Found Docker image f73799b (3 weeks old) from
  workstation.lab.example.com:5000 for "workstation.lab.example.com:5000/node-
hello"

  Node.js 6
  -----
  Platform for building and running Node.js 6 applications

  Tags: builder, nodejs, nodejs6

  * An image stream will be created as "load:latest" that will track this
  image
    * This image will be deployed in deployment config "load"
    * Ports 3000/tcp, 8080/tcp will be load balanced by service "load"
      * Other containers can access this service through the hostname "load"

--> Creating resources ...
  imagestream "load" created
  deploymentconfig "load" created
  service "load" created
--> Success
  Run 'oc status' to view your app.
```

2.3. Create a route for the application:

```
[student@workstation ~]$ oc expose svc/load
route "load" exposed
```

2.4. Wait until the application pod is ready and running:

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

NAME	READY	STATUS	...
load-1-kg98w	1/1	Running	...

3. Install the *httpd-tools* package to generate some load.

- 3.1. Use the **yum** command to install the *httpd-tools* package on **workstation**. When prompted, use **student** as the password.

```
[student@workstation ~]$ sudo yum -y install httpd-tools
...
[sudo] password for student: student
...

Updated:
    httpd-tools.x86_64 0:2.4.6-45.el7_3.4

Dependency Updated:
    httpd.x86_64 0:2.4.6-45.el7_3.4

Complete!
```

- 3.2. Run the **ab** command to generate some load on the application. Note that the trailing forward slash is mandatory at the end of the URL. The application is available at **http://load-load.cloudapps.lab.example.com**.

```
[student@workstation ~]$ ab -n 3000000 -c 20 \
    http://load-load.cloudapps.lab.example.com/
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking load-load.cloudapps.lab.example.com (be patient)
Completed 30000 requests
...
```

4. Log in to the OpenShift web console and scale up the pods.

- 4.1. From **workstation**, open Firefox and navigate to **https://master.lab.example.com:8443**. If prompted, accept the self-signed certificate.

- 4.2. Log in to the web console using **developer** as the user name, and **openshift** as the password.

- 4.3. Click the **load** entry to access the **load** project.

- 4.4. Review the **Overview** page. Ensure that there is one pod running, indicated by a blue donut. Notice the graphs next to the ring. Highlight the first graph, which corresponds to the memory used by the pod. As you highlight the graph, a box should appear, indicating how much memory is used by the pod.

Highlight the second graph, which indicates the number of CPUs used by the pods.

Highlight the third graph, which indicates the network traffic for the pods.

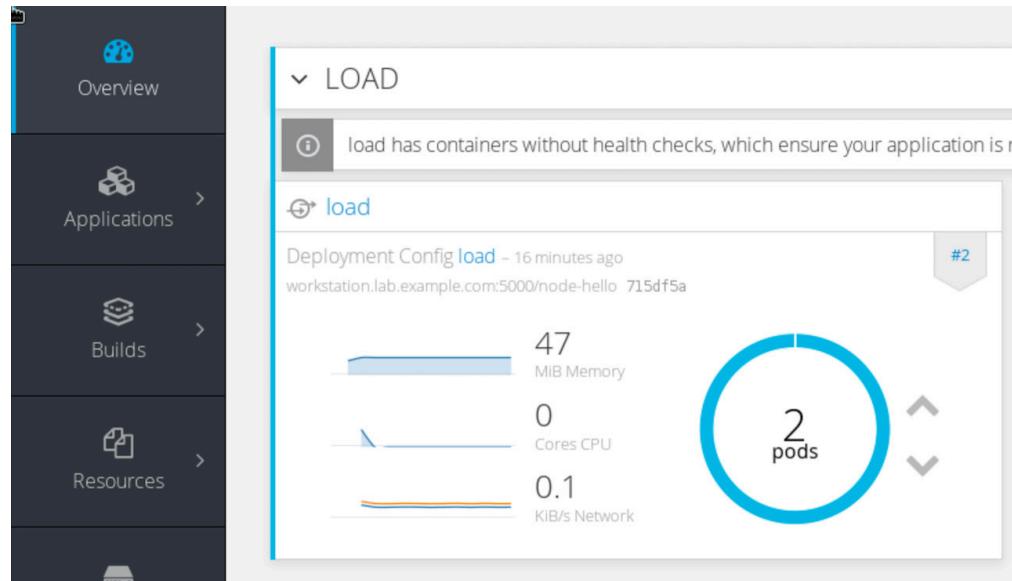


Figure 9.21: The Overview page

4.5. Click the up-arrow next to the blue donut to scale up the number of pods for this application to two.

4.6. Navigate to **Applications > Deployments** to access the deployments for the project.

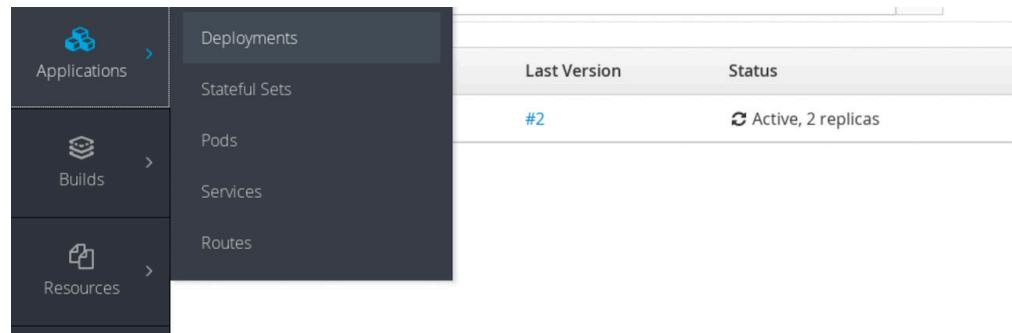


Figure 9.22: The Deployments page

From the **Deployments** page, click the **load** entry to access the deployment.

4.7. Notice the **Actions** button on the right side. Click it and select **Edit YAML** to edit the deployment configuration. Click **Actions** on the upper-right of the page, and then click **Edit YAML** to edit the deployment configuration.

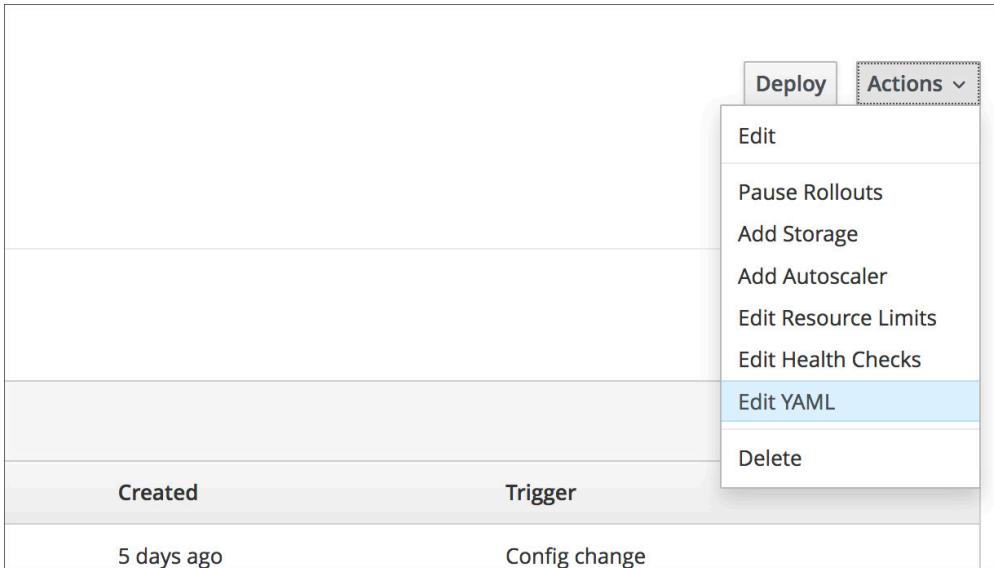


Figure 9.23: Editing YAML

Review the YAML file for the deployment. Scroll down and ensure that the **replicas** entry has a value of 2, which matches the number of pods running for this deployment. Click **Cancel** to return to the previous screen.

5. Review the metrics for the pods in the project.
 - 5.1. Click the latest deployment from the **Deployment** table to access the overview of the current deployment.
 - 5.2. Click the **Metrics** tab to access the metrics for the project. You should see four graphs for the application: the amount of used memory, the number of used CPUs, the amount of network packets that are received, and the amount of network packets that are sent. For each graph, there are two plots, each being assigned to one pod.



Figure 9.24: Metrics for the application

- 5.3. Highlight the graphs for the memory and notice the box that displays detailed values for the two pods running in the application.
6. Review the **Monitoring** section of the web console.
 - 6.1. From the side pane, click **Monitoring** to access the monitoring page. There should be two entries under the **Pods** section and one entry under the **Deployments** section.
 - 6.2. Click the arrow next to the first pod to access its details. You should see a console that reads **nodejs server running on http://0.0.0.0:3000**
Scroll down to access the graphs for the pod. There should be three graphs: one that indicates the amount of memory used by the pod, one that indicates the number of CPUS used by the pods, and one that indicate the network packets sent and received by the pod.
 - 6.3. Highlight the plots on the first graph, **Memory**. Notice the box that displays, which shows how much memory the pod uses.
 7. Create a volume claim for your application. The persistent volume that the claim will bind to is already provided by this exercise environment.
 - 7.1. Click **Storage** to create a persistent volume claim.
 - 7.2. Click **Create Storage** to define the claim.
 - 7.3. Enter **web-storage** as the **Name**. Select **Shared Access (RWX)** as the **Access Mode**. Enter **1** for the size and leave the unit as **GiB**.

The screenshot shows a 'Create Storage' dialog box. At the top, there's a breadcrumb navigation: 'load-review > Storage > Create Storage'. The main title is 'Create Storage'. Below it, a sub-instruction says 'Create a request for an administrator-defined storage asset by specifying size and permissions for a best fit.' A 'Learn More' link is available. The form fields are as follows:

- * Name:** A text input field containing 'web-storage'.
- * Access Mode:** A radio button group where 'Shared Access (RWX)' is selected, while 'Single User (RWO)' and 'Read Only (RO)' are unselected.
- * Size:** A text input field containing '1' with a dropdown menu showing 'GiB'.

Below the form, there's a note: 'Desired storage capacity.' followed by a 'What are GiB?' link. There's also a note about using 'label selectors' to request storage. At the bottom, there are 'Create' and 'Cancel' buttons, with 'Create' being the active button.

Figure 9.25: Creating a volume claim

Click **Create** to create the persistent volume claim. On the next page, ensure that the **Status** reads **Bound to volume web-storage**.

8. Add storage to your application.

8.1. Navigate to **Applications > Deployments** to manage the deployment.

Click the **Load** entry to access the deployment.

8.2. Click **Actions** for the deployment then select the **Add Storage** option. This option allows you to add an existing persistent volume claim to the template of the deployment configuration. Select **web-storage** as the storage claim.

8.3. Enter **/web-storage** as the **Mount Path** and **web-storage** as the **Volume Name**.

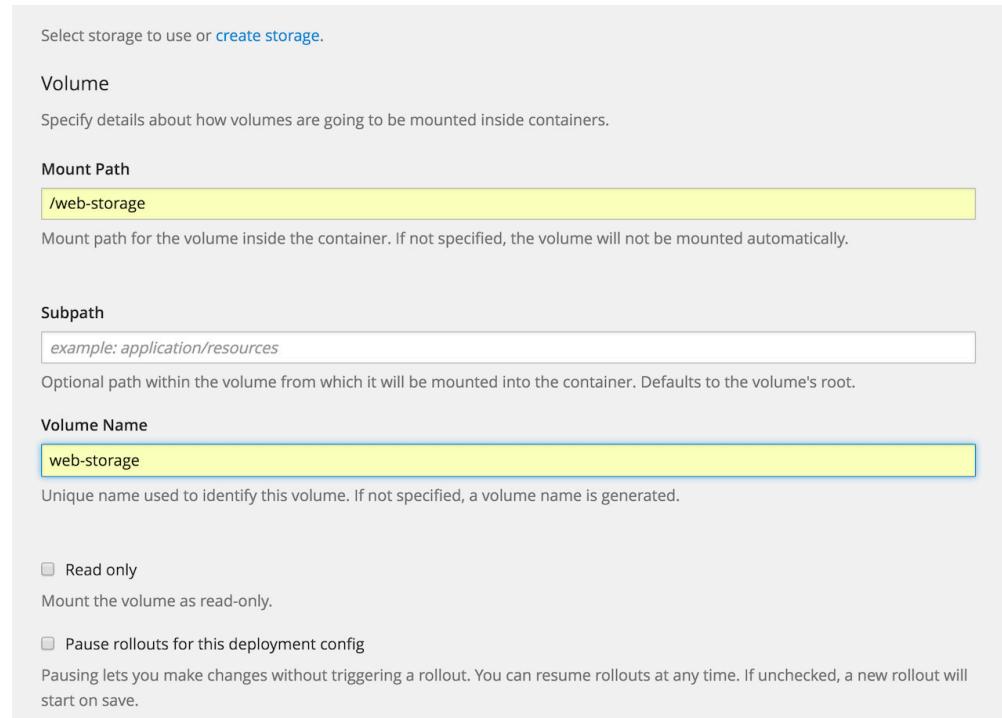


Figure 9.26: Adding storage to a deployment

Click **Add** to add the storage. The action triggers a new deployment and returns you to the **History** tab of the deployment.

9. Review the storage.

9.1. From the **Deployments** page, click the latest deployment. Wait until two replicas are marked as **Active**.

9.2. Ensure that the **Volumes** section has the volume **web-storage** as a persistent volume. From the **Pods** section, select one of the running pods.

9.3. Click the **Terminal** tab to open a shell for the pod.

9.4. Run the following command to locate the volume:

```
sh-4.2$ mount | grep web-storage
master.lab.example.com:/var/export/web-storage-ge on /web-storage type nfs4 ...
```

-
10. Clean up.
- 10.1. On the **workstation** host, delete the **load** project, which also deletes all the pods created during this lab:

```
[student@workstation ~]$ oc delete project load  
project "load" deleted
```

Lab: Managing and Monitoring OpenShift

In this lab, you will manage resource limits and monitor applications deployed on OpenShift.

Outcomes

You should be able to:

- Apply quotas and limits to a project.
- Verify quotas and limits for a project.
- Implement liveness probes for an application.

Before you begin

All the labs from *Chapter 2, Installing OpenShift Container Platform*, and the section called “*Guided Exercise: Installing the Metrics Subsystem*” should be completed.

You should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts, and then run the following commands on the **workstation** host to ensure your environment is correctly configured for this exercise:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab review-monitor setup
```

Steps

1. On **workstation**, log in to the OpenShift cluster as the **developer** user and create the **load-review** project.
2. On **workstation**, inspect the **limits.yml** file, which defines limit ranges, located at **/home/student/D0280/labs/monitor-review**. Set limits to provide default resource requests for pods created inside the project.
- As the cluster administrator, set the limits as defined by the file and review the limits created.
3. As the **developer** user, create the **load** application in the **load-review** project using the **node-hello** image at **workstation.lab.example.com:5000**. The application exposes two HTTP **GET** URLs at **/health** and **/ready**.
4. Ensure that the default requested limit by the pod matches the limits set for the project.
5. Update the limit for the deployment configuration by requesting 350 MiB. Review the events in the project. Look for the entry that indicates that the request was rejected because of limit violation. This should also prevent the deployment of a new version of the application. a new version of the deployment. Revert the request to 200 MiB.

-
6. In the `/home/student/D0280/labs/monitor-review/` directory, inspect the `quotas.yml` file, which defines quotas for the `load-review` project.

As the cluster administrator, define quotas for the `load-review` project and review the quotas.
 7. As the **developer** user, scale up the application by adding four replicas, and review the events for the project. Ensure that the deployment configuration cannot create the fourth pod due to quota violation. Use the `grep` command to filter on **Warning** messages. Scale down the number of replicas to one.
 8. Expose a route for the `load` service to allow external clients to access the application.
 9. Log in to the web console to create a liveness probe. Use the information provided in the following table.

Readiness Probe

Field	Value
Type	HTTP GET
Path	/health
Port	3000
Initial Delay	10
Timeout	3

10. Ensure that the liveness probe has been successfully created.
11. Grade your work. Run the following command to grade your work:

```
[student@workstation ~]$ lab review-monitor grade
```

If you do not get **PASS** grades for all tasks, review your work and run the grading command again.

12. Clean up.

Delete the `load-review` project:

Solution

In this lab, you will manage resource limits and monitor applications deployed on OpenShift.

Outcomes

You should be able to:

- Apply quotas and limits to a project.
- Verify quotas and limits for a project.
- Implement liveness probes for an application.

Before you begin

All the labs from ???, and ??? should be completed.

You should have an OpenShift Container Platform cluster running with a master and two nodes. If not, reset the **master**, **node1**, and **node2** hosts, and then run the following commands on the **workstation** host to ensure your environment is correctly configured for this exercise:

```
[student@workstation ~]$ lab install-post setup  
[student@workstation ~]$ ansible-playbook -i ~/inventory ~/full_classroom_install.yml
```

To verify that the **master**, **node1**, and **node2** hosts are started, and to download the files needed by this guided exercise, open a terminal on **workstation** and run the following command:

```
[student@workstation ~]$ lab review-monitor setup
```

Steps

1. On **workstation**, log in to the OpenShift cluster as the **developer** user and create the **load-review** project.
 - 1.1. On **workstation**, open a terminal and log in to the OpenShift cluster as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \  
https://master.lab.example.com:8443  
Login successful.  
  
You don't have any projects. You can try to create a new project, by running  
oc new-project <projectname>
```

- 1.2. Create the **load-review** project:

```
[student@workstation ~]$ oc new-project load-review  
Now using project "load-review" on server "https://master.lab.example.com:8443".  
  
You can add applications to this project with the 'new-app' command. For  
example, try:  
  
oc new-app centos/ruby-22-centos7-https://github.com/openshift/ruby-ex.git  
to build a new example application in Ruby.
```

2. On **workstation**, inspect the **limits.yml** file, which defines limit ranges, located at **/home/student/D0280/labs/monitor-review**. Set limits to provide default resource requests for pods created inside the project.

As the cluster administrator, set the limits as defined by the file and review the limits created.

- 2.1. Inspect the **limits.yml** file in the lab directory, **/home/student/D0280/labs/monitor-review**. The file reads as follows:

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "review-limits"
spec:
  limits:
    - type: "Container"
      max:
        memory: "300Mi"
      default:
        memory: "200Mi"
```

- 2.2. Log in to the OpenShift cluster as the administrator:

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...
```

- 2.3. Enter the **load-review** project:

```
[student@workstation ~]$ oc project load-review
Now using project "load-review" on server "https://master.lab.example.com:8443".
```

- 2.4. As the cluster administrator, set the limits as defined by the file for the **load-review** project:

```
[student@workstation ~]$ oc create -f \
  /home/student/D0280/labs/monitor-review/limits.yml
limitrange "review-limits" created
```

- 2.5. Run the **oc describe** command to review the limits created:

limits						
Name:	review-limits					
Namespace:		load-review				
Type	Resource	Min	Max	Default Request	Default Limit	
Container	memory	-	300Mi	200Mi	200Mi	-

3. As the **developer** user, create the **load** application in the **load-review** project using the **node-hello** image at **workstation.lab.example.com:5000**. The application exposes two HTTP **GET** URLs at **/health** and **/ready**.

- 3.1. Log in as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
```

- 3.2. Use the **oc new-app** command to create the **load** application:

```
[student@workstation ~]$ oc new-app --name=load \
--docker-image=workstation.lab.example.com:5000/node-hello \
--insecure-registry
```

4. Ensure that the default requested limit by the pod matches the limits set for the project.

- 4.1. Run the **oc get pods** command to list the pods present in the environment:

```
[student@workstation ~]$ oc get pods
NAME      READY   STATUS    RESTARTS   AGE
load-1-fpm1v  1/1     Running   0          3m
```

- 4.2. Inspect the pod returned by the previous command and locate the **Limits** and **Requests** in the **Container** section:

```
[student@workstation ~]$ oc describe pod load-1-fpm1v
...
Containers:
...
  Limits:
    memory: 200Mi
  Requests:
    memory: 200Mi
...
```

5. Update the limit for the deployment configuration by requesting 350 MiB. Review the events in the project. Look for the entry that indicates that the request was rejected because of limit violation. This should also prevent the deployment of a new version of the application. a new version of the deployment. Revert the request to 200 Mib.

- 5.1. Request 350 MiB by running the **oc set resources** command:

```
[student@workstation ~]$ oc set resources dc load --requests=memory=350Mi
deploymentconfig "load" resource requirements updated
```

- 5.2. Run the **oc get events** command and locate the entry that indicates a limits violation. It might take a few moments until you see the error message.

```
[student@workstation ~]$ oc get events | grep Warning
Warning FailedCreate {replication-controller} Error creating:
pods "load-2-" is forbidden: maximum memory usage per Container is 300Mi, but
request is 350Mi.
```

- 5.3. Revert the limit to 200 Mib:

```
[student@workstation ~]$ oc set resources dc load --requests=memory=200Mi
```

- 5.4. Wait until the new pod, from the third deployment, is ready and running:

```
[student@workstation ~]$ oc status ; oc get pod
In project load-review on server https://master.lab.example.com:8443

svc/load - 172.30.16.168 ports 3000, 8080
dc/load deploys istag/load:latest
  deployment #3 deployed about a minute ago - 1 pod
  deployment #2 failed about a minute ago: newer deployment was found running
  deployment #1 deployed 2 minutes ago

View details with 'oc describe <resource>/<name>' or list everything with 'oc
get all'.
NAME        READY     STATUS    RESTARTS   AGE
load-3-bjsql 1/1      Running   0          1m
```

6. In the **/home/student/D0280/labs/monitor-review/** directory, inspect the **quotas.yml** file, which defines quotas for the **load-review** project.

As the cluster administrator, define quotas for the **load-review** project and review the quotas.

- 6.1. Inspect the **quotas.yml** file to set quotas for the **load-review** project. The file reads as follows:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: review-quotas
spec:
  hard:
    requests.memory: "600Mi"
```

- 6.2. Log in as the **admin** user:

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...
```

- 6.3. Set quotas for the **load-review** project:

```
[student@workstation ~]$ oc create -f \
/home/student/D0280/labs/monitor-review/quotas.yml
resourcequota "review-quotas" created
```

- 6.4. Review the quotas set for the **load-review** project by running the **oc describe** command to review the quotas that were set for the **load-review** project:

```
[student@workstation ~]$ oc describe quota
Name:           review-quotas
Namespace:      load-review
```

Resource	Used	Hard
requests.memory	400Mi	600Mi

7. As the **developer** user, scale up the application by adding four replicas, and review the events for the project. Ensure that the deployment configuration cannot create the fourth pod due to quota violation. Use the **grep** command to filter on **Warning** messages. Scale down the number of replicas to one.

- 7.1. Log in as the **developer** user:

```
[student@workstation ~]$ oc login -u developer -p openshift
Login successful.

You have one project on this server: "load-review"

Using project "load-review".
```

- 7.2. Request four pods for the application by running the **oc set resources** command:

```
[student@workstation ~]$ oc scale --replicas=4 dc load
deploymentconfig "load" scaled
```

- 7.3. Run the **oc get pods** command to list the number of running pods. Wait for three pods to be ready and running. Notice that the fourth pod is not created:

```
[student@workstation ~]$ oc get pods
NAME      READY     STATUS    RESTARTS   AGE
load-1-stdnx  1/1      Running   0          29s
load-1-vg1m1  1/1      Running   0          29s
load-1-zwdh0  1/1      Running   0          2m
```

- 7.4. Review the events for the project. Locate the entry that indicates that the quota was applied, which prevents the fourth pod from being created:

```
[student@workstation ~]$ oc get events | grep Warning
...
Warning FailedCreate           {replication-controller }
Error creating: pods "load-3-" is forbidden: exceeded quota: review-quotas,
requested: requests.memory=200Mi, used: requests.memory=600Mi, limited:
requests.memory=600Mi
```

- 7.5. Run the **oc scale** command to scale down the number of replicas to one:

```
[student@workstation ~]$ oc scale --replicas=1 dc load
deploymentconfig "load" scaled
```

8. Expose a route for the **load** service to allow external clients to access the application.

- 8.1. Expose a route for the **load** service, and use **load-review.cloudapps.lab.example.com** as the host name:

```
[student@workstation ~]$ oc expose svc load \
--hostname=load-review.cloudapps.lab.example.com
route "load" exposed
```

9. Log in to the web console to create a liveness probe. Use the information provided in the following table.

Readiness Probe

Field	Value
Type	HTTP GET
Path	/health
Port	3000
Initial Delay	10
Timeout	3

- 9.1. From **workstation**, open Firefox and navigate to the OpenShift web console at **https://master.lab.example.com:8443**. Use **developer** as the user name, and **openshift** as the password. Click **load-review** to access the project.
 - 9.2. To add a liveness probe, navigate to **Applications > Deployments**, and click the **load** deployment.
- Click **Actions** and select the **Edit Health Checks** entry to add a liveness probe.

- 9.3. Click the **Add Liveness Probe** link to create the probe. Select **HTTP GET** for the **Type** field, and **/health** for the **Path** field. Enter **10** for the **Initial Delay** field, and **3** for the **Timeout** field.

Click **Save** to create the liveness probe.

10. Ensure that the liveness probe has been successfully created.

- 10.1. Navigate to **Applications > Deployments** and select the **load** deployment.

Select the latest deployment for the application.

- 10.2. In the **Template** section, locate the following entry:

```
Liveness Probe: GET /health on port 3000 (HTTP) 10s delay, 3s timeout
```

11. Grade your work. Run the following command to grade your work:

```
[student@workstation ~]$ lab review-monitor grade
```

If you do not get **PASS** grades for all tasks, review your work and run the grading command again.

12. Clean up.

Delete the **load-review** project:

```
[student@workstation ~]$ oc delete project load-review
```

Summary

In this chapter, you learned:

- OpenShift Container Platform can enforce quotas that track and limit the usage of two kinds of resources: object counts and compute resources.
- There are two methods for performing OpenShift Container Platform cluster upgrades: in-place upgrades (automated or manual), and upgrading using a blue-green deployment method.
- OpenShift applications can become unhealthy due to temporary connectivity loss, configuration errors, application errors, and similar issues. Developers can use probes to monitor their applications to help manage these issues.
- The web console integrates a set of features that provide real-time feedback, such as the state of a deployment, pod, service, and other resources, as well as providing information about system-wide events.



CHAPTER 10

COMPREHENSIVE REVIEW: RED HAT OPENSHIFT ADMINISTRATION I

Overview	
Goal	Practice and demonstrate knowledge and skills learned in this course.
Objectives	Review information needed to successfully complete the review lab.
Sections	Comprehensive Review
Lab	OpenShift Container Platform Administration I Comprehensive Review

Comprehensive Review

Objectives

After completing this section, students should be able to review and refresh knowledge and skills learned in *Red Hat OpenShift Administration I*.

Reviewing Red Hat OpenShift Administration I

Before beginning the comprehensive review for this course, students should be comfortable with the topics covered in each chapter.

Students can refer to earlier sections in the textbook for extra study.

Chapter 1, Introducing Red Hat OpenShift Container Platform

List the features and describe the architecture of the OpenShift Container Platform.

- Describe the typical use of the product and list its features.
- Describe the architecture of OpenShift.

Chapter 2, Installing OpenShift Container Platform

Install OpenShift and configure the cluster.

- Prepare the servers for installation.
- Run the OCP installer to configure the cluster.
- Execute postinstallation tasks and verify the cluster configuration.

Chapter 3, Describing and Exploring OpenShift Networking Concepts

Describe and explore OpenShift networking concepts.

- Describe how OpenShift implements software-defined networking.
- Describe how OpenShift routing works and create a route.

Chapter 4, Executing Commands

Execute commands using the command-line interface.

- Configure OpenShift resources using the command-line interface.
- Execute commands that assist in troubleshooting common problems.

Chapter 5, Controlling Access to OpenShift Resources

Control access to OpenShift resources.

- Segregate resources and control access to them using OpenShift security features.
- Create and apply secrets to manage sensitive information.
- Manage security policies using the command-line interface.

Chapter 6, Allocating Persistent Storage

Implement persistent storage.

- Provision persistent storage for use by applications.
- Configure the internal container registry for persistence.

Chapter 7, Managing Application Deployments

Manipulate resources to manage deployed applications.

- Control the number of replications of a pod.
- Describe and control how pods are scheduled on the cluster.
- Manage the images, image streams, and templates used in application builds.

Chapter 8, Installing and Configuring the Metrics Subsystem

Install and configure the metrics gathering subsystem.

- Describe the architecture and operation of the metrics subsystem.
- Install the metrics subsystem.

Chapter 9, Managing and Monitoring OpenShift Container Platform

Manage and monitor OpenShift resources and software.

- Limit the amount of resources consumed by an application.
- Upgrade an instance of OpenShift.
- Configure probes to monitor application health.
- Monitor OpenShift resources using data obtained from the web console.

General OpenShift Container Platform Hints

These hints may save some time and simplify the implementation:

- The **oc new-app** command can create pods from any source, and also most other application resources. It can also be used to generate resource definition files for customization.
- Use the **oc get** and **oc describe** commands to inspect existing resources. Use the **oc export** command to export definitions to resource definition files.
- If a set of application resources has a matching label, you can use a single **oc delete all -l name=value** command to delete all of them.
- OpenShift standard templates are in the **openshift** name space.
- Most OpenShift operations are asynchronous and may take a few seconds to complete. Having pods in a *Pending* state usually indicates that OpenShift is still creating the pod.
- The master can directly access any pod's internal IP address, even when there are no routes or services configured.
- The **oc exec** command can run commands from inside a pod, even from a developer's workstation.
- The **oc port-forward** command allows a developer's workstation to connect to network services inside a pod, even without any services or routes configured.
- The Linux **root** user on the master can always use the **oc login** command as the OpenShift cluster administrator (**system:admin**) without a password.



Note

Previous chapter labs and demonstrations contain troubleshooting hints that will be useful if problems occur during the completion of this lab.

Lab: Installing OpenShift

In this review, you will install OpenShift in a three-node cluster and verify that the OpenShift cluster is fully functional after installation.

Outcomes

You should be able to:

- Prepare each host for installation as an OpenShift node.
- Perform an offline installation of OpenShift, with one master and two additional nodes.
- Perform postinstallation tasks to make the registry console and S2I image stream work in an environment without internet access.
- Configure the OpenShift internal registry to use persistent storage.
- Create two initial users, one of them being a cluster administrator.

Before you begin

Reset the **master**, **node1**, and **node2** VMs. There is no need to reset the **workstation** VM.

Run the following command to verify that your environment is ready for starting this review lab and also to download sample solution files:

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

Instructions

Install Red Hat OpenShift Container Platform 3.5, using the quick installer, with the following configuration:

- The OpenShift cluster has three nodes:
 - **master.lab.example.com** is the OpenShift master and is an *unschedulable* node.
 - **node1.lab.example.com** is an OpenShift node that may run both application and infrastructure pods.
 - **node2.lab.example.com** is another OpenShift node that may run both application and infrastructure pods.
- All nodes use LVM thin volumes for Docker storage. The second disk (**vdb**) in each node is reserved for Docker storage.
- All nodes are to use an RPM-based installation.
- The default domain for routes is **cloudapps.lab.example.com**. The classroom DNS server is already configured to resolve all host names in this domain to **node1.lab.example.com**. The domain is also known as the wildcard domain.
- All container images used by the OpenShift cluster are stored in a private registry available at **workstation.lab.example.com** on port 5000. This container image registry should be considered insecure because it is not configured with a valid SSL certificate.

- Authentication uses an Apache HTPasswd file. Create the **/etc/origin/openshift-passwd** file in the **master** host. Create two initial users: **developer** with password **openshift** as a regular user, and **admin** with password **redhat** as a cluster administrator.
- Container images built by S2I are retained on restart. Persistent storage is provided by an NFS server configured on the **master.lab.example.com** host by the OpenShift quick installer, and it exports shares from the **/exports** folder. Allocate 15GiB of storage to the internal registry.

RPM packages required to install OpenShift are defined by Yum configuration files already installed on all hosts. Enable the required repositories.

Some of the configuration required by this exercise cannot be done using the quick installer in interactive mode. You are free to perform configuration fixes after running the installer, but it is recommended to use the unattended mode with a configuration file hand-crafted for a cluster without internet access.

The **/home/student/D0280/labs/review-install** folder provides sample scripts and configuration files to run the OpenShift quick installer in unattended mode. This folder also provides sample YAML files for resources such as persistent volumes. There is also a copy of these files in the **/root/D0280/labs/review-install** folder, in the **master** host.

A test application is provided by a Git server at **http://workstation.lab.example.com/php-helloworld**. It is a simple "hello, world" application. You may deploy this application using S2I to verify that the OpenShift internal registry and the router are performing as expected.

The Red Hat OpenShift Container Platform 3.5 product documentation is provided in the **workstation** host desktop. Use the *Installation Guide* and the *Developer Guide* as reference about installation prerequisites, configuration files, and specific commands to perform installation and configuration of the OpenShift cluster.

Proposed Solution Steps

The following is a summary of steps to perform this exercise. Perform these steps from the **master** host. To save time, most of these steps are already scripted in the **/root/D0280/labs/review-install** folder. For each script, use the **TODO** comments as a guide to complete the script, and then run it. Each of the following steps is performed by one script:

1. Prepare the hosts for installing OpenShift. Complete the **pre-install.sh** script to do the following:
 - 1.1. Check that all host names are resolvable using DNS.
 - 1.2. Check that the default wild card domain resolves a test host name.
 - 1.3. Generate SSH keys and copy them to all hosts.
 - 1.4. Check that all hosts have the Network Manager service active.
 - 1.5. Enable the Yum repositories required by the OpenShift installer.
 - 1.6. Install the prerequisite packages.
 - 1.7. Install the OpenShift excluder and installer packages.
2. Install and configure Docker on all hosts. Complete the **configure-docker.sh** script to do the following:

-
- 2.1. Install Docker.
 - 2.2. Configure Docker storage to use LVM thin pools.
 - 2.3. Start and enable the Docker service.
 3. Run the OpenShift quick installer. Copy the installer configuration file **ocp35-1master-2nodes.cfg.yml** to the **/root** folder and complete the file for one master and two nodes. Complete the **install-ocp.sh** script to do the following:
 - 3.1. Create a quick installer configuration file from the provided samples.
 - 3.2. Add to the configuration file the Ansible variables required for using a private, insecure registry.
 - 3.3. Run the quick installer in unattended mode.
 - 3.4. Check that all nodes are ready.
 - 3.5. Check that the router and registry pods are ready and running.
 4. Fix the S2I image streams and the router console deployment for offline usage. Complete the **post-install.sh** script to do the following:
 - 4.1. Fix the router console deployment to use the private, insecure registry.
 - 4.2. Delete all public image streams.
 - 4.3. Modify the example image stream JSON file to use the private, insecure registry.
 - 4.4. Recreate the example image streams.
 - 4.5. Verify that OpenShift was able to fetch metadata for the images provided by the private registry.
 5. Make the OpenShift internal registry persistent. Complete the **registry-pv.yml** file to point to the NFS share created by the quick installer. Also complete the **persistent-registry.sh** script to do the following:
 - 5.1. Create a persistent volume for the NFS share at **/exports/registry**.
 - 5.2. Add a persistent volume claim to the registry deployment.
 - 5.3. Check that a new registry pod is ready and running.
 - 5.4. Check that the new pod is using the persistent volume.
 6. Configure OpenShift for Apache HTPasswd authentication and create initial users. Complete the **configure-auth.sh** script to do the following:
 - 6.1. Install the Apache httpd-tools package.
 - 6.2. Configure the OpenShift master to use the HTPasswd password identity provider.
 - 6.3. Create the HTPasswd file.

- 6.4. Restart the OpenShift master service.
 - 6.5. Add users **developer** and **admin** to the HTPasswd file.
 - 6.6. Grant the cluster administrator role to the **admin** user.
7. Install the OpenShift client on the workstation VM.
 8. Validate your installation by deploying the test application using S2I. The **test-app.sh** script is ready to use.

Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab review-install grade
```

If you do not get an overall PASS grade, review your work and run the grading command again.

Solution

In this review, you will install OpenShift in a three-node cluster and verify that the OpenShift cluster is fully functional after installation.

Outcomes

You should be able to:

- Prepare each host for installation as an OpenShift node.
- Perform an offline installation of OpenShift, with one master and two additional nodes.
- Perform postinstallation tasks to make the registry console and S2I image stream work in an environment without internet access.
- Configure the OpenShift internal registry to use persistent storage.
- Create two initial users, one of them being a cluster administrator.

Before you begin

Reset the **master**, **node1**, and **node2** VMs. There is no need to reset the **workstation** VM.

Run the following command to verify that your environment is ready for starting this review lab and also to download sample solution files:

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

Instructions

Install Red Hat OpenShift Container Platform 3.5, using the quick installer, with the following configuration:

- The OpenShift cluster has three nodes:
 - **master.lab.example.com** is the OpenShift master and is an *unschedulable* node.
 - **node1.lab.example.com** is an OpenShift node that may run both application and infrastructure pods.
 - **node2.lab.example.com** is another OpenShift node that may run both application and infrastructure pods.
- All nodes use LVM thin volumes for Docker storage. The second disk (**vdb**) in each node is reserved for Docker storage.
- All nodes are to use an RPM-based installation.
- The default domain for routes is **cloudapps.lab.example.com**. The classroom DNS server is already configured to resolve all host names in this domain to **node1.lab.example.com**. The domain is also known as the wildcard domain.
- All container images used by the OpenShift cluster are stored in a private registry available at **workstation.lab.example.com** on port 5000. This container image registry should be considered insecure because it is not configured with a valid SSL certificate.

- Authentication uses an Apache HTPasswd file. Create the **/etc/origin/openshift-passwd** file in the **master** host. Create two initial users: **developer** with password **openshift** as a regular user, and **admin** with password **redhat** as a cluster administrator.
- Container images built by S2I are retained on restart. Persistent storage is provided by an NFS server configured on the **master.lab.example.com** host by the OpenShift quick installer, and it exports shares from the **/exports** folder. Allocate 15GiB of storage to the internal registry.

RPM packages required to install OpenShift are defined by Yum configuration files already installed on all hosts. Enable the required repositories.

Some of the configuration required by this exercise cannot be done using the quick installer in interactive mode. You are free to perform configuration fixes after running the installer, but it is recommended to use the unattended mode with a configuration file hand-crafted for a cluster without internet access.

The **/home/student/D0280/labs/review-install** folder provides sample scripts and configuration files to run the OpenShift quick installer in unattended mode. This folder also provides sample YAML files for resources such as persistent volumes. There is also a copy of these files in the **/root/D0280/labs/review-install** folder, in the **master** host.

A test application is provided by a Git server at **http://workstation.lab.example.com/php-helloworld**. It is a simple "hello, world" application. You may deploy this application using S2I to verify that the OpenShift internal registry and the router are performing as expected.

The Red Hat OpenShift Container Platform 3.5 product documentation is provided in the **workstation** host desktop. Use the *Installation Guide* and the *Developer Guide* as reference about installation prerequisites, configuration files, and specific commands to perform installation and configuration of the OpenShift cluster.

Proposed Solution Steps

The following is a summary of steps to perform this exercise. Perform these steps from the **master** host. To save time, most of these steps are already scripted in the **/root/D0280/labs/review-install** folder. For each script, use the **TODO** comments as a guide to complete the script, and then run it. Each of the following steps is performed by one script:

1. Prepare the hosts for installing OpenShift. Complete the **pre-install.sh** script to do the following:
 - 1.1. Check that all host names are resolvable using DNS.
 - 1.2. Check that the default wild card domain resolves a test host name.
 - 1.3. Generate SSH keys and copy them to all hosts.
 - 1.4. Check that all hosts have the Network Manager service active.
 - 1.5. Enable the Yum repositories required by the OpenShift installer.
 - 1.6. Install the prerequisite packages.
 - 1.7. Install the OpenShift excluder and installer packages.
2. Install and configure Docker on all hosts. Complete the **configure-docker.sh** script to do the following:

- 2.1. Install Docker.
- 2.2. Configure Docker storage to use LVM thin pools.
- 2.3. Start and enable the Docker service.
3. Run the OpenShift quick installer. Copy the installer configuration file **ocp35-1master-2nodes.cfg.yml** to the **/root** folder and complete the file for one master and two nodes. Complete the **install-ocp.sh** script to do the following:
 - 3.1. Create a quick installer configuration file from the provided samples.
 - 3.2. Add to the configuration file the Ansible variables required for using a private, insecure registry.
 - 3.3. Run the quick installer in unattended mode.
 - 3.4. Check that all nodes are ready.
 - 3.5. Check that the router and registry pods are ready and running.
4. Fix the S2I image streams and the router console deployment for offline usage. Complete the **post-install.sh** script to do the following:
 - 4.1. Fix the router console deployment to use the private, insecure registry.
 - 4.2. Delete all public image streams.
 - 4.3. Modify the example image stream JSON file to use the private, insecure registry.
 - 4.4. Recreate the example image streams.
 - 4.5. Verify that OpenShift was able to fetch metadata for the images provided by the private registry.
5. Make the OpenShift internal registry persistent. Complete the **registry-pv.yml** file to point to the NFS share created by the quick installer. Also complete the **persistent-registry.sh** script to do the following:
 - 5.1. Create a persistent volume for the NFS share at **/exports/registry**.
 - 5.2. Add a persistent volume claim to the registry deployment.
 - 5.3. Check that a new registry pod is ready and running.
 - 5.4. Check that the new pod is using the persistent volume.
6. Configure OpenShift for Apache HTPasswd authentication and create initial users. Complete the **configure-auth.sh** script to do the following:
 - 6.1. Install the Apache httpd-tools package.
 - 6.2. Configure the OpenShift master to use the HTPasswd password identity provider.
 - 6.3. Create the HTPasswd file.

- 6.4. Restart the OpenShift master service.
 - 6.5. Add users **developer** and **admin** to the HTPasswd file.
 - 6.6. Grant the cluster administrator role to the **admin** user.
7. Install the OpenShift client on the workstation VM.
 8. Validate your installation by deploying the test application using S2I. The **test-app.sh** script is ready to use.

Steps

1. Prepare the hosts for installing OpenShift.

Verify and configure the OpenShift installation prerequisites on the **master**, **node1**, and **node2** hosts.

All commands in this step are in the **pre-install.sh** script in the **/root/D0280/labs/review-install** folder. Notice from the **TODO** comments in the script that Step 1.9 is not complete. Complete the script and run it from the **master** host, or follow the manual steps below.



Important

Read each step carefully for instructions about repeating commands and copying files in the **node1** and **node2** hosts. Most times, the narrative shows only the commands in the **master** host, but not all commands performed in the **master** are to be performed in the **node1** and **node2** hosts.

- 1.1. Open an SSH session to the **master** host, and perform the next steps from there.

```
[student@workstation ~]$ ssh root@master
```

- 1.2. Verify that each host name is resolved by DNS:

```
[root@master ~]# dig master.lab.example.com +short  
172.25.250.10  
[root@master ~]# dig node1.lab.example.com +short  
172.25.250.11  
[root@master ~]# dig node2.lab.example.com +short  
172.25.250.12
```

- 1.3. Verify that the wildcard DNS entry for **cloudapps.lab.example.com** points to the IP address of the **node1** host:

```
[root@master ~]# dig test.cloudapps.lab.example.com +short  
172.25.250.11
```

- 1.4. Generate an SSH key pair in the **master** and copy the public key to all hosts. Key-based SSH access is required by the OpenShift installer and also eases performing prerequisite steps. The root password in all hosts is **redhat**.

```
[root@master ~]# ssh-keygen -f /root/.ssh/id_rsa -N ''
...
[root@master ~]# ssh-copy-id root@master
...
[root@master ~]# ssh-copy-id root@node1
...
[root@master ~]# ssh-copy-id root@node2
```

- 1.5. Verify that Network Manager is active on all hosts.

```
[root@master ~]# systemctl status NetworkManager
● NetworkManager.service - Network Manager
  Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled;
  vendor preset: enabled)
    Active: active (running) since ...
...
[root@master ~]# ssh node1 systemctl status NetworkManager
● NetworkManager.service - Network Manager
  Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled;
  vendor preset: enabled)
    Active: active (running) since ...
...
[root@master ~]# ssh node2 systemctl status NetworkManager
● NetworkManager.service - Network Manager
  Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled;
  vendor preset: enabled)
    Active: active (running) since ...
```

- 1.6. Enable only the required Yum repositories by the OpenShift Container Platform installer.

```
[root@master ~]# yum-config-manager --disable "*"
...
[root@master ~]# yum-config-manager \
    --enable "rhel_dvd" \
    --enable "rhel-7-server-extras-rpms" \
    --enable "rhel-7-server-updates-rpms" \
    --enable "rhel-7-server-ose-3.5-rpms" \
    --enable "rhel-7-fast-datapath-rpms"
```

Use SSH to perform the same commands in **node1** host:

```
[root@master ~]# ssh node1 yum-config-manager --disable "*"
...
[root@master ~]# ssh node1 yum-config-manager \
    --enable "rhel_dvd" \
    --enable "rhel-7-server-extras-rpms" \
    --enable "rhel-7-server-updates-rpms" \
    --enable "rhel-7-server-ose-3.5-rpms" \
    --enable "rhel-7-fast-datapath-rpms"
```

Use SSH to perform the same commands in **node2** host.

- 1.7. Install prerequisite packages on all hosts.

```
[root@master ~]# yum -y install wget git net-tools bind-utils \
```

```
iptables-services bridge-utils bash-completion kexec-tools sos psacct
```

Use SSH to perform the same command on **node1** and **node2** hosts.

- 1.8. Install the OpenShift excluder packages in all hosts, and unexclude the production Docker packages.

```
[root@master ~]# yum -y install \
atomic-openshift-excluder atomic-openshift-docker-excluder
[root@master ~]# atomic-openshift-excluder unexclude
```

Use SSH to repeat the same commands in **node1** and **node2**.

- 1.9. Install the OpenShift utilities package on the **master** host. This package provides the OpenShift installers.

```
[root@master ~]# yum -y install atomic-openshift-utils
```

2. Install and configure Docker on all hosts.

All commands in this step are in the **configure-docker.sh** script in the **/root/D0280/labs/review-install** folder. Notice from the **TODO** comments in the script that *Step 2.1* is not complete. Complete the script and run it, or follow the manual steps below.

- 2.1. Install Docker on all hosts:

```
[root@master ~]# yum -y install docker
```

Use SSH to repeat the commands on **node1** and **node2** hosts.

- 2.2. Configure Docker on all hosts to use LVM thin pools on the **vdb** disk.

Create the **docker-storage-setup** configuration file:

```
[root@master ~]# echo -e 'DEVS=/dev/vdb\nVG=docker-vg' \
> /etc/sysconfig/docker-storage-setup
```

Run the **docker-storage-setup** command.

```
[root@master ~]# docker-storage-setup
...
INFO: Device node /dev/vdb1 exists.
Physical volume "/dev/vdb1" successfully created.
Volume group "docker-vg" successfully created
Using default stripesize 64.00 KiB.
Rounding up size to full physical extent 24.00 MiB
Logical volume "docker-pool" created.
Logical volume docker-vg/docker-pool changed.
```

Use SSH to copy the configuration file and repeat the commands on **node1** and **node2** hosts.

```
[root@master ~]# scp /etc/sysconfig/docker-storage-setup root@node1:/etc/
sysconfig/
[root@master ~]# scp /etc/sysconfig/docker-storage-setup root@node2:/etc/
sysconfig/
```

2.3. Start and enable the docker service on all hosts:

```
[root@master ~]# systemctl start docker
[root@master ~]# systemctl enable docker
```

Use SSH to repeat the commands in **node1** and **node2** hosts.

3. Install OpenShift using the quick installation method from the **master** host.

Copy the **ocp35-1master-2nodes.cfg.yml** installer configuration file from the **/root/D0280/labs/review-install** folder to the **/root** folder and edit it to include information for the **node2** host. The IP address is 172.25.250.12.

After changing the installer configuration file, edit the **install-ocp.sh** script in the **/root/D0280/labs/review-install** folder. Notice from the **TODO** comments in the script that *Step 3.3* is not complete in the script. Complete the script and run it, or follow the manual steps below.

3.1. Create the quick installer configuration file.

Copy the configuration file from the **/root/D0280/labs/review-install** folder in the **master** host to the **root** user home folder in the **master** host.

```
[root@master ~]# cp ~/D0280/labs/review-install/ocp35-1master-2nodes.cfg.yml ~
```

3.2. Edit the configuration file copy in the **root** user home folder to add information about the **node2** host.

The Ansible variables required for an offline installation are already in the file. The private registry is at **workstation.lab.example.com:5000** and is insecure. These are the required variables:

- **openshift_docker_additional_registries**: points to the private registry at **workstation.lab.example.com** on port 5000.
- **openshift_docker_blocked_registries**: points to the public Docker Hub registry at **docker.io** and also to the Red Hat registry at **registry.access.redhat.com**.
- **openshift_docker_insecure_registries**: points to the private registry at **workstation.lab.example.com:5000** and also to the OpenShift internal registry at 172.30.0.0/16.

More information about these variables is available in the OpenShift Container Platform 3.5 *Installation and Configuration Guide*, section *Configuring Ansible Inventory Files*.

A ready-to-use, already edited, example quick installer configuration file is also available at **ocp35-1master-2nodes.cfg.yml** in the **/root/D0280/solutions/review-install** folder on the **master** host.

Here are the configuration file contents after edits:

```
ansible_callback_facts_yaml: /root/.config/openshift/.ansible/
callback_facts.yaml
ansible_inventory_path: /root/.config/openshift/hosts
ansible_log_path: /tmp/ansible.log
deployment:
  ansible_ssh_user: root
  hosts:
    - connect_to: master.lab.example.com
      hostname: master.lab.example.com
      ip: 172.25.250.10
      public_hostname: master.lab.example.com
      public_ip: 172.25.250.10
      roles:
        - master
        - etcd
        - node
        - storage
    - connect_to: node1.lab.example.com
      hostname: node1.lab.example.com
      ip: 172.25.250.11
      node_labels: '{"region": "infra"}'
      public_hostname: node1.lab.example.com
      public_ip: 172.25.250.11
      roles:
        - node
    - connect_to: node2.lab.example.com
      hostname: node2.lab.example.com
      ip: 172.25.250.12
      node_labels: '{"region": "infra"}'
      public_hostname: node2.lab.example.com
      public_ip: 172.25.250.12
      roles:
        - node
  master_routingconfig_subdomain: clouddaps.lab.example.com
  openshift_master_cluster_hostname: None
  openshift_master_cluster_public_hostname: None
  proxy_exclude_hosts: ''
  proxy_http: ''
  proxy_https: ''
  roles:
    etcd: {}
    master: {
      'openshift_docker_additional_registries' :
      'workstation.lab.example.com:5000',
      'openshift_docker_blocked_registries' : { 'docker.io',
      'registry.access.redhat.com' },
      'openshift_docker_insecure_registries' :
      { 'workstation.lab.example.com:5000', '172.30.0.0/16' }
    }
    node: {
      'openshift_docker_additional_registries' :
      'workstation.lab.example.com:5000',
      'openshift_docker_blocked_registries' : { 'docker.io',
      'registry.access.redhat.com' },
```

```

'openshift_docker_insecure_registries' :
{ 'workstation.lab.example.com:5000', '172.30.0.0/16' }
}
storage: {}
variant: openshift-enterprise
variant_version: '3.5'
version: v2

```

Notice that the value for each variable should be in a single line, without line breaks. Remember, YAML files are sensitive to white space.

Some of the configuration required by this exercise cannot be done using the quick installer in interactive mode. For example, it is not possible to add these variables using the interactive mode.

3.3. Run the OpenShift quick installer in unattended mode.

```
[root@master ~]# atomic-openshift-installer -u -c \
~/ocp35-1master-2nodes.cfg.yml install
```

Wait for the quick installer to finish. It may take 20-30 minutes for the installation to be completed. Watch the Ansible output carefully for error messages. The failed count should be zero for all nodes.

If there are errors, you can run the quick installer again after making fixes. Most times, there is no need to reset your VMs.

3.4. Verify that all nodes are ready:

```
[root@master ~]# oc get nodes -o wide -l region
NAME           STATUS    AGE      EXTERNAL-IP   REGION
master.lab.example.com Ready     5m      <none>       <none>
node1.lab.example.com  Ready     5m      <none>       infra
node2.lab.example.com  Ready     5m      <none>       infra
```

When run interactively, the quick installer configures all nodes with the label **region=infra**. This was not changed in the installation configuration file.

3.5. Verify that the registry and router pods are ready and running, but that the registry console pod is not started:

```
[root@master ~]# oc get pods -o wide
NAME          READY   STATUS    ...   IP            NODE
docker-registry-1-t283s  1/1    Running  ...  10.130.0.4
node1.lab.example.com
registry-console-1-deploy 0/1    Error    ...  10.129.0.3
node2.lab.example.com
router-1-5xq18          1/1    Running  ...  172.25.250.11
node1.lab.example.com
router-1-x94zc          1/1    Running  ...  172.25.250.12
node2.lab.example.com
```

The quick installer configures one router pod for each node when run interactively. This was not changed in the installation configuration file.

The registry and router pods may take a few moments to complete their initialization. Repeat the command until they are in *Running* state. If there are errors in pods other than the registry console, please review your previous steps and run the quick installer again.

It is expected that the registry console pod fails to start because the cluster has no access to the Internet.

4. Modify the image streams and the router console deployment for offline usage.

All commands in this step are in the **post-install.sh** script in the **/root/D0280/labs/review-install** folder. Notice from the **TODO** comments in the script that *Step 4.6* is not complete. Complete the script and run it, or follow the manual steps below.

- 4.1. Reinstate the OpenShift package exclusions on all hosts:

```
[root@master ~]# atomic-openshift-excluder exclude
```

Use SSH to repeat the commands in **node1** and **node2** hosts.

- 4.2. Fix the registry console deployment. The OpenShift quick installer does not change the registry console deployment configuration to use the private registry. Fix the deployment configuration using the **oc set** command:

```
[root@master ~]# oc set image --source=docker dc/registry-console \
    registry-console=workstation.lab.example.com:5000\
    /openshift3/registry-console:3.5
```

Notice that in the previous command there should be no white space before and after the second "\".

Check that the registry console pod is ready and running before moving to the next step:

```
[root@master ~]# oc get pods -o wide -n default
  NAME           READY   STATUS    ...     IP
  NODE
  docker-registry-1-t283s   1/1     Running   0      1h     10.130.0.4
    node1.lab.example.com
  registry-console-1-deploy  0/1     Error     0      1h     10.129.0.3
    node2.lab.example.com
  registry-console-2-04kkp   1/1     Running   0      27s    10.129.0.4
    node2.lab.example.com
  router-1-5xq18            1/1     Running   0      1h     172.25.250.11
    node1.lab.example.com
  router-1-x94zc            1/1     Running   0      1h     172.25.250.12
    node2.lab.example.com
```

The **registry-console-1-deploy** pod with an error status is a result of the first attempt to deploy the registry console. It can be safely ignored.

- 4.3. Delete all predefined image streams:

```
[root@master ~]# oc delete is -n openshift --all
```

4.4. Create a copy of the example image stream definition files:

```
[root@master ~]# cp -r /usr/share/openshift/examples/image-streams \
~/openshift-examples
```

Notice that many of the example image streams are not used in the classroom because the container images they reference are not in the private registry. Keeping these extra image streams is fine because they are not supposed to be used during the review labs.

4.5. Edit the copy to replace the Red Hat registry URL with the classroom registry, and to configure the image streams for insecure registries. Instead of making all edits manually, you can copy the **image-streams-rhel7.json** file in the **/root/D0280/labs/review-install** folder, which already has all required changes.

The annotation **openshift.io/image.insecureRepository** has to be added to all image streams. The **importPolicy** attribute has to be added to all tags with **from.kind** equal to **DockerImage**, inside all image streams.

The following listing shows the edits made to the **php** image stream:

```
...
{
  "kind": "ImageStream",
  "apiVersion": "v1",
  "metadata": {
    "name": "php",
    "annotations": {
      "openshift.io/image.insecureRepository": "true",
      "openshift.io/display-name": "PHP"
    }
  },
  "spec": {
    "tags": [
      ...
      {
        "name": "7.0",
        "annotations": {
          "openshift.io/display-name": "PHP 7.0",
          ...
        },
        "from": {
          "kind": "DockerImage",
          "name": "workstation.lab.example.com:5000/rhscl/php-70-
rhel7:latest"
        },
        "importPolicy": {
          "insecure": true
        }
      }
    ...
  }
}
```

There should be no line breaks in string attribute values.

Do not forget to make the same changes to all other image streams and all tags in the **image-streams-rhel7.json** file.

4.6. Recreate the image streams using the modified JSON file.

```
[root@master ~]# oc create -n openshift -f \
~/openshift-examples/image-streams-rhel7.json
```

OpenShift tries to fetch metadata for each image referenced by each image stream. You can verify that the images available in the private registry were found using the **oc describe** command, for example:

```
[root@master ~]# oc describe is php -n openshift
...
7.0 (latest)
tagged from workstation.lab.example.com:5000/rhscl/php-70-rhel7:latest
  will use insecure HTTPS or HTTP connections
...
* workstation.lab.example.com:5000/rhscl/php-70-
rhel7@sha256:3ff8e5a2d3f0933acadd30898ab5451392eb2189483bcb34c735e55de097a169
  2 minutes ago
...
```

Images not available in the private registry display errors such as:

```
[root@master ~]# oc describe is php -n openshift
...
5.6
tagged from workstation.lab.example.com:5000/rhscl/php-56-rhel7:latest
  will use insecure HTTPS or HTTP connections
...
! error: Import failed (NotFound): dockerimage
"workstation.lab.example.com:5000/rhscl/php-56-rhel7:latest" not found
  2 minutes ago
...
```

5. Make the OpenShift internal registry persistent.

Copy the **registry-pv.yml** resource definition file from the **/root/D0280/labs/review-install** folder to the **/root** folder and edit it to include the correct path for the NFS share created by the quick installer. The path is **/exports/registry**.

After changing the PV definition file, edit the **persistent-registry.sh** script in the **/root/D0280/labs/review-install** folder. Notice from the **TODO** comments in the script that Step 5.2 is not complete. Complete the script and run it, or follow the manual steps below.

5.1. Create a persistent volume YAML file to use the NFS share, created by the quick installer, at **/exports/registry**.

Copy the **registry-pv.yml** file from the **/root/D0280/labs/review-install** folder to the **/root** folder.

Edit the sample to use the NFS share created by the quick installer, and to make sure that only the registry pod can bind to it. The file should read as follows:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: registry-data
spec:
  capacity:
    storage: 15Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /exports/registry
    server: master.lab.example.com
  persistentVolumeReclaimPolicy: Recycle
  claimRef:
    name: registry-claim
    namespace: default
```

5.2. Create the persistent volume using the **oc create** command:

```
[root@master ~]# oc create -f ~/registry-pv.yml
```

5.3. Change the registry deployment to use the persistent volume.

Use the **oc set volume** command to change the deployment configuration and create a persistent volume claim in a single step:

```
[root@master ~]# oc set volume dc/docker-registry -n default \
--add --overwrite --name=registry-storage -t pvc \
--claim-name=registry-claim --claim-size=15Gi --claim-mode=ReadWriteOnce
```

5.4. Check that the persistent volume is bound to the correct claim and that the registry pod is using the persistent volume.

```
[root@master ~]# oc get pv registry-data
NAME      CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
        CLAIM          REASON     AGE
registry-data  15Gi      RWO        Recycle       Bound
              default/registry-claim           1m

[root@master ~]# oc get pvc registry-claim -n default
NAME      STATUS  VOLUME      CAPACITY  ACCESSMODES  AGE
registry-claim  Bound   registry-data  15Gi      RWO         1m

[root@master ~]# oc set volume dc docker-registry -n default
deploymentconfigs/docker-registry
  pvc/registry-claim (allocated 15GiB) as registry-storage
    mounted at /registry
  secret/registry-certificates as registry-certificates
    mounted at /etc/secrets
```

5.5. Wait for a new registry pod to be ready and running.

```
[root@master ~]# oc get pod -n default
NAME          READY   STATUS    ...
docker-registry-2-9744k   1/1     Running   ...
registry-console-2-pb2bz  1/1     Running   ...
router-1-6xj7t           1/1     Running   ...
router-1-gbd6c           1/1     Running   ...
```

6. Configure OpenShift for Apache HTPasswd authentication and create the initial users.

All commands in this step are in the **configure-auth.sh** script in the **/root/D0280/labs/review-install** folder. Notice from the **TODO** comments in the script that *Step 6.6* and *Step 6.7* are not complete. Complete the script and run it, or follow the manual steps below.

6.1. Verify that the *httpd-tools* package is installed on the **master** host:

```
[root@master ~]# rpm -q httpd-tools
```

6.2. Edit the master configuration file **/etc/origin/master/master-config.yaml** on the **master** host. Replace the line:

```
kind: DenyAllPasswordIdentityProvider
```

With the following lines:

```
kind: HTPasswdPasswordIdentityProvider
file: /etc/origin/openshift-passwd
```

Be sure to preserve the indentation. The number of spaces before the **kind** attribute should be the same as before the changes, and the **file** attribute needs the same number of indentation spaces.

6.3. Create the HTPasswd file:

```
[root@master ~]# touch /etc/origin/openshift-passwd
```

6.4. Restart the OpenShift master service on the **master** host:

```
[root@master ~]# systemctl restart atomic-openshift-master
```

6.5. Create an initial regular user with the user name **developer** and a password of **openshift**:

```
[root@master ~]# htpasswd -b /etc/origin/openshift-passwd developer openshift
```

6.6. Create a cluster administrator user with the user name **admin** and a password of **redhat**:

```
[root@master ~]# htpasswd -b /etc/origin/openshift-passwd admin redhat
```

- 6.7. Assign the required privileges to the administrator user:

```
[root@master ~]# oc adm policy add-cluster-role-to-user cluster-admin admin
```

- 6.8. Log in as the regular user and request a list of all pods. The list is empty:

```
[root@master ~]# oc login -u developer -p openshift
...
[root@master ~]# oc get pod --all-namespaces
Error from server (Forbidden): User "developer" cannot list all pods in the
cluster
```

- 6.9. Log in as the cluster administrator user and request a list of all pods. The list includes the router and registry pods:

```
[root@master ~]# oc login -u admin -p redhat
...
[root@master ~]# oc get pod --all-namespaces
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
default       docker-registry-1-t283s           1/1     Running   0          12m
...
```

- 6.10.Return the **root** operating system user in the **master** host to the default OpenShift administrator user:

```
[root@master ~]# oc login -u system:admin
```

7. Install OpenShift client on the workstation.

Open a new terminal window on the **workstation** host and run the following command:

```
[root@workstation ~]$ sudo yum install -y atomic-openshift-clients
...
Installed:
  atomic-openshift-clients.x86_64 0:3.5.5.8-1.git.0.1a85a97.el7
Complete!
```

8. Deploy the test application to verify if the OpenShift router and registry are working as expected.

All commands in this step are in the **test-app.sh** script, in the **/home/student/D0280/labs/review-install** folder. Run the script or follow the manual steps below.

- 8.1. From the **workstation** VM, open a new terminal and log in to the OpenShift master (<https://master.lab.example.com:8443>) as **developer** and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \
```

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

8.2. Create the **test-s2i** project:

```
[student@workstation ~]$ oc new-project test-s2i
```

8.3. Deploy the application from the Git repository:

```
[student@workstation ~]$ oc new-app --name=hello -i php:7.0 \
http://workstation.lab.example.com/php-helloworld
```

8.4. Wait until the application pod is ready and running:

```
[student@workstation ~]$ oc get pods -w
NAME      READY   STATUS    RESTARTS   AGE
hello-1-build  0/1     Completed   0          1m
hello-1-g13jp  1/1     Running    0          59s
```

8.5. Expose the service to external users:

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

8.6. Access the application using the URL from the route:

```
[student@workstation ~]$ curl http://hello-test-s2i.cloudapps.lab.example.com
...
<p>Hello World</p>
...
```



Note

It may take a while for the application to be built from source and deployed on OpenShift. You may get an error like the following when invoking the route URL:

```
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  ...
    <h1>Application is not available</h1>
    <p>The application is currently not serving requests at this
       endpoint. It may not have been started or is still starting.</p>
  ...
```

The error is due to the fact that the application pods are not yet in *Running* state, and hence the route has no registered endpoints. Give the application some more time to deploy, and try invoking the URL again after a while.

8.7. Delete the test application project:

```
[student@workstation ~]$ oc delete project test-s2i
```

This concludes the review lab.

Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab review-install grade
```

If you do not get an overall PASS grade, review your work and run the grading command again.

Lab: Deploy an Application

In this review, you will deploy a multi-container application on OpenShift and fix issues to ensure that it is successfully deployed.

Outcomes

You should be able to:

- Allow users to create new applications within a project. Do not allow users to create new projects. Apply resource quotas to the project to control resource usage.
- Deploy a simple Source-to-Image (S2I) based application to validate quota settings. Troubleshoot and fix issues during application deployment.
- Set up persistent storage for the MySQL database used by the application.
- Deploy a multi-container application on OpenShift using a template. The template should be made available to users from the web console.
- Create a custom docker image and import it into OpenShift. Create a new application using the image stream for the custom image.

Before you begin

You must have completed the comprehensive review lab *Installing OpenShift* in the previous section, and you must have an OpenShift cluster running with a master and two nodes.

Run the following command to verify that your environment is ready for starting this review lab and also to download files required for this lab:

```
[student@workstation ~]$ lab review-deploy setup
```

The **/home/student/D0280/labs/review-deploy** folder on **workstation** provides sample configuration files and scripts that are used in this lab. This folder also provides sample YAML files for resources such as persistent volumes. There is also a copy of these files in the **/root/D0280/labs/review-deploy** folder in the **master** VM.

The Red Hat OpenShift Container Platform 3.5 product documentation is provided in the **workstation** host desktop. Use the documentation as a reference for the commands you need for this lab.

The **TODO List** application consists of three containers:

- A **MySQL** database container that stores data about tasks in the TODO list.
- An Apache httpd web server front-end container (**todooui**) that has the static HTML, CSS, and Javascript assets for the application. The user interface for the TODO List application is written using the Angular.js framework.
- An API back-end container (**todoapi**) based on Node.js exposing a RESTful interface to the front-end container. The **todoapi** container connects to the **MySQL** database container to manage the data in the application.

Instructions

Read these instructions carefully before implementing them. There are two users used in this lab. The cluster administrator user **admin** has a password of **redhat**, and the **developer** user has a password of **openshift**. These users were created in the previous review lab.

1. From the **workstation** VM, log in as the cluster administrator user **admin**. Prevent all regular users from creating new projects in OpenShift. Only a cluster administrator should be allowed to create new projects.
2. Create a new project called **todoapp**. Allow the **developer** user access to this project. Allow the **developer** user to create new applications in this project.
Set quotas on the **todoapp** project by executing the script at **/home/student/D0280/labs/review-deploy/set-quotas.sh** on the **workstation** VM.
3. Login as the **developer** user. Deploy the **php-helloworld** application to validate the quota settings. The source code for the application is available at **http://workstation.lab.example.com/php-helloworld**. The name of the application should be **hello**.
4. Verify that the deployment of the **php-helloworld** application is successful. Deployment will fail.
5. Troubleshoot why deployment did not succeed, and fix the issues. Ensure that the application pod is in the *Running* state. To cancel a failed deployment, you can use the **oc rollout cancel** command. To redeploy an application, use the **oc rollout latest** command. Both of these commands take a deployment configuration as parameters.
6. Delete all the resources in the **php-helloworld** application. Do not delete the **todoapp** project.
7. Provision persistent storage for the MySQL database. Create a new NFS share on the **master** VM at **/var/export/dbvol**. Export this share. Verify that the **node1** and **node2** VMs can read and write from this NFS shared folder.

A sample script to automate the NFS share creation is provided for you in the **/root/D0280/labs/review-deploy/config-nfs.sh** file on the **master** VM.

8. Log in to OpenShift as the **admin** user. Create a new PersistentVolume named **mysql-pv** backed by an NFS share. The persistent volume should be **2GB** in size, with an access mode that allows the volume to be written and read by multiple clients simultaneously. Use the **/var/export/dbvol** NFS share from the **master** VM. A template YAML configuration file for this persistent volume is provided for you in the **/home/student/D0280/labs/review-deploy/todoapi/openshift/mysql-pv.yaml** file in the **workstation** VM.

Verify that the persistent volume is available to be claimed by projects.

9. The **todoapi** back end consists of a MySQL database container, and a Node.js container. To simplify deployment, an OpenShift template that combines these two containers, along with other required resources, is provided in the **/home/student/D0280/labs/review-deploy/todoapi/openshift/nodejs-mysql-template.yaml** file on the **workstation** VM. Briefly review the contents of this file. Do not make any changes to it.

10. As the **admin** user, import the template into OpenShift. Ensure that the template appears in the OpenShift web console under the **JavaScript** category. This allows developers to create new applications from this template using a graphical user interface.
11. The **Dockerfile** for the **todooui** application and its associated build artifacts are available in the **/root/D0280/labs/review-deploy/todooui** folder on the **master** VM. Briefly review the provided **Dockerfile** to understand how the Apache httpd web server container containing the HTML, CSS, and Javascript assets, is built as a docker image.
12. Run the provided **build.sh** script to build the **todooui** Docker image. Tag the resulting image as **workstation.lab.example.com:5000/todoapp/todooui:latest**, and push this image to the classroom private Docker registry on the **workstation** VM.
13. Import the **workstation.lab.example.com:5000/todoapp/todooui:latest** image from the classroom private docker registry into OpenShift. Ensure that an image stream called **todooui** is created in the **todoapp** namespace.
14. Log in to the OpenShift web console as the **developer** user with a password of **openshift**, and verify that a single project called **todoapp** is visible.
15. Create a new application using the imported **nodejs-mysql-persistent** template using the values in the following table:

Node.js + MySQL (Persistent) Template Parameters

Name	Value
Name	todoapi
Git Repository URL	http://workstation.lab.example.com/todoapi
Application Hostname	todoapi.cloudapps.lab.example.com
MySQL Username	todoapp
MySQL Password	todoapp
Database name	todoappdb
Database Administrator Password	redhat

Add a label called **app=todoapi** to the application.



Note

Default values are populated from the **nodejs-mysql-persistent** template created by the cluster administrator. Apart from the options listed in the previous table, leave all other options on this page at default values.

16. Verify that the application is built and deployed. You must see two pods, one each for the **todoapi** application and the MySQL database.
17. Sample data for the **TODO List** application is provided for you in the **/home/student/D0280/labs/review-deploy/todoapi/sql/db.sql** file on the **workstation** VM.

Use OpenShift port-forwarding to execute the SQL statements and import data into the **todoappdb** database in MySQL.

Forward local port **3306** on workstation to port **3306** in the MySQL database pod. For your convenience, a utility script to import the data into the MySQL database is available at **/home/student/D0280/labs/review-deploy/todoapi/sql/import-data.sh**.

Review the script and execute it on **workstation** after you set up port-forwarding to the MySQL pod.

18. Verify that the **todoapi** back-end API is deployed and running without any errors. Verify that the application is working correctly by using the **curl** command to access the URL **http://todoapi.cloudapps.lab.example.com/todo/api/host**, which prints the pod name on which the application is running.
19. Verify that the **todoapi** application fetches data from MySQL by using the **curl** command to access the URL **http://todoapi.cloudapps.lab.example.com/todo/api/items**, which prints the list of TODO items in JSON format.
20. On the **workstation** VM, log in as the **developer** user with a password of **openshift**. Verify that there is a single project called **todoapp** after you log in.
21. Create a new application named **todoui** using the **todoui** image stream created by the cluster administrator.
22. Verify that the deployment is successful. You should see a new pod in the OpenShift web console in the **Overview** page for the project.
23. Create a new route for the **todoui** service with a *hostname* of **todo.cloudapps.lab.example.com**.
24. Using the **curl** command, verify that the URL **http://todo.cloudapps.lab.example.com** returns the HTML content of the home page for the **todoui** front-end application.
25. Finally, use a browser on the **workstation** VM to navigate to the home page of the **TODO List** application at **http://todo.cloudapps.lab.example.com**. You should see the **TODO List** application with a list of items stored in the database. Add, Edit, and Delete tasks to ensure that the application is working correctly.

Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab review-deploy grade
```

If you do not get an overall PASS grade, review your work and run the grading command again.

Solution

In this review, you will deploy a multi-container application on OpenShift and fix issues to ensure that it is successfully deployed.

Outcomes

You should be able to:

- Allow users to create new applications within a project. Do not allow users to create new projects. Apply resource quotas to the project to control resource usage.
- Deploy a simple Source-to-Image (S2I) based application to validate quota settings. Troubleshoot and fix issues during application deployment.
- Set up persistent storage for the MySQL database used by the application.
- Deploy a multi-container application on OpenShift using a template. The template should be made available to users from the web console.
- Create a custom docker image and import it into OpenShift. Create a new application using the image stream for the custom image.

Before you begin

You must have completed the comprehensive review lab *Installing OpenShift* in the previous section, and you must have an OpenShift cluster running with a master and two nodes.

Run the following command to verify that your environment is ready for starting this review lab and also to download files required for this lab:

```
[student@workstation ~]$ lab review-deploy setup
```

The **/home/student/D0280/labs/review-deploy** folder on **workstation** provides sample configuration files and scripts that are used in this lab. This folder also provides sample YAML files for resources such as persistent volumes. There is also a copy of these files in the **/root/D0280/labs/review-deploy** folder in the **master** VM.

The Red Hat OpenShift Container Platform 3.5 product documentation is provided in the **workstation** host desktop. Use the documentation as a reference for the commands you need for this lab.

The **TODO List** application consists of three containers:

- A **MySQL** database container that stores data about tasks in the TODO list.
- An Apache httpd web server front-end container (**todooui**) that has the static HTML, CSS, and Javascript assets for the application. The user interface for the TODO List application is written using the Angular.js framework.
- An API back-end container (**todoapi**) based on Node.js exposing a RESTful interface to the front-end container. The **todoapi** container connects to the **MySQL** database container to manage the data in the application.

Instructions

Read these instructions carefully before implementing them. There are two users used in this lab. The cluster administrator user **admin** has a password of **redhat**, and the **developer** user has a password of **openshift**. These users were created in the previous review lab.

1. From the **workstation** VM, log in as the cluster administrator user **admin**. Prevent all regular users from creating new projects in OpenShift. Only a cluster administrator should be allowed to create new projects.

2. Create a new project called **todoapp**. Allow the **developer** user access to this project. Allow the **developer** user to create new applications in this project.

Set quotas on the **todoapp** project by executing the script at **/home/student/D0280/labs/review-deploy/set-quotas.sh** on the **workstation** VM.

3. Login as the **developer** user. Deploy the **php-helloworld** application to validate the quota settings. The source code for the application is available at **http://workstation.lab.example.com/php-helloworld**. The name of the application should be **hello**.
4. Verify that the deployment of the **php-helloworld** application is successful. Deployment will fail.
5. Troubleshoot why deployment did not succeed, and fix the issues. Ensure that the application pod is in the *Running* state. To cancel a failed deployment, you can use the **oc rollout cancel** command. To redeploy an application, use the **oc rollout latest** command. Both of these commands take a deployment configuration as parameters.
6. Delete all the resources in the **php-helloworld** application. Do not delete the **todoapp** project.
7. Provision persistent storage for the MySQL database. Create a new NFS share on the **master** VM at **/var/export/dbvol**. Export this share. Verify that the **node1** and **node2** VMs can read and write from this NFS shared folder.

A sample script to automate the NFS share creation is provided for you in the **/root/D0280/labs/review-deploy/config-nfs.sh** file on the **master** VM.

8. Log in to OpenShift as the **admin** user. Create a new PersistentVolume named **mysql-pv** backed by an NFS share. The persistent volume should be **2GB** in size, with an access mode that allows the volume to be written and read by multiple clients simultaneously. Use the **/var/export/dbvol** NFS share from the **master** VM. A template YAML configuration file for this persistent volume is provided for you in the **/home/student/D0280/labs/review-deploy/todoapi/openshift/mysql-pv.yaml** file in the **workstation** VM.

Verify that the persistent volume is available to be claimed by projects.

9. The **todoapi** back end consists of a MySQL database container, and a Node.js container. To simplify deployment, an OpenShift template that combines these two containers, along with other required resources, is provided in the **/home/student/D0280/labs/review-deploy/todoapi/openshift/nodejs-mysql-template.yaml** file on the **workstation** VM. Briefly review the contents of this file. Do not make any changes to it.

10. As the **admin** user, import the template into OpenShift. Ensure that the template appears in the OpenShift web console under the **JavaScript** category. This allows developers to create new applications from this template using a graphical user interface.
11. The **Dockerfile** for the **todooui** application and its associated build artifacts are available in the **/root/D0280/labs/review-deploy/todooui** folder on the **master** VM. Briefly review the provided **Dockerfile** to understand how the Apache httpd web server container containing the HTML, CSS, and Javascript assets, is built as a docker image.
12. Run the provided **build.sh** script to build the **todooui** Docker image. Tag the resulting image as **workstation.lab.example.com:5000/todoapp/todooui:latest**, and push this image to the classroom private Docker registry on the **workstation** VM.
13. Import the **workstation.lab.example.com:5000/todoapp/todooui:latest** image from the classroom private docker registry into OpenShift. Ensure that an image stream called **todooui** is created in the **todoapp** namespace.
14. Log in to the OpenShift web console as the **developer** user with a password of **openshift**, and verify that a single project called **todoapp** is visible.
15. Create a new application using the imported **nodejs-mysql-persistent** template using the values in the following table:

Node.js + MySQL (Persistent) Template Parameters

Name	Value
Name	todoapi
Git Repository URL	http://workstation.lab.example.com/todoapi
Application Hostname	todoapi.cloudapps.lab.example.com
MySQL Username	todoapp
MySQL Password	todoapp
Database name	todoappdb
Database Administrator Password	redhat

Add a label called **app=todoapi** to the application.



Note

Default values are populated from the **nodejs-mysql-persistent** template created by the cluster administrator. Apart from the options listed in the previous table, leave all other options on this page at default values.

16. Verify that the application is built and deployed. You must see two pods, one each for the **todoapi** application and the MySQL database.
17. Sample data for the **TODO List** application is provided for you in the **/home/student/D0280/labs/review-deploy/todoapi/sql/db.sql** file on the **workstation** VM.

Use OpenShift port-forwarding to execute the SQL statements and import data into the **todoappdb** database in MySQL.

Forward local port **3306** on workstation to port **3306** in the MySQL database pod. For your convenience, a utility script to import the data into the MySQL database is available at **/home/student/D0280/labs/review-deploy/todoapi/sql/import-data.sh**.

Review the script and execute it on **workstation** after you set up port-forwarding to the MySQL pod.

18. Verify that the **todoapi** back-end API is deployed and running without any errors. Verify that the application is working correctly by using the **curl** command to access the URL **http://todoapi.cloudapps.lab.example.com/todo/api/host**, which prints the pod name on which the application is running.
19. Verify that the **todoapi** application fetches data from MySQL by using the **curl** command to access the URL **http://todoapi.cloudapps.lab.example.com/todo/api/items**, which prints the list of TODO items in JSON format.
20. On the **workstation** VM, log in as the **developer** user with a password of **openshift**. Verify that there is a single project called **todoapp** after you log in.
21. Create a new application named **todoui** using the **todoui** image stream created by the cluster administrator.
22. Verify that the deployment is successful. You should see a new pod in the OpenShift web console in the **Overview** page for the project.
23. Create a new route for the **todoui** service with a *hostname* of **todo.cloudapps.lab.example.com**.
24. Using the **curl** command, verify that the URL **http://todo.cloudapps.lab.example.com** returns the HTML content of the home page for the **todoui** front-end application.
25. Finally, use a browser on the **workstation** VM to navigate to the home page of the **TODO List** application at **http://todo.cloudapps.lab.example.com**. You should see the **TODO List** application with a list of items stored in the database. Add, Edit, and Delete tasks to ensure that the application is working correctly.

Steps

1. From **workstation**, log in as the cluster administrator. Prevent regular users from creating new projects in OpenShift.
 - 1.1. Open a terminal window on the **workstation** VM, and log in as the **admin** user with a password of **redhat**.

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

- 1.2. Restrict project creation to only cluster administrator roles. Regular users cannot create new projects.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \

```

```
self-provisioner system:authenticated system:authenticated:oauth
cluster role "self-provisioner" removed: ["system:authenticated"
"system:authenticated:oauth"]
```

2. Create a new project. Allow the **developer** user to access this project, and set resource quotas on the project.

2.1. Create a new project called **todoapp**:

```
[student@workstation ~]$ oc new-project todoapp
Now using project "todoapp" on server "https://master.lab.example.com:8443".
...
```

Allow **developer** to access the **todoapp** project. Ensure that **todoapp** is your current project:

```
[student@workstation ~]$ oc policy add-role-to-user edit developer
role "edit" added: "developer"
```

- 2.2. Set quotas on the **todoapp** project by running the script at **/home/student/D0280/labs/review-deploy/set-quotas.sh** on the **workstation** VM:

```
[student@workstation ~]$ bash \
/home/student/D0280/labs/review-deploy/set-quotas.sh
Setting quotas on the todoapp project...
Already on project "todoapp" on server "https://master.lab.example.com:8443".
resourcequota "todoapp-quota" created
```

3. Deploy the Source-to-Image (S2I) based **php-helloworld** application to validate quota settings on the project. The source code is available in the Git repository at **http://workstation.lab.example.com/php-helloworld**.

- 3.1. From the **workstation** VM, open a new terminal and log in to OpenShift (**https://master.lab.example.com:8443**) as **developer** with a password of **openshift**, and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
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): y

Login successful.

You have one project on this server: "todoapp"

Using project "todoapp".
```

- 3.2. Create a new application named **hello**:

```
[student@workstation ~]$ oc new-app --name=hello \
http://workstation.lab.example.com/php-helloworld
```

3.3. Verify that the application was successfully built:

```
[student@workstation ~]$ oc logs -f bc/hello
Cloning "http://workstation.lab.example.com/php-helloworld" ...
Commit: f2515f8da088a566bf79b33296dbadcee6dc588c (Initial commit)
Author: root <root@workstation.lab.example.com>
Date: Fri Jul 28 16:59:33 2017 +0530
---> Installing application source...
Pushing image 172.30.238.48:5000/todoapp/hello:latest ...
Pushed 0/5 layers, 7% complete
Pushed 1/5 layers, 20% complete
Pushed 2/5 layers, 44% complete
Pushed 3/5 layers, 77% complete
Pushed 4/5 layers, 97% complete
Pushed 5/5 layers, 100% complete
Push successful
```

3.4. Verify that the application builds successfully, but that the pod is not created.

NAME	READY	STATUS	RESTARTS	AGE
hello-1-build	0/1	Completed	0	3m

4. Troubleshoot why the **php-helloworld** application is not being deployed, and fix the issues.

4.1. Check the event logs for the project:

```
[student@workstation ~]$ oc get events
...
Warning FailedRetry {deployments-controller} ... couldn't create deployer
pod for todoapp/hello-1:
pods "hello-1-deploy" is forbidden:
exceeded quota: todoapp-quota, requested: pods=1, used: pods=1, limited: pods=1
```

The event log indicates that the number of pods requested by the application is more than the number of pods allowed for the project. Check the state of the deployment config:

NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY
hello	1	1	0	config,image(hello:latest)

Observe that the **CURRENT** column shows a value of zero.

4.2. Check the quota settings for the project.:

```
[student@workstation ~]$ oc describe quota
Name: todoapp-quota
Namespace: todoapp
Resource Used Hard
-----
pods     0      1
```

The output of the command shows that the hard limit on the pod count is set to 1. This is a very low number and is not enough to build, deploy, and run the application. Recall that you executed a script in *Step 2.2* to set quotas for the project. Review the contents of the **/home/student/D0280/labs/review-deploy/set-quotas.sh** file on the **workstation** VM, and observe that the cluster administrator has inadvertently made a typo in the command that creates the quota for the **todoapp** project.

```
...
oc create quota todoapp-quota --hard=pods=1
...
```

The pod count was incorrectly typed as 1 instead of 10.

- 4.3. Correct the quota settings by executing the following commands as the cluster administrator from the **workstation** VM. Ensure that **todoapp** is your current active project. The commands are also available as a script at **/home/student/D0280/labs/review-deploy/fix-quotas.sh**.

```
[student@workstation ~]$ oc login -u admin -p redhat
[student@workstation ~]$ oc project todoapp
[student@workstation ~]$ oc patch resourcequota/todoapp-quota \
--patch '{"spec":{"hard":{"pods":"10"}}}'
"todoapp-quota" patched
```



Note

You can also use the **oc edit resourcequota todoapp-quota** command to change the quota.

- 4.4. Switch back to the OpenShift **developer** user on **workstation**. Verify that the quota for the pod count has been incremented to 10. Cancel the existing deployment for the application:

```
[student@workstation ~]$ oc login -u developer -p openshift
[student@workstation ~]$ oc describe quota
Name: todoapp-quota
Namespace: todoapp
Resource Used Hard
-----
pods      0    10
```

```
[student@workstation ~]$ oc rollout cancel dc/hello
deploymentconfig "hello" cancelling
```

- 4.5. Redeploy the application:

```
[student@workstation ~]$ oc rollout latest dc/hello
deploymentconfig "hello" rolled out
```

4.6. Check the status of the pods in the project:

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
hello-1-build  0/1     Completed  0          1h
hello-2-lxqkr  1/1     Running   0          8s
```

The deployment of the application should now succeed and the pod must be in *Running* state.

4.7. Delete the sample S2I application:

```
[student@workstation ~]$ oc delete all -l app=hello
buildconfig "hello" deleted
imagestream "hello" deleted
deploymentconfig "hello" deleted
service "hello" deleted
```

5. You can now proceed with deploying the **TODO List** application. As a first step, provision persistent storage for the MySQL database used in the application.

5.1. Create a new NFS share on the **master** VM.

Open a new terminal on **workstation** and log in to the **master** VM as **root** using SSH. Run the **config-nfs.sh** script located in the **/root/D0280/labs/review-deploy** folder on the **master** VM.

```
[root@master ~]# bash /root/D0280/labs/review-deploy/config-nfs.sh
Export directory /var/export/dbvol created.
Export list for master.lab.example.com:
(exports/logging-es-ops *
(exports/logging-es *
(exports/metrics *
(exports/registry *
/var/export/dbvol *
```

- 5.2. Verify that both **node1** and **node2** hosts can access the NFS-exported volume from the OpenShift **master** VM.

Open a new terminal window on **workstation** and log in to the **node1** host as the user **root** to confirm that the **node1** host can access the NFS share on the **master** VM.

```
[root@node1 ~]# mount -t nfs master.lab.example.com:/var/export/dbvol /mnt
```

- 5.3. On **node1**, verify that the file system has the correct permissions from **node1**:

```
[root@node1 ~]# ls -la /mnt ; mount | grep /mnt
total 0
drwx----- 2 nfsnobody nfsnobody 6 Jul 18 02:07 .
dr-xr-xr-x 17 root      root      224 Jun  5 08:54 ..
master.lab.example.com:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.0,rsize=262144,wszie=262144,namlen=255,hard,
proto=tcp,port=0,timeo=600,retrans=2,sec=sys,
```

```
clientaddr=172.25.250.11,local_lock=none,addr=172.25.250.10)
```

5.4. Unmount the NFS share:

```
[root@node1 ~]# umount /mnt
```

5.5. Similarly, log in to the **node2** VM as the **root** user to confirm that the **node2** VM can access the NFS share on **master**:

```
[root@node2 ~]# mount -t nfs master.lab.example.com:/var/export/dbvol /mnt
```

5.6. On **node2**, verify that the file system has correct permissions from **node2**:

```
[root@node2 ~]# ls -la /mnt ; mount | grep /mnt
total 0
drwx----- 2 nfsnobody nfsnobody 6 Jul 18 02:07 .
dr-xr-xr-x. 17 root      root    224 Jun  5 08:54 ..
master.lab.example.com:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.0,rsize=262144,wszie=262144,namlen=255,hard,
proto=tcp,port=0,timeo=600,retrans=2,sec=sys,
clientaddr=172.25.250.12,local_lock=none,addr=172.25.250.10)
```

5.7. Unmount the NFS share:

```
[root@node2 ~]# umount /mnt
```

5.8. Create a new PersistentVolume for the MySQL database.

On the **workstation** VM, edit the YAML resource file provided in the **/home/student/D0280/labs/review-deploy/todoapi/openshift/mysql-pv.yaml** file. Change the attributes to match the values provided in the lab instructions to create a persistent volume. The final contents should be:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /var/export/dbvol
    server: master.lab.example.com
```

5.9. Log in to OpenShift as the cluster administrator user **admin**. Create the persistent volume using the provided YAML resource definition file:

```
[student@workstation ~]$ oc login -u admin -p redhat
[student@workstation ~]$ oc create -f \
  /home/student/D0280/labs/review-deploy/todoapi/openshift/mysql-pv.yaml
```

```
persistentvolume "mysql-pv" created
```

- 5.10. Verify that the persistent volume is available to be claimed by projects:

```
[student@workstation ~]$ oc get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS     ...   AGE
mysql-pv   2Gi        RWX          Retain          Available   ...   8s
...
```

6. Import the **nodejs-mysql-persistent** template into OpenShift.

An OpenShift template combining the **todoapi** and **mysql** pods, and their associated services, is provided to you in the **/home/student/D0280/labs/review-deploy/todoapi/openshift/nodejs-mysql-template.yaml** file on the **workstation** VM.

- 6.1. Briefly review the content of the template file. Do not edit or make any changes to this file.
- 6.2. Import the template into the **openshift** namespace so that the template is available to users for creating applications:

```
[student@workstation ~]$ oc create -n openshift -f \
/home/student/D0280/labs/review-deploy/todoapi/openshift/nodejs-mysql-
template.yaml
template "nodejs-mysql-persistent" created
```

7. Build the **todoui** docker image on the **master** VM. Tag it and push it to the workstation private registry.

The **Dockerfile** for the **todoui** application and its associated build artifacts are available in the **/root/D0280/labs/review-deploy/todoui/** folder on the **master** VM. Briefly review the provided **Dockerfile**.

- 7.1. Build the **todoui** docker image by running the provided **build.sh** script in the **/root/D0280/labs/review-deploy/todoui/** folder on the **master** VM:

```
[root@master ~]# cd /root/D0280/labs/review-deploy/todoui/
[root@master todoui]# ./build.sh
Sending build context to Docker daemon 647.2 kB
Step 1 : FROM rhel7:7.3
Trying to pull repository workstation.lab.example.com:5000/rhel7 ...
sha256:fcf297c67951b871f42c6e9abcf09854a0e49519bda9156301b52900e457716c: Pulling
from workstation.lab.example.com:5000/rhel7
...
Installed:
    httpd.x86_64 0:2.4.6-45.el7_3.4           less.x86_64 0:458-9.el7
    openssl-devel.x86_64 1:1.0.1e-60.el7_3.1    unzip.x86_64 0:6.0-16.el7
    wget.x86_64 0:1.14-13.el7                  which.x86_64 0:2.20-7.el7
...
Removing intermediate container 7239fc24b2e1
Step 11 : CMD /usr/sbin/apachectl -DFOREGROUND
--> Running in 4f9bf9fe954e
--> 531b63448385
Removing intermediate container 4f9bf9fe954e
Successfully built 531b63448385
```

- 7.2. Tag the newly built **todoui** docker image:

```
[root@master ~]# docker tag todoapp/todoui:latest \
workstation.lab.example.com:5000/todoapp/todoui:latest
```

- 7.3. Push the newly tagged **todoui** docker image to the classroom private docker registry on the **workstation** VM:

```
[root@master ~]# docker push \
workstation.lab.example.com:5000/todoapp/todoui:latest
The push refers to a repository [workstation.lab.example.com:5000/todoapp/
todoui]
3fe8032789c3: Pushed
0aecdcea6e69: Pushed
bf2a7dfdfcc: Pushed
6c833695ddf6: Pushed
6adc7bfe790b: Pushed
f483edd7a42b: Mounted from rhel7
f7b626558f10: Mounted from rhel7
latest: digest: sha256:a147349...
[root@master todoui]# cd ~
```

8. Import the **todoui** docker image into OpenShift and verify that the image streams are created.

- 8.1. Import the **todoui** docker image from the private registry on **workstation**:

```
[root@master ~]# oc import-image todoui \
--from=workstation.lab.example.com:5000/todoapp/todoui \
--confirm --insecure=true -n todoapp
The import completed successfully.

Name: todoui
Namespace: todoapp
Created: Less than a second ago
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2017-07-27T13:25:10Z
Docker Pull Spec: 172.30.238.48:5000/todoapp/todoui
Unique Images: 1
Tags: 1

latest
tagged from workstation.lab.example.com:5000/todoapp/todoui
will use insecure HTTPS or HTTP connections

* workstation.lab.example.com:5000/todoapp/todoui@sha256:a147349e...
Less than a second ago
```

- 8.2. Verify that the **todoui** image stream has been created:

```
[root@master ~]# oc get is -n todoapp | grep todoui
todoui 172.30.238.48:5000/todoapp/todoui latest About a minute ago
```

- 8.3. Describe the **todoui** image stream and verify that the latest **todoui** docker image is referenced:

```
[root@master ~]# oc describe is todoui -n todoapp
Name: todoui
Namespace: todoapp
Created: 2 minutes ago
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2017-07-27T13:25:10Z
Docker Pull Spec: 172.30.238.48:5000/todoapp/todoui
Unique Images: 1
Tags: 1

latest
tagged from workstation.lab.example.com:5000/todoapp/todoui
will use insecure HTTPS or HTTP connections

* workstation.lab.example.com:5000/todoapp/todoui@sha256:a147349e...
2 minutes ago
```

9. Create a new application using the **nodejs-mysql-persistent** template from the OpenShift web console.
 - 9.1. From the **workstation** VM, launch a web browser and log in to the OpenShift web console (<https://master.lab.example.com:8443>) as the **developer** user with a password of **openshift**, and verify that a single project called **todoapp** is visible.

The screenshot shows the OpenShift web console's 'Projects' page. At the top, it says 'OPENSHIFT CONTAINER PLATFORM'. Below that, there's a search bar and a 'Sort by' dropdown set to 'Display Name'. A message at the bottom says 'A cluster admin can create a project for you by running the command `oadm new-project <projectname> --admin=user1`'. The main list contains one item: 'todoapp', which was 'created by system:admin 2 hours ago'. There are icons for editing and deleting the project.

- 9.2. Click **todoapp**, and then click **Add to Project**. On the **Browse Catalog** page, click **JavaScript** to get a list of Javascript-based templates.
- 9.3. Select the **Node.js + MySQL (Persistent)** template, and create a new application called **todoapi** using the values provided in the table below:

Node.js + MySQL (Persistent) Template Parameters

Name	Value
Name	todoapi
Git Repository URL	http://workstation.lab.example.com/todoapi
Application Hostname	todoapi.cloudapps.lab.example.com
MySQL Username	todoapp
MySQL Password	todoapp
Database name	todoappdb
Database Administrator Password	redhat

In the **Labels** section below the form, add a new label below the **Each label is applied to each created resource** text. Enter **app** in the **Name** field, and **todoapi** in the **Value** field.



Note

Default values are populated from the **nodejs-mysql-persistent** template created by the cluster administrator. Apart from the options listed in the table above, leave all other options in this page at default values.

Click **Create** to create the application.

- 9.4. A confirmation page confirms that the **todoapi** application was created successfully, as shown below:

OPENSHIFT CONTAINER PLATFORM

todoapp » Add to Project » nodejs-mysql-persistent » Next Steps

Application created. [Continue to overview.](#)

i The following service(s) have been created in your project: todoapi, mysql.

Click **Continue to overview** to navigate to the **Overview** page for the project. Confirm that there are two pods, **todoapi** and **mysql**, running successfully. It may take some time for the **todoapi** application to be built from source and deployed.

TODOAPI

Build todoapi, #1 ✓ Complete. an hour ago [View Log](#)

todoapi

Deployment Config todoapi – an hour ago #1

CONTAINER: NODEJS-MYSQL-PERSISTENT

Image: todoapp/todoapi

Ports: 8080/TCP

mysql

Deployment Config mysql – an hour ago #1

CONTAINER: MYSQL

Image: rhsc/mariadb-101-rhel7

Ports: 3306/TCP

10. Import the SQL data for the **TODO List** application into the MySQL database. Test the **todoapi** back-end services API and ensure that it is working correctly.

- 10.1. From the **workstation** VM, open a new terminal and log in to OpenShift (<https://master.lab.example.com:8443>) as **developer** with a password of **openshift**, and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p openshift \
https://master.lab.example.com:8443
...
Using project "todoapp".
```

- 10.2. You must have access to a single project called **todoapp**:

```
[student@workstation ~]$ oc projects
You have one project on this server: "todoapp".

Using project "todoapp" on server "https://master.lab.example.com:8443".
```

- 10.3. Sample data for the TODO List application is provided for you in the **/home/student/D0280/labs/review-deploy/todoapi/sql/db.sql** file on the **workstation** VM. Use the **oc port-forward** command to connect directly to the MySQL database pod. Get the name of the database pod from the **oc get pods** command:

```
[student@workstation ~]$ oc port-forward mysql-1-crxnw 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
```

This command does not return to the shell prompt. Leave this terminal window as is, and open a new terminal window on the **workstation** VM to execute the next step.

- 10.4. Import the SQL data in **db.sql** file into the **todoappdb** database.

Execute the script available at **/home/student/D0280/labs/review-deploy/todoapi/sql/import-data.sh** to import data into the database. If the script executes successfully, it prints a list of records imported into the database:

```
[student@workstation ~]$ bash \
/home/student/D0280/labs/review-deploy/todoapi/sql/import-data.sh
Importing data into database...

The following records have been imported into the database:

+---+-----+---+
| id | description | done |
+---+-----+---+
| 1 | Pick up newspaper |   |
| 2 | Buy groceries | x |
+---+-----+---+

DONE!
```



Note

The **done** column in the query result displays binary Unicode text on the screen because the field is declared as a **BIT** type (indicating true or false values) in the schema. The 'X' shown in the output above is just for convenience and indicates whether a particular task is complete.

- 10.5.Verify that the application is working correctly by using the **curl** command to access the URL **http://todoapi.cloudapps.lab.example.com/todo/api/host**, which prints the pod name and the pod IP address where the application is running.

```
[student@workstation ~]$ curl -s \
  http://todoapi.cloudapps.lab.example.com/todo/api/host \
  | python -m json.tool
{
  "hostname": "todoapi-2-gzb54",
  "ip": "10.130.0.20"
}
```

- 10.6.Verify that the application fetches data from the MySQL database correctly by using the **curl** command to access the URL **http://todoapi.cloudapps.lab.example.com/todo/api/items**, which prints the list of tasks stored in the database.

```
[student@workstation ~]$ curl -s \
  http://todoapi.cloudapps.lab.example.com/todo/api/items \
  | python -m json.tool
{
  "currentPage": 1,
  "list": [
    {
      "description": "Pick up newspaper",
      "done": false,
      "id": 1
    },
    {
      "description": "Buy groceries",
      "done": true,
      "id": 2
    }
  ],
  "pageSize": 10,
  "sortDirections": "asc",
  "sortFields": "id",
  "totalResults": 2
}
```

- 10.7.In the terminal window running the **oc port-forward** command, stop port-forwarding by pressing **Ctrl+C**.

11. Create a new application using the **todoui** image stream.

- 11.1. Create a new application called **todoui**, based on the **todoui** image stream:

```
[student@workstation ~]$ oc new-app --name=todoui -i todoui
--> Found image 531b634 (14 minutes old) in image stream "todoapp/todoui" under
tag "latest" for "todoui"

Red Hat Enterprise Linux 7
-----
Tags: base rhel7

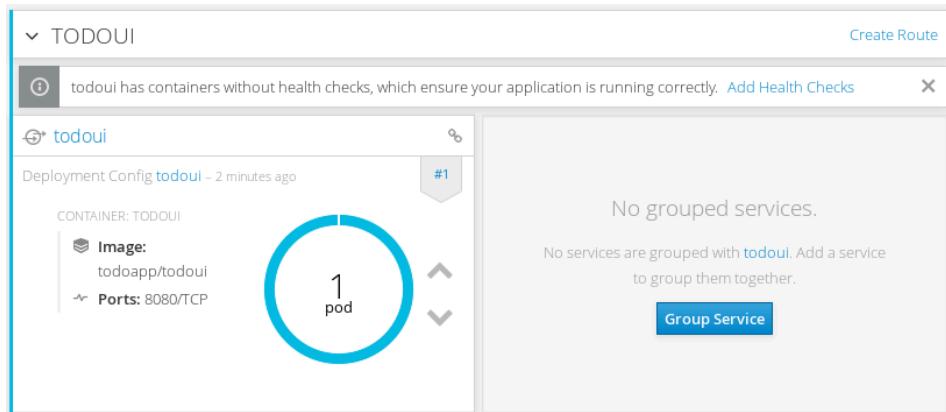
* This image will be deployed in deployment config "todoui"
* Port 8080/tcp will be load balanced by service "todoui"
* Other containers can access this service through the hostname "todoui"

--> Creating resources ...
deploymentconfig "todoui" created
service "todoui" created
--> Success
Run 'oc status' to view your app.
```

11.2. Review the status of the pods in the project:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql-1-crxnw 1/1     Running   0          1h
todoapi-1-build 0/1     Completed  0          1h
todoapi-1-h37hc 1/1     Running   0          1h
todoui-1-dmw7j 1/1     Running   0          9s
```

In the OpenShift web console, you should now see a new **todoui** pod in the **Overview** page for the project:



12. Create a route for the **todoui** application.

12.1. Expose the service to external users:

```
[student@workstation ~]$ oc expose svc todoui \
--hostname=todo.cloudapps.lab.example.com
route "todoui" exposed
```

12.2. Access the **todoui** application using the URL from the route:

```
[student@workstation ~]$ curl http://todo.cloudapps.lab.example.com
<!DOCTYPE html>

<html ng-app="items">
<head>
    <title>To Do List</title>
    ...
    <!-- Form buttons. The 'Save' button is only enabled when the form is valid. -->
    <div class="buttons">
        <button type="button" class="btn btn-primary" ng-click="clearForm()">Clear</button>
        <button type="submit" class="btn btn-primary" ng-disabled="itemForm.$invalid">Save</button>
    </div>
</form>
</div>
</div>

</body>
</html>
```

13. Finally, use a browser on the **workstation** VM to navigate to the home page of the **TODO List** application at **http://todo.cloudapps.lab.example.com**. You should see the **TODO List** application with a list of items stored in the database. Add, Edit, and Delete tasks to ensure that the application is working correctly.

To Do List Application

To Do List

Id	Description	Done	
1	Pick up news...	false	
2	Buy groceries	true	

[First](#) [Previous](#) [1](#) [Next](#) [Last](#)

Add Task

Description:

Add Description.

Completed:

[Clear](#) [Save](#)



Note

Do not delete the first two tasks preloaded from the database. These are used by the grading script to verify that the lab was completed successfully.

Evaluation

Run the following command to grade your work:

```
[student@workstation ~]$ lab review-deploy grade
```

If you do not get an overall PASS grade, review your work and run the grading command again.

