

Join the explorers, builders, and individuals who boldly offer new solutions to old problems. For open source, innovation is only possible because of the people behind it.

RED HAT® TRAINING+ CERTIFICATION

STUDENT WORKBOOK (ROLE)

OCP 3.9 DO285

**CONTAINERS, KUBERNETES, AND RED HAT
OPENSHIFT ADMINISTRATION I**

Edition 1



CONTAINERS, KUBERNETES, AND RED HAT OPENSHIFT ADMINISTRATION I



OCP 3.9 DO285
Containers, Kubernetes, and Red Hat OpenShift
Administration I
Edition 1 20181002
Publication date 20181002

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

Copyright © 2018 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are
Copyright © 2018 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	ix
Introduction	xi
Containers, Kubernetes, and Red Hat OpenShift Administration I	xii
Orientation to the Classroom Environment	xiii
Internationalization	xvi
1. Describing Container Technology	1
Describing Container Architecture	2
Quiz: Describing Container Architecture	4
Describing Docker Architecture	8
Quiz: Describing Docker Architecture	11
Describing the OpenShift Container Platform Architecture	13
Quiz: Describing Kubernetes and OpenShift	21
Summary	25
2. Creating Containerized Services	27
Provisioning a Database Server	28
Guided Exercise: Creating a MySQL Database Instance	37
Lab: Creating Containerized Services	40
Summary	45
3. Managing Containers	47
Managing the Life Cycle of Containers	48
Guided Exercise: Managing a MySQL Container	58
Attaching Docker Persistent Storage	62
Guided Exercise: Persisting a MySQL Database	65
Accessing Docker Networks	68
Guided Exercise: Loading the Database	71
Lab: Managing Containers	74
Summary	82
4. Managing Container Images	83
Accessing Registries	84
Quiz: Working With Registries	90
Manipulating Container Images	94
Guided Exercise: Creating a Custom Apache Container Image	99
Lab: Managing Images	105
Summary	112
5. Creating Custom Container Images	113
Design Considerations for Custom Container Images	114
Quiz: Approaches to Container Image Design	118
Building Custom Container Images with Dockerfile	120
Guided Exercise: Creating a Basic Apache Container Image	129
Lab: Creating Custom Container Images	133
Summary	140
6. Deploying Multi-Container Applications	141
Considerations for Multi-Container Applications	142
Quiz: Multi-Container Application Considerations	147
Deploying a Multi-Container Application with Docker	149
Guided Exercise: Deploying the Web Application and MySQL Containers	153
Lab: Deploying Multi-Container Applications	160
7. Installing OpenShift Container Platform	167
Preparing Servers for Installation	168
Workshop: Preparing for Installation	174
Installing Red Hat OpenShift Container Platform	181
Guided Exercise: Installing Red Hat OpenShift Container Platform	190

Executing Postinstallation Tasks	200
Workshop: Completing Postinstallation Tasks	204
Summary	209
8. Describing and Exploring OpenShift Networking Concepts	211
Describing OpenShift's Implementation of Software-Defined Networking	212
Guided Exercise: Exploring Software-Defined Networking	218
Creating Routes	224
Guided Exercise: Creating a Route	230
Lab: Exploring OpenShift Networking Concepts	235
Summary	242
9. Deploying Containerized Applications on OpenShift	245
Creating Kubernetes Resources	246
Guided Exercise: Deploying a Database Server on OpenShift	254
Creating Applications with Source-to-Image	259
Guided Exercise: Creating a Containerized Application with Source-to-Image	268
Creating Routes	275
Guided Exercise: Exposing a Service as a Route	278
Creating Applications with the OpenShift Web Console	281
Guided Exercise: Creating an Application with the Web Console	283
Lab: Deploying Containerized Applications on OpenShift	290
Summary	293
10. Deploying a Multi-Container Application on OpenShift	295
Deploying a Multi-Container Application on OpenShift	296
Guided Exercise: Creating an Application with a Template	303
Lab: Deploying Multi-Container Applications	309
Summary	317
11. Executing Commands	319
Configuring Resources with the CLI	320
Guided Exercise: Managing an OpenShift Instance Using <code>oc</code>	327
Executing Troubleshooting Commands	335
Guided Exercise: Troubleshooting Common Problems	341
Lab: Executing Commands	349
Summary	359
12. Controlling Access to OpenShift Resources	361
Securing Access to OpenShift Resources	362
Guided Exercise: Managing Projects and Accounts	371
Managing Sensitive Information with Secrets	379
Guided Exercise: Protecting a Database Password	383
Managing Security Policies	387
Quiz: Managing Security Policies	392
Lab: Controlling Access to OpenShift Resources	394
Summary	404
13. Allocating Persistent Storage	405
Provisioning Persistent Storage	406
Guided Exercise: Implementing Persistent Database Storage	412
Describing Persistence for the Internal Registry	420
Quiz: Describing Persistence for the Internal Registry	423
Lab: Allocating Persistent Storage	425
Summary	436
14. Managing Application Deployments	437
Scaling an Application	438
Guided Exercise: Scaling An Application	442
Controlling Pod Scheduling	449

Guided Exercise: Controlling Pod Scheduling	454
Managing Images, Image Streams, and Templates	459
Guided Exercise: Managing Image Streams	464
Lab: Managing Application Deployments	469
Summary	477
15. Installing and Configuring the Metrics Subsystem	479
Describing the Architecture of the Metrics Subsystem	480
Quiz: Describing the Architecture of the metrics Subsystem	485
Installing the Metrics Subsystem	489
Guided Exercise: Installing the Metrics Subsystem	494
Summary	504
16. Managing and Monitoring OpenShift Container Platform	505
Limiting Resource Usage	506
Workshop: Limiting Resource Usage	513
Upgrading the OpenShift Container Platform	522
Quiz: Upgrading OpenShift	529
Monitoring Applications with Probes	531
Guided Exercise: Monitoring Applications with Probes	536
Monitoring Resources with the Web Console	543
Guided Exercise: Exploring Metrics with the Web Console	550
Lab: Managing and Monitoring OpenShift	558
Summary	567
17. Comprehensive Review of Introduction to Containers, Kubernetes, and Red Hat OpenShift	569
Comprehensive Review	570
Lab: Containerizing and Deploying a Software Application	573
Comprehensive Review	583
Lab: Deploying an Application	586

DOCUMENT CONVENTIONS



REFERENCES

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



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.

INTRODUCTION

CONTAINERS, KUBERNETES, AND RED HAT OPENSHIFT ADMINISTRATION I

One of the key tenants of the DevOps movement is continuous integration and continuous deployment. Containers have become a key technology for the configuration and deployment of applications and microservices. Kubernetes is a container orchestration platform that provides foundational services in Red Hat OpenShift Container Platform. Introduction to Containers, Kubernetes, and Red Hat OpenShift Administration I (DO285) introduces the student to building and managing docker containers for deployment on a Kubernetes cluster. This course helps students build core knowledge and skills in managing containers through hands-on experience with Docker, Kubernetes, and the Red Hat OpenShift Container Platform. OpenShift is a containerized application platform that allows enterprises to manage container deployments and scale their applications using Kubernetes. Students will also build core administration skills through the installation, configuration, and management of an OpenShift cluster.

COURSE OBJECTIVES

- Containerize simple software applications and services.
- Deploy applications and services with docker, Kubernetes, and Red Hat OpenShift.
- Test containerized applications, and troubleshoot issues with deployment.

AUDIENCE

- Developers who wish to containerize software applications.
- Administrators who are new to container technology and container orchestration.
- Architects who are considering using container technologies in software architectures.
- System administrators
- System architects
- Architects and developers who want to install and configure OpenShift

PREREQUISITES

- Ability to use a Linux terminal session and issue operating system commands.
- RHCSA certification or equivalent knowledge.
- Experience with web application architectures and their corresponding technologies.

ORIENTATION TO THE CLASSROOM ENVIRONMENT

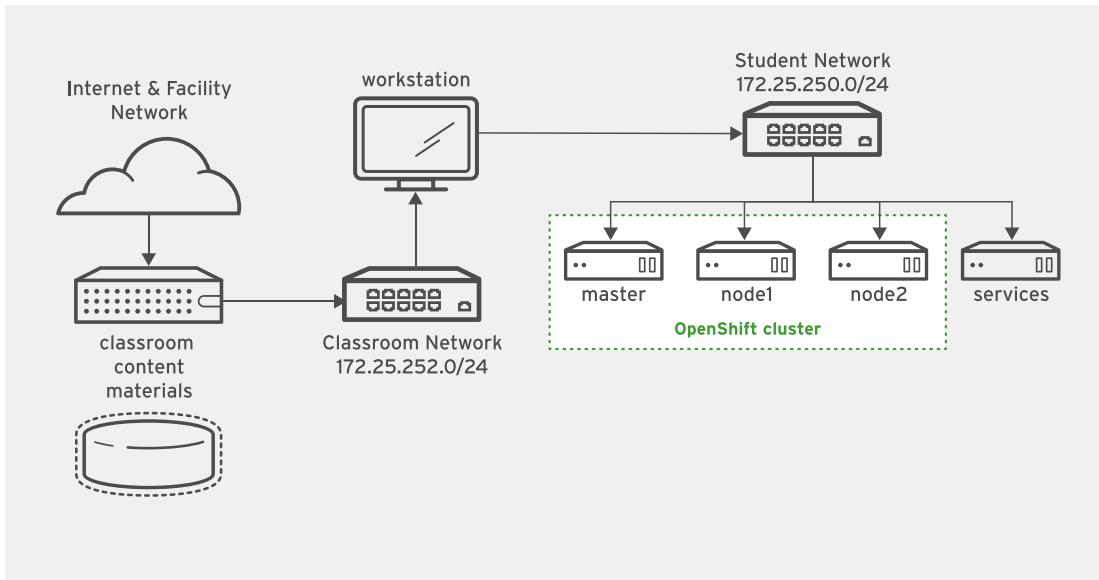


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is **workstation**. Four other machines will also be used by students for these activities. These are **master**, **node1**, **node2**, and **services**. All four of these systems are in the **lab.example.com** DNS domain.

All student computer systems have a standard user account, **student**, which has the password **student**. The **root** password on all student systems is **redhat**.

A third VM, **master**, with the host name **master.lab.example.com**, hosts the OpenShift Container Platform cluster.

The fourth and fifth VMs, **node1** and **node2**, with the host names **node1.lab.example.com**, and **node2.lab.example.com** respectively are nodes part of the OpenShift Container Platform cluster.

- 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 Nexus server with a repository of modules for Node.js development.

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

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	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 cluster server
node1.lab.example.com	172.25.250.11	OpenShift Container Platform cluster node
node2.lab.example.com	172.25.250.12	OpenShift Container Platform cluster node
services.lab.example.com , registry.lab.example.com	172.25.250.13	Classroom private registry

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

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

LAB EXERCISE SETUP AND GRADING

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

- The **setup** subcommand is used at the beginning of an exercise. It verifies that the systems are ready for the activity, possibly making some configuration changes to them.
- The **grade** subcommand is executed at the end of an exercise. It provides external confirmation that the activity's requested steps were performed correctly.
- The **reset** subcommand can be used to return the VM to its original state and restart. This is usually followed by the **lab setup** command.

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

Controlling the stations

The state of each virtual machine in the classroom is displayed on the page found under the Online Lab tab.

Machine States

MACHINE STATE	DESCRIPTION
STARTING	The machine is in the process of booting.
STARTED	The machine is running and available (or, when booting, soon will be).
STOPPING	The machine is in the process of shutting down.
STOPPED	The machine is completely shut down. Upon starting, the machine will boot into the same state as when it was shut down (the disk will have been preserved).
PUBLISHING	The initial creation of the virtual machine is being performed.
WAITING_TO_START	The virtual machine is waiting for other virtual machines to start.

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

Classroom/Machine Actions

BUTTON OR ACTION	DESCRIPTION
PROVISION LAB	Create the ROL classroom. This creates all of the virtual machines needed for the classroom and starts them. This will take several minutes to complete.
DELETE LAB	Delete the ROL classroom. This destroys all virtual machines in the classroom. Caution: Any work generated on the disks is lost.
START LAB	Start all machines in the classroom.
SHUTDOWN LAB	Stop all machines in the classroom.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. Students can log in directly to the machine and run commands. In most cases, students should log in to the workstation.lab.example.com machine and use ssh to connect to the other virtual machines.
ACTION → Start	Start (“power on”) the machine.
ACTION → Shutdown	Gracefully shut down the machine, preserving the contents of its disk.
ACTION → Power Off	Forcefully shut down the machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION → Reset	Forcefully shut down the machine and reset the disk to its initial state. Caution: Any work generated on the disk is lost.

At the start of a lab exercise, if an instruction to reset **workstation** appears, click ACTION → Reset for the **workstation** virtual machine. Likewise, if an instruction to reset **infrastructure** appears, click ACTION → Reset for the **infrastructure** virtual machine.

At the start of a lab exercise, if an instruction to reset all virtual machines appears, click **DELETE LAB** to delete the classroom environment. After it has been deleted, click **PROVISION LAB** to create a fresh version of the classroom systems.

The Autostop Timer

The Red Hat Online Learning enrollment entitles students to a certain amount of computer time. To help conserve time, the ROL classroom has an associated countdown timer, which will shut down the classroom environment when the timer expires.

To adjust the timer, click **MODIFY**. A **New Autostop Time** dialog opens. Set the autostop time in hours and minutes (note: there is a ten hour maximum time). Click **ADJUST TIME** to adjust the time accordingly.

INTERNATIONALIZATION

LANGUAGE SUPPORT

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

PER-USER LANGUAGE SELECTION

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

Language Settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the Region & Language application. Run the command **gnome-control-center region**, or from the top bar, select (User) → Settings. In the window that opens, select Region & Language. Click the Language box and select the preferred language from the list that appears. This also updates 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, and 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
```

Introduction

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to determine 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 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.

When 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 also find 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, the **root** user can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it displays 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 language packs, run **yum langavailable**. To view the list of language packs currently installed on the system, run **yum langlist**. To add an additional language pack 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

LANGUAGE	\$LANG VALUE
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

DESCRIBING CONTAINER TECHNOLOGY

GOAL

Describe how software can run in containers orchestrated by Red Hat OpenShift Container Platform.

OBJECTIVES

- Describe the architecture of Linux containers.
 - Describe how containers are implemented using Docker.
 - Describe the architecture of a Kubernetes cluster running on the Red Hat OpenShift Container Platform.
-
- Describing Container Architecture (and Quiz)
 - Describing Docker Architecture (and Quiz)
 - Describing Kubernetes and OpenShift Architecture (and Quiz)

SECTIONS

DESCRIBING CONTAINER ARCHITECTURE

OBJECTIVES

After completing this section, students should be able to:

- Describe the architecture of Linux containers.
- Describe the characteristics of software applications.
- List the approaches of using a container

CONTAINERIZED APPLICATIONS

Software applications are typically deployed as a single set of libraries and configuration files to a runtime environment. They are traditionally deployed to an operating system with a set of services running, such as a database server or an HTTP server, but they can also be deployed to any environment that can provide the same services, such as a virtual machine or a physical host.

The major drawback to using a software application is that it is entangled with the runtime environment and any updates or patches applied to the base OS might break the application. For example, an OS update might include multiple dependency updates, including libraries (that is, operating system libraries shared by multiple programming languages) that might affect the running application with incompatible updates.

Moreover, if another application is sharing the same host OS and the same set of libraries, as described in the Figure 1.1, there might be a risk of preventing the application from properly running it if an update that fixes the first application libraries affects the second application.

Therefore, for a company developing typical software applications, any maintenance on the running environment might require a full set of tests to guarantee that any OS update does not affect the application as well.

Depending on the complexity of an application, the regression verification might not be an easy task and might require a major project. Furthermore, any update normally requires a full application stop. Normally, this implies an environment with high-availability features enabled to minimize the impact of any downtime, and increases the complexity of the deployment process. The maintenance might become cumbersome, and any deployment or update might become a complex process. The Figure 1.1 describes the difference between applications running as containers and applications running on the host operating system.

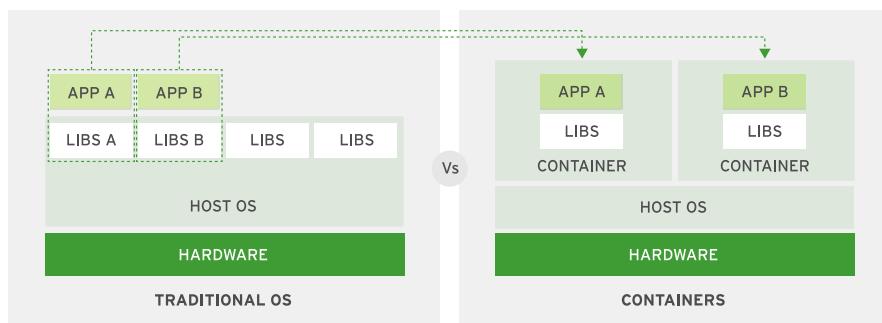


Figure 1.1: Container versus operating system differences

Alternatively, a system administrator can work with *containers*, which are a kind of isolated partition 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. They also isolate the libraries and the runtime environment (such as CPU and storage) used by an application to minimize the impact of any OS update to the host OS, as described in Figure 1.1.

The use of containers helps not only with the efficiency, elasticity, and reusability of the hosted applications, but also with portability of the platform and applications. There are many container providers available, such as Rocket, Drawbridge, and LXC, but one of the major providers is Docker.

Some of the major advantages of containers are listed below.

Low hardware footprint

Uses OS internal features to create an isolated environment where resources are managed using OS facilities such as namespaces and cgroups. This approach minimizes the amount of CPU and memory overhead compared to a virtual machine hypervisor. Running an application in a VM is a way to create isolation from the running environment, but it requires a heavy layer of services to support the same low hardware footprint isolation provided by containers.

Environment isolation

Works in a closed environment where changes made to the host OS or other applications do not affect the container. Because the libraries needed by a container are self-contained, the application can run without disruption. For example, each application can exist in its own container with its own set of libraries. An update made to one container does not affect other containers, which might not work with the update.

Quick deployment

Deploys any container quickly because there is no need to install the entire underlying operating system. Normally, to support the isolation, a new OS installation is required on a physical host or VM, and any simple update might require a full OS restart. A container only requires a restart without stopping any services on the host OS.

Multiple environment deployment

In a traditional deployment scenario using a single host, any environment differences might potentially break the application. Using containers, however, the differences and incompatibilities are mitigated because the same container image is used.

Reusability

The same container can be reused by multiple applications without the need to set up a full OS. A database container can be used to create a set of tables for a software application, and it can be quickly destroyed and recreated without the need to run a set of housekeeping tasks. Additionally, the same database container can be used by the production environment to deploy an application.

Often, a software application with all its dependent services (databases, messaging, file systems) are made to run in a single container. However, container characteristics and agility requirements might make this approach challenging or ill-advised. In these instances, a multi-container deployment may be more suitable. Additionally, be aware that some application actions may not be suited for a containerized environment. For example, applications accessing low-level hardware information, such as memory, file-systems and devices may fail due to container constraints.

Finally, containers boost the microservices development approach because they provide a lightweight and reliable environment to create and run services that can be deployed to a production or development environment without the complexity of a multiple machine environment.

► QUIZ

DESCRIBING CONTAINER ARCHITECTURE

Choose the correct answers to the following questions:

► 1. **Which two options are examples of software applications that might run in a container? (Select two.)**

- a. A database-driven Python application accessing services such as a MySQL database, a file transfer protocol (FTP) server, and a web server on a single physical host.
- b. A Java Enterprise Edition application, with an Oracle database, and a message broker running on a single VM.
- c. An I/O monitoring tool responsible for analyzing the traffic and block data transfer.
- d. A memory dump application tool capable of taking snapshots from all the memory CPU caches for debugging purposes.

► 2. **Which of the two following use cases are better suited for containers? (Select two.)**

- a. A software provider needs to distribute software that can be reused by other companies in a fast and error-free way.
- b. A company is deploying applications on a physical host and would like to improve its performance by using containers.
- c. A data center is looking for alternatives to shared hosting for database applications to minimize the amount of hardware processing needed.
- d. A financial company is implementing a CPU-intensive risk analysis tool on their own containers to minimize the number of processors needed.

► 3. **A company is migrating their PHP and Python applications running on the same host to a new architecture. Due to internal policies, both are using a set of custom-made shared libraries from the OS, but the latest update applied to them as a result of a Python development team request broke the PHP application. Which two architectures would provide the best support for both applications? (Select two.)**

- a. Deploy each application to different VMs and apply the custom-made shared libraries individually to each VM host.
- b. Deploy each application to different containers and apply the custom-made shared libraries individually to each container.
- c. Deploy each application to different VMs and apply the custom-made shared libraries to all VM hosts.
- d. Deploy each application to different containers and apply the custom-made shared libraries to all containers.

► **4. Which three kinds of applications can be packaged as containers for immediate consumption? (Select three.)**

- a. A virtual machine hypervisor.
- b. A blog software, such as WordPress.
- c. A database.
- d. A local file system recovery tool.
- e. A web server.

► SOLUTION

DESCRIBING CONTAINER ARCHITECTURE

Choose the correct answers to the following questions:

► 1. **Which two options are examples of software applications that might run in a container? (Select two.)**

- a. A database-driven Python application accessing services such as a MySQL database, a file transfer protocol (FTP) server, and a web server on a single physical host.
- b. A Java Enterprise Edition application, with an Oracle database, and a message broker running on a single VM.
- c. An I/O monitoring tool responsible for analyzing the traffic and block data transfer.
- d. A memory dump application tool capable of taking snapshots from all the memory CPU caches for debugging purposes.

► 2. **Which of the two following use cases are better suited for containers? (Select two.)**

- a. A software provider needs to distribute software that can be reused by other companies in a fast and error-free way.
- b. A company is deploying applications on a physical host and would like to improve its performance by using containers.
- c. A data center is looking for alternatives to shared hosting for database applications to minimize the amount of hardware processing needed.
- d. A financial company is implementing a CPU-intensive risk analysis tool on their own containers to minimize the number of processors needed.

► 3. **A company is migrating their PHP and Python applications running on the same host to a new architecture. Due to internal policies, both are using a set of custom-made shared libraries from the OS, but the latest update applied to them as a result of a Python development team request broke the PHP application. Which two architectures would provide the best support for both applications? (Select two.)**

- a. Deploy each application to different VMs and apply the custom-made shared libraries individually to each VM host.
- b. Deploy each application to different containers and apply the custom-made shared libraries individually to each container.
- c. Deploy each application to different VMs and apply the custom-made shared libraries to all VM hosts.
- d. Deploy each application to different containers and apply the custom-made shared libraries to all containers.

► **4. Which three kinds of applications can be packaged as containers for immediate consumption? (Select three.)**

- a. A virtual machine hypervisor.
- b. A blog software, such as WordPress.
- c. A database.
- d. A local file system recovery tool.
- e. A web server.

DESCRIBING DOCKER ARCHITECTURE

OBJECTIVES

After completing this section, students should be able to:

- Describe how containers are implemented using Docker.
- List the key components of the Docker architecture.
- Describe the architecture behind the Docker command-line interface (CLI).

DESCRIBING THE DOCKER ARCHITECTURE

Docker is one of the container implementations available for deployment and supported by companies such as Red Hat in their Red Hat Enterprise Linux Atomic Host platform. Docker Hub provides a large set of containers developed by the community.

Docker uses a client-server architecture, as described below:

Client

The command-line tool (**docker**) is responsible for communicating with a server using a RESTful API to request operations.

Server

This service, which runs as a **daemon** on an operating system, does the heavy lifting of building, running, and downloading container images.

The daemon can run either on the same system as the **docker** client or remotely.



NOTE

For this course, both the client and the server will be running on the **workstation** machine.



NOTE

In a Red Hat Enterprise Linux environment, the daemon is represented by a **systemd** unit called **docker.service**.

DOCKER CORE ELEMENTS

Docker depends on three major elements:

Images

Images are read-only templates that contain a runtime environment that includes application libraries and applications. Images are used to create containers. Images can be created, updated, or downloaded for immediate consumption.

Registries

Registries store images for public or private use. The well-known public registry is Docker Hub, and it stores multiple images developed by the community, but private registries can be

created to support internal image development under a company's discretion. This course runs on a private registry in a virtual machine where all the required images are stored for faster consumption.

Containers

Containers are segregated user-space environments for running applications isolated from other applications sharing the same host OS.



REFERENCES

Docker Hub website
<https://hub.docker.com>



NOTE

In a RHEL environment, the registry is represented by a **systemd** unit called **docker-registry.service**.

CONTAINERS AND THE LINUX KERNEL

Containers created by Docker, from Docker-formatted container images, are isolated from each other by several standard features of the Linux kernel. These include:

Namespaces

The kernel can place specific system resources that are normally visible to all processes into a namespace. Inside a namespace, only processes that are members of that namespace can see those resources. Resources that can be placed into a namespace include network interfaces, the process ID list, mount points, IPC resources, and the system's own hostname information. As an example, two processes in two different mounted namespaces have different views of what the mounted root file system is. Each container is added to a specific set of namespaces, which are only used by that container.

Control groups (cgroups)

Control groups partition sets of processes and their children into groups in order to manage and limit the resources they consume. Control groups place restrictions on the amount of system resources the processes belonging to a specific container might use. This keeps one container from using too many resources on the container host.

SELinux

SELinux is a mandatory access control system that is used to protect containers from each other and to protect the container host from its own running containers. Standard SELinux type enforcement is used to protect the host system from running containers. Container processes run as a confined SELinux type that has limited access to host system resources. In addition, sVirt uses SELinux *Multi-Category Security (MCS)* to protect containers from each other. Each container's processes are placed in a unique category to isolate them from each other.

DOCKER CONTAINER IMAGES

Each image in Docker consists of a series of layers that are combined into what is seen by the containerized applications a single virtual file system. Docker images are immutable; any extra layer added over the preexisting layers overrides their contents without changing them directly. Therefore, any change made to a container image is destroyed unless a new image is generated using the existing extra layer. The UnionFS file system provides containers with a single file system view of the multiple image layers.



REFERENCES

UnionFS wiki page

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

In summary, there are two approaches to create a new image:

- *Using a running container:* An immutable image is used to start a new container instance and any changes or updates needed by this container are made to a read/ write extra layer. **Docker** commands can be issued to store that read/write layer over the existing image to generate a new image. Due to its simplicity, this approach is the easiest way to create images, but it is not a recommended approach because the image size might become large due to unnecessary files, such as temporary files and logs.
- *Using a Dockerfile:* Alternatively, container images can be built from a base image using a set of steps called *instructions*. Each instruction creates a new layer on the image that is used to build the final container image. This is the suggested approach to building images, because it controls which files are added to each layer.

► QUIZ

DESCRIBING DOCKER ARCHITECTURE

Choose the correct answers to the following questions:

- ▶ **1. Which of the following three tasks are managed by a component other than the Docker client? (Select three.)**
 - a. Downloading container image files from a registry.
 - b. Requesting a container image deployment from a server.
 - c. Searching for images from a registry.
 - d. Building a container image.

- ▶ **2. Which of the following best describes a container image?**
 - a. A virtual machine image from which a container will be created.
 - b. A container blueprint from which a container will be created.
 - c. A runtime environment where an application will run.
 - d. The container's index file used by a registry.

- ▶ **3. Which two kernel components does Docker use to create and manage the runtime environment for any container? (Choose two.)**
 - a. Namespaces
 - b. iSCSI
 - c. Control groups
 - d. LVM
 - e. NUMA support

- ▶ **4. An existing image of a WordPress blog was updated on a developer's machine to include new homemade extensions. Which is the best approach to create a new image with those updates provided by the developer? (Select one.)**
 - a. The updates made to the developer's custom WordPress should be copied and transferred to the production WordPress, and all the patches should be made within the image.
 - b. The updates made to the developer's custom WordPress should be assembled as a new image using a Dockerfile to rebuild the container image.
 - c. A **diff** should be executed on the production and the developer's WordPress image, and all the binary differences should be applied to the production image.
 - d. Copy the updated files from the developer's image to the **/tmp** directory from the production environment and request an image update.

► SOLUTION

DESCRIBING DOCKER ARCHITECTURE

Choose the correct answers to the following questions:

- ▶ **1. Which of the following three tasks are managed by a component other than the Docker client? (Select three.)**
 - a. Downloading container image files from a registry.
 - b. Requesting a container image deployment from a server.
 - c. Searching for images from a registry.
 - d. Building a container image.

- ▶ **2. Which of the following best describes a container image?**
 - a. A virtual machine image from which a container will be created.
 - b. A container blueprint from which a container will be created.
 - c. A runtime environment where an application will run.
 - d. The container's index file used by a registry.

- ▶ **3. Which two kernel components does Docker use to create and manage the runtime environment for any container? (Choose two.)**
 - a. Namespaces
 - b. iSCSI
 - c. Control groups
 - d. LVM
 - e. NUMA support

- ▶ **4. An existing image of a WordPress blog was updated on a developer's machine to include new homemade extensions. Which is the best approach to create a new image with those updates provided by the developer? (Select one.)**
 - a. The updates made to the developer's custom WordPress should be copied and transferred to the production WordPress, and all the patches should be made within the image.
 - b. The updates made to the developer's custom WordPress should be assembled as a new image using a Dockerfile to rebuild the container image.
 - c. A `diff` should be executed on the production and the developer's WordPress image, and all the binary differences should be applied to the production image.
 - d. Copy the updated files from the developer's image to the `/tmp` directory from the production environment and request an image update.

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:

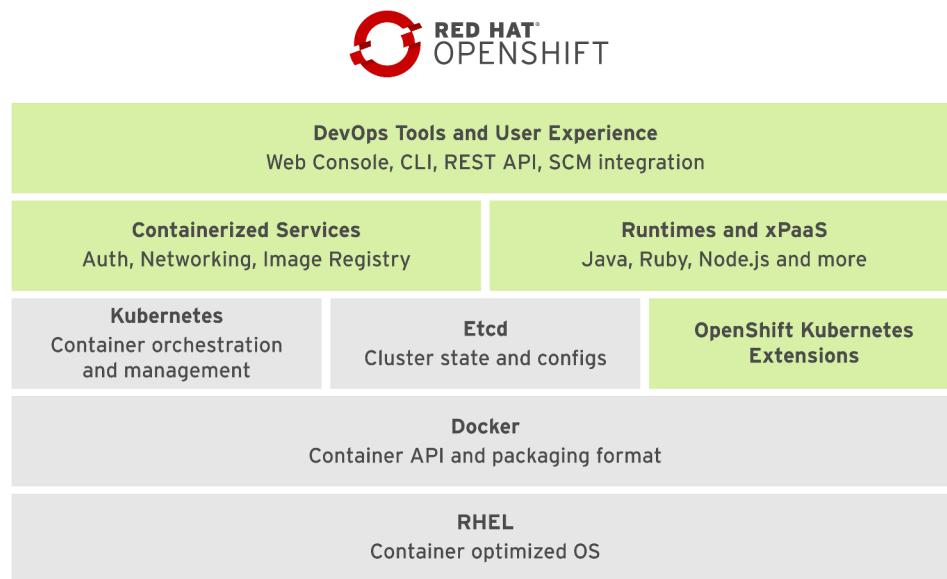


Figure 1.2: OpenShift architecture

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 multicontainer 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.

A Kubernetes 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 server and a node, but those roles are usually segregated for increased stability.

Kubernetes Terminology

TERM	DEFINITION
Master	A server that manages the workload and communications in a Kubernetes cluster.
Node	A server that host applications in a Kubernetes cluster.
Label	A key/value pair that can be assigned to any Kubernetes resource. A selector uses labels to filter eligible resources for scheduling and other operations.

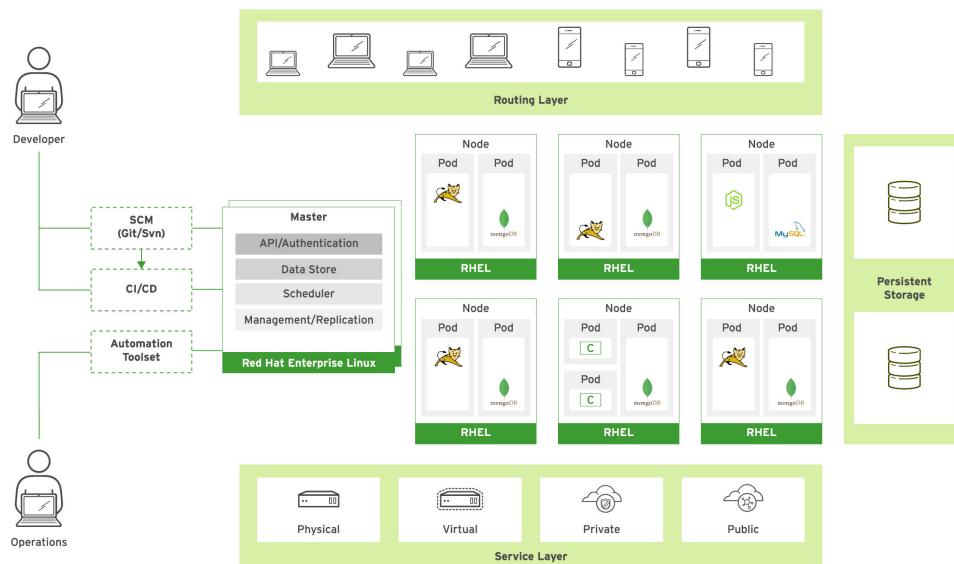


Figure 1.3: OpenShift and Kubernetes architecture

An OpenShift cluster is a Kubernetes cluster which can be managed the same way, but using the management tools provided OpenShift, such as the command-line interface or the web console. This allows for more productive workflows and makes common tasks much easier.

KUBERNETES RESOURCE TYPES

Kubernetes has five main resource types that can be created and configured using a YAML or a JSON file, or using OpenShift management tools:

Pods

Represent a collection of containers that share resources, such as IP addresses and persistent storage volumes. It is the basic unit of work for Kubernetes.

Services

Define a single IP/port combination that provides access to a pool of pods. By default, services connect clients to pods in a round-robin fashion.

Replication Controllers

A framework for defining pods that are meant to be horizontally scaled. A replication controller includes a pod definition that is to be replicated, and the pods created from it can be scheduled to different nodes.

Persistent Volumes (PV)

Provision persistent networked storage to pods that can be mounted inside a container to store data.

Persistent Volume Claims (PVC)

Represent a request for storage by a pod to Kubernetes.



NOTE

For the purpose of this course, the PVs are provisioned on local storage, not on networked storage. This is a valid approach for development purposes, but it is not a recommended approach for a production environment.

Although Kubernetes pods can be created standalone, they are usually created by high-level resources such as replication controllers.

OPENSHIFT RESOURCE TYPES

The main resource types added by OpenShift Container Platform to Kubernetes are as follows:

Deployment Configurations (dc)

Represent a set of pods created from the same container image, managing workflows such as rolling updates. A **dc** also provides a basic but extensible continuous delivery workflow.

Build Configurations (bc)

Used by the OpenShift *Source-to-Image (S2I)* feature to build a container image from application source code stored in a Git server. A **bc** works together with a **dc** to provide a basic but extensible continuous integration and continuous delivery workflows.

Routes

Represent a DNS host name recognized by the OpenShift router as an ingress point for applications and microservices.

Although Kubernetes replication controllers can be created standalone in OpenShift, they are usually created by higher-level resources such as deployment controllers.

NETWORKING

Each container deployed by a **docker** daemon has an IP address assigned from an internal network that is accessible only from the host running the container. Because of the container's ephemeral nature, IP addresses are constantly assigned and released.

Kubernetes provides a *software-defined network (SDN)* that spawns the internal container networks from multiple nodes and allows containers from any pod, inside any host, to access pods from other hosts. Access to the SDN only works from inside the same Kubernetes cluster.

Containers inside Kubernetes pods are not supposed to connect to each other's dynamic IP address directly. It is recommended that they connect to the more stable IP addresses assigned to services, and thus benefit from scalability and fault tolerance.

External access to containers, without OpenShift, requires redirecting a port from the router to the host, then to the internal container IP address, or from the node to a service IP address in the SDN. A Kubernetes service can specify a **NodePort** attribute that is a network port redirected by all the cluster nodes to the SDN. Unfortunately, none of these approaches scale well.

OpenShift makes external access to containers both scalable and simpler, by defining *route* resources. HTTP and TLS accesses to a route are forwarded to service addresses inside the Kubernetes SDN. The only requirement is that the desired DNS host names are mapped to the OCP routers nodes' external IP addresses.

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.

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>

**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:

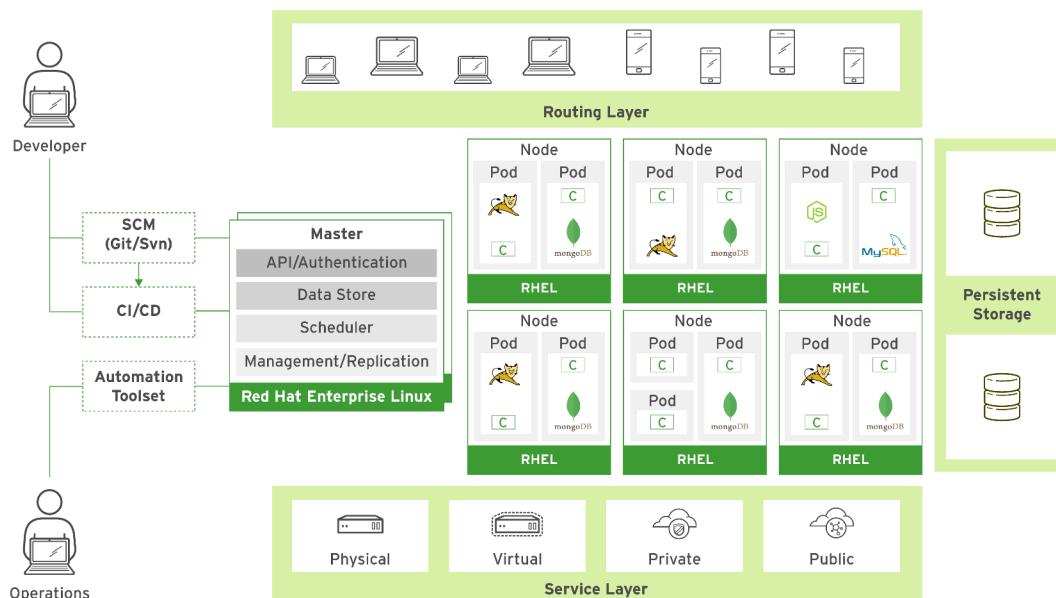


Figure 1.4: OpenShift Container Platform master and nodes

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.

OPENSIFT 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 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 OPENSIFT 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.

OPENSFIFT 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 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

DESCRIBING KUBERNETES AND OPENSIFT

Choose the correct answers to the following questions:

► 1. Which three sentences are correct regarding Kubernetes architecture? (Choose three.)

- a. Kubernetes nodes can be managed without a master.
- b. Kubernetes masters manage pod scaling.
- c. Kubernetes masters schedule pods to specific nodes.
- d. A pod is a set of containers managed by Kubernetes as a single unit.
- e. Kubernetes tools cannot be used to manage resources in an OpenShift cluster.

► 2. Which two sentences are correct regarding Kubernetes and OpenShift resource types? (Choose two.)

- a. A pod is responsible for provisioning its own persistent storage.
- b. All pods generated from the same replication controller have to run in the same node.
- c. A route is responsible for providing IP addresses for external access to pods.
- d. A replication controller is responsible for monitoring and maintaining the number of pods for a particular application.
- e. Containers created from Kubernetes pods cannot be managed using standard Docker tools.

► 3. Which two statements are true regarding Kubernetes and OpenShift networking? (Select two.)

- a. A Kubernetes service can provide an IP address to access a set of pods.
- b. Kubernetes is responsible for providing internal IP addresses for each container.
- c. Kubernetes is responsible for providing a fully qualified domain name for a pod.
- d. A replication controller is responsible for routing external requests to the pods.
- e. A route is responsible for providing DNS names for external access.

► 4. Which statement is correct regarding persistent storage in OpenShift and Kubernetes?

- a. A PVC represents a storage area that a pod can use to store data and is provisioned by the application developer.
- b. PVC represents a storage area that can be requested by a pod to store data but is provisioned by the cluster administrator.
- c. A PVC represents the amount of memory that can be allocated on a node, so that a developer can state how much memory he requires for his application to run.
- d. A PVC represents the number of CPU processing units that can be allocated on a node, subject to a limit managed by the cluster administrator.

► **5. Which statement is correct regarding OpenShift additions over Kubernetes?**

- a. OpenShift adds features required for real-world usage of Kubernetes.
- b. Container images created for OpenShift cannot be used with plain Kubernetes much less with Docker alone.
- c. Red Hat maintains forked versions of Docker and Kubernetes internal to the OCP product.
- d. Doing Continuous Integration and Continuous Deployment with OCP requires external tools.

► **6. Which two statements are true regarding image streams? (Select two.)**

- a. An image stream comprises any number of container images identified by tags.
- b. OpenShift does not provide any default image streams.
- c. Deployed pods do not require updating if the builder image changes.
- d. Images watch image streams to receive notifications when new images are added.
- e. Applications are built against image streams.

► **7. Which two statements are true regarding OpenShift High Availability? (Select two.)**

- a. OpenShift Container Platform HA provides the OpenShift infrastructure itself and HA for the applications running inside the OpenShift cluster.
- b. Masters nodes require plugins to support HA.
- c. Deployed pods do not require updating if the builder image changes.
- d. OpenShift provides a fully supported native HA mechanism for masters by default.
- e. OpenShift networking is responsible for the state of the pod.

► SOLUTION

DESCRIBING KUBERNETES AND OPENSHIFT

Choose the correct answers to the following questions:

► 1. **Which three sentences are correct regarding Kubernetes architecture? (Choose three.)**

- a. Kubernetes nodes can be managed without a master.
- b. Kubernetes masters manage pod scaling.
- c. Kubernetes masters schedule pods to specific nodes.
- d. A pod is a set of containers managed by Kubernetes as a single unit.
- e. Kubernetes tools cannot be used to manage resources in an OpenShift cluster.

► 2. **Which two sentences are correct regarding Kubernetes and OpenShift resource types? (Choose two.)**

- a. A pod is responsible for provisioning its own persistent storage.
- b. All pods generated from the same replication controller have to run in the same node.
- c. A route is responsible for providing IP addresses for external access to pods.
- d. A replication controller is responsible for monitoring and maintaining the number of pods for a particular application.
- e. Containers created from Kubernetes pods cannot be managed using standard Docker tools.

► 3. **Which two statements are true regarding Kubernetes and OpenShift networking? (Select two.)**

- a. A Kubernetes service can provide an IP address to access a set of pods.
- b. Kubernetes is responsible for providing internal IP addresses for each container.
- c. Kubernetes is responsible for providing a fully qualified domain name for a pod.
- d. A replication controller is responsible for routing external requests to the pods.
- e. A route is responsible for providing DNS names for external access.

► 4. **Which statement is correct regarding persistent storage in OpenShift and Kubernetes?**

- a. A PVC represents a storage area that a pod can use to store data and is provisioned by the application developer.
- b. PVC represents a storage area that can be requested by a pod to store data but is provisioned by the cluster administrator.
- c. A PVC represents the amount of memory that can be allocated on a node, so that a developer can state how much memory he requires for his application to run.
- d. A PVC represents the number of CPU processing units that can be allocated on a node, subject to a limit managed by the cluster administrator.

► **5. Which statement is correct regarding OpenShift additions over Kubernetes?**

- a. OpenShift adds features required for real-world usage of Kubernetes.
- b. Container images created for OpenShift cannot be used with plain Kubernetes much less with Docker alone.
- c. Red Hat maintains forked versions of Docker and Kubernetes internal to the OCP product.
- d. Doing Continuous Integration and Continuous Deployment with OCP requires external tools.

► **6. Which two statements are true regarding image streams? (Select two.)**

- a. An image stream comprises any number of container images identified by tags.
- b. OpenShift does not provide any default image streams.
- c. Deployed pods do not require updating if the builder image changes.
- d. Images watch image streams to receive notifications when new images are added.
- e. Applications are built against image streams.

► **7. Which two statements are true regarding OpenShift High Availability? (Select two.)**

- a. OpenShift Container Platform HA provides the OpenShift infrastructure itself and HA for the applications running inside the OpenShift cluster.
- b. Masters nodes require plugins to support HA.
- c. Deployed pods do not require updating if the builder image changes.
- d. OpenShift provides a fully supported native HA mechanism for masters by default.
- e. OpenShift networking is responsible for the state of the pod.

SUMMARY

In this chapter, you learned:

- Containers are an isolated application runtime created with very little overhead.
- A container image packages an application with all its dependencies, making it easier to run the application in different environments.
- Docker creates containers using features of the standard Linux kernel.
- Container image registries are the preferred mechanism for distributing container images to multiple users and hosts.
- OpenShift orchestrates applications composed of multiple containers using Kubernetes.
- Kubernetes manages load balancing, high availability and persistent storage for containerized applications.
- OpenShift adds to Kubernetes multitenancy, security, ease of use, and continuous integration and continuous development features.
- OpenShift routes are key to exposing containerized applications to external users in a manageable way

CHAPTER 2

CREATING CONTAINERIZED SERVICES

GOAL

Provision a server using container technology.

OBJECTIVES

- Create a database server from a container image stored on Docker Hub.

SECTIONS

- Provisioning a Database Server (and Guided Exercise)

LAB

- Creating Containerized Services

PROVISIONING A DATABASE SERVER

OBJECTIVES

After completing this section, students should be able to:

- Create a database server from a container image stored on Docker Hub.
- Search for containers on the Docker Hub site.
- Start containers using the **docker** command.
- Access containers from the command line.
- Leverage the Red Hat Container Catalog.

FINDING AN IMAGE ON DOCKER HUB

Many container images are available for download from the Docker community website at <https://docker.io>. It is a large repository where developers and administrators can get a number of container images developed by the community and some companies.

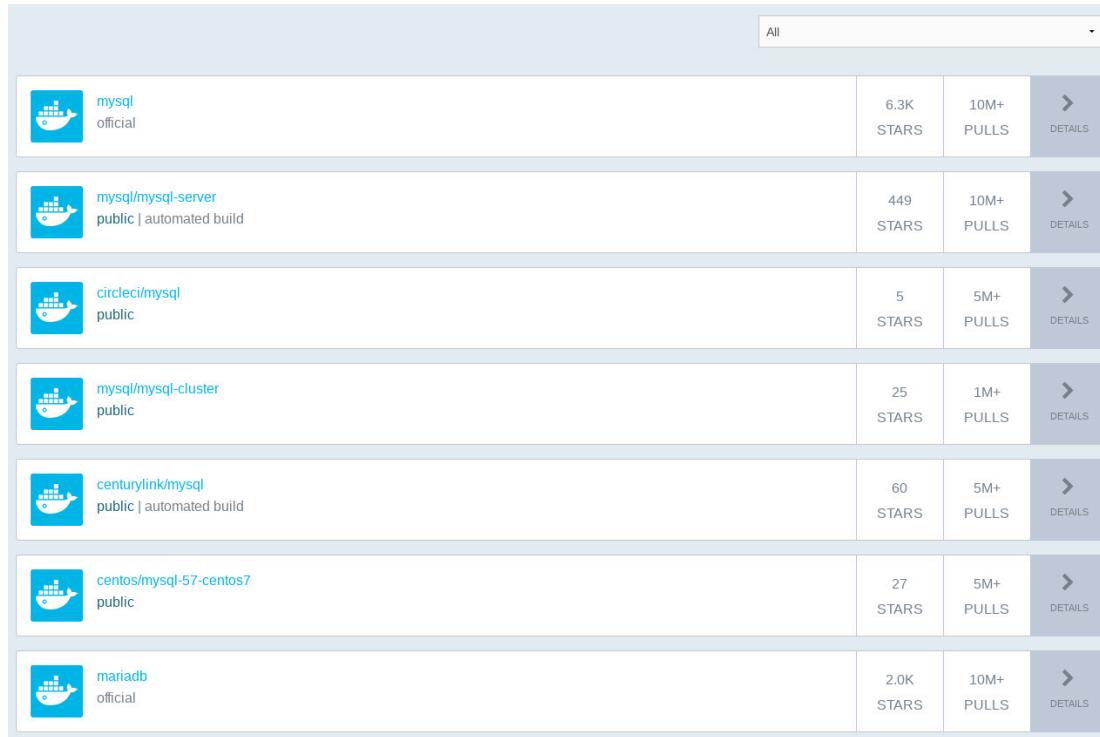
By default, Docker downloads image layers from the Docker Hub image registry. However, images do not provide textual information about themselves, and a search engine tool called Docker Hub was created to look for information about each image and its functionality.



NOTE

Red Hat also provides a private registry with tested and certified container images. By default, RHEL 7 is configured to look for the Red Hat registry in addition to Docker Hub.

The Docker Hub search engine is a simple but effective search engine used to find container images. It looks for a project name and all similar image names from the Docker Hub container image registry. The search results page lists the string used to pull the image files. For example, for the following screen output, the first column is the name of the container image.

**Figure 2.1: Search results page from Docker Hub**

Click an image name to display the container image details page. The details page is not guaranteed to display any particular information, because each author provides different levels of information about their images.

OFFICIAL REPOSITORY

mysql ☆

Last pushed: a month ago

Repo Info Tags

Short Description

MySQL is a widely used, open-source relational database management system (RDBMS).

Docker Pull Command

```
docker pull mysql
```

Full Description

Supported tags and respective Dockerfile links

- 8.0.11, 8.0, 8, latest ([8.0/Dockerfile](#))
- 5.7.22, 5.7, 5 ([5.7/Dockerfile](#))
- 5.6.40, 5.6 ([5.6/Dockerfile](#))
- 5.5.60, 5.5 ([5.5/Dockerfile](#))

Quick reference

- Where to get help:**
[the Docker Community Forums](#), [the Docker Community Slack](#), or [Stack Overflow](#)
- Where to file issues:**
<https://github.com/docker-library/mysql/issues>
- Maintained by:**
[the Docker Community](#) and the MySQL Team
- Supported architectures:** ([more info](#))
amd64

- **Published image artifact details:**
[repo-info](#) repo's repos/mysql/ [directory](#) ([history](#))
(image metadata, transfer size, etc)
- **Image updates:**
[official-images](#) PRs with label library/mysql
[official-images](#) repo's library/mysql [file](#) ([history](#))
- **Source of this description:**
[docs](#) repo's mysql/ [directory](#) ([history](#))
- **Supported Docker versions:**
[the latest release](#) (down to 1.6 on a best-effort basis)

What is MySQL?

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, covering the entire range from personal projects and websites, via e-commerce and information services, all the way to high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.

Figure 2.3: Detailed information about the image

SEARCHING FROM THE DOCKER CLIENT

The **docker** command can also be used to search for container images:

```
[root@workstation ~]# docker search mysql
```

The **search** verb uses the Docker Hub registry but also any other registries running version 1 of the API.

Running the **docker** command requires special privileges. A development environment usually manages that requirement by assigning the developer to the **docker** group. Without the correct privileges to run the **docker** command, an error message such as the following appears:

```
Got permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.26/images/
search?limit=25&term=mysql: dial unix /var/run/docker.sock: connect: permission
denied
```



NOTE

For a production environment, the **docker** command access should be given with the **sudo** command because the **docker** group is vulnerable to privilege escalation attacks.

FETCHING AN IMAGE

To download an image from the Docker Hub container image registry, look for the first column name from the Docker Hub search results page, or the second column from the output of **docker search** command, and use **docker pull** command:

```
[root@workstation ~]# docker pull mysql
```

Many versions of the same image can be provided. Each one receives a different tag name. For example, from the MySQL image details page, there are three different image names available, each one having multiple tag names assigned to them.

If no tag name is provided, then **docker pull** uses the tag called **latest** by default.

To download a specific tag, append the tag name to the image name separated by a colon (:) in the **docker pull** command:

```
[root@workstation ~]# docker pull mysql:5.5
```

LISTING THE IMAGES AVAILABLE IN THE LOCAL CACHE

To list all images that were already downloaded by the local Docker daemon, use the **docker images** command:

```
[root@workstation ~]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
docker.io/mysql    5.5      f13c4be36ec5   3 weeks ago   205 MB
docker.io/mysql    latest    a8a59477268d   3 weeks ago   445 MB
```

The REPOSITORY column contains the image name as the last path component. A Docker daemon installation comes without any images, so the images list is empty until the system administrator downloads images.

An image name is prefixed with a registry name, which is usually the FQDN name of the registry host, but could be any string. An image name can also include a tag. Thus, the full syntax for an image name, including all optional parameters, is as follows:

[registry_uri/] [user_name/] image_name[:tag]

For example: `docker .io/library/mysql:latest`.

Image names from the Docker Hub without a user name actually use the *library* user, so the previous image name is the same as: `docker .io/mysql:latest`. The user *library* is associated with the formally organized images, in official Docker Hub Registry.

Multiple tags can be applied to the same image, as noted previously on MySQL container images. Each tag is listed individually, even though they may represent the same image.

The **docker images** command shows that each image has a unique ID. This allows you to see which image names and tag names refer to the same container image contents. Most **docker** commands that work on images can take either an image name or an image ID as argument.

CREATING A CONTAINER

To create and start a process within a new container, use the **docker run** command. The container is created from the container image name passed as argument.

```
[root@workstation ~]# docker run mysql
```

If the image is not available from the local Docker daemon cache, the **docker run** command tries to pull the image as if a **docker pull** command had been used.

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

To start a container image as a background process, pass the **-d** to the **docker run** command:

```
[root@workstation ~]# docker run -d mysql:5.5
```

Each container has to be assigned a name when created. Docker automatically generates a name if one is not provided. To make container tracking easier, the **--name** option may be passed to the **docker run** command:

```
[root@workstation ~]# docker run --name mysql-container mysql:5.5
```

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

```
[root@workstation ~]# docker run --name mysql-container -it mysql:5.5 /bin/bash  
[root@f0f46d3239e0:/]#
```

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

Many container images require parameters to be started, such as the MySQL official image. They should be provided using the **-e** option from the **docker** command, and are seen as environment variables by the processes inside the container. The following image is a snapshot from the MySQL official image documentation listing all environment variables recognized by the container image:

Environment Variables

When you start the `mysql` image, you can adjust the configuration of the MySQL instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

See also <https://dev.mysql.com/doc/refman/5.7/en/environment-variables.html> for documentation of environment variables which MySQL itself respects (especially variables like `MYSQL_HOST`, which is known to cause issues when used with this image).

`MYSQL_ROOT_PASSWORD`

This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account. In the above example, it was set to `my-secret-pw`.

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access ([corresponding to GRANT ALL](#)) to this database.

`MYSQL_USER`, `MYSQL_PASSWORD`

These variables are optional, used in conjunction to create a new user and to set that user's password. This user will be granted superuser permissions (see above) for the database specified by the `MYSQL_DATABASE` variable. Both variables are required for a user to be created.

Do note that there is no need to use this mechanism to create the root superuser, that user gets created by default with the password specified by the `MYSQL_ROOT_PASSWORD` variable.

`MYSQL_ALLOW_EMPTY_PASSWORD`

This is an optional variable. Set to `yes` to allow the container to be started with a blank password for the root user. *NOTE:* Setting this variable to `yes` is not recommended unless you really know what you are doing, since this will leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access.

`MYSQL_RANDOM_ROOT_PASSWORD`

This is an optional variable. Set to `yes` to generate a random initial password for the root user (using `pwgen`). The generated root password will be printed to stdout (`GENERATED ROOT PASSWORD:`).

`MYSQL_ONETIME_PASSWORD`

Sets root (not the user specified in `MYSQL_USER`!) user as expired once init is complete, forcing a password change on first login. *NOTE:* This feature is supported on MySQL 5.6+ only. Using this option on MySQL 5.5 will throw an appropriate error during initialization.

Figure 2.5: Environment variables supported by the official MySQL Docker Hub image

To start the MySQL server with different user credentials, pass the following parameters to the `docker run` command:

```
[root@workstation ~]# docker run --name mysql-custom \
-e MYSQL_USER=redhat -e MYSQL_PASSWORD=r3dh4t \
-d mysql:5.5
```

LEVERAGING RED HAT CONTAINER CATALOG

Red Hat maintains its own repository of finely-tuned container images. Using this repository provides customers with a layer of protection and reliability against known vulnerabilities, which could potentially be caused by images that are not tested. Whereas the standard `docker` command works perfectly fine with this repository, there is a web portal called *Red Hat Container Catalog*, or *RHCC*, providing a user-friendly interface to search and explore container images from Red Hat. This web portal is available at <https://access.redhat.com/containers/>.

RHCC also serves as a single interface, providing access to different aspects of all the available container images in the repository. It is useful in determining the best among multiple versions of container image on the basis of health index grades. The health index grade indicates how current an image is, and whether the latest security updates have been applied. These grades range from **A** through **F**, with grade **A** being the best.

RHCC also gives access to the errata documentation of an image. It describes the latest bug fixes and enhancements in each update. It also suggests the best technique for pulling an image on each operating system.

The following screenshots highlight some of the features of *Red Hat Container Catalog*.

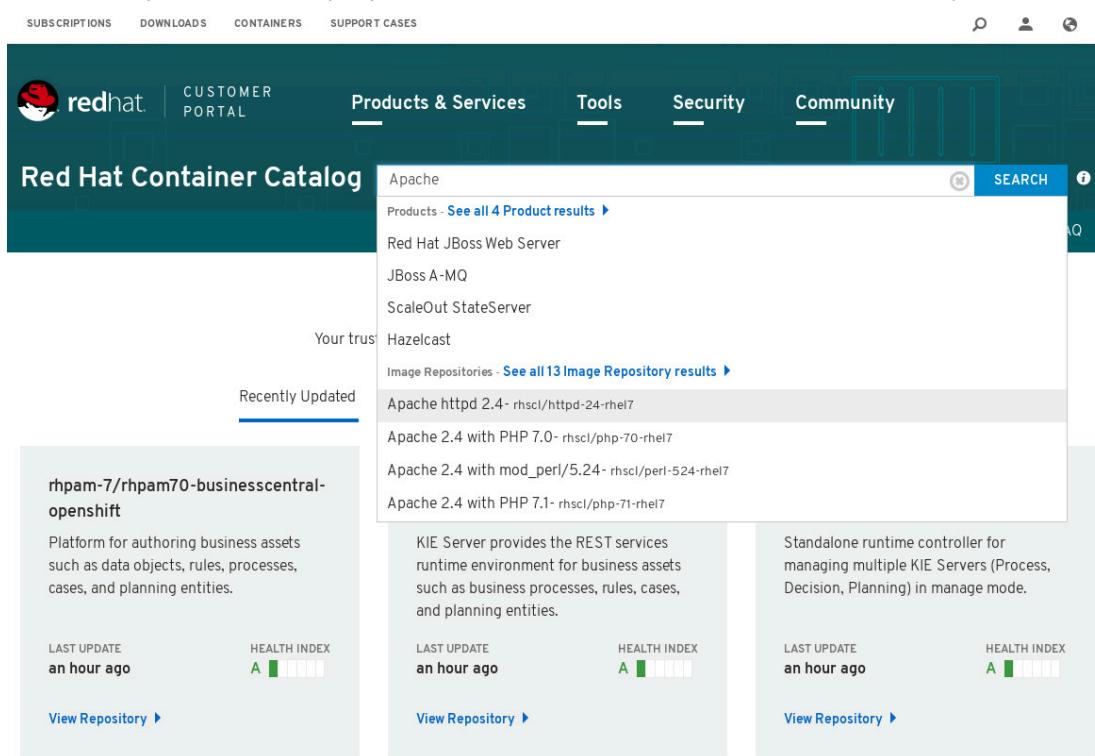


Figure 2.6: Red Hat Container Catalog home page

As displayed in the screenshot above, typing *Apache* in the search box of *Red Hat Container Catalog* displays a suggested list of products and image repositories matching the search pattern. To access the **Apache httpd 2.4** image page, select *Apache httpd 2.4-rhel7/rhel7* from the suggested list.

Apache httpd 2.4 ☆

by Red Hat, Inc. | in Product Red Hat Enterprise Linux

registry.access.redhat.com/rhscl/httpd-24-rhel7 | Updated 6 days ago 2.4-55 : Health Index A [██████]

Description

Apache httpd 2.4 available as container, is a powerful, efficient, and extensible web server. Apache supports a variety of features, many implemented as compiled modules which extend the core functionality. These can range from server-side programming language support to authentication schemes. Virtual hosting allows one Apache installation to serve many different Web sites.

Evaluate Image

There is a preview of this image available on OpenShift Online. Deploy it from the Console and try it out.
[Log in to OpenShift Online](#)

Repository Specifications

Registry	registry.access.redhat.com
Namespace/Repository	rhscl/httpd-24-rhel7
Release Category	Generally Available
Application Categories	Web Services

Most recent tag [View All Tags](#)

- Updated 6 days ago 2.4-55
- Health Index A [██████]
- Security: Signed, Unprivileged
- Size: 111.9 MB
- RPM Packages: 222

Figure 2.7: Apache httpd 2.4 (rhscl/httpd-24-rhel7) image page

Details and several tabs are displayed under the *Apache httpd 2.4* panel. This page states that **Red Hat, Inc.** maintains the image repository. It also indicates that the image repository is associated with Red Hat Enterprise Linux. Under the *Overview* tab, there are other details:

- **Description:** A short summary of what the image's capabilities.
- **Evaluate Image:** Using **OpenShift Online** (a public PaaS cloud), users can try out the image to validate the functional state of the application in the image.
- **Most Recent Tag:** When the image received its latest update, the latest applied to the image, the health of the image, and more.

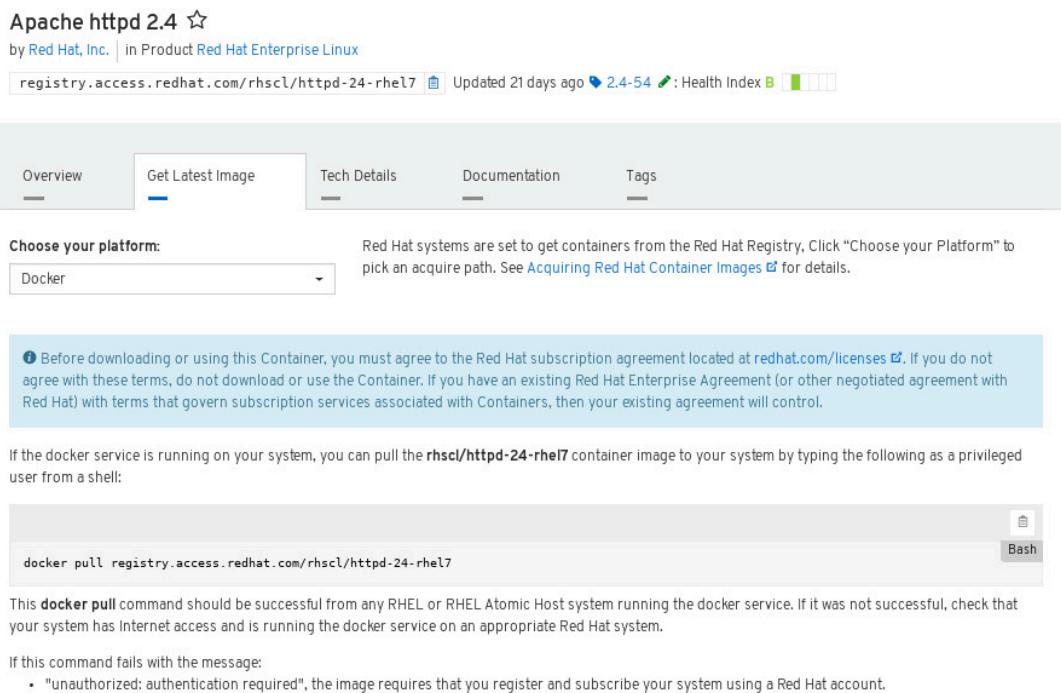


Figure 2.8: Apache httpd 2.4 (rhscl/httpd-24-rhel7) image page

The **Get Latest Image** tab provides the procedure to get the most current version of the image. Specify the intended platform for the image under **Choose your platform** drop-down menu, and the page provides the appropriate command to be executed.



REFERENCES

Docker Hub website

<https://hub.docker.com>

Red Hat Registry website

<https://registry.access.redhat.com>

► GUIDED EXERCISE

CREATING A MYSQL DATABASE INSTANCE

In this exercise, you will start a MySQL database inside a container, and then create and populate a database.

OUTCOMES

You should be able to start a database from a container image and store information inside the database.

BEFORE YOU BEGIN

Your workstation must have Docker running. To verify that Docker is running, use the following command in a terminal:

```
[student@workstation ~]$ lab create-basic-mysql setup
```

► 1. Create a MySQL container instance.

- 1.1. Start a container from the Red Hat Software Collections Library MySQL image.

```
[student@workstation ~]$ docker run --name mysql-basic \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
-d rhsc1/mysql-57-rhel7:5.7-3.14
```

This command downloads the `mysql` container image with the **5.7-3.14 tag**, and then starts a container-based image. It creates a database named *items*, owned by a user named *user1* with *mypa55* as the password. The database administrator password is set to *r00tpa55* and the container runs in the background.

- 1.2. Verify that the container started without errors.

[student@workstation ~]\$ docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
<code>2ae7af15746d</code>	<code>rhsc1/mysql-57-rhel7:5.7-3.14</code>	<code>"container-entrypo..."</code>	<code>4 seconds ago</code>	<code>Up 3 seconds</code>	<code>3306/tcp</code>	<code>mysql-basic</code>

► 2. Access the container sandbox by running the following command:

```
[student@workstation ~]$ docker exec -it mysql-basic bash
```

This command starts a Bash shell, running as the `mysql` user inside the **MySQL** container.

► 3. Add data to the database.

- 3.1. Log in to MySQL as the database administrator user (root).

Run the following command from the container terminal to connect to the database:

```
bash-4.2$ mysql -uroot
```

The **mysql** command opens the MySQL database interactive prompt. Run the following command to check the database availability:

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| mysql          |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.00 sec)
```

- 3.2. Create a new table in the *items* database. Run the following command to access the database.

```
mysql> use items;
Database changed
```

- 3.3. Create a table called *Projects* in the *items* database.

```
mysql> CREATE TABLE Projects (id int(11) NOT NULL, name varchar(255) DEFAULT NULL,
   code varchar(255) DEFAULT NULL, PRIMARY KEY (id));
Query OK, 0 rows affected (0.29 sec)
```



NOTE

You can optionally use the `~student/D0285/solutions/create-mysql-basic/create_table.txt` file to copy and paste the **CREATE TABLE** mysql statement as given above.

- 3.4. Use the **show tables** command to verify that the table was created.

```
mysql> show tables;
+-----+
| Tables_in_items      |
+-----+
| Projects             |
+-----+
```

- 3.5. Use the **insert** command to insert a row into the table.

```
mysql> insert into Projects (id, name, code) values (1, 'DevOps', 'D0285');
```

```
Query OK, 1 row affected (0.02 sec)
```

- 3.6. Use the **select** command to verify that the project information was added to the table.

```
mysql> select * from Projects;
+----+-----+-----+
| id | name      | code   |
+----+-----+-----+
| 1  | DevOps    | D0285 |
+----+-----+-----+
```

- 3.7. Exit from the MySQL prompt and the MySQL container:

```
mysql> exit
Bye
bash-4.2$ exit
exit
```

- 4. Verify that the database was configured correctly by running the following script:

```
[student@workstation ~]$ lab create-basic-mysql grade
```

- 5. Revert the modifications made during the lab.

- 5.1. Use the command **docker stop** to stop the container.

```
[student@workstation ~]$ docker stop mysql-basic
mysql-basic
```

- 5.2. Remove the data from the stopped container using the **docker rm** command.

```
[student@workstation ~]$ docker rm mysql-basic
mysql-basic
```

- 5.3. Use the command **docker rmi** to remove the container image.

```
[student@workstation ~]$ docker rmi rhscl/mysql-57-rhel7:5.7-3.14
Untagged: rhscl/mysql-57-rhel7:5.7-3.14
Untagged: registry.lab.example.com/rhscl/mysql-57-
rhel7@sha256:0a8828385c63d6a7cdb1cfccf899303357d5cbe500fa1761114256d5966a
Deleted: sha256:4ae3a3f4f409a8912cab9fbf71d3564d011ed2e68f926d50f88f2a3a72c809c5
Deleted: sha256:aeecef47e8af04bce5918cf2ecd5bc2e6440c34af30ad36cfb902f8814e9002
Deleted: sha256:238aeee18c5ad3ebcab48db60799cc60607d9f630881a6530dcfe58d117adabf2
Deleted: sha256:d4d40807755515379fa58b2b225b72f7b8e1722ff2bc0ecd39f01f6f8174b3e3
```

This concludes the guided exercise.

► LAB

CREATING CONTAINERIZED SERVICES

In this lab, you create an Apache HTTP Server container with a custom welcome page.

OUTCOMES

You should be able to start and customize a container using a container image from Docker Hub.

BEFORE YOU BEGIN

Open a terminal on **workstation** as **student** user and run the following command to verify that Docker is running:

```
[student@workstation ~]$ lab httpd-basic setup
```

1. Start a container named **httpd-basic** in the background, and forward port 8080 to port 80 in the container. Use the **httpd** container image with the **2.4** tag.



NOTE

Use the **-p 8080:80** option with **docker run** command to forward the port.

This command starts the Apache HTTP server in the background and returns to the Bash prompt.

2. Test the **httpd-basic** container.

From **workstation**, attempt to access **http://localhost:8080** using any web browser.

An "It works!" message is displayed, which is the **index.html** page from the Apache HTTP server container running on **workstation**.

3. Customize the **httpd-basic** container.

- 3.1. Start a Bash session inside the container and customize the existing content of **index.html** page.
- 3.2. From the Bash session, verify the **index.html** file under **/usr/local/apache2/htdocs** directory using the **ls -la** command.
- 3.3. Change the **index.html** page to contain the text **Hello World**, replacing all of the existing content.
- 3.4. Attempt to access **http://localhost:8080** again, and verify that the web page has been updated.

4. Grade your work.

Run the following command:

```
[student@workstation ~]$ lab httpd-basic grade
```

5. Clean up.

- 5.1. Stop and remove the **httpd-basic** container.
 - 5.2. Remove the **httpd-basic** container image from the local Docker cache.
- This concludes this lab.

► SOLUTION

CREATING CONTAINERIZED SERVICES

In this lab, you create an Apache HTTP Server container with a custom welcome page.

OUTCOMES

You should be able to start and customize a container using a container image from Docker Hub.

BEFORE YOU BEGIN

Open a terminal on **workstation** as **student** user and run the following command to verify that Docker is running:

```
[student@workstation ~]$ lab httpd-basic setup
```

1. Start a container named **httpd-basic** in the background, and forward port 8080 to port 80 in the container. Use the **httpd** container image with the **2.4** tag.



NOTE

Use the **-p 8080:80** option with **docker run** command to forward the port.

Run the following command:

```
[student@workstation ~]$ docker run -d -p 8080:80 \
--name httpd-basic \
httpd:2.4
Unable to find image 'httpd:2.4' locally
Trying to pull repository registry.lab.example.com/httpd ...
2.4: Pulling from registry.lab.example.com/httpd
3d77ce4481b1: Pull complete
73674f4d9403: Pull complete
d266646f40bd: Pull complete
ce7b0dda0c9f: Pull complete
01729050d692: Pull complete
014246127c67: Pull complete
7cd2e04cf570: Pull complete
Digest: sha256:58270ec746bed598ec109aef58d495fca80ee0a89f520bd2430c259ed31ee144
Status: Downloaded newer image for registry.lab.example.com/httpd:2.4
45ede3ccf26ff43079603cec68bc7a7c9105ac1ce8334052afab48e77ee65a03
```

This command starts the Apache HTTP server in the background and returns to the Bash prompt.

2. Test the **httpd-basic** container.

From **workstation**, attempt to access **http://localhost:8080** using any web browser.

An "It works!" message is displayed, which is the **index.html** page from the Apache HTTP server container running on **workstation**.

```
[student@workstation ~]$ curl http://localhost:8080
<html><body><h1>It works!</h1></body></html>
```

3. Customize the **httpd-basic** container.

- 3.1. Start a Bash session inside the container and customize the existing content of **index.html** page.

Run the following command:

```
[student@workstation ~]$ docker exec -it httpd-basic bash
```

- 3.2. From the Bash session, verify the **index.html** file under **/usr/local/apache2/htdocs** directory using the **ls -la** command.

```
root@2d2417153059:/usr/local/apache2# ls -la /usr/local/apache2/htdocs
total 3
drwxr-sr-x. 2 root      www-data 24 Jun 15 11:02 .
drwxr-sr-x. 1 www-data www-data 18 Apr 30 04:30 ..
-rw-rw-r--. 1      1000     1000 10 Jun 15 11:02 index.html
```

- 3.3. Change the **index.html** page to contain the text **Hello World**, replacing all of the existing content.

From the Bash session in the container, run the following command:

```
root@2d2417153059:/usr/local/apache2# echo "Hello World" > \
/usr/local/apache2/htdocs/index.html
```

- 3.4. Attempt to access **http://localhost:8080** again, and verify that the web page has been updated.

```
root@2d2417153059:/usr/local/apache2# exit
[student@workstation ~]$ curl http://localhost:8080
Hello World
```

4. Grade your work.

Run the following command:

```
[student@workstation ~]$ lab httpd-basic grade
```

5. Clean up.

- 5.1. Stop and remove the **httpd-basic** container.

```
[student@workstation ~]$ docker stop httpd-basic
httpd-basic
[student@workstation ~]$ docker rm httpd-basic
httpd-basic
```

- 5.2. Remove the **httpd-basic** container image from the local Docker cache.

```
[student@workstation ~]$ docker rmi httpd:2.4
```

```
Untagged: httpd:2.4
Untagged: registry.lab.example.com/
httpd@sha256:58270ec746bed598ec109aef58d495fca80ee0a89f520bd2430c259ed31
Deleted: sha256:fb2f3851a97186bb0eaf551a40b94782712580c2feac0d15ba925bef2da5fc18
Deleted: sha256:aed0dee33d7b24c5d60b42e6a9e5f15fd040dbeb9b7f2fbf6ec93f205fc7e5a4
Deleted: sha256:2f645c8ada1c73b101ceb7729275cebed1fa643a2b056412934a100d1faf615
Deleted: sha256:838f5356324c57929eb42e6885c0dde76703e9d2c11bf0b6a39afbaa3e6cf443
Deleted: sha256:8837e56bcf087feea70b2b5162365a46a41cbf3bbcfa643f2c3b7e39fe7448d2
Deleted: sha256:1b2a502dceb0d29a50064d64d96282063e6dd1319de112ab18e69557dcc5c915
Deleted: sha256:13f2da842fdccaa301d6b853a9644476e6e1fed9150c32bbc4c3f9d5699169a2
Deleted: sha256:2c833f307fd8f18a378b71d3c43c575fabdb88955a2198662938ac2a08a99928
```

This concludes this lab.

SUMMARY

In this chapter, you learned:

- Red Hat OpenShift Container Platform can be installed from RPM packages, from the client as a container, and in a dedicated virtual machine from Red Hat Container Development Kit (CDK).
 - The RPM installation usually configures a cluster of multiple Linux nodes for production, QA, and testing environments.
 - The containerized and CDK installations can be performed on a developer workstation, and supports Linux, MacOS, and Windows operating systems.
- The **oc cluster up** command from the OpenShift client starts the local all-in-one cluster inside a container. It provides many command-line options to adapt to offline environments.
- Before starting the local all-in-one cluster, the Docker daemon must be configured to allow accessing the local insecure registry.
- The Docker Hub website provides a web interface to search for container images developed by the community and corporations. The Docker client can also search for images in Docker Hub.
- The **docker run** command creates and starts a container from an image that can be pulled by the local Docker daemon.
- Container images might require environment variables that are set using the **-e** option from the **docker run** command.
- *Red Hat Container Catalog* assists in searching, exploring, and analyzing container images from Red Hat's official container image repository.

CHAPTER 3

MANAGING CONTAINERS

GOAL

Manipulate pre-built container images to create and manage containerized services.

OBJECTIVES

- Manage the life cycle of a container from creation to deletion.
- Save application data across container restarts through the use of persistent storage.
- Describe how Docker provides network access to containers, and access a container through port forwarding.

SECTIONS

- Managing the Life Cycle of Containers (and Guided Exercise)
- Attaching Docker Persistent Storage (and Guided Exercise)
- Accessing Docker Networks (and Guided Exercise)

LAB

- Managing Containers

MANAGING THE LIFE CYCLE OF CONTAINERS

OBJECTIVES

After completing this section, students should be able to manage the life cycle of a container from creation to deletion.

DOCKER CLIENT VERBS

The Docker client, implemented by the `docker` command, provides a set of verbs to create and manage containers. The following figure shows a summary of the most commonly used verbs that change container state.

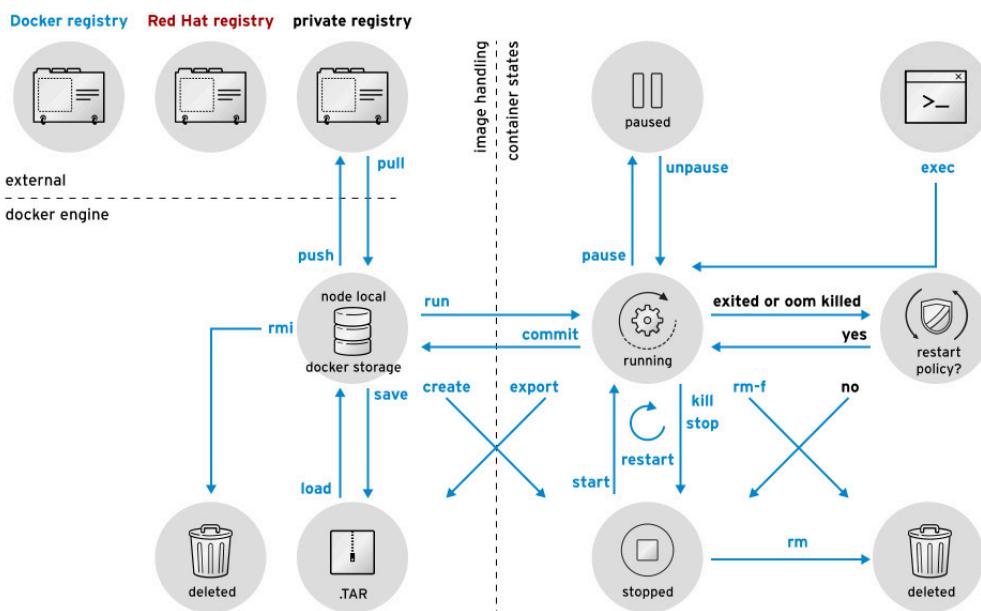


Figure 3.1: Docker client action verbs

The Docker client also provides a set of verbs to obtain information about running and stopped containers. The following figure shows a summary of the most commonly used verbs that query information related to Docker containers.

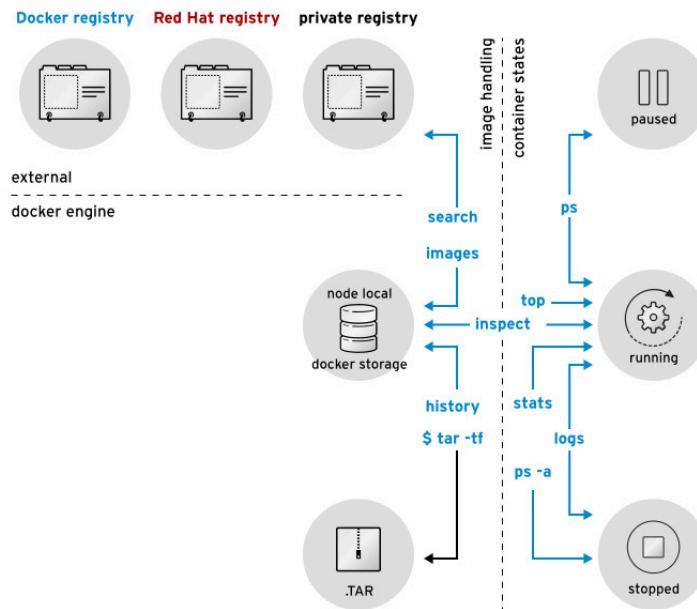


Figure 3.2: Docker client query verbs

Use these two figures as a reference while you learn about the **docker** command verbs along this course.

CREATING CONTAINERS

The **docker run** command creates a new container from an image and starts a process inside the new container. If the container image is not available, this command also tries to download it:

```
$ docker run rhscl/httpd-24-rhel7
Unable to find image 'rhscl/httpd-24-rhel7:latest' locally
Trying to pull repository registry.lab.example.com/rhscl/httpd-24-rhel7 ...
latest: Pulling from registry.lab.example.com/rhscl/httpd-24-rhel7
b12636467c49: Pull complete
b538cc6febe6: Pull complete
fd87b2ca7715: Pull complete
6861c26d5818: Pull complete
Digest: sha256:35f2b43891a7ebfa5330ef4c736e171c42380aec95329d863dcde0e608ffff1e
...
[Wed Jun 13 09:40:34.098087 2018] [core:notice] [pid 1] AH00094: Command line:
'httpd -D FOREGROUND'
$ ^C
```

In the previous output sample, the container was started with a noninteractive process, and stopping that process with **Ctrl+C (SIGINT)** also stops the container.

The management docker commands require an ID or a name. The **docker run** command generates a random ID and a random name that are unique. The **docker ps** command is responsible for displaying these attributes:

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
347c9aad6049❶	rhscl/httpd-24-rhel7	"httpd -D FOREGROUND"	31 seconds
ago	80/tcp	<i>focused_fermat</i> ❷	

- ❶ This ID is generated automatically and must be unique.
- ❷ This name can be generated automatically or manually specified.

If desired, the container name can be explicitly defined. The **--name** option is responsible for defining the container name:

```
$ docker run --name my-nginx-container do285/nginx
```



NOTE

The name must be unique. An error is thrown if another container has the same name, including containers that are stopped.

Another important option is to run the container as a daemon, running the containerized process in the background. The **-d** option is responsible for running in detached mode. Using this option, the container ID is displayed on the screen:

```
$ docker run --name my-nginx-container -d do285/nginx
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

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

```
$ docker run do285/nginx ls /tmp
anaconda-post.log
ks-script-1j4CXN
yum.log
```

The specified command must exist inside the container image.



NOTE

Since a specified command was provided in the previous example, the *HTTPD* service does not start.

Sometimes it is desired to run a container executing a Bash shell. This can be achieved with:

```
$ docker run --name my-nginx-container -it do285/nginx /bin/bash
bash-4.2#
```

Options **-t** and **-i** are usually needed for interactive text-based programs, so they get a proper terminal, but not for background daemons.

RUNNING COMMANDS IN A CONTAINER

When a container is created, a default command is executed according to what is specified by the container image. However, it may be necessary to execute other commands to manage the running container.

The **docker exec** command starts an additional process inside a running container:

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

The previous example used the container ID to execute the command. It is also possible to use the container name:

```
$ docker exec my-httdp-container cat /etc/hostname
7ed6e671a600
```

DEMONSTRATION: CREATING CONTAINERS

- Run the following command:

```
[student@workstation ~]$ docker run --name demo-container rhel7 \
dd if=/dev/zero of=/dev/null
```

This command downloads the official Red Hat Enterprise Linux 7 container and starts it using the **dd** command. The container exits when the **dd** command returns the result. For educational purposes, the provided **dd** never stops.

- Open a new terminal window from the workstation VM and check if the container is running:

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

Some information about the container, including the container name **demo-container** specified in the last step, is displayed.

- Open a new terminal window and stop the container using the provided name:

```
[student@workstation ~]$ docker stop demo-container
```

- Return to the original terminal window and verify that the container was stopped:

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

- Start a new container without specifying a name:

```
[student@workstation ~]$ docker run rhel7 dd if=/dev/zero of=/dev/null
```

If a container name is not provided, **docker** generates a name for the container automatically.

- Open a terminal window and verify the name that was generated:

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

An output similar to the following will be listed:

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
05b725c0fd5a	rhel7	"dd if=/dev/zero of=/"	13 seconds ago
Up 11 seconds		reverent_blackwell	

The `reverent_blackwell` is the generated name. Students probably will have a different name for this step.

7. Stop the container with the generated name:

```
[student@workstation ~]$ docker stop reverent_blackwell
```

8. Containers can have a default long-running command. For these cases, it is possible to run a container as a daemon using the `-d` option. For example, when a MySQL container is started it creates the databases and keeps the server actively listening on its port. Another example using `dd` as the long-running command is as follows:

```
[student@workstation ~]$ docker run --name demo-container-2 -d rhel7 \
dd if=/dev/zero of=/dev/null
```

9. Stop the container:

```
[student@workstation ~]$ docker stop demo-container-2
```

10. Another possibility is to run a container to just execute a specific command:

```
[student@workstation ~]$ docker run --name demo-container-3 rhel7 ls /etc
```

This command starts a new container, lists all files available in the `/etc` directory in the container, and exits.

11. Verify that the container is not running:

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

12. It is possible to run a container in interactive mode. This mode allows for staying in the container when the container runs:

```
[student@workstation ~]$ docker run --name demo-container-4 -it rhel7 \
/bin/bash
```

The `-i` option specifies that this container should run in interactive mode, and the `-t` allocates a pseudo-TTY.

13. Exit the Bash shell from the container:

```
[root@8b1580851134 /]# exit
```

14. Remove all stopped containers from the environment by running the following from a terminal window:

```
[student@workstation ~]$ docker rm demo-container demo-container-2 \
demo-container-3 demo-container-4
```

15. Remove the container started without a name. Replace the <container_name> with the container name from the step 7:

```
[student@workstation ~]$ docker rm <container_name>
```

16. Remove the rhel7 container image:

```
[student@workstation ~]$ docker rmi rhel7
```

This concludes the demonstration.

MANAGING CONTAINERS

Docker provides the following commands to manage containers:

- **docker ps**: This command is responsible for listing running containers:

\$ docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED			
STATUS	PORTS	NAMES				
77d4b7b8ed1f ①	do285/httpd ②	"httpd -D FOREGROUND" ③	15 hours ago ④			
Up 15 hours ⑤	80/tcp ⑥	my-httpd-container ⑦				

- ① Each container, when created, gets a **container ID**, which is a hexadecimal number and looks like an image ID, but is actually unrelated.
- ② Container image that was used to start the container.
- ③ Command that was executed when the container started.
- ④ Date and time the container was started.
- ⑤ Total container uptime, if still running, or time since terminated.
- ⑥ Ports that were exposed by the container or the port forwards, if configured.
- ⑦ The container name.

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

\$ docker ps -a						
CONTAINER ID	IMAGE	COMMAND	CREATED			
STATUS	PORTS	NAMES				
4829d82fbbff	do285/httpd	"httpd -D FOREGROUND"	15 hours ago			
Exited (0) 3 seconds ago		my-httpd-container				

- **docker inspect**: This command is responsible for listing metadata about a running or stopped container. The command produces a **JSON** output:

```
$ docker inspect my-httpd-container
[
```

```
{
  "Id": "980e45b5376a4e966775fb49cbef47ee7bbd461be8bfd1a75c2cc5371676c8be",
  ...OUTPUT OMITTED...
  "NetworkSettings": {
    "Bridge": "",
    "EndpointID":
    "483fc91363e5d877ea8f9696854a1f14710a085c6719afc858792154905d801a",
    "Gateway": "172.17.42.1",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "HairpinMode": false,
    "IPAddress": "172.17.0.9",
  ...OUTPUT OMITTED...
}
```

This command allows formatting of the output string using the given **go** template with the **-f** option. For example, to retrieve only the IP address, the following command can be executed:

```
$ docker inspect -f '{{ .NetworkSettings.IPAddress }}' my-httdp-container
```

- **docker stop**: This command is responsible for stopping a running container gracefully:

```
$ docker stop my-httdp-container
```

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

- **docker kill**: This command is responsible for stopping a running container forcefully:

```
$ docker kill my-httdp-container
```

It is possible to specify the signal with the **-s** option:

```
$ docker kill -s SIGKILL my-httdp-container
```

The following signals are available:

SIGNAL	DEFAULT ACTION	DESCRIPTION
SIGHUP	Terminate process	Terminate line hangup
SIGINT	Terminate process	Interrupt program
SIGQUIT	Create core image	Quit program
SIGABRT	Create core image	Abort program
SIGKILL	Terminate process	Kill program
SIGTERM	Terminate process	Software termination signal
SIGUSR1	Terminate process	User-defined signal 1
SIGUSR2	Terminate process	User-defined signal 2

- **docker restart**: This command is responsible for restarting a stopped container:

```
$ docker restart my-httdp-container
```

The **docker restart** command creates a new container with the same container ID, reusing the stopped container state and filesystem.

- **docker rm**: This command is responsible for deleting a container, discarding its state and filesystem:

```
$ docker rm my-httdp-container
```

It is possible to delete all containers at the same time. The **docker ps** command has the **-q** option that returns only the ID of the containers. This list can be passed to the **docker rm** command:

```
$ docker rm $(docker ps -q)
```

Before deleting all containers, all running containers must be stopped. It is possible to stop all containers with:

```
$ docker stop $(docker ps -q)
```



NOTE

The commands **docker inspect**, **docker stop**, **docker kill**, **docker restart**, and **docker rm** can use the container ID instead of the container name.

DEMONSTRATION: MANAGING A CONTAINER

1. Run the following command:

```
[student@workstation ~]$ docker run --name demo-container -d rhsc1/httpd-24-rhel7
```

This command will start a **HTTPD** container as a daemon.

2. List all running containers:

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

3. Stop the container with the following command:

```
[student@workstation ~]$ docker stop demo-container
```

4. Verify that the container is not running:

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

5. Run a new container with the same name:

```
[student@workstation ~]$ docker run --name demo-container -d rhsc1/httpd-24-rhel7
```

A conflict error is displayed. Remember that a stopped container is not discarded immediately and their local file systems and other states are preserved so they can be inspected for *post-mortem* analysis.

6. It is possible to list all containers with the following command:

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

7. Start a new **HTTPD** container:

```
[student@workstation ~]$ docker run --name demo-1-httdp -d rhel7/httpd-24-rhel7
```

8. An important feature is the ability to list metadata about a running or stopped container. The following command returns the metadata:

```
[student@workstation ~]$ docker inspect demo-1-httdp
```

9. It is possible to format and retrieve a specific item from the **inspect** command. To retrieve the **IPAddress** attribute from the **NetworkSettings** object, use the following command:

```
[student@workstation ~]$ docker inspect -f '{{ .NetworkSettings.IPAddress }}' \
demo-1-httdp
```

Make a note about the IP address from this container. It will be necessary for a further step.

10. Run the following command to access the container **bash**:

```
[student@workstation ~]$ docker exec -it demo-1-httdp /bin/bash
```

11. Create a new HTML file on the container and exit:

```
bash-4.2# echo do285 > \
/opt/rh/httpd24/root/var/www/html/do285.html
bash-4.2# exit
```

12. Using the IP address from step 8, try to access the previously created page:

```
[student@workstation ~]$ curl IP:8080/do285.html
```

The following output is be displayed:

```
do285
```

13. It is possible to restart the container with the following command:

```
[student@workstation ~]$ docker restart demo-1-httdp
```

14. When the container is restarted, the data is preserved. Verify the IP address from the restarted container and check that the do285 page is still available:

```
[student@workstation ~]$ docker inspect demo-1-httdp | grep IPAddress
```

```
[student@workstation ~]$ curl IP:8080/do285.html
```

15. Stop the **HTTP** container:

```
[student@workstation ~]$ docker stop demo-1-httdp
```

16. Start a new **HTTP** container:

```
[student@workstation ~]$ docker run --name demo-2-httdp -d rhsc1/httdp-24-rhel7
```

17. Verify the IP address from the new container and check if the do285 page is available:

```
[student@workstation ~]$ docker inspect demo-2-httdp | grep IPAddress  
[student@workstation ~]$ curl IP:8080/do285.html
```

The page is not available because this page was created just for the previous container. New containers will not have the page since the container image did not change.

18. In case of a freeze, it is possible to kill a container like any process. The following command will kill a container:

```
[student@workstation ~]$ docker kill demo-2-httdp
```

This command kills the container with the **SIGKILL** signal. It is possible to specify the signal with the **-s** option.

19. Containers can be removed, discarding their state and filesystem. It is possible to remove a container by name or by its ID. Remove the **demo-httdp** container:

```
[student@workstation ~]$ docker ps -a  
[student@workstation ~]$ docker rm demo-1-httdp
```

20. It is also possible to remove all containers at the same time. The **-q** option returns the list of container IDs and the **docker rm** accepts a list of IDs to remove all containers:

```
[student@workstation ~]$ docker rm $(docker ps -aq)
```

21. Verify that all containers were removed:

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

22. Clean up the images downloaded by running the following from a terminal window:

```
[student@workstation ~]$ docker rmi rhsc1/httdp-24-rhel7
```

This concludes the demonstration.

► GUIDED EXERCISE

MANAGING A MYSQL CONTAINER

In this exercise, you will create and manage a MySQL database container.

OUTCOMES

You should be able to create and manage a MySQL database container.

BEFORE YOU BEGIN

The workstation should have docker running. To check if this is true, run the following command from a terminal window:

```
[student@workstation ~]$ lab managing-mysql setup
```

- ▶ 1. Open a terminal window from the workstation VM (Applications → Utilities → Terminal) and run the following command:

```
[student@workstation ~]$ docker run --name mysql-db rhsc1/mysql-57-rhel7
```

This command downloads the MySQL database container image and tries to start it, but it does not start. The reason for this is the image requires a few environment variables to be provided.



NOTE

If you try to run the container as a daemon (**-d**), the error message about the required variables is not displayed. However, this message is included as part of the container logs, which can be viewed using the following command:

```
[student@workstation ~]$ docker logs mysql-db
```

- ▶ 2. Start the container again, providing the required variables. Give it a name of **mysql**. Specify each variable using the **-e** parameter.



NOTE

Make sure to start the new container with the correct name.

```
[student@workstation ~]$ docker run --name mysql \
-d -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
rhsc1/mysql-57-rhel7
```

- 3. Verify that the container was started correctly. Run the following command:

```
[student@workstation ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
5cd89eca81dd        rh scl/mysql-57-rhel7   "container-entrypoint"   9 seconds ago    Up 8 seconds      mysql
```

- 4. Inspect the container metadata to obtain the IP address from the MySQL database:

```
[student@workstation ~]$ docker inspect -f \
'{{ .NetworkSettings.IPAddress }}' mysql
172.17.0.2
```

**NOTE**

You can get other important information with the **docker inspect** command. For example, if you forgot the root password, it is available in the **Env** section.

- 5. Connect to the MySQL database from the host:

```
[student@workstation ~]$ mysql -uuser1 -h IP -p items
```

Use **mypa55** as password.

- 6. You are connected to the **items** database. Create a new table:

```
MySQL [items]> CREATE TABLE Projects (id int(11) NOT NULL,
name varchar(255) DEFAULT NULL, code varchar(255) DEFAULT NULL,
PRIMARY KEY (id));
```

**NOTE**

Alternatively you can upload the database using the provided file:

```
[student@workstation ~]$ mysql -uuser1 -h IP -pmypa55 items \
< D0285/labs/managing-mysqldb/db.sql
```

- 7. Insert a row into the table by running the following command:

```
MySQL [items]> insert into Projects (id, name, code) values (1,'DevOps','D0285');
```

- 8. Exit from the MySQL prompt:

```
MySQL [items]> exit
```

- ▶ 9. Create another container using the same container image from the previous container executing the **/bin/bash** shell:

```
[student@workstation ~]$ docker run --name mysql-2 -it \
rhscl/mysql-57-rhel7 /bin/bash
bash-4.2$
```

- ▶ 10. Try to connect to the MySQL database:

```
bash-4.2$ mysql -uroot
```

The following error is displayed:

```
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/var/lib/
mysql/mysql.sock' (2)
```

The reason for this error is that the MySQL database server is not running because we changed the default command responsible for starting the database to **/bin/bash**.

- ▶ 11. Exit from the **bash** shell:

```
bash-4.2$ exit
```

- ▶ 12. When you exit the **bash** shell, the container was stopped. Verify that the container **mysql-2** is not running:

```
[student@workstation ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
8b2c0ee86419        rhscl/mysql-57-rhel7   "container-entrypoint"   4 minutes ago
Up 4 minutes          3306/tcp           mysql
```

- ▶ 13. Verify that the database was correctly set up. Run the following from a terminal window:

```
[student@workstation ~]$ lab managing-mysql grade
```

- ▶ 14. Delete the containers and resources created by this lab.

14.1. Stop the running container, by running the following commands:

```
[student@workstation ~]$ docker stop mysql
```

14.2. Remove the container data by running the following commands:

```
[student@workstation ~]$ docker rm mysql
[student@workstation ~]$ docker rm mysql-2
```

```
[student@workstation ~]$ docker rm mysql-db
```

14.3. Remove the container image by running the following command:

```
[student@workstation ~]$ docker rmi rhsc1/mysql-57-rhel7
```

This concludes the guided exercise.

ATTACHING DOCKER PERSISTENT STORAGE

OBJECTIVES

After completing this section, students should be able to:

- Save application data across container restarts through the use of persistent storage.
- Configure host directories for use as container volumes.
- Mount a volume inside the container.

PREPARING PERMANENT STORAGE LOCATIONS

Container storage is said to be **ephemeral**, meaning its contents are not preserved after the container is removed. Containerized applications are supposed to work on the assumption that they always start with empty storage, and this makes creating and destroying containers relatively inexpensive operations.

Ephemeral container storage is **not** sufficient for applications that need to keep data over restarts, like databases. To support such applications, the administrator must provide a container with persistent storage.

Previously in this course, container images were characterized as **immutable** and **layered**, meaning they are never changed, but composed of layers that add or override the contents of layers below.

A running container gets a new layer over its base container image, and this layer is the **container storage**. At first, this layer is the only read-write storage available for the container, and it is used to create working files, temporary files, and log files. Those files are considered volatile. An application does not stop working if they are lost. The container storage layer is exclusive to the running container, so if another container is created from the same base image, it gets another read-write layer.

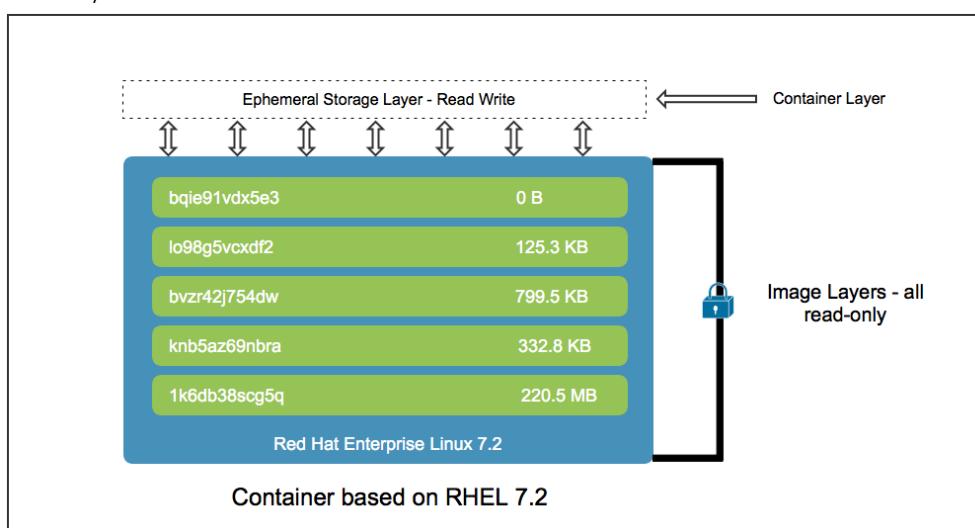


Figure 3.3: Container layers

Containerized applications should not try to use the container storage to store persistent data, as they cannot control how long its contents will be preserved. Even if it were possible to keep

container storage around for a long time, the layered file system does not perform well for intensive I/O workloads and would not be adequate for most applications requiring persistent storage.

Reclaiming Storage

Docker tries to keep old stopped container storage available for a while to be used by troubleshooting operations, such as reviewing a failed container logs for error messages. But this container storage can be reclaimed at any time to create new containers, including replacements for the old ones; for example, when the host is rebooted.

If the administrator needs to reclaim old containers storage sooner, the stopped container IDs can be found using `docker ps -a`, and the container then can be deleted using `docker rm container_id`. This last command also deletes the container storage.

Preparing the Host Directory

The Docker daemon can be requested to bind mount a host directory inside a running container. The host directory is seen by the containerized application as part of the container storage, much like a remote network volume is seen by applications as if it were part of the host file system. But these host directory contents are not reclaimed after the container is stopped, and it can be bind mounted to new containers whenever needed.

For example, a database container could be started using a host directory to store database files. If this database container dies, a new container can be created using the same host directory, keeping the database data available to client applications. To the database container, it does not matter where this host directory is stored from the host point of view; it could be anything from a local hard disk partition to a remote networked file system.

A container runs as a host operating system process, under a host operating system user and group ID, so the host directory needs to be configured with ownership and permissions allowing access to the container. In RHEL, the host directory also needs to be configured with the appropriate SELinux context, which is `svirt_sandbox_file_t`.

One way to set up the host directory is:

1. Create a directory with owner and group `root` (notice the root prompt #):

```
# mkdir /var/dbfiles
```

2. The container user must be capable of writing files on the directory. If the host machine does not have the container user, the permission should be defined with the numeric user ID (UID) from the container. In case of the mysql service provided by Red Hat, the UID is 27:

```
# chown -R 27:27 /var/dbfiles
```

3. Allow containers (and also virtual machines) access to the directory:

```
# chcon -t svirt_sandbox_file_t /var/dbfiles
```

The host directory has to be configured **before** starting the container using it.

Mounting a Volume

After creating and configuring the host directory, the next step is to mount this directory to a container. To bind mount a host directory to a container, add the `-v` option to the `docker run` command, specifying the host directory path and the container storage path, separated by a colon (:).

For example, to use the **/var/dbfiles** host directory for MySQL server database files, which are expected to be under **/var/lib/mysql** inside a MySQL container image named **mysql**, use the following command:

```
# docker run -v /var/dbfiles:/var/lib/mysql mysql # other options required by the  
MySQL image omitted
```

In the previous command, if the **/var/lib/mysql** already exists inside the **mysql** container image, the **/var/dbfiles** mount overlays but does not remove the content from the container image. If the mount is removed, the original content is accessible again.

► GUIDED EXERCISE

PERSISTING A MYSQL DATABASE

In this lab, you will create a container that persists the MySQL database data into a host directory.

OUTCOMES

You should be able to deploy a persistent database.

BEFORE YOU BEGIN

The workstation should not have any container images running. To achieve this goal, run from a terminal window the command:

```
[student@workstation ~]$ lab persist-mysqldb setup
```

- 1. Create a directory with the correct permissions.

- 1.1. Open a terminal window from the **workstation** VM (Applications → System Tools → Terminal) and run the following command:

```
[student@workstation ~]$ sudo mkdir -p /var/local/mysql
```

- 1.2. Apply the appropriate SELinux context to the mount point.

```
[student@workstation ~]$ sudo chcon -R -t svirt_sandbox_file_t \
/var/local/mysql
```

- 1.3. Change the owner of the mount point to the mysql user and mysql group:

```
[student@workstation ~]$ sudo chown -R 27:27 /var/local/mysql
```



NOTE

The container user must be capable of writing files in the directory. If the host machine does not have the container user, set the permission to the numeric user ID (UID) from the container. In case of the **mysql** service provided by Red Hat, the UID is 27.

- 2. Create a MySQL container instance with persistent storage.

- 2.1. Pull the MySQL container image from the internal registry:

```
[student@workstation ~]$ docker pull rhel/mysql-57-rhel17
```

- 2.2. Create a new container specifying the mount point to store the MySQL database data:

```
[student@workstation ~]$ docker run --name persist-mysqldb \
-d -v /var/local/mysql:/var/lib/mysql/data \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
rhscl/mysql-57-rhel7
```

This command mounts the host **/var/local/mysql** directory in the container **/var/lib/mysql/data** directory. The **/var/lib/mysql/data** is the directory where the MySQL database stores the data.

- 2.3. Verify that the container was started correctly. Run the following command:

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

An output similar to the following will be listed:

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
8d6acfaa55a5 seconds ago	rhscl/mysql-57-rhel7 3306/tcp	"container-entrypoint" persist-mysqldb	11 seconds ago 9

- 3. Verify that the **/var/local/mysql** directory contains an **items** directory:

```
[student@workstation ~]$ ls -l /var/local/mysql
total 41032
-rw-r-----. 1 27 27      56 Jun 13 07:50 auto.cnf
-rw-----. 1 27 27     1676 Jun 13 07:50 ca-key.pem
-rw-r--r--. 1 27 27    1075 Jun 13 07:50 ca.pem
-rw-r--r--. 1 27 27    1079 Jun 13 07:50 client-cert.pem
-rw-----. 1 27 27    1680 Jun 13 07:50 client-key.pem
-rw-r-----. 1 27 27      2 Jun 13 07:51 e89494e64d5b.pid
-rw-r-----. 1 27 27    349 Jun 13 07:51 ib_buffer_pool
-rw-r-----. 1 27 27 12582912 Jun 13 07:51 ibdata1
-rw-r-----. 1 27 27 8388608 Jun 13 07:51 ib_logfile0
-rw-r-----. 1 27 27 8388608 Jun 13 07:50 ib_logfile1
-rw-r-----. 1 27 27 12582912 Jun 13 07:51 ibtmp1
drwxr-x---. 2 27 27      20 Jun 13 07:50 items
drwxr-x---. 2 27 27    4096 Jun 13 07:50 mysql
drwxr-x---. 2 27 27    8192 Jun 13 07:50 performance_schema
-rw-----. 1 27 27    1680 Jun 13 07:50 private_key.pem
-rw-r--r--. 1 27 27     452 Jun 13 07:50 public_key.pem
-rw-r--r--. 1 27 27    1079 Jun 13 07:50 server-cert.pem
-rw-----. 1 27 27    1676 Jun 13 07:50 server-key.pem
drwxr-x---. 2 27 27    8192 Jun 13 07:50 sys
```

This directory persists data related to the **items** database that was created by this container. If this directory is not available, the mount point was not defined correctly in the container creation.

- 4. Verify if the database was correctly set up. Run the following from a terminal window:

```
[student@workstation ~]$ lab persist-mysqldb grade
```

- 5. Delete the containers and resources created by this lab.

5.1. Stop the running container, by running the following commands:

```
[student@workstation ~]$ docker stop persist-mysqldb
```

5.2. Remove the container data by running the following commands:

```
[student@workstation ~]$ docker rm persist-mysqldb
```

5.3. Remove the container image by running the following command:

```
[student@workstation ~]$ docker rmi rhsc1/mysql-57-rhel7
```

5.4. Delete the volume directory created earlier in the exercise by running the following command:

```
[student@workstation ~]$ sudo rm -rf /var/local/mysql
```

This concludes the guided exercise.

ACCESSING DOCKER NETWORKS

OBJECTIVES

After completing this section, students should be able to:

- Describe the basics of networking with containers.
- Connect to services within a container remotely.

INTRODUCING NETWORKING WITH CONTAINERS

By default, the Docker engine uses a bridged network mode, which through the use of **iptables** and NAT, allows containers to connect to the host machines network.

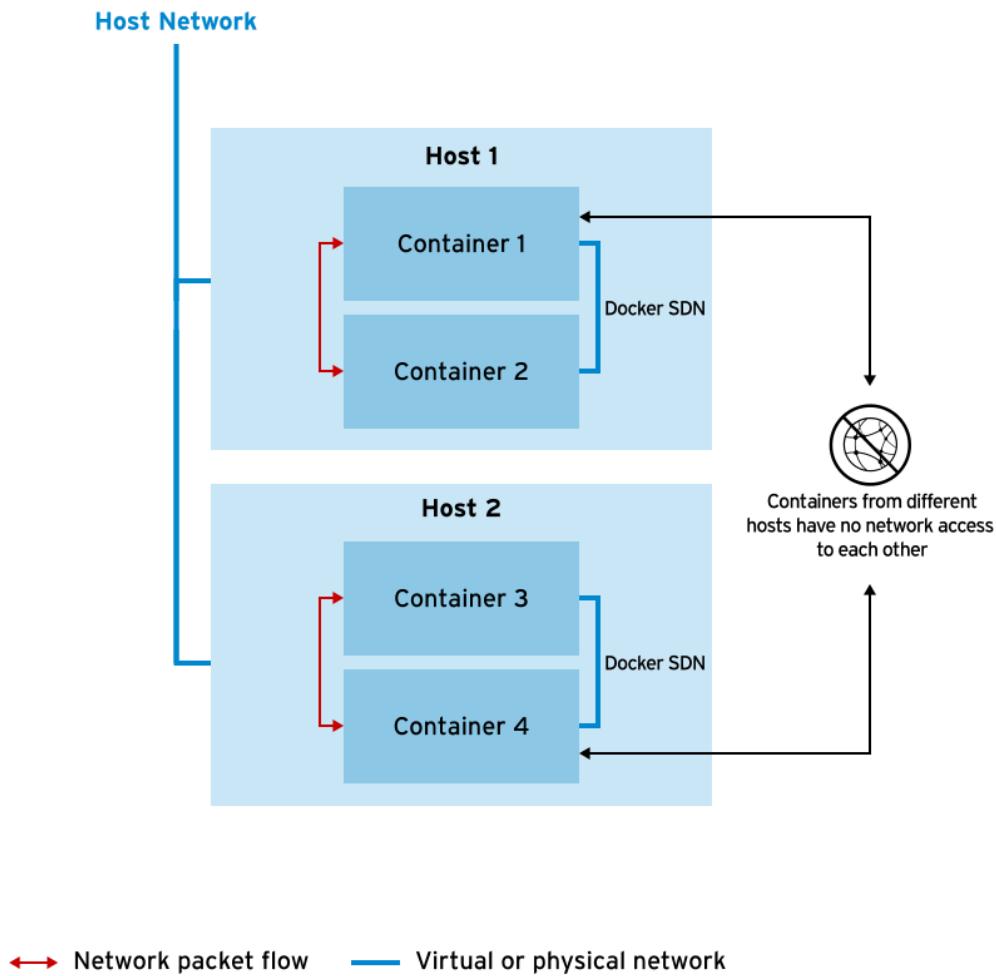


Figure 3.4: Basic Linux containers networking

Each container gets a networking stack, and Docker provides a bridge for these containers to communicate using a virtual switch. Containers running on a shared host each have a unique IP address and containers running on different Docker hosts can share an IP address. Moreover, all containers that connect to the same bridge on the same host can communicate with each other freely by IP address.

It is also important to note that by default all container networks are hidden from the real network. That is, containers typically can access the network outside, but without explicit configuration, there is no access back into the container network.

MAPPING NETWORK PORTS

Accessing the container from the external world can be a challenge. It is not possible to specify the IP address for the container that will be created, and the IP address changes for every new container. Another problem is that the container network is only accessible by the container host.

To solve these problems, it is possible to use the container host network model combined with network address translation (NAT) rules to allow the external access. To achieve this, use the **-p** option should be used:

```
# docker run -d --name httpd -p 8080:80 do276/httpd
```

In the previous example, requests received by the container host on port 8080 from any IP address will be forwarded to port 80 in the container.

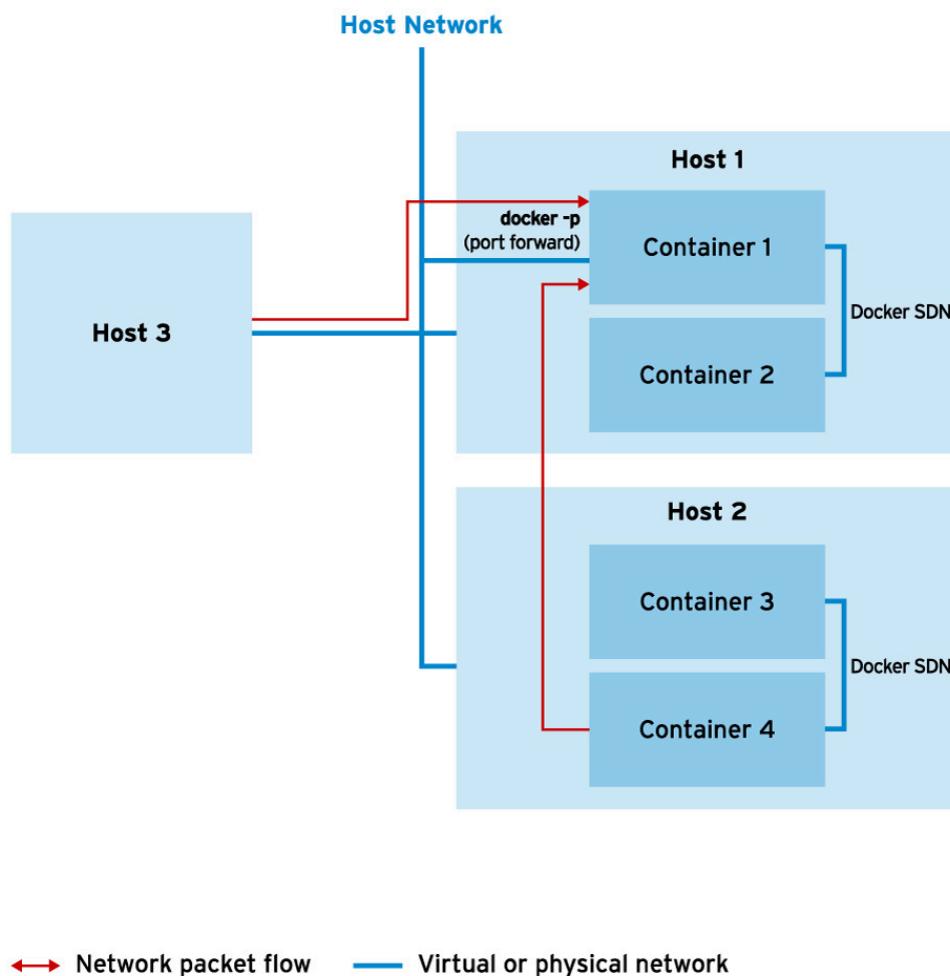


Figure 3.5: Allowing external accesses to Linux containers

It is also possible to specify an IP address for the port forward:

```
# docker run -d --name httpd -p 192.168.1.5:8080:80 do276/httpd
```

If a port is not specified for the host port, the container picks a random available port:

```
# docker run -d --name httpd -p 192.168.1.5::80 do276/httpd
```

Finally, it is possible to listen on all interfaces and have an available port picked automatically:

```
# docker run -d --name httpd -p 80 do276/httpd
```

► GUIDED EXERCISE

LOADING THE DATABASE

In this lab, you will create a MySQL database container. You forward ports from the container to the host in order to load the database with a SQL script.

RESOURCES

Files:	NA
Application URL:	NA
Resources:	RHSCl MySQL 5.7 image (rhscl/mysql-57-rhel7)

OUTCOMES

You should be able to deploy a database container and load a SQL script.

BEFORE YOU BEGIN

The workstation should have a directory to persist data from the database container. To ensure these requirements are supported by the workstation, the setup script creates the necessary directory. Run the following command from a terminal window:

```
[student@workstation ~]$ lab load-mysqldb setup
```

- 1. Create a MySQL container instance with persistent storage and port forward:

```
[student@workstation ~]$ docker run --name mysqldb-port \
-d -v /var/local/mysql:/var/lib/mysql/data \
-p 13306:3306 \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
rhscl/mysql-57-rhel7
```

The **-p** parameter is responsible for the port forward. In this case, every connection on the host IP using the port 13306 is forwarded to this container in port 3306.



NOTE

The **/var/local/mysql** directory was created and configured by the setup script to have the permissions required by the containerized database.

- 2. Verify that the container was started correctly. Run the following command:

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

An output similar to the following will be listed. Look at the **PORTS** column and see the port forward.

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
ad697775565b	rhscl/mysql-5.7-rhel7	"container-entrypoint"	4 seconds ago
Up 2 seconds	0.0.0.0:13306->3306/tcp	mysqlldb-port	

- 3. Load the database:

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
-P13306 items < /home/student/DO285/labs/load-mysql ldb/db.sql
```

- 4. Verify that the database was successfully loaded:

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
-P13306 items -e "SELECT * FROM Item"
```

An output similar to the following will be listed:

```
+-----+-----+
| id | description      | done |
+-----+-----+
| 1  | Pick up newspaper |  0  |
| 2  | Buy groceries     |  1  |
+-----+-----+
```

- 5. Another way to verify that the database was successfully loaded is by running the **mysql** command inside the container. To do that, access the container **bash**:

```
[student@workstation ~]$ docker exec -it mysql ldb-port /bin/bash
```

- 6. Verify that the database contains data:

```
bash-4.2$ mysql -uroot items -e "SELECT * FROM Item"
```

- 7. Exit from the **bash** shell inside the container:

```
bash-4.2$ exit
```

- 8. There is a third option to verify that the database was successfully loaded. It is possible to inject a process into the container to check if the database contains data:

```
[student@workstation ~]$ docker exec -it mysql ldb-port \
```

```
/opt/rh/rh-mysql57/root/usr/bin/mysql -uroot items -e "SELECT * FROM Item"
```

**NOTE**

The **mysql** command is not in the \$PATH variable and, for this reason, you must use an absolute path.

- ▶ 9. Verify that the databases were correctly set up. Run the following from a terminal window:

```
[student@workstation ~]$ lab load-mysqldb grade
```

- ▶ 10. Delete the container and volume created by this lab.

- 10.1. To stop the container, run the following command:

```
[student@workstation ~]$ docker stop mysqldb-port
```

- 10.2. To remove the data stored by the stopped container, run the following command:

```
[student@workstation ~]$ docker rm mysqldb-port
```

- 10.3. To remove the container image, run the following command:

```
[student@workstation ~]$ docker rmi rhsc1/mysql-57-rhel7
```

- 10.4. To remove the directory with the database data, run the following command:

```
[student@workstation ~]$ sudo rm -rf /var/local/mysql
```

This concludes the guided exercise.

► LAB

MANAGING CONTAINERS

In this lab, you will deploy a container that persists the MySQL database data into a host folder, load the database, and manage the container.

RESOURCES	
Files:	/home/student/D0285/labs/work-containers
Application URL:	NA
Resources:	RHSCl MySQL 5.7 image (rhsc1/mysql-57-rhel7)

OUTCOMES

You should be able to deploy and manage a persistent database using a shared volume. You should also be able to start a second database using the same shared volume and observe that the data is consistent between the two containers as they are using the same directory on the host to store the MySQL data.

BEFORE YOU BEGIN

The workstation should have Docker running already. To verify this and download the necessary files for the lab, run the following command from a terminal window:

```
[student@workstation ~]$ lab work-containers setup
```

1. Create the **/var/local/mysql** directory with the correct permission.
 - 1.1. Create the host folder to store the MySQL database data.
 - 1.2. Apply the appropriate SELinux context to the host folder.
 - 1.3. Change the owner of the host folder to the mysql user (**uid**=27) and mysql group (**gid** = 27).
2. Deploy a MySQL container instance using the following characteristics:
 - **Name:** mysql-1
 - **Run as daemon:** yes
 - **Volume:** from **/var/local/mysql** host folder to **/var/lib/mysql/data** container folder
 - **Container image:** rhsc1/mysql-57-rhel7
 - **Port forward:** no
 - **Environment variables:**
 - **MYSQL_USER:** user1
 - **MYSQL_PASSWORD:** mypa55

- **MYSQL_DATABASE: items**
 - **MYSQL_ROOT_PASSWORD: r00tpa55**
- 2.1. Create and start the container.
 - 2.2. Verify that the container was started correctly.
 3. Load the **items** database using the **/home/student/D0285/labs/work-containers/db.sql** script.
 - 3.1. Get the container IP.
 - 3.2. Load the database.
 - 3.3. Verify that the database was loaded.
 4. Stop the container gracefully.

**NOTE**

This step is very important since a new container will be created sharing the same volume for database data. Having two containers using the same volume can corrupt the database. Do not restart the **mysql-1** container.

5. Create a new container with the following characteristics:
 - **Name:** mysql-2
 - **Run as a daemon:** yes
 - **Volume:** from **/var/local/mysql** host folder to **/var/lib/mysql/data** container folder
 - **Container image:** **rhsc1/mysql-57-rhel7**
 - **Port forward:** yes, from host port 13306 to container port 3306
 - **Environment variables:**
 - **MYSQL_USER:** user1
 - **MYSQL_PASSWORD:** mypa55
 - **MYSQL_DATABASE:** items
 - **MYSQL_ROOT_PASSWORD:** r00tpa55
- 5.1. Create and start the container.
- 5.2. Verify that the container was started correctly.
6. Save the list of all containers (including stopped ones) to the **/tmp/my-containers** file.

7. Access the **bash** shell inside the container and verify that the **items** database and the **Item** table are still available. Confirm also that the table contains data.
 - 7.1. Access the **bash** shell inside the container.
 - 7.2. Connect to the MySQL server.
 - 7.3. List all databases and confirm that the **items** database is available.
 - 7.4. List all tables from the **items** database and verify that the **Item** table is available.
 - 7.5. View the data from the table.
 - 7.6. Exit from the MySQL client and from the container shell.
8. Using the port forward, insert a new **Finished lab** row in the **Item** table.
 - 8.1. Connect to the MySQL database.
 - 8.2. Insert the new **Finished lab** row.
 - 8.3. Exit from the MySQL client.
9. Since the first container is not required any more, remove it from the Docker daemon to release resources.
10. Verify that the lab was correctly executed. Run the following from a terminal window:

```
[student@workstation ~]$ lab work-containers grade
```

11. Delete the containers and resources created by this lab.
 - 11.1. Stop the running container.
 - 11.2. Remove the container storage.
 - 11.3. Remove the container image.
 - 11.4. Remove the file created to store the information about the containers.
 - 11.5. Remove the host directory used by the container volumes.

Cleanup

From **workstation**, run the **lab work-containers cleanup** command to clean up the environment.

```
[student@workstation ~]$ lab work-containers cleanup
```

This concludes the lab.

► SOLUTION

MANAGING CONTAINERS

In this lab, you will deploy a container that persists the MySQL database data into a host folder, load the database, and manage the container.

RESOURCES	
Files:	/home/student/D0285/labs/work-containers
Application URL:	NA
Resources:	RHSCl MySQL 5.7 image (rhscl/mysql-57-rhel7)

OUTCOMES

You should be able to deploy and manage a persistent database using a shared volume. You should also be able to start a second database using the same shared volume and observe that the data is consistent between the two containers as they are using the same directory on the host to store the MySQL data.

BEFORE YOU BEGIN

The workstation should have Docker running already. To verify this and download the necessary files for the lab, run the following command from a terminal window:

```
[student@workstation ~]$ lab work-containers setup
```

1. Create the **/var/local/mysql** directory with the correct permission.
 - 1.1. Create the host folder to store the MySQL database data.
Open a terminal window from the workstation VM (Applications → Utilities → Terminal) and run the following command:


```
[student@workstation ~]$ sudo mkdir -p /var/local/mysql
```
 - 1.2. Apply the appropriate SELinux context to the host folder.


```
[student@workstation ~]$ sudo chcon -R -t svirt_sandbox_file_t \
/var/local/mysql
```
- 1.3. Change the owner of the host folder to the mysql user (**uid=27**) and mysql group (**gid = 27**).


```
[student@workstation ~]$ sudo chown -R 27:27 /var/local/mysql
```

2. Deploy a MySQL container instance using the following characteristics:
 - **Name: mysql-1**

- **Run as daemon:** yes
- **Volume:** from `/var/local/mysql` host folder to `/var/lib/mysql/data` container folder
- **Container image:** `rhscl/mysql-57-rhel7`
- **Port forward:** no
- **Environment variables:**
 - `MYSQL_USER: user1`
 - `MYSQL_PASSWORD: mypa55`
 - `MYSQL_DATABASE: items`
 - `MYSQL_ROOT_PASSWORD: r00tpa55`

2.1. Create and start the container.

```
[student@workstation ~]$ docker run --name mysql-1 \
-d -v /var/local/mysql:/var/lib/mysql/data \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mpa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
rhscl/mysql-57-rhel7
```

2.2. Verify that the container was started correctly.

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

An output similar to the following will be listed:

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
616azfaa55x8	rhscl/mysql-57-rhel7	"container-entrypoint"	11 seconds ago
9 seconds ago	3306/tcp	mysql-1	

3. Load the `items` database using the `/home/student/D0285/labs/work-containers/db.sql` script.

3.1. Get the container IP.

```
[student@workstation ~]$ docker inspect -f \
'{{ .NetworkSettings.IPAddress }}' mysql-1
```

3.2. Load the database.

```
[student@workstation ~]$ mysql -uuser1 -h CONTAINER_IP -pmypa55 items \
< /home/student/D0285/labs/work-containers/db.sql
```

Where `CONTAINER_IP` is the IP address returned by the previous command.

3.3. Verify that the database was loaded.

```
[student@workstation ~]$ mysql -uuser1 -h CONTAINER_IP -pmypa55 items \
```

```
-e "SELECT * FROM Item"
```

4. Stop the container gracefully.

**NOTE**

This step is very important since a new container will be created sharing the same volume for database data. Having two containers using the same volume can corrupt the database. Do not restart the **mysql-1** container.

Stop the container using the following command:

```
[student@workstation ~]$ docker stop mysql-1
```

5. Create a new container with the following characteristics:

- **Name:** mysql-2
- **Run as a daemon:** yes
- **Volume:** from `/var/local/mysql` host folder to `/var/lib/mysql/data` container folder
- **Container image:** `rhsc1/mysql-57-rhel7`
- **Port forward:** yes, from host port 13306 to container port 3306
- **Environment variables:**
 - `MYSQL_USER: user1`
 - `MYSQL_PASSWORD: mypa55`
 - `MYSQL_DATABASE: items`
 - `MYSQL_ROOT_PASSWORD: r00tpa55`

- 5.1. Create and start the container.

```
[student@workstation ~]$ docker run --name mysql-2 \
-d -v /var/local/mysql:/var/lib/mysql/data \
-p 13306:3306 \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
rhsc1/mysql-57-rhel7
```

- 5.2. Verify that the container was started correctly.

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

An output similar to the following will be listed:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					

```
281c0e2790e5 rhsc1/mysql-57-rhel7 "container-entrypoint" 14 seconds ago 11  
seconds ago 0.0.0.0:13306->3306/tcp mysql-2
```

6. Save the list of all containers (including stopped ones) to the `/tmp/my-containers` file.
Save the information with the following command:

```
[student@workstation ~]$ docker ps -a > /tmp/my-containers
```

7. Access the `bash` shell inside the container and verify that the `items` database and the `Item` table are still available. Confirm also that the table contains data.

- 7.1. Access the `bash` shell inside the container.

```
[student@workstation ~]$ docker exec -it mysql-2 /bin/bash
```

- 7.2. Connect to the MySQL server.

```
bash-4.2$ mysql -uroot
```

- 7.3. List all databases and confirm that the `items` database is available.

```
mysql> show databases;
```

- 7.4. List all tables from the `items` database and verify that the `Item` table is available.

```
mysql> use items;  
mysql> show tables;
```

- 7.5. View the data from the table.

```
mysql> SELECT * FROM Item;  
+----+-----+---+  
| id | description | done |  
+----+-----+---+  
| 1 | Pick up newspaper | 0 |  
| 2 | Buy groceries | 1 |  
+----+-----+---+
```

- 7.6. Exit from the MySQL client and from the container shell.

```
mysql> exit  
bash-4.2$ exit
```

8. Using the port forward, insert a new `Finished lab` row in the `Item` table.

- 8.1. Connect to the MySQL database.

```
[student@workstation ~]$ mysql -uuser1 -h workstation.lab.example.com \
```

```
-pmypa55 -P13306 items
```

- 8.2. Insert the new **Finished lab** row.

```
MySQL[items]> insert into Item (description, done) values ('Finished lab', 1);
```

- 8.3. Exit from the MySQL client.

```
MySQL[items]> exit
```

9. Since the first container is not required any more, remove it from the Docker daemon to release resources.

Remove the container with the following command:

```
[student@workstation ~]$ docker rm mysql-1
```

10. Verify that the lab was correctly executed. Run the following from a terminal window:

```
[student@workstation ~]$ lab work-containers grade
```

11. Delete the containers and resources created by this lab.

- 11.1. Stop the running container.

```
[student@workstation ~]$ docker stop mysql-2
```

- 11.2. Remove the container storage.

```
[student@workstation ~]$ docker rm mysql-2
```

- 11.3. Remove the container image.

```
[student@workstation ~]$ docker rmi rhscl/mysql-57-rhel7
```

- 11.4. Remove the file created to store the information about the containers.

```
[student@workstation ~]$ rm /tmp/my-containers
```

- 11.5. Remove the host directory used by the container volumes.

```
[student@workstation ~]$ sudo rm -rf /var/local/mysql
```

Cleanup

From **workstation**, run the **lab work-containers cleanup** command to clean up the environment.

```
[student@workstation ~]$ lab work-containers cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- A set of commands are provided to create and manage containers.
 - **docker run**: Create a new container.
 - **docker ps**: List containers.
 - **docker inspect**: List metadata about a container.
 - **docker stop**: Stop a container.
 - **docker kill**: Stop a container forcefully.
 - **docker restart**: Restart a stopped container.
 - **docker rm**: Delete a container.
- Container storage is said to be ephemeral, meaning its contents are not preserved after the container is removed.
- To work with persistent data, a folder from the host can be used.
- It is possible to mount a volume with the **-v** option in the **docker run** command.
- The **docker exec** command starts an additional process inside a running container.
- A port mapping can be used with the **-p** option in the **docker run** command.

CHAPTER 4

MANAGING CONTAINER IMAGES

GOAL

Manage the life cycle of a container image from creation to deletion.

OBJECTIVES

- Search for and pull images from remote registries.
- Export, import, and manage container images locally and in a registry.

SECTIONS

- Accessing Registries (and Quiz)
- Manipulating Container Images (and Guided Exercise)

LAB

- Managing Container Images

ACCESSING REGISTRIES

OBJECTIVES

After completing this section, students should be able to:

- Search for and pull images from remote registries .
- List the advantages of using a certified public registry to download secure images.
- Customize the **docker** daemon to access alternative container image registries.
- Search for container images using **docker** command and the REST API.
- Pull images from a registry.
- List images downloaded from a registry to the daemon cache.
- Manage tags to pull tagged images.

PUBLIC REGISTRIES

The **docker** daemon searches for and downloads container images from a public registry provided by Docker. Docker Hub is the public registry managed by Docker, and it hosts a large set of container images, including those provided by major open source projects, such as Apache, MySQL, and Jenkins. It also hosts customized container images developed by the community.

Some images provided by the community do not take security concerns into consideration, and might put data or the application running in a production environment at risk, because anyone can get an account and publish custom images.

For example, root-based access containers and security-flawed tools (such as the Bash with ShellShock security vulnerability) might be some of the issues encountered in such containers.

Alternatively, Red Hat also has a public registry where certified and tested container images are available for consumption by customers with a valid Red Hat subscription.

Red Hat container images provide the following benefits:

- *Trusted source*: All container images are built from a known source by Red Hat.
- *Original dependencies*: None of the container packages have been tampered with, and only include known libraries.
- *Vulnerability-free*: Container images are free of known vulnerabilities in the platform components or layers.
- *Red Hat Enterprise Linux (RHEL) compatible*: Container images are compatible with all Red Hat Enterprise Linux platforms, from bare metal to cloud.
- *Red Hat support*: The complete stack is commercially supported by Red Hat.

PRIVATE REGISTRY

Some teams might need to distribute custom container images for internal use. Even though it is possible to use a public registry to make them available for download, a better approach would be to publish them to a private registry. A private registry can be installed as a service on a host for

development purposes. To use this internal registry, developers only need to add it to the list of registries to use by updating the **ADD_REGISTRY** variable.

**NOTE**

The **docker-registry** service installation and customization process is beyond the scope of this course.

To configure extra registries for the **docker** daemon, you need to update the **/etc/sysconfig/docker** file. On a RHEL host, add the following extra parameter:

```
ADD_REGISTRY='--add-registry registry.access.redhat.com \  
--add-registry ① services.lab.example.com:5000 ②'
```

- ① The **--add-registry** parameter requires the registry FQDN host and port.
- ② The FQDN host and port number where the **docker-registry** service is running.

**NOTE**

The **docker** daemon requires a full restart to make them effective by running **systemctl restart docker.service** command.

To access a registry, a secure connection is needed with a certificate. For a closed environment where only known hosts are allowed, the **/etc/sysconfig/docker** file can be customized to support insecure connections from a RHEL host:

```
ADD_REGISTRY='--add-registry ① registry.lab.example.com ②'
```

- ① The **--add-registry** parameter requires the FQDN of the registry. Specify the port if it is different than port 5000.
- ② The FQDN host and port number that the **docker-registry** service uses.

ACCESSING REGISTRIES

A container image registry is accessed via the Docker daemon service from the **docker** command. Because the **docker** command line uses a RESTful API to request process execution by the daemon, most of the commands from the client are translated into an HTTP request, and can be sent using **curl**.

**NOTE**

This capability can be used to get additional information from the registries and troubleshoot **docker** client problems that are not clearly stated by the logs.

Searching for Images in Registries

The subcommand **search** is provided by the **docker** command to find images by image name, user name, or description from all the registries listed in the **/etc/sysconfig/docker** configuration file. The syntax for the verb is:

```
# docker search [OPTIONS] <term>
```

The following table shows the options available for the **search** verb to limit the output from the command:

OPTION	DESCRIPTION
--automated=true	List only automated builds, where the image files are built using a Dockerfile.
--no-trunc=true	Do not truncate the output.
--stars=N	Display images with at least N stars provided by users from the registry.

**NOTE**

The command returns up to 25 results from the registry and does not display which tags are available for download.

To overcome the limitations of the **search** verb, the RESTful API can be used instead.

**NOTE**

To send an HTTP request to a container registry, a tool with HTTP support should be used, such as **curl** or a web browser.

**NOTE**

Images are stored in collections, known as a *repositories*, as seen throughout the API specification. Registry instances may contain several repositories.

To customize the number of container images listed from a registry, a parameter called **n** is used to return a different number of images:

```
GET /v2/_catalog?n=<number>
```

For example, to get the list of images present in the registry, run the following **curl** command:

```
# curl https://registry.lab.example.com/v2/_catalog?n=4 | python -m json.tool
```

```
{
  "repositories": [
    "jboss-eap-7/eap70-openshift",
    "jboss-fuse-6/fis-java-openshift",
    "jboss-fuse-6/fis-karaf-openshift",
    "jboss-webserver-3/webserver31-tomcat8-openshift"
  ]
}
... output omitted ...
```

Searching for Image Tags in Registries

To get the tags from any image, use the RESTful API. The HTTP request must be similar to:

```
GET /v2/repository name/tags/list
```

The following procedure describes the required steps for authenticating with the V2 API of the official Docker registry.

1. Define variables for your user name and password.

```
# UNAME="username"
# UPASS="password"
```

2. Retrieve the authentication token. Pass the user name and password with the **POST** method.

```
# curl -s -H "Content-Type: application/json" \
-X POST -d '{"username": "'${UNAME}'", "password": "'${UPASS}'"}' \
https://hub.docker.com/v2/users/login/ | jq -r .token
```

3. Retrieve the list of repositories that you have access to. The **Authorization: JWT** header is used for authenticating the request.

```
# curl -s -H "Authorization: JWT token" \
https://hub.docker.com/v2/repositories/user name?page_size=10000 | \
jq -r '.results|[[]].name'
```

4. You can also retrieve a list of all images and tags by iterating on the output of the previous command.

```
# repository=$(curl -s -H "Authorization: JWT token" \
https://hub.docker.com/v2/repositories/user name?page_size=10000 | \
jq -r '.results|[[]].name')

# for image in ${repository}; do \
curl -s -H "Authorization: JWT token" \
https://hub.docker.com/v2/repositories/user name/${image}/tags/?page_size=100 | \
jq -r '.results|[[]].name')
```

Searching for Images in Private Registries

You can use the **docker** command with the **search** option to browse a Docker registry. Public registries, such as `registry.access.redhat.com` support the **search** verb, as they are exposing the version 1 of the API. The **search** verb does not work in this classroom because the registry exposes the version 2 of the API.

To use the search feature in registries running the version 2 of the API, a Python script can be used. The script is provided as an open source project hosted at <https://github.com> and it is referred in the References section.

In the classroom environment, the script is provided as a bash script named **docker-registry-cli**. The script can list and search all the repositories available. It also supports HTTPS and basic authentication. To get all the images available at a private registry, use the following syntax:

```
# docker-registry-cli <docker-registry-host>:<port> <list|search> [options] [ssl]
```

For example, to get the list of all images available at `registry.lab.example.com`:

```
# docker-registry-cli registry.lab.example.com list all ssl
-----
1) Name: rhscl/postgresql-95-rhel7
Tags: latest
-----
2) Name: openshift3/container-engine
Tags: latest v3.9 v3.9.14
-----
... output omitted ...
```

To search for a specific string, use the following command:

```
# docker-registry-cli registry.lab.example.com search mysql ssl
-----
1) Name: rhscl/mysql-57-rhel7
Tags: 5.7-3.14 latest
-----
2) Name: openshift3/mysql-55-rhel7
Tags: latest
-----
3) Name: rhscl/mysql-56-rhel7
Tags: latest
... output omitted ...
```

Pulling Images

To pull container images from a registry, use the **docker** command with the **pull** option.

```
# docker pull [OPTIONS] NAME[:TAG] | [REGISTRY_HOST[:REGISTRY_PORT]/]NAME[:TAG]
```

The following table shows the options available for the **pull** verb:

OPTION	DESCRIPTION
--all-tags=true	Download all tagged images in the repository.
--disable-content-trust=true	Skip image verification.

To pull an image from a registry, **docker pull** will use the image name obtained from the **search** verb. The **docker pull** command supports the fully qualified domain name to identify from which registry the image should be pulled. This option is supported because multiple registries can be used by **docker** for searching purposes, and the same image name can be used by multiple registries for different images.

For example, to pull an NGINX container from the docker .io registry, use the following command:

```
# docker pull docker.io/nginx
```



NOTE

If no registry is provided, the first registry listed in the **/etc/sysconfig/docker** configuration file from the **ADD_REGISTRY** line is used.

Listing Cached Copies of Images

Any image files pulled from a registry are stored on the same host where the **docker** daemon is running to avoid multiple downloads, and to minimize the deployment time for a container. Moreover, any custom container image built by a developer is saved to the same cache. To list all the container images cached by the daemon, **docker** provides a verb called **images**.

```
# docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
docker.io/httpd  latest  1c0076966428  4 weeks ago  193.4 MB
```



NOTE

The image files are stored in the **/var/lib/docker** directory from the **docker** daemon's host if the default configuration is used. If LVM thin storage is used to store images, the LVM volume mount point is used instead.

Image Tags

An image tag is a mechanism from the **docker-registry** service to support multiple releases of the same project. This facility is useful when multiple versions of the same software are provided, such as a production-ready container or the latest updates of the same software developed for community evaluation. Any operation where a container image is requested from a registry accepts a tag parameter to differentiate between multiple tags. If no tag is provided, then **latest** is used, which indicates the latest build of the image that has been published. For example, to pull an image with the tag **5.5** from **mysql**, use the following command:

```
# docker pull mysql:5.6
```

To start a new container based on the **mysql:5.6** image, use the following command:

```
# docker run mysql:5.6
```



REFERENCES

Docker Hub website

<https://hub.docker.com>

The Docker Registry API documentation

https://docs.docker.com/v1.6/reference/api/registry_api/#search

Docker registry v2 CLI

https://github.com/vivekjuneja/docker_registry_cli/

Docker remote API documentation

https://docs.docker.com/engine/reference/api/docker_remote_api/

Red Hat Container Catalog

<https://registry.access.redhat.com>

Red Hat container certification program website

<https://connect.redhat.com/zones/containers/why-certify-containers>

Setting up a docker-registry container

<https://docs.docker.com/registry/deploying/>

► QUIZ

WORKING WITH REGISTRIES

Choose the correct answers to the following questions, based on the following information:

A docker daemon is installed on a RHEL host with the following `/etc/sysconfig/docker` file:

```
ADD_REGISTRY="--add-registry registry.access.redhat.com --add-registry
docker.io"
```

The `registry.access.redhat.com` and `docker.io` hosts have a registry running, both have valid certificates, and use the version 1 registry. The following images are available for each host:

- `registry.access.redhat.com`:

image names/tags:

- `nginx/1.0`
- `mysql/5.6`
- `httpd/2.2`

- `docker.io`:

image names/tags:

- `mysql/5.5`
- `httpd/2.4`

No images were downloaded by the daemon.

► 1. Which two commands search for the `mysql` image available for download from `registry.access.redhat.com`? (Select two.)

- a. `docker search registry.access.redhat.com/mysql`
- b. `docker images`
- c. `docker pull mysql`
- d. `docker search mysql`

► 2. Which command is used to list all the available image tags from the `httpd` container image?

- a. `docker search httpd`
- b. `docker images httpd`
- c. `docker pull --all-tags=true httpd`
- d. There is no docker command available to search for tags.

► 3. Which two commands pull the httpd image with the 2.2 tag? (Select two.)

- a. `docker pull httpd:2.2`
- b. `docker pull httpd:latest`
- c. `docker pull docker.io/httpd`
- d. `docker pull registry.access.redhat.com/httpd:2.2`

► 4. After running the following commands, what is the output of the `docker images` command?

```
docker pull registry.access.redhat.com/httpd:2.2  
docker pull docker.io/mysql:5.6
```

a. Option 1:

REPOSITORY	TAG
docker.io/httpd	2.2
registry.access.redhat.com/mysql	5.6

b. Option 2:

REPOSITORY	TAG
registry.access.redhat.com/httpd	2.2
registry.access.redhat.com/mysql	5.6

c. Option 3:

REPOSITORY	TAG
registry.access.redhat.com/httpd	2.2

d. Option 4:

REPOSITORY	TAG
docker.io/httpd	2.2

► SOLUTION

WORKING WITH REGISTRIES

Choose the correct answers to the following questions, based on the following information:

A docker daemon is installed on a RHEL host with the following `/etc/sysconfig/docker` file:

```
ADD_REGISTRY="--add-registry registry.access.redhat.com --add-registry
docker.io"
```

The `registry.access.redhat.com` and `docker.io` hosts have a registry running, both have valid certificates, and use the version 1 registry. The following images are available for each host:

- `registry.access.redhat.com`:

image names/tags:

- `nginx/1.0`
- `mysql/5.6`
- `httpd/2.2`

- `docker.io`:

image names/tags:

- `mysql/5.5`
- `httpd/2.4`

No images were downloaded by the daemon.

► 1. Which two commands search for the `mysql` image available for download from `registry.access.redhat.com`? (Select two.)

- a. `docker search registry.access.redhat.com/mysql`
- b. `docker images`
- c. `docker pull mysql`
- d. `docker search mysql`

► 2. Which command is used to list all the available image tags from the `httpd` container image?

- a. `docker search httpd`
- b. `docker images httpd`
- c. `docker pull --all-tags=true httpd`
- d. There is no docker command available to search for tags.

► 3. Which two commands pull the httpd image with the 2.2 tag? (Select two.)

- a. `docker pull httpd:2.2`
- b. `docker pull httpd:latest`
- c. `docker pull docker.io/httpd`
- d. `docker pull registry.access.redhat.com/httpd:2.2`

► 4. After running the following commands, what is the output of the `docker images` command?

```
docker pull registry.access.redhat.com/httpd:2.2  
docker pull docker.io/mysql:5.6
```

a. Option 1:

REPOSITORY	TAG
docker.io/httpd	2.2
registry.access.redhat.com/mysql	5.6

b. Option 2:

REPOSITORY	TAG
registry.access.redhat.com/httpd	2.2
registry.access.redhat.com/mysql	5.6

c. Option 3:

REPOSITORY	TAG
registry.access.redhat.com/httpd	2.2

d. Option 4:

REPOSITORY	TAG
docker.io/httpd	2.2

MANIPULATING CONTAINER IMAGES

OBJECTIVES

After completing this section, students should be able to:

- Export, import, and manage container images locally and in a registry.
- Create a new container image using the **commit** command.
- Identify the changed artifacts in a container.
- Manage image tags for distribution purposes.

INTRODUCTION

There are various ways to manage image containers in a devops fashion. For example, a developer finished testing a custom container in a machine, and needs to transfer this container image to another host for another developer, or to a production server. There are two ways to accomplish this:

1. Save the container image to a **.tar** file.
2. Publish (*push*) the container image to an image registry.



NOTE

One of the ways a developer could have created this custom container is discussed later in this chapter (**docker commit**). However, the recommended way to do so, that is, using **Dockerfiles** is discussed in next chapters.

SAVING AND LOADING IMAGES

Existing images from the Docker cache can be saved to a **.tar** file using the **docker save** command. The generated file is not a regular **tar** file: it contains image metadata and preserves the original image layers. By doing so, the original image can be later recreated exactly as it was.

The general syntax of the **docker** command **save** verb is as follows::

```
# docker save [-o FILE_NAME] IMAGE_NAME[:TAG]
```

If the **-o** option is not used the generated image is sent to the standard output as binary data.

In the following example, the MySQL container image from the Red Hat Container Catalog is saved to the **mysql.tar** file:

```
# docker save -o mysql.tar registry.access.redhat.com/rhscl/mysql-57-rhel7
```

.tar files generated using the **save** verb can be used for backup purposes. To restore the container image, use the **docker load** command. The general syntax of the command is as follows:

```
# docker load [-i FILE_NAME]
```

If the **.tar** file given as an argument is not a container image with metadata, the **docker load** command fails.

An image previously saved can be restored to the Docker cache by using the **load** verb.

```
# docker load -i mysql.tar
```



NOTE

To save disk space, the file generated by the **save** verb can be compressed as a Gzip file. The **load** verb uses the **gunzip** command before importing the file to the cache directory.

PUBLISHING IMAGES TO A REGISTRY

To publish an image to the registry, it must be stored in the Docker's cache cache, and should be tagged for identification purposes. To tag an image, use the **tag** verb, as follows:

```
# docker tag IMAGE[:TAG] [REGISTRYHOST/]USERNAME/]NAME[:TAG]
```

For example, to tag the **nginx** image with the **latest** tag, run the following command:

```
# docker tag nginx nginx
```

To push the image to the registry, use the **push** verb, as follows:

```
# docker push nginx
```

DELETING IMAGES

Any image downloaded to the Docker cache is kept there even if it is not used by any container. However, images can become outdated, and should be subsequently replaced.



NOTE

Any updates to images in a registry are not automatically updated. The image must be removed and then pulled again to guarantee that the cache has the latest version of an image.

To delete an image from the cache, run the **docker rmi** command. The syntax for this command is as follows:

```
# docker rmi [OPTIONS] IMAGE [IMAGE...]
```

The major option available for the **rmi** subcommand is **--force=true**. The option forces the removal of an image. An image can be referenced using its name or its ID for removal purposes.

A single image might use more than one tag, and using the **docker rmi** command with the image ID will fail in this case. To avoid removing each tag individually, the simplest approach is to use the **--force** option.

Any container using the image will block any attempt to delete an image. All the containers using that image must be stopped and removed before it can be deleted.

DELETING ALL IMAGES

To delete all images that are not used by any container, use the following command:

```
# docker rmi $(docker images -q)
```

The command returns all the image IDs available in the cache, and passes them as a parameter to the **docker rmi** command for removal. Images that are in use are not deleted, however, this does not prevent any unused images from being removed.

MODIFYING IMAGES

Ideally, all container images should be built using a **Dockerfile**, in order to create a clean lightweight set of image layers without log files, temporary files, or other artifacts created by the container customization. However, some container images may be provided as they are, without any **Dockerfile**. As an alternative approach to creating new images, a running container can be changed in place and its layers saved to create a new container image. This feature is provided by the **docker commit** command.



WARNING

Even though the **docker commit** command is the simplest approach to creating new images, it is not recommended because of the image size (logs and process ID files are kept in the captured layers during the **commit** execution), and the lack of change traceability. A **Dockerfile** provides a robust mechanism to customize and implement changes to a container using a readable set of commands, without the set of files that are generated by the operating system.

The syntax for the **docker commit** command is as follows:

```
# docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

The following table shows the important options available for the **docker commit** command:

OPTION	DESCRIPTION
--author=""	Identifies who created the container image.
--message=""	Includes a commit message to the registry.

To identify a running container in Docker, run the **docker ps** command:

```
# docker ps
CONTAINER ID IMAGE COMMAND                  CREATED        STATUS          PORTS
 NAMES
87bdfcc7c656 mysql "/entrypoint.sh mysql" 14 seconds ago Up 13 seconds 3306/tcp
mysql-basic
```

Eventually, administrators might customize the image and set the container to the desired state. To identify which files were changed, created, or deleted since the container was started, use the **diff** verb. The verb only requires the container name or container ID:

```
# docker diff mysql-basic
C /run
C /run/mysqld
A /run/mysqld/mysqld.pid
A /run/mysqld/mysqld.sock
A /run/mysqld/mysqld.sock.lock
A /run/secrets
```

Any added file is tagged with an **A**, and any changed file is tagged with a **C**.

To commit the changes to another image, run the following command:

```
# docker commit mysql-basic mysql-custom
```

TAGGING IMAGES

A project with multiple images based on the same software could be distributed, creating individual projects for each image; however, this approach requires more maintenance for managing and deploying the images to the correct locations.

Container image registries support tags in order to distinguish multiple releases of the same project. For example, a customer might use a container image to run with a MySQL or PostgreSQL database, using a tag as a way to differentiate which database is to be used by a container image.



NOTE

Usually, the tags are used by container developers to distinguish between multiple versions of the same software. Multiple tags are provided to easily identify a release. On the official MySQL container image website, the version is used as the tag's name (**5.5.16**). In addition, the same image has a second tag with the minor version, for example 5.5, to minimize the need to get the latest release for a certain version.

To tag an image, use the **docker tag** command:

```
# docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/]USERNAME/]NAME[:TAG]
```

The **IMAGE** argument is the image name with an optional tag which is managed by Docker. The following argument refers to alternative names for the image that is stored locally. If the tag value is not provided, Docker assumes the latest version, as indicated by the **latest** tag. For example, to tag an image, use the following command:

```
# docker tag mysql-custom devops/mysql
```

The **mysql-custom** option corresponds to the image name in the Docker registry.

To use a different tag name, use the following command instead:

```
# docker tag mysql-custom devops/mysql:snapshot
```

Removing Tags from Images

To associate multiple tags with a single image, use the `docker tag` command. Tags can be removed by using the `docker rmi` command, as mentioned earlier:

```
# docker rmi devops/mysql:snapshot
```



NOTE

Because multiple tags can point to the same image, to remove an image referred to by multiple tags, each tag should be individually removed first. Alternatively, use the `docker rmi --force` command.

BEST PRACTICES FOR TAGGING IMAGES

The **latest** tag is automatically added by Docker if you do not specify any tag, as Docker consider the image to be the latest build. However, this may not be true depending on how the tags are used. For example, many open source projects consider the **latest** tag to match the most recent release, but not the latest build.

Moreover, multiple tags are provided to minimize the need to remember the latest release of a certain version of a project. Thus, if there is a project version release, for example, **2.1.10**, another tag called **2.1** can be created and pointed to the same image from the **2.1.10** release. This simplifies how the image is pulled from the registry.



REFERENCES

Docker documentation

<https://docs.docker.com/>

► GUIDED EXERCISE

CREATING A CUSTOM APACHE CONTAINER IMAGE

In this guided exercise, you will create a custom Apache container image using the **docker commit** command.

OUTCOMES

You should be able to create a custom container image.

BEFORE YOU BEGIN

To set up the environment for the exercise, open a terminal and run the following command. The script ensures that the Docker daemon is running on the **workstation** VM.

```
[student@workstation ~]$ lab container-images-manipulating setup
```

- ▶ 1. Open a terminal on **workstation** (Applications → System Tools → Terminal) and start a container by using the image available at **centos/httpd**. The **-p** option allows you to specify a redirect port. In this case, Docker forwards incoming request on TCP port 8180 to the TCP port 80.

```
[student@workstation ~]$ docker run -d \
--name official-httdp -p 8180:80 \
centos/httpd
... output omitted ...
Digest: sha256:35f2b43891a7ebfa5330ef4c736e171c42380aec95329d863dcde0e608ffff1e
Status: Downloaded newer image for registry.lab.example.com/centos/httpd:latest
e37d6532638fe85b8d92ef8a60dcfb22bf154d6f6852dd4c4baef670f3d11f4a
```

- ▶ 2. Create an HTML page on the **official-httdp** container.
- 2.1. Access the shell of the container by using the **exec** option and create an HTML page.

```
[student@workstation ~]$ docker exec -it official-httdp /bin/bash
[root@3e1ab7d3aa94 /]# echo "DO285 Page" > /var/www/html/do285.html
```

- 2.2. Exit the container.

```
[root@3e1ab7d3aa94 /]# exit
```

- 2.3. Ensure that the HTML file is reachable from the **workstation** VM by using the **curl** command.

```
[student@workstation ~]$ curl 127.0.0.1:8180/do285.html
```

You should see the following output:

DO285 Page

- 3. Examine the differences in the container between the image and the new layer created by the container. To do so, use the **diff** option.

```
[student@workstation ~]$ docker diff official-httdp
```

```
C /root
A /root/.bash_history
C /run
C /run/httpd
A /run/httpd/authdigest_shm.7
D /run/httpd/htcacheclean
A /run/httpd/httpd.pid
D /run/secrets
C /tmp
C /var
C /var/log
C /var/log/httpd
A /var/log/httpd/access_log
A /var/log/httpd/error_log
C /var/www
C /var/www/html
A /var/www/html/do285.html
```

The previous output lists the directories and files that were changed or added to the **official-httdp** container. Remember that these changes are only for this container.

- 4. Create a new image with the changes created by the running container.

4.1. Stop the **official-httdp** container.

```
[student@workstation ~]$ docker stop official-httdp
official-httdp
```

4.2. Commit the changes to a new container image. Use your name as the author of the changes.

```
[student@workstation ~]$ docker commit \
-a 'Your Name' \
-m 'Added do285.html page' \
official-httdp
sha256:6ad5919cb2cf89843c2c15b429c2eab29358887421d97f5e35a6b7fa35576888
```

4.3. List the available container images.

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

The expected output is similar to the following:

REPOSITORY	TAG
<none>	<none>

registry.lab.example.com/centos/httpd		latest
IMAGE ID	CREATED	SIZE
6ad5919cb2cf	25 seconds ago	305 MB
1c0961bdb0f3	4 weeks ago	305 MB

The image ID matches the first 12 characters of the hash. Most recent images are listed at the top.

- 4.4. The new container image has neither a name, as listed in the **REPOSITORY** column, nor a tag. Tag the image with a custom name of **do285/custom-htpd**.

```
[student@workstation ~]$ docker tag 6ad5919cb2cf do285/custom-htpd
```



NOTE

The container image ID, **6ad5919cb2cf**, is the truncated version of the ID returned by the previous step.

- 4.5. List the available container images again to ensure that the name and tag were applied to the correct image.

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

The expected output is similar to the following:

REPOSITORY		TAG
do285/custom-htpd		latest
registry.lab.example.com/centos/httpd		latest
IMAGE ID	CREATED	SIZE
6ad5919cb2cf	6 minutes ago	305 MB
1c0961bdb0f3	4 weeks ago	305 MB

- 5. Publish the saved container image to the Docker registry.

- 5.1. To tag the image with the registry host name and port, run the following command.

```
[student@workstation ~]$ docker tag do285/custom-htpd \
registry.lab.example.com/do285/custom-htpd:v1.0
```

- 5.2. Run the **docker images** command to ensure that the new name has been added to the cache.

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

REPOSITORY		TAG
do285/custom-htpd		latest
registry.lab.example.com/do285/custom-htpd		v1.0
registry.lab.example.com/centos/httpd		latest
IMAGE ID	CREATED	SIZE
6ad5919cb2cf	8 minutes ago	305 MB
6ad5919cb2cf	8 minutes ago	305 MB

```
1c0961bdb0f3        4 weeks ago      305 MB
```

- 5.3. Publish the image to the private registry, accessible at `registry.lab.example.com`.

```
[student@workstation ~]$ docker push \
registry.lab.example.com/do285/custom-httdp:v1.0
... output omitted ...
8049b1db64b5: Mounted from centos/httdp
v1.0: digest:
sha256:804109820ce13f540f916fabc742f0a241f6a3647723bcd0f4a83c39b26580f7 size:
1368
```



NOTE

Each student only has access to their own private registry, it is therefore impossible that they will interfere with each. Even though these private registries do not require authentication, most public registries require that you authenticate before pushing any image.

- 5.4. Verify that the image is returned by the `docker-registry-cli` command.

```
[student@workstation ~]$ docker-registry-cli \
registry.lab.example.com search custom-httdp ssl
```

available options:-

1) Name: do285/custom-httdp
Tags: v1.0

1 images found !

- ▶ 6. Create a container from the newly published image.

- 6.1. Use the `docker run` command to start a new container. Use `do285/custom-httdp:v1.0` as the base image.

```
[student@workstation ~]$ docker run -d --name test-httdp -p 8280:80 \
do285/custom-httdp:v1.0
5ada5f7e53d1562796b4848928576c1c35df120faee38805a7d1e7e6c43f8e3f
```

- ▶ 7. Create an HTML page on the `test-httdp` container.

- 7.1. Access the shell of the container by using the `exec` option and create an HTML page.

```
[student@workstation ~]$ docker exec -it test-httdp /bin/bash
```

```
[root@081155772ec5 /]# echo "DO285 Page" > /var/www/html/do285.html
```

- 7.2. From the **workstation** VM, use **curl** to access the HTML page. Make sure to use the port 8280.

The HTML page created in the previous step should be displayed.

```
[student@workstation ~]$ curl http://localhost:8280/do285.html  
DO285 Page
```

- 8. Grade your work. From the **workstation** VM, run the **lab container-images-manipulating grade** command.

```
[student@workstation ~]$ lab container-images-manipulating grade
```

- 9. Delete the containers and images created in this exercise.

- 9.1. Use the **docker stop** command to stop the running containers.

```
[student@workstation ~]$ docker stop test-httdp  
test-httdp
```

- 9.2. Remove the container images.

```
[student@workstation ~]$ docker rm official-httdp test-httdp  
official-httdp  
test-httdp
```

- 9.3. Delete the exported container image.

```
[student@workstation ~]$ docker rmi do285/custom-httdp  
Untagged: do285/custom-httdp:latest  
Untagged: registry.lab.example.com/do285/custom-httdp@sha256:(...)
```

Remove the committed container image.

```
[student@workstation ~]$ docker rmi \  
registry.lab.example.com/do285/custom-httdp:v1.0  
Untagged: registry.lab.example.com/do285/custom-httdp:v1.0  
... output omitted ...
```

- 9.4. Remove the **centos/httpd** container image:

```
[student@workstation ~]$ docker rmi centos/httpd  
Untagged: centos/httpd:latest  
... output omitted ...
```

Cleanup

Clean your exercise by running the following command.

```
[student@workstation ~]$ lab container-images-manipulating gradeclean
```

This concludes the guided exercise.

▶ LAB

MANAGING IMAGES

In this lab, you will create and manage container images.

RESOURCES

Application URL:	<code>http://127.0.0.1:8080, http://127.0.0.1:8280</code>
Resources	Nginx image

OUTCOMES

You should be able to create a custom container image and manage container images.

BEFORE YOU BEGIN

To set the environment for this lab, run the following command. The script ensures the Docker daemon is running on **workstation**.

```
[student@workstation ~]$ lab container-images-lab setup
```

- From **workstation**, open a terminal and use the **docker-registry-cli** command to locate the **nginx** image and pull it into your local Docker cache.



NOTE

There are three Nginx images in the repository, and two of them were retrieved from the Red Hat Software Collection Library (*rh scl*). The Nginx container image for this lab is the same as the one that is available at [docker .io](https://hub.docker.com/_/nginx).

Ensure that the image has been successfully retrieved.

- Start a new container using the Nginx image, according to the specifications listed in the following list. Forward the server port 80 to port 8080 from the host using the **-p** option.
 - Name:** `official-nginx`
 - Run as daemon:** yes
 - Container image:** `nginx`
 - Port forward:** from host port 8080 to container port 80.
- Log in to the container using the **exec** verb and update the content `index.html` file with **D0285 Page**. The web server directory is located at `/usr/share/nginx/html`. Once the page is updated, exit the container and use the **curl** command to access the web page.

4. Stop the running container and commit your changes to create a new container image. Give the new image a name of **do285/mynginx** and a tag of **v1.0**. Use the following specifications:
 - Image name: **do285/mynginx**
 - Image tag: **v1.0**
 - Author name: *your name*
 - Commit message: **Changed index.html page**
5. Use the image tagged **do285/mynginx:v1.0** to create a new container with the following specifications:
 - Container name: **my-nginx**
 - Run as daemon: yes
 - Container image: **do285/mynginx:v1.0**
 - Port forward: from host port 8280 to container port 80
- From **workstation**, use the **curl** command to access the web server, accessible from the port 8280.
6. Grade your work. From the **workstation** VM, run the **lab container-images-manipulating grade** command.

```
[student@workstation ~]$ lab container-images-lab grade
```

7. Stop the running container and delete the two containers by using the **rm** option.

Cleanup

Clean your lab by running the following command.

```
[student@workstation ~]$ lab container-images-lab cleanup
```

This concludes the lab.

► SOLUTION

MANAGING IMAGES

In this lab, you will create and manage container images.

RESOURCES

Application URL:	<code>http://127.0.0.1:8080, http://127.0.0.1:8280</code>
Resources	Nginx image

OUTCOMES

You should be able to create a custom container image and manage container images.

BEFORE YOU BEGIN

To set the environment for this lab, run the following command. The script ensures the Docker daemon is running on **workstation**.

```
[student@workstation ~]$ lab container-images-lab setup
```

- From **workstation**, open a terminal and use the **docker-registry-cli** command to locate the **nginx** image and pull it into your local Docker cache.



NOTE

There are three Nginx images in the repository, and two of them were retrieved from the Red Hat Software Collection Library (*rh scl*). The Nginx container image for this lab is the same as the one that is available at [docker .io](https://hub.docker.com/_/nginx).

Ensure that the image has been successfully retrieved.

- Open a terminal on the workstation VM (Applications → System Tools → Terminal) and pull the HTTPd container image.
- Use the **docker-registry-cli** command to search for the Nginx container image **nginx**.

```
[student@workstation ~]$ docker-registry-cli \
registry.lab.example.com search nginx ssl
-----
1) Name: nginx
Tags: latest

1 images found !
```

- Pull the Nginx container image by using the **docker pull** command.

```
[student@workstation ~]$ docker pull nginx
```

```
... output omitted ...
Status: Downloaded newer image for registry.lab.example.com/nginx:latest
```

- 1.4. Ensure that the container image is available in the cache by running the **docker images** command.

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

This command produces output similar to the following:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
registry.lab.example.com/nginx	latest	cd5239a0906a	8 days ago	109 MB

2. Start a new container using the Nginx image, according to the specifications listed in the following list. Forward the server port 80 to port 8080 from the host using the **-p** option.

- **Name:** **official-nginx**
- **Run as daemon:** yes
- **Container image:** **nginx**
- **Port forward:** from host port 8080 to container port 80.

- 2.1. From **workstation**, use the **docker run** command to create a container. Give the container a name of **official-nginx**.

```
[student@workstation ~]$ docker run \
--name official-nginx \
-d -p 8080:80 \
registry.lab.example.com/nginx
02dbc348c7dcf8560604a44b11926712f018b0ac44063d34b05704fb8447316f
```

3. Log in to the container using the **exec** verb and update the content **index.html** file with **DO285 Page**. The web server directory is located at **/usr/share/nginx/html**.

Once the page is updated, exit the container and use the **curl** command to access the web page.

- 3.1. Log in to the container by using the **docker exec** command.

```
[student@workstation ~]$ docker exec -it \
official-nginx /bin/bash
```

```
root@f22c60d901fa:/#
```

- 3.2. Update the **index.html** file located at **/usr/share/nginx/html**. The file should read **D0285 Page**.

```
root@f22c60d901fa:/# echo 'D0285 Page' > /usr/share/nginx/html/index.html
```

- 3.3. Exit the container.

```
root@f22c60d901fa:/# exit
```

- 3.4. Use the **curl** command to ensure that the **index.html** file is updated.

```
[student@workstation ~]$ curl 127.0.0.1:8080
D0285 Page
```

4. Stop the running container and commit your changes to create a new container image. Give the new image a name of **do285/mynginx** and a tag of **v1.0**. Use the following specifications:

- Image name: **do285/mynginx**
- Image tag: **v1.0**
- Author name: *your name*
- Commit message: **Changed index.html page**

- 4.1. Use the **docker stop** to stop the **official-nginx** container.

```
[student@workstation ~]$ docker stop official-nginx
official-nginx
```

- 4.2. Commit your changes to a new container image. Use your name as the author of the changes, with a commit message of **Changed index.html page**.

```
[student@workstation ~]$ docker commit -a 'Your Name' \
-m 'Changed index.html page' \
official-nginx
sha256:d77b234dec1cc96df8d6ce95d395c8fc69664ab70f412557a54fffffebd180d6
```



NOTE

Note the container ID that is returned. This ID will be required for the image tagging.

- 4.3. List the available container images in order to locate your newly created image.

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

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	d77b234dec1c	7 seconds ago	109 MB

```
registry.lab.example.com/nginx    latest cd5239a0906a 8 days ago  109 MB
```

- 4.4. Set the name and tag for the new image. Pass the ID to the **docker tag** command.

```
[student@workstation ~]$ docker tag d77b234dec1c \
do285/mynginx:v1.0
```

 **NOTE**

The *d77b234dec1c* ID corresponds to the truncated version of the ID returned in step 4.3.

5. Use the image tagged **do285/mynginx:v1.0** to create a new container with the following specifications:

- Container name: **my-nginx**
- Run as daemon: yes
- Container image: **do285/mynginx:v1.0**
- Port forward: from host port 8280 to container port 80

From **workstation**, use the **curl** command to access the web server, accessible from the port 8280.

- 5.1. Use the **docker run** command to create the **my-nginx** container, according to the specifications.

```
[student@workstation ~]$ docker run -d \
--name my-nginx \
-p 8280:80 \
do285/mynginx:v1.0
c1cba44fa67bf532d6e661fc5e1918314b35a8d46424e502c151c48fb5fe6923
```

- 5.2. Use the **curl** command to ensure that the **index.html** page is available and returns the custom content.

```
[student@workstation ~]$ curl 127.0.0.1:8280
DO285 Page
```

6. Grade your work. From the **workstation** VM, run the **lab container-images-manipulating grade** command.

```
[student@workstation ~]$ lab container-images-lab grade
```

7. Stop the running container and delete the two containers by using the **rm** option.

7.1. Stop the **my-nginx** container.

```
[student@workstation ~]$ docker stop my-nginx
```

7.2. Delete the two containers.

```
[student@workstation ~]$ docker rm my-nginx official-nginx
```

7.3. Delete the container images.

```
[student@workstation ~]$ docker rmi do285/mynginx:v1.0
```

Cleanup

Clean your lab by running the following command.

```
[student@workstation ~]$ lab container-images-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Registries must be used to pull and push container images to private registries for internal use, or to public registries for outside consumption.
 - The Red Hat Software Collections Library (RHSCl) provides tested and certified images at `registry.access.redhat.com`.
 - Docker daemons support extra registries by editing the `/etc/sysconfig/docker` file and by adding new registries to the `ADD_REGISTRY` variable.
 - In order to support registries with self-signed certificates, add the registry to the `INSECURE_REGISTRY` variable of the `/etc/sysconfig/docker` configuration file.
 - Registries implement a RESTful API to pull, push, and manipulate objects. The API is used by the Docker daemon, but it can also be queried via tools such as `curl`.
 - To search for an image in a public registry, use the `docker search` command.
 - To search for an image in a private registry, use the `docker-registry-cli` command.
 - To pull an image from a registry, use the `docker pull` command.
 - Registries use tags as a mechanism to support multiple image releases.
- The Docker daemon supports export and import procedures for image files using the `docker export`, `docker import`, `docker save`, and `docker load` commands.
 - For most scenarios, using `docker save` and `docker load` command is the preferred approach.
- The Docker daemon cache can be used as a staging area to customize and push images to a registry.
- Docker also supports container image publication to a registry using the `docker push` command.
- Container images from a daemon cache can be removed using the `docker rmi` command.

CHAPTER 5

CREATING CUSTOM CONTAINER IMAGES

GOAL

Design and code a Dockerfile to build a custom container image.

OBJECTIVES

- Describe the approaches for creating custom container images.
- Create a container image using common Dockerfile commands.

SECTIONS

- Design Considerations for Custom Container Images (and Quiz)
- Building Custom Container Images with Dockerfile (and Guided Exercise)

LAB

- Creating Custom Container Images

DESIGN CONSIDERATIONS FOR CUSTOM CONTAINER IMAGES

OBJECTIVES

After completing this section, students should be able to:

- Describe the approaches for creating custom container images.
- Find existing Dockerfiles to use as a starting point for creating a custom container image.
- Define the role played by the Red Hat Software Collections Library (RHSCL) in designing container images from the Red Hat registry.
- Describe the Source-to-Image (S2I) alternative to Dockerfiles.

REUSING EXISTING DOCKERFILES

Two common ways of building a new container image are as follows:

1. Run operating system commands inside a container and then commit the image.
2. Use a Dockerfile that invokes operating system commands and uses an operating system image as the parent.

Both methods involve extra effort when the same runtimes and libraries are required by different application images. There is not much to be done to improve the first option, but the second option can be improved by selecting a better parent image.

Many popular application platforms are already available in public image registries like Docker Hub. It is not trivial to customize an application configuration to follow recommended practices for containers, and so starting from a proven parent image usually saves a lot of work.

Using a high quality parent image enhances maintainability, especially if the parent image is kept updated by its author to account for bug fixes and security issues.

Typical scenarios to create a Dockerfile for building a child image from an existing container image include:

- Add new runtime libraries, such as database connectors.
- Include organization-wide customizations such as SSL certificates and authentication providers.
- Add internal libraries, to be shared as a single image layer by multiple container images for different applications.

Changing an existing Dockerfile to create a new image can also be a sensible approach in other scenarios. For example:

- Trim the container image by removing unused material (such as libraries).
- Lock either the parent image or some included software package to a specific release to lower the risk related to future software updates.

Two sources of container images to use either as parent images or for changing their Dockerfiles are the Docker Hub and the Red Hat Software Collections Library (RHSCL).

WORKING WITH THE RED HAT SOFTWARE COLLECTIONS LIBRARY

The *Red Hat Software Collections Library (RHSCl)*, or simply Software Collections, is Red Hat's solution for developers who need to use the latest development tools, and which usually do not fit the standard RHEL release schedule.

Red Hat Enterprise Linux (RHEL) provides a stable environment for enterprise applications. This requires RHEL to keep the major releases of upstream packages at the same level to prevent API and configuration file format changes. Security and performance fixes are back-ported from later upstream releases, but new features that would break backward-compatibility are not back-ported.

The RHSCl allows software developers to use the latest version without impacting RHEL, because the RHSCl packages do not replace or conflict with default RHEL packages. Default RHEL packages and RHSCl packages are installed side-by-side.



NOTE

All RHEL subscribers have access to the RHSCl. To enable a particular *software collection* for a specific user or application environment (for example, MySQL 5.7, which is named `rh-mysql157`), enable the RHSCl software Yum repositories and follow a few simple steps.

FINDING DOCKERFILES FROM THE RED HAT SOFTWARE COLLECTIONS LIBRARY

The RHSCl is the source of most container images provided by the Red Hat image registry for use by RHEL Atomic Host and OpenShift Container Platform customers.

Red Hat provides the RHSCl Dockerfiles and related sources in the *rhscl-dockerfiles* package available from the RHSCl repository. Community users can get the Dockerfiles for CentOS-based equivalent container images from <https://github.com/sclorg?q=-container>.

Many RHSCl container images include support for *Source-to-Image (S2I)* which is best known as an OpenShift Container Platform feature. Having support for S2I does not affect the usage of these container images with docker.

CONTAINER IMAGES IN RED HAT CONTAINER CATALOG (RHCC)

Mission-critical applications require trusted containers. The *Red Hat Container Catalog* is a repository of reliable, tested, certified, and curated collection of container images built on versions of Red Hat Enterprise Linux (RHEL) and related systems. Container images available through RHCC have undergone a quality assurance process. The components have been rebuilt by Red Hat to avoid known security vulnerabilities. They are upgraded on a regular basis so that they contain the required version of software even when a new image is not yet available. Using RHCC you can browse and search for images, you can access information about each image, such as its version, contents, and usage.

FINDING DOCKERFILES ON DOCKER HUB

The Docker Hub web site is a popular search site for container images. Anyone can create a Docker Hub account and publish container images there. There are no general assurances about quality and security; images on Docker Hub range from professionally supported to one-time experiments. Each image has to be evaluated individually.

After searching for an image, the documentation page might provide a link to its Dockerfile. For example, the first result when searching for **mysql** is the documentation page for the MySQL official image at https://hub.docker.com/_/mysql/.

On that page, the link for the 5.5/Dockerfile image points to the **docker-library** GitHub project, which hosts **Dockerfiles** for images built by the Docker community automatic build system.

The direct URL for the Docker Hub MySQL 5.5 **Dockerfile** tree is <https://github.com/docker-library/mysql/blob/master/5.5/>.

USING THE OPENSHIFT SOURCE-TO-IMAGE TOOL

Source-to-Image (S2I) provides an alternative to using Dockerfiles to create new container images and can be used either as a feature from OpenShift or as the standalone **s2i** utility. S2I allows developers to work using their usual tools, instead of learning Dockerfile syntax and using operating system commands such as **yum**, and usually creates slimmer images, with fewer layers.

S2I uses the following process to build a custom container image for an application:

1. Start a container from a base container image called the *builder image*, which includes a programming language runtime and essential development tools such as compilers and package managers.
2. Fetch the application source code, usually from a Git server, and send it to the container.
3. Build the application binary files inside the container.
4. Save the container, after some clean up, as a new container image, which includes the programming language runtime and the application binaries.

The builder image is a regular container image that follows a standard directory structure and provides scripts that are called during the S2I process. Most of these builder images can also be used as base images for Dockerfiles, outside the S2I process.

The **s2i** command is used to run the S2I process outside of OpenShift, in a Docker-only environment. It can be installed on a RHEL system from the *source-to-image* RPM package, and on other platforms, including Windows and MacOS, from the installers available in the S2I project on GitHub.



REFERENCES

Red Hat Software Collections Library (RHSCl)

<https://access.redhat.com/documentation/en/red-hat-software-collections/>

Red Hat Container Catalog (RHCC)

<https://access.redhat.com/containers/>

RHSCl Dockerfiles on GitHub

<https://github.com/sclorg?q=-container>

Using Red Hat Software Collections Container Images

<https://access.redhat.com/articles/1752723>

Docker Hub

<https://hub.docker.com/>

Docker Library GitHub project

<https://github.com/docker-library>

The S2I GitHub project

<https://github.com/openshift/source-to-image>

► QUIZ

APPROACHES TO CONTAINER IMAGE DESIGN

Choose the correct answers to the following questions:

► 1. Which method for creating container images is recommended by the Docker community? (Choose one.)

- a. Run commands inside basic OS containers, commit the container, and save or export it as a new container image.
- b. Run commands from a Dockerfile and push the generated container image to an image registry.
- c. Create the container image layers manually from tar files.
- d. Run the `docker build` command to process a container image description in YAML format.

► 2. What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)

- a. Requires no additional tools apart from a basic Docker setup.
- b. Creates smaller container images, having fewer layers.
- c. Reuses high-quality builder images.
- d. Automatically updates the child image as the parent image changes (for example, with security fixes).
- e. Creates images compatible with OpenShift, unlike container images created from Docker tools.

► 3. What are the typical scenarios for creating a Dockerfile to build a child image from an existing image (Choose three):

- a. Adding new runtime libraries.
- b. Setting constraints to the container's access to the host machine's CPU.
- c. Including organization-wide customizations, for example, SSL certificates and authentication providers.
- d. Adding internal libraries, to be shared as a single image layer by multiple container images for different applications.

► SOLUTION

APPROACHES TO CONTAINER IMAGE DESIGN

Choose the correct answers to the following questions:

► 1. **Which method for creating container images is recommended by the Docker community? (Choose one.)**

- a. Run commands inside basic OS containers, commit the container, and save or export it as a new container image.
- b. Run commands from a Dockerfile and push the generated container image to an image registry.
- c. Create the container image layers manually from tar files.
- d. Run the `docker build` command to process a container image description in YAML format.

► 2. **What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)**

- a. Requires no additional tools apart from a basic Docker setup.
- b. Creates smaller container images, having fewer layers.
- c. Reuses high-quality builder images.
- d. Automatically updates the child image as the parent image changes (for example, with security fixes).
- e. Creates images compatible with OpenShift, unlike container images created from Docker tools.

► 3. **What are the typical scenarios for creating a Dockerfile to build a child image from an existing image (Choose three):**

- a. Adding new runtime libraries.
- b. Setting constraints to the container's access to the host machine's CPU.
- c. Including organization-wide customizations, for example, SSL certificates and authentication providers.
- d. Adding internal libraries, to be shared as a single image layer by multiple container images for different applications.

BUILDING CUSTOM CONTAINER IMAGES WITH DOCKERFILE

OBJECTIVES

After completing this section, students should be able to create a container image using common Dockerfile commands.

BASE CONTAINERS

A **Dockerfile** is the mechanism that the Docker packaging model provides to automate the building of container images. Building an image from a Dockerfile is a three-step process:

1. Create a working directory.
2. Write the **Dockerfile** specification.
3. Build the image with the **docker** command.

Create a Working Directory

The **docker** command can use the files in a working directory to build an image. An empty working directory should be created to keep from incorporating unnecessary files into the image. For security reasons, the root directory, `/`, should never be used as a working directory for image builds.

Write the Dockerfile Specification

A **Dockerfile** is a text file that should exist in the working directory. The basic syntax of a **Dockerfile** is shown below:

```
# Comment
INSTRUCTION arguments
```

Lines that begin with a pound sign (#) are comments. Inline comments are not supported. **INSTRUCTION** is a **Dockerfile** keyword. Keywords are not case-sensitive, but a common convention is to make instructions all uppercase to improve visibility.

Instructions in a Dockerfile are executed in the order they appear. The first non-comment instruction must be a **FROM** instruction to specify the base image to build upon. Each Dockerfile instruction is run independently (so **RUN cd /var/tmp** will not have an effect on the commands that follow).

The following is an example Dockerfile for building a simple Apache web server container:

```
# This is a comment line ①
FROM rhel7:7.5 ②
LABEL description="This is a custom httpd container image" ③
MAINTAINER John Doe <jdoe@xyz.com> ④
RUN yum install -y httpd ⑤
EXPOSE 80 ⑥
ENV LogLevel "info" ⑦
```

```

ADD http://someserver.com/filename.pdf /var/www/html ⑧
COPY ./src/ /var/www/html/ ⑨
USER apache ⑩
ENTRYPOINT ["/usr/sbin/httpd"] ⑪
CMD ["-D", "FOREGROUND"] ⑫

```

- ➊ Lines that begin with a pound sign (#) are comments.
- ➋ The new container image will be constructed using **rhel7:7.5** container base image. You can use any other container image as a base image, not only images from operating system distributions. Red Hat provides a set of container images that are certified and tested. Red Hat highly recommends that you use these container images as a base.
- ➌ **LABEL** is responsible for adding generic metadata to an image. A **LABEL** is a simple key/value pair.
- ➍ **MAINTAINER** is responsible for setting the **Author** field of the generated container image. You can use the **docker inspect** command to view image metadata.
- ➎ **RUN** executes commands in a new layer on top of the current image, then commits the results. The committed result is used in the next step in the **Dockerfile**. The shell that is used to execute commands is **/bin/sh**.
- ➏ **EXPOSE** indicates that the container listens on the specified network port at runtime. The **EXPOSE** instruction defines metadata only; it does not make ports accessible from the host. The **-p** option in the **docker run** command exposes a port from the host and the port does not need to be listed in an **EXPOSE** instruction.
- ➐ **ENV** is responsible for defining environment variables that will be available to the container. You can declare multiple **ENV** instructions within the **Dockerfile**. You can use the **env** command inside the container to view each of the environment variables.
- ➑ **ADD** copies files from local or remote source and adds them to the container's file system.
- ➒ **COPY** also copies files from local source and adds them to the container's file system. It is not possible to copy a remote file using its URL with this Dockerfile instruction.
- ➓ **USER** specifies the username or the UID to use when running the container image for the **RUN**, **CMD**, and **ENTRYPOINT** instructions in the **Dockerfile**. It is a good practice to define a different user other than **root** for security reasons.
- ➔ **ENTRYPOINT** specifies the default command to execute when the container is created. By default, the command that is executed is **/bin/sh -c** unless an **ENTRYPOINT** is specified.
- ➕ **CMD** provides the default arguments for the **ENTRYPOINT** instruction.

USING CMD AND ENTRYPOINT INSTRUCTIONS IN THE DOCKERFILE

The **ENTRYPOINT** and **CMD** instructions have two formats:

- Using a **JSON** array:

```
ENTRYPOINT ["command", "param1", "param2"]
```

```
CMD ["param1", "param2"]
```

This is the preferred form.

- Using a shell form:

```
ENTRYPOINT command param1 param2
```

```
CMD param1 param2
```

The **Dockerfile** should contain at most one **ENTRYPOINT** and one **CMD** instruction. If more than one of each is written, then only the last instruction takes effect. Because the default **ENTRYPOINT** is **/bin/sh -c**, a **CMD** can be passed in without specifying an **ENTRYPOINT**.

The **CMD** instruction can be overridden when starting a container. For example, the following instruction causes any container that is run to display the current time:

```
ENTRYPOINT ["/bin/date", "+%H:%M"]
```

The following example provides the same functionality, with the added benefit of being overwritable when a container is started:

```
ENTRYPOINT ["/bin/date"]
CMD ["+%H:%M"]
```

When a container is started without providing a parameter, the current time is displayed:

```
[student@workstation ~]$ docker run -it do285/rhel
11:41
```

If a parameter is provided after the image name in the **docker run** command, it overwrites the **CMD** instruction. For example, the following command will display the current day of the week instead of the time:

```
[student@workstation demo-basic]$ docker run -it do285/rhel +%A
Tuesday
```

As previously mentioned, because the default **ENTRYPOINT** is **/bin/sh -c**, the following instruction also displays the current time, with the added benefit of being able to be overridden at run time.

```
CMD ["date", "+%H:%M"]
```

USING ADD AND COPY INSTRUCTIONS IN THE DOCKERFILE

The **ADD** and **COPY** instructions have two forms:

- Using a shell form:

```
ADD <source>... <destination>
```

```
COPY <source>... <destination>
```

- Using a **JSON** array:

```
ADD [<source>, ... <destination>"]
```

```
COPY ["<source>", ... "<destination>"]
```

The **source** path must be inside the same folder as the **Dockerfile**. The reason for this is that the first step of a **docker build** command is to send all files from the **Dockerfile** folder to the **docker** daemon, and the **docker** daemon cannot see folders or files that are in another folder.

The **ADD** instruction also allows you to specify a resource using a URL:

```
ADD http://someserver.com/filename.pdf /var/www/html
```

If the **source** is a compressed file, the **ADD** instruction decompresses the file to the **destination** folder. The **COPY** instruction does not have this functionality.



WARNING

Both the **ADD** and **COPY** instructions copy the files, retaining the permissions, and with **root** as the owner, even if the **USER** instruction is specified. Red Hat recommends that you use a **RUN** instruction after the copy to change the owner and avoid "permission denied" errors.

IMAGE LAYERING

Each instruction in a **Dockerfile** creates a new layer. Having too many instructions in a **Dockerfile** causes too many layers, resulting in large images. For example, consider the following **RUN** instructions in a **Dockerfile**:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms"
RUN yum update -y
RUN yum install -y httpd
```

The previous example is not a good practice when creating container images, because three layers are created for a single purpose. Red Hat recommends that you minimize the number of layers. You can achieve the same objective using the **&&** conjunction:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" && yum update -y && yum
install -y httpd
```

The problem with this approach is that the readability of the **Dockerfile** is compromised, but it can be easily fixed:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" && \
yum update -y && \
yum install -y httpd
```

The example creates only one layer and the readability is not compromised. This layering concept also applies to instructions such as **ENV** and **LABEL**. To specify multiple **LABEL** or **ENV** instructions, Red Hat recommends that you use only one instruction per line, and separate each key-value pair with an equals sign (=):

```
LABEL version="2.0" \
description="This is an example container image" \
```

```
creationDate="01-09-2017"
```

```
ENV MYSQL_ROOT_PASSWORD="my_password" \
    MYSQL_DATABASE "my_database"
```

BUILDING IMAGES WITH THE DOCKER COMMAND

The **docker build** command processes the **Dockerfile** and builds a new image based on the instructions it contains. The syntax for this command is as follows:

```
$ docker build -t NAME:TAG DIR
```

DIR is the path to the working directory. It can be the current directory as designated by a period (.) if the working directory is the current directory of the **shell**. *NAME:TAG* is a name with a tag that is assigned to the new image. It is specified with the **-t** option. If the *TAG* is not specified, then the image is automatically tagged as **latest**.

DEMONSTRATION: BUILDING A SIMPLE CONTAINER

1. Briefly review the provided **Dockerfile** by opening it in a text editor:

```
[student@workstation ~]$ vim /home/student/DO285/labs/simple-container/Dockerfile
```

2. Observe the **FROM** instruction on the first line of the **Dockerfile**:

```
FROM rhel7:7.5
```

rhel7:7.5 is the base image from which the container is built and where subsequent instructions in the **Dockerfile** are executed.

3. Observe the **MAINTAINER** instruction:

```
MAINTAINER username <username@example.com>
```

The **MAINTAINER** instruction generally indicates the author of the **Dockerfile**. It sets the *Author* field of the generated image. You can run the **docker inspect** command on the generated image to view the *Author* field.

4. Observe the **LABEL** instruction, which sets multiple key-value pair labels as metadata for the image:

```
LABEL version="1.0" \
      description="This is a simple container image" \
      creationDate="31 March 2018"
```

You can run the **docker inspect** command to view the labels in the generated image.

5. Observe the **ENV** instruction, which injects environment variables into the container at runtime. These environment variables can be overridden in the **docker run** command:

```
ENV VAR1="hello" \
    VAR2="world"
```

- In the **ADD** instruction, the **training.repo** file that points to the classroom yum repository, is copied to **/etc/yum.repos.d/training.repo** from the current working directory:

```
ADD training.repo /etc/yum.repos.d/training.repo
```

**NOTE**

The **training.repo** file configures **yum** to use the local repository instead of relying on the default Red Hat Yum repositories.

- Observe the **RUN** instruction where the **yum update** command is executed, and the *bind-utils* package is installed in the container image:

```
RUN yum install -y bind-utils && \
yum clean all
```

The **yum update** command updates the RHEL 7.5 operating system, while the second command installs the DNS utility package *bind-utils*. Notice that both commands are executed with a single **RUN** instruction. Each **RUN** instruction in a **Dockerfile** creates a new image layer to execute the subsequent commands. Minimizing the number of **RUN** commands therefore makes for less overhead when actually running the container.

- Save the **Dockerfile** and run the following commands in the terminal to begin building the new image:

```
[student@workstation ~]$ cd /home/student/D0285/labs/simple-container
[student@workstation simple-container]$ docker build -t do285/rhel .
Sending build context to Docker daemon 3.584 kB
Step 1/6 : FROM rhel7:7.5
Trying to pull repository registry.lab.example.com/rhel7 ...
7.5: Pulling from registry.lab.example.com/rhel7
845d34b6bc6a: Pull complete
c7cbbc13fe2e: Pull complete
Digest: sha256:ca84777dbf5c4574f103c5e9f270193eb9bf8732198e86109c5aa26fdb6fdb0b
Status: Downloaded newer image for registry.lab.example.com/rhel7:7.5
--> 93bb76ddeb7a
Step 2/6 : MAINTAINER John Doe <jdoe@abc.com>
--> Running in a52aa1d5e94b
--> 78ea134831fc
Removing intermediate container a52aa1d5e94b
Step 3/6 : LABEL version "1.0" description "This is a simple container image"
creationDate "31 March 2017"
--> Running in 37871b3a4d69
--> a83e12fc5f8a
Removing intermediate container 37871b3a4d69
Step 4/6 : ENV VAR1 "hello" VAR2 "world"
--> Running in 83d6b5c8546f
--> cc0de1e2423a
Removing intermediate container 83d6b5c8546f
Step 5/6 : ADD training.repo /etc/yum.repos.d/training.repo
--> d6193393a573
Removing intermediate container f08a97a2851b
Step 6/6 : RUN yum install -y bind-utils && yum clean all
--> Running in dc57663d28b1
...
```

```
Installed:  
bind-utils.x86_64 32:9.9.4-61.el7  
  
Dependency Installed:  
GeoIP.x86_64 0:1.5.0-11.el7           bind-libs.x86_64 32:9.9.4-61.el7  
bind-license.noarch 32:9.9.4-61.el7  
  
Complete!  
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager  
This system is not receiving updates. You can use subscription-manager on the host  
to register and assign subscriptions.  
Cleaning repos: rhel-7-server-optional-rpms rhel-server-rhscl-7-rpms rhel_dvd  
Cleaning up everything  
Maybe you want: rm -rf /var/cache/yum, to also free up space taken by orphaned  
data from disabled or removed repos  
--> 93b8610f643e  
Removing intermediate container dc57663d28b1  
Successfully built 93b8610f643e
```

9. After the build completes, run the **docker images** command to verify that the new image is built successfully.

```
[student@workstation simple-container]$ docker images  
REPOSITORY      TAG      IMAGE ID      CREATED          SIZE  
do285/rhel      latest   93b8610f643e   About a minute ago   381 MB  
...
```

10. Inspect the created image to verify that the **MAINTAINER** and **LABEL Dockerfile** instructions indeed manipulated the metadata of the image:

```
[student@workstation simple-container]$ docker inspect do285/rhel | grep Author  
        "Author": "John Doe <jdoe@abc.com>",  
[student@workstation simple-container]$ docker inspect do285/rhel \  
| grep 'version\|description\|creationDate'  
        "creationDate": "31 March 2018",  
        "description": "This is a simple container image",  
        "version": "1.0"  
...
```

11. Execute the following command to run a new container from the generated image with an interactive Bash terminal:

```
[student@workstation simple-container]$ docker run --name simple-container \  
-it do285/rhel /bin/bash
```

12. In the RHEL 7.5 container, verify that the environment variables set in the **Dockerfile** are injected into the container:

```
[root@5c62bfb50dc8 /]# env | grep 'VAR1\|VAR2'  
VAR1=hello  
VAR2=world
```

13. Execute a **dig** command from the container to verify that the *bind-utils* package is installed and working correctly:

```
[root@5c62bfb50dc8 /]# dig materials.example.com
; <>> DiG 9.9.4-RedHat-9.9.4-37.el7 <>> materials.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20867
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;materials.example.com. IN A

;; ANSWER SECTION:
materials.example.com. 3600 IN A 172.25.254.254

;; Query time: 0 msec
;; SERVER: 172.25.250.254#53(172.25.250.254)
;; WHEN: Fri Mar 31 16:33:38 UTC 2017
;; MSG SIZE rcvd: 55
```

Examine the output to confirm that the answer section provides the IP address of the classroom materials server.

14. Exit from the Bash shell in the container:

```
[root@5c62bfb50dc8 thu /]# exit
```

Exiting Bash terminates the running container.

15. Remove the **simple-container** container:

```
[student@workstation simple-container]$ docker rm simple-container
```

16. Remove the **do285/rhel** container image:

```
[student@workstation simple-container]$ docker rmi do285/rhel
```

17. Remove the intermediate container images generated by the **docker build** command.

```
[student@workstation simple-container]$ docker rmi -f $(docker images -q)
```

This concludes the demonstration.



REFERENCES

OpenShift Container Platform documentation: Creating Images

https://docs.openshift.com/container-platform/3.9/creating_images

Dockerfile Reference Guide

<https://docs.docker.com/engine/reference/builder/>

Creating base images

<https://docs.docker.com/engine/userguide/eng-image/baseimages/>

Implementing a base image based on RHEL-based distros

<https://github.com/docker/docker/blob/master/contrib/mkimage-yum.sh>

► GUIDED EXERCISE

CREATING A BASIC APACHE CONTAINER IMAGE

In this exercise, you will create a basic Apache container image.

OUTCOMES

You should be able to create a basic Apache container image built on a RHEL 7.5 image.

BEFORE YOU BEGIN

Run the following command to download the relevant lab files and to verify that docker is running:

```
[student@workstation ~]$ lab basic-apache setup
```

► 1. Create the Apache Dockerfile

- 1.1. Open a terminal on **workstation**. Use your preferred editor and create a new Dockerfile:

```
[student@workstation ~]$ vim /home/student/D0285/labs/basic-apache/Dockerfile
```

- 1.2. Use RHEL 7.5 as a base image by adding the following **FROM** instruction at the top of the new Dockerfile:

```
FROM rhel7:7.5
```

- 1.3. Below the **FROM** instruction, include the **MAINTAINER** instruction to set the **Author** field in the new image. Replace the values to include your name and email address:

```
MAINTAINER Your Name <youremail>
```

- 1.4. Below the **MAINTAINER** instruction, add the following **LABEL** instruction to add description metadata to the new image:

```
LABEL description="A basic Apache container on RHEL 7"
```

- 1.5. Add a **RUN** instruction with a **yum install** command to install Apache on the new container:

```
ADD training.repo /etc/yum.repos.d/training.repo
RUN yum -y update && \
    yum install -y httpd && \
    yum clean all
```

**NOTE**

The **ADD** instruction configures **yum** to use the local repository instead of relying on the default Red Hat Yum repositories.

- 1.6. Use the **EXPOSE** instruction below the **RUN** instruction to document the port that the container listens to at runtime. In this instance, set the port to 80, because it is the default for an Apache server:

```
EXPOSE 80
```

**NOTE**

The **EXPOSE** instruction does not actually make the specified port available to the host; rather, the instruction serves more as metadata about which ports the container is listening to.

- 1.7. At the end of the file, use the following **CMD** instruction to set **httpd** as the default executable when the container is run:

```
CMD ["httpd", "-D", "FOREGROUND"]
```

- 1.8. Verify that your Dockerfile matches the following before saving and proceeding with the next steps:

```
FROM rhel7:7.5

MAINTAINER Your Name <youremail>

LABEL description="A basic Apache container on RHEL 7"

ADD training.repo /etc/yum.repos.d/training.repo
RUN yum -y update && \
    yum install -y httpd && \
    yum clean all

EXPOSE 80

CMD ["httpd", "-D", "FOREGROUND"]
```

► 2. Build and verify the Apache container image.

- 2.1. Use the following commands to create a basic Apache container image using the newly created Dockerfile:

```
[student@workstation ~]$ cd /home/student/D0285/labs/basic-apache
[student@workstation basic-apache]$ docker build -t do285/apache .
Sending build context to Docker daemon 3.584 kB
Step 1/7 : FROM rhel7:7.5
Trying to pull repository registry.lab.example.com/rhel7 ...
7.5: Pulling from registry.lab.example.com/rhel7
845d34b6bc6a: Pull complete
c7cbcb13fe2e: Pull complete
Digest: sha256:ca84777dbf5c4574f103c5e9f270193eb9bf8732198e86109c5aa26fdb6fdb0b
```

```
Status: Downloaded newer image for registry.lab.example.com/rhel7:7.5
---> 93bb76ddeb7a
Step 2/7 : MAINTAINER Your Name <youremail>
---> Running in 3d4d201a3730
---> 6c0c7ca9d614
Removing intermediate container 3d4d201a3730
Step 3/7 : LABEL description "A basic Apache container on RHEL 7"
---> Running in 0e9ec16edf9b
---> 0541f6f3d393
Removing intermediate container 0e9ec16edf9b
Step 4/7 : ADD training.repo /etc/yum.repos.d/training.repo
---> 642dc842d870
Removing intermediate container 33e2dc33f198
Step 5/7 : RUN yum install -y httpd && yum clean all
---> Running in 50e68830492b
...
Installed:
httpd.x86_64 0:2.4.6-80.el7
...
Complete!

...
Maybe you want: rm -rf /var/cache/yum, to also free up space taken by orphaned
data from disabled or removed repos
---> 550f4c27c866
Removing intermediate container 50e68830492b
Step 6/7 : EXPOSE 80
---> Running in 63c30a79a047
---> 517c0dd6f2ae
Removing intermediate container 63c30a79a047
Step 7/7 : CMD httpd -D FOREGROUND
---> Running in 612a9802c872
---> 6b291a377825
Removing intermediate container 612a9802c872
Successfully built 6b291a377825
```

- 2.2. After the build process has finished, run **docker images** to see the new image in the image repository:

```
[student@workstation basic-apache]$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
do285/apache    latest   6b291a377825  4 minutes ago  393 MB
...
```

► 3. Run the Apache Container

- 3.1. Use the following command to run a container using the Apache image:

```
[student@workstation basic-apache]$ docker run --name lab-apache \
-d -p 10080:80 do285/apache
693da7820edb...
```

- 3.2. Run the **docker ps** command to see the running container:

```
[student@workstation basic-apache]$ docker ps
```

CONTAINER ID NAMES	IMAGE	COMMAND	... PORTS
693da7820edb tcp lab-apache	do285/apache	"httpd -D FOREGROUND"	0.0.0.0:10080->80/

3.3. Use **curl** command to verify that the server is running:

```
[student@workstation basic-apache]$ curl 127.0.0.1:10080
```

If the server is running, you should see HTML output for an Apache server test home page.

- 4. Run the following command to verify that the image was correctly built:

```
[student@workstation basic-apache]$ lab basic-apache grade
```

- 5. Stop and then remove the **lab-apache** container:

```
[student@workstation basic-apache]$ docker stop lab-apache  
[student@workstation basic-apache]$ docker rm lab-apache
```

- 6. Remove the **do285/apache** container image:

```
[student@workstation basic-apache]$ docker rmi -f do285/apache
```

- 7. Remove the intermediate container images generated by the **docker build** command.

```
[student@workstation simple-container]$ docker rmi -f $(docker images -q)
```

This concludes the guided exercise.

▶ LAB

CREATING CUSTOM CONTAINER IMAGES

In this lab, you will create a Dockerfile to run an Apache Web Server container that hosts a static HTML file. The Apache image will be based on a base RHEL 7.5 image that serves a custom `index.html` page.

OUTCOMES

You should be able to create a custom Apache Web Server container that hosts static HTML files.

BEFORE YOU BEGIN

Run the following command to download the relevant lab files and to verify that there are no running or stopped containers that will interfere with completing the lab:

```
[student@workstation ~]$ lab custom-images setup
```

1. Review the provided Dockerfile stub in the `/home/student/D0285/labs/custom-images/` folder. Edit the **Dockerfile** and ensure that it meets the following specifications:
 - Base image: `rhel7:7.5`
 - Set the desired author name and email ID with **MAINTAINER** instruction.
 - Environment variable: `PORT` set to 8080
 - Update the RHEL packages and install Apache (`httpd` package) using the classroom Yum repository. Also ensure you run the `yum clean all` command as a best practice.
 - Change the Apache configuration file `/etc/httpd/conf/httpd.conf` to listen to port 8080 instead of the default port 80.
 - Change ownership of the `/etc/httpd/logs` and `/run/httpd` folders to user and group `apache` (UID and GID are 48).
 - So that container users know how to access the Apache Web Server, expose the value set in the `PORT` environment variable.
 - Copy the contents of the `src/` folder in the lab directory to the Apache **DocumentRoot** (`/var/www/html/`) inside the container.

The `src/` folder contains a single `index.html` file that prints a **Hello World!** message.

- Start Apache `httpd` in the foreground using the following command:

```
httpd -D FOREGROUND
```

- 1.1. Open a terminal (Applications → System Tools → Terminal) and use your preferred editor to modify the Dockerfile located in the `/home/student/D0285/labs/custom-images/` folder.

- 1.2. Set the base image for the Dockerfile to **rhel7:7.5**.
 - 1.3. Set your name and email with a **MAINTAINER** instruction.
 - 1.4. Create an environment variable called **PORT** and set it to 8080.
 - 1.5. Add the classroom Yum repositories configuration file. Update the RHEL packages and install Apache with a single **RUN** instruction.
 - 1.6. Change the Apache HTTP Server configuration file to listen to port 8080 and change ownership of the server working folders with a single **RUN** instruction.
 - 1.7. Use the **USER** instruction to run the container as the **apache** user. Use the **EXPOSE** instruction to document the port that the container listens to at runtime. In this instance, set the port to the **PORT** environment variable, which is the default for an Apache server.
 - 1.8. Copy all the files from the **src** folder to the Apache **DocumentRoot** path at **/var/www/html**.
 - 1.9. Finally, insert a **CMD** instruction to run **httpd** in the foreground, and then save the Dockerfile.
2. Build the custom Apache image with the name **do285/custom-apache**.
 - 2.1. Verify the Dockerfile for the custom Apache image.
 - 2.2. Run a **docker build** command to build the custom Apache image and name it **do285/custom-apache**.
 - 2.3. Run the **docker images** command to verify that the custom image is built successfully.
 3. Create a new container in detached mode with the following characteristics:
 - **Name:** **lab-custom-images**
 - **Container image:** **do285/custom-apache**
 - **Port forward:** from host port 20080 to container port 8080
 - 3.1. Create and run the container.
 - 3.2. Verify that the container is ready and running.
 4. Verify that the server is running and that it is serving the HTML file.
 - 4.1. Run a **curl** command on **127.0.0.1:20080**
 5. Verify that the lab was correctly executed. Run the following from a terminal:

```
[student@workstation custom-images]$ lab custom-images grade
```

6. Stop and then remove the container started in this lab.
7. Remove the custom container image created in this lab.
8. Remove the intermediate container images generated by the **docker build** command.

```
[student@workstation simple-container]$ docker rmi -f $(docker images -q)
```

This concludes the lab.

► SOLUTION

CREATING CUSTOM CONTAINER IMAGES

In this lab, you will create a Dockerfile to run an Apache Web Server container that hosts a static HTML file. The Apache image will be based on a base RHEL 7.5 image that serves a custom `index.html` page.

OUTCOMES

You should be able to create a custom Apache Web Server container that hosts static HTML files.

BEFORE YOU BEGIN

Run the following command to download the relevant lab files and to verify that there are no running or stopped containers that will interfere with completing the lab:

```
[student@workstation ~]$ lab custom-images setup
```

- Review the provided Dockerfile stub in the `/home/student/D0285/labs/custom-images/` folder. Edit the **Dockerfile** and ensure that it meets the following specifications:
 - Base image: **rhel7:7.5**
 - Set the desired author name and email ID with **MAINTAINER** instruction.
 - Environment variable: **PORT** set to 8080
 - Update the RHEL packages and install Apache (`httpd` package) using the classroom Yum repository. Also ensure you run the `yum clean all` command as a best practice.
 - Change the Apache configuration file `/etc/httpd/conf/httpd.conf` to listen to port 8080 instead of the default port 80.
 - Change ownership of the `/etc/httpd/logs` and `/run/httpd` folders to user and group **apache** (UID and GID are 48).
 - So that container users know how to access the Apache Web Server, expose the value set in the **PORT** environment variable.
 - Copy the contents of the `src/` folder in the lab directory to the Apache **DocumentRoot** (`/var/www/html/`) inside the container.

The `src/` folder contains a single `index.html` file that prints a **Hello World!** message.

- Start Apache **httpd** in the foreground using the following command:

```
httpd -D FOREGROUND
```

- Open a terminal (Applications → System Tools → Terminal) and use your preferred editor to modify the Dockerfile located in the `/home/student/D0285/labs/custom-images/` folder.

```
[student@workstation ~]$ cd /home/student/D0285/labs/custom-images/  
[student@workstation custom-images]$ vim Dockerfile
```

- 1.2. Set the base image for the Dockerfile to **rhel7:7.5**.

```
FROM rhel7:7.5
```

- 1.3. Set your name and email with a **MAINTAINER** instruction.

```
MAINTAINER Your Name <youremail>
```

- 1.4. Create an environment variable called **PORT** and set it to 8080.

```
ENV PORT 8080
```

- 1.5. Add the classroom Yum repositories configuration file. Update the RHEL packages and install Apache with a single **RUN** instruction.

```
ADD training.repo /etc/yum.repos.d/training.repo  
RUN yum install -y httpd && \  
yum clean all
```

- 1.6. Change the Apache HTTP Server configuration file to listen to port 8080 and change ownership of the server working folders with a single **RUN** instruction.

```
RUN sed -ri -e '/^Listen 80/c\Listen ${PORT}' /etc/httpd/conf/httpd.conf \  
&& chown -R apache:apache /etc/httpd/logs/ \  
&& chown -R apache:apache /run/httpd/
```

- 1.7. Use the **USER** instruction to run the container as the **apache** user. Use the **EXPOSE** instruction to document the port that the container listens to at runtime. In this instance, set the port to the **PORT** environment variable, which is the default for an Apache server.

```
USER apache  
EXPOSE ${PORT}
```

- 1.8. Copy all the files from the **src** folder to the Apache **DocumentRoot** path at **/var/www/html**.

```
COPY ./src/ /var/www/html/
```

- 1.9. Finally, insert a **CMD** instruction to run **httpd** in the foreground, and then save the Dockerfile.

```
CMD ["httpd", "-D", "FOREGROUND"]
```

2. Build the custom Apache image with the name **do285/custom-apache**.

- 2.1. Verify the Dockerfile for the custom Apache image.

The Dockerfile for the custom Apache image should look like the following:

```
FROM rhel7:7.5
```

```
MAINTAINER Your Name <youremail>

ENV PORT 8080

ADD training.repo /etc/yum.repos.d/training.repo
RUN yum install -y httpd && \
yum clean all

RUN sed -ri -e '/^Listen 80/c\Listen ${PORT}' /etc/httpd/conf/httpd.conf \
&& chown -R apache:apache /etc/httpd/logs/ \
&& chown -R apache:apache /run/httpd/

USER apache
EXPOSE ${PORT}

COPY ./src/ /var/www/html/

CMD ["httpd", "-D", "FOREGROUND"]
```

2.2. Run a **docker build** command to build the custom Apache image and name it **do285/custom-apache**.

```
[student@workstation custom-images]$ docker build -t do285/custom-apache .
Sending build context to Docker daemon 5.632 kB
Step 1/10 : FROM rhel7:7.5
--> 93bb76ddeb7a
Step 2/10 : MAINTAINER Your Name <youremail>
--> Running in fd2701e2dc31
--> 2a0cb88ba0d9
Removing intermediate container fd2701e2dc31
Step 3/10 : ENV PORT 8080
--> Running in 5274fb5b5017
--> 4f347c7508e3
Removing intermediate container 5274fb5b5017
Step 4/10 : ADD training.repo /etc/yum.repos.d/training.repo
--> a99cd105c046
Removing intermediate container 0e8c11d7a1f6
Step 5/10 : RUN yum install -y httpd && yum clean all
--> Running in 594f9e7b594a
...
Installed:
httpd.x86_64 0:2.4.6-80.el7
...
Complete!
...
--> dd9ed43b1683
Removing intermediate container 92baa6734de3
Step 6/10 : RUN sed -ri -e '/^Listen 80/c\Listen ${PORT}' /etc/httpd/conf/
httpd.conf && chown -R apache:apache /etc/httpd/logs/ && chown -R apache:apache /run/httpd/
--> Running in 24911b35ea8c
--> 0e556d9cb284
Removing intermediate container 24911b35ea8c
Step 7/10 : USER apache
```

```
--> Running in 7f769d2f9eda
--> 95e648d72347
Removing intermediate container 7f769d2f9eda
Step 8/10 : EXPOSE ${PORT}
--> Running in de11b4f17026
--> b3ad5660808c
Removing intermediate container de11b4f17026
Step 9/10 : COPY ./src/ /var/www/html
--> 804879072dc7
Removing intermediate container 51526a53cf1b
Step 10/10 : CMD httpd -D FOREGROUND
--> Running in 2dfffc816173
--> 35175850d37d
Removing intermediate container 2dfffc816173
Successfully built 35175850d37d
```

- 2.3. Run the **docker images** command to verify that the custom image is built successfully.

```
[student@workstation custom-images]$ docker images
REPOSITORY          TAG      IMAGE ID      ...
do285/custom-apache    latest   35175850d37d   ...
registry.lab.example.com/rhel7     7.5      41a4953dbf95   ...
```

3. Create a new container in detached mode with the following characteristics:

- **Name: lab-custom-images**
- **Container image: do285/custom-apache**
- **Port forward:** from host port 20080 to container port 8080

- 3.1. Create and run the container.

```
[student@workstation custom-images]$ docker run --name lab-custom-images \
-d -p 20080:8080 do285/custom-apache
367823e35c4a...
```

- 3.2. Verify that the container is ready and running.

```
[student@workstation custom-images]$ docker ps
...     IMAGE                  COMMAND            ... PORTS
NAMES
...     do285/custom-apache   "httpd -D FOREGROUND" ...  0.0.0.0:20080->8080/tcp
lab-custom-images
```

4. Verify that the server is running and that it is serving the HTML file.

- 4.1. Run a **curl** command on 127.0.0.1:20080

```
[student@workstation custom-images]$ curl 127.0.0.1:20080
```

The output should be as follows:

```
<html>
<header><title>D0285 Hello!</title></header>
```

```
<body>
Hello World! The custom-images lab works!
</body>
</html>
```

5. Verify that the lab was correctly executed. Run the following from a terminal:

```
[student@workstation custom-images]$ lab custom-images grade
```

6. Stop and then remove the container started in this lab.

```
[student@workstation custom-images]$ docker stop lab-custom-images
[student@workstation custom-images]$ docker rm lab-custom-images
```

7. Remove the custom container image created in this lab.

```
[student@workstation custom-images]$ docker rmi -f do285/custom-apache
```

8. Remove the intermediate container images generated by the **docker build** command.

```
[student@workstation simple-container]$ docker rmi -f $(docker images -q)
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The usual method of creating container images is using Dockerfiles.
- Dockerfiles provided by Red Hat or Docker Hub are a good starting point for creating custom images using a specific language or technology.
- The Source-to-Image (S2I) process provides an alternative to Dockerfiles. S2I spares developers the need to learn low-level operating system commands and usually generates smaller images.
- Building an image from a Dockerfile is a three-step process:
 1. Create a working directory.
 2. Write the **Dockerfile** specification.
 3. Build the image using the **docker build** command.
- The **RUN** instruction is responsible for modifying image contents.
- The following instructions are responsible for adding metadata to images:
 - **LABEL**
 - **MAINTAINER**
 - **EXPOSE**
- The default command that runs when the container starts can be changed with the **RUN** and **ENTRYPOINT** instructions.
- The following instructions are responsible for managing the container environment:
 - **WORKDIR**
 - **ENV**
 - **USER**
- The **VOLUME** instruction creates a mount point in the container.
- The **Dockerfile** provides two instructions to include resources in the container image:
 - **ADD**
 - **COPY**

CHAPTER 6

DEPLOYING MULTI-CONTAINER APPLICATIONS

GOAL

Deploy containerized applications using multiple container images.

OBJECTIVES

- Describe the considerations for containerizing applications with multiple container images.
- Deploy a multi-container application with user-defined Docker network.

SECTIONS

- Considerations for Multi-Container Applications (and Quiz)
- Deploying a Multi-Container Application with Docker (and Guided Exercise)

LAB

- Deploying Multi-Container Applications

CONSIDERATIONS FOR MULTI-CONTAINER APPLICATIONS

OBJECTIVES

After completing this section, students should be able to:

- Describe considerations for containerizing applications with multiple container images.
- Inject environment variables into a container.
- Describe the architecture of the To Do List application.

DISCOVERING SERVICES IN A MULTI-CONTAINER APPLICATION

Due to the dynamic nature of container IP addresses, applications cannot rely on either fixed IP addresses or fixed DNS host names to communicate with middleware services and other application services. Containers with dynamic IP addresses can become a problem when working with multi-container applications in which each container must be able to communicate with other containers in order to use the services on which it depends.

For example, consider an application which is composed of a front-end container, a back-end container, and a database. The front-end container needs to retrieve the IP address of the back-end container. Similarly, the back-end container needs to retrieve the IP address of the database container. Additionally, the IP address could change if a container is restarted, so a process is needed to ensure any change in IP triggers an update to existing containers.

Both Docker and Kubernetes provide potential solutions to the issue of service discoverability and the dynamic nature of container networking, some of these solutions are covered later in the section.

INJECTING ENVIRONMENT VARIABLES INTO A DOCKER CONTAINER

It is a recommended practice to parameterize application connection information to outside services, and one common way to do that is the usage of operating system environment variables. The **docker** command provides a few options for interacting with container environment variables:

- The **docker run** command provides the **-e** option to define environment variables when starting a container, and this could be used to pass parameters to an application such as a database server IP address or user credentials. The **-e** option can be used multiple times to define more than one environment variable for the same container.
- The **docker inspect** command can be used to check a running container for environment variables specified either when starting the container or defined by the container image **Dockerfile** instructions. However, it does not show environment variables inherited by the container by the operating system or defined by shell scripts inside the image.
- The **docker exec** command can be used to inspect all environment variables known to a running container using regular shell commands. For example:

```
$ docker exec mysql env | grep MYSQL
```

Additionally, the **Dockerfile** for a container image may contain instructions related to the management of a container environment variables. Use the **ENV** instruction to expose an environment variable to the container:

```
ENV MYSQL_ROOT_PASSWORD="my_password"
ENV MYSQL_DATABASE="my_database"
```

It is recommended to declare all environment variables using only one **ENV** instruction to avoid creating multiple layers:

```
ENV MYSQL_ROOT_PASSWORD="my_password" \
    MYSQL_DATABASE="my_database"
```

COMPARING PLAIN DOCKER AND KUBERNETES

Using environment variables allows you to share information between containers with Docker, which makes service discovery technically possible. However, there are still some limitations and some manual work involved in ensuring that all environment variables stay in sync, especially when working with many containers. Kubernetes provides an approach to solve this problem by creating services for your containers, as covered in previous chapters.

Pods are attached to a Kubernetes namespace, which OpenShift calls a *project*. When a pod starts, Kubernetes automatically adds a set of environment variables for each service defined on the same namespace.

Any service defined on Kubernetes, generates environment variables for the IP address and port number where the service is available. Kubernetes automatically injects these environment variables into the containers from pods in the same namespace. These environment variables usually follow a convention:

- **Uppercase:** All environment variables are set using uppercase names.
- **Words separated with underscore:** Any environment variable created by a service normally are created with multiple words and they are separated with an underscore (_).
- **Service name first:** The first word for an environment variable created by a service is the service name.
- **Protocol type:** Most network environment variables are declared with the protocol type (TCP or UDP).

These are the environment variables generated by Kubernetes for a service:

- **<SERVICE_NAME>_SERVICE_HOST:** Represents the IP address enabled by a service to access a pod.
- **<SERVICE_NAME>_SERVICE_PORT:** Represents the port where the server port is listed.
- **<SERVICE_NAME>_PORT:** Represents the address, port, and protocol provided by the service for external access.
- **<SERVICE_NAME>_PORT_<PORT_NUMBER>_<PROTOCOL>:** Defines an alias for the **<SERVICE_NAME>_PORT**.
- **<SERVICE_NAME>_PORT_<PORT_NUMBER>_<PROTOCOL>_PROTO:** Identifies the protocol type (TCP or UDP).
- **<SERVICE_NAME>_PORT_<PORT_NUMBER>_<PROTOCOL>_PORT:** Defines an alias for the **<SERVICE_NAME>_SERVICE_PORT**.

- <SERVICE_NAME>_PORT_<PORT_NUMBER>_<PROTOCOL>_ADDR: Defines an alias for the <SERVICE_NAME>_SERVICE_HOST.

Note that the variables following the convention <SERVICE_NAME>_PORT_* emulate the variables created by Docker for associated containers in the pod.

For instance, if the following service is deployed on Kubernetes:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: mysql
    name: mysql
spec:
  ports:
    - protocol: TCP
      port: 3306
  selector:
    name: mysql
```

The following environment variables are available for each pod created after the service, on the same namespace:

```
MYSQL_SERVICE_HOST=10.0.0.11
MYSQL_SERVICE_PORT=3306
MYSQL_PORT=tcp://10.0.0.11:3306
MYSQL_PORT_3306_TCP=tcp://10.0.0.11:3306
MYSQL_PORT_3306_TCP_PROTO=tcp
MYSQL_PORT_3306_TCP_PORT=3306
MYSQL_PORT_3306_TCP_ADDR=10.0.0.11
```



NOTE

On the basis of the protocol, IP address and port number as set against <SERVICE_NAME>_PORT environment variable, the other relevant <SERVICE_NAME>_PORT_* environment variables' names are set. For example, **MYSQL_PORT=tcp://10.0.0.11:3306** variable leads to the creation of the environment variables with names such as **MYSQL_PORT_3306_TCP**, **MYSQL_PORT_3306_TCP_PROTO**, **MYSQL_PORT_3306_TCP_PORT** and **MYSQL_PORT_3306_TCP_ADDR**. If the protocol component is not set against the said environment variable, Kubernetes uses TCP protocol and assigns the variable names accordingly.

EXAMINING THE To Do List APPLICATION

Many labs from this course are based on a To Do List application. This application is divided in three tiers as illustrated by the following figure:

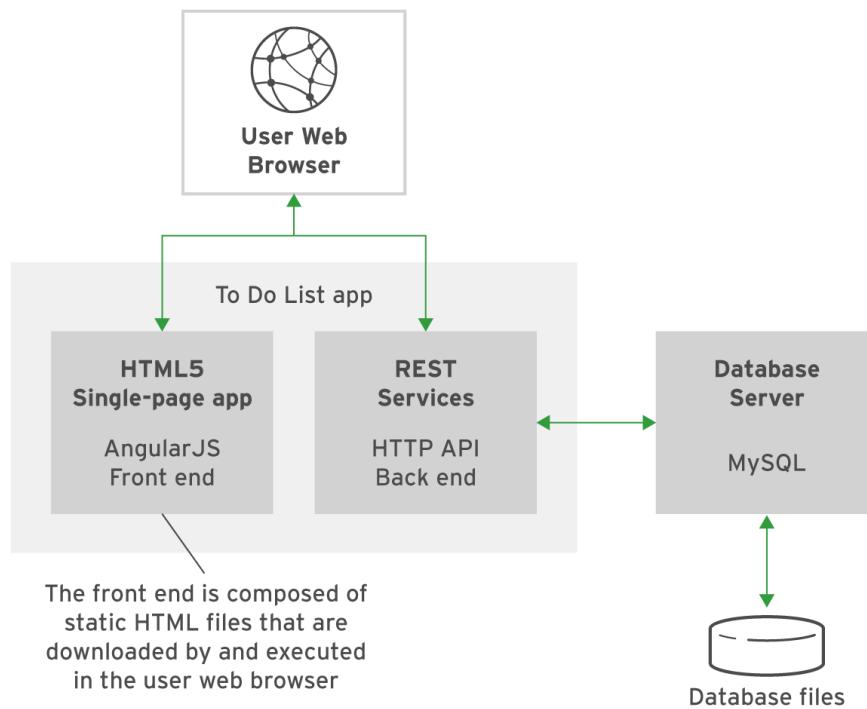


Figure 6.1: To Do List application logical architecture

- The presentation tier is built as a single-page HTML5 front-end using **AngularJS**.
- The business tier is built as an HTTP API back-end, with **Node.js**.
- Persistence tier is based on a **MySQL** database server.

The following figure is a screen capture of the application web interface.

To Do List Application

To Do List

Id	Description	Done	
1	Pick up new...	false	×
2	Buy groceries	true	×

Add Task

Description:

Completed:

Clear Save

First
Previous
1
Next
Last

Figure 6.2: The To Do List application

On the left is a table with items to complete, and on the right is a form to add a new item.

The classroom private registry server, **services.lab.example.com**, provides the application in two versions:

nodejs

Represents how a typical developer would create the application as a single unit, without caring to break it into tiers or services.

nodejs_api

Shows the changes needed to break the application presentation and business tiers so they can be deployed into different containers.

The sources of both of these application versions are available under *todoapp* GIT repository at:
<http://services.lab.example.com/todoapp/apps/nodejs>

► QUIZ

MULTI-CONTAINER APPLICATION CONSIDERATIONS

Choose the correct answer(s) to the following questions:

► 1. **Why is it difficult for Docker containers to communicate with each other?**

- a. There is no connectivity between containers using the default Docker networking.
- b. The container's firewall must always be disabled in order to enable communication between containers.
- c. Containers use dynamic IP addresses and host names, so it is difficult for one container to reliably find another container's IP address without explicit configuration.
- d. Containers require a VPN client and server in order to connect to each other.

► 2. **What are three benefits of using a multi-container architecture? (Choose three.)**

- a. Keeping images as simple as possible by only including a single application or service in each container provides for easier deployment, maintenance, and administration.
- b. The more containers used in an application, the better the performance will be.
- c. When using multiple containers for an application, each layer of the application can be scaled independently from the others, thus conserving resources.
- d. Monitoring, troubleshooting, and debugging are simplified when a container only provides a single purpose.
- e. Applications using multiple containers typically contain fewer defects than those built inside a single container.

► 3. **How does Kubernetes solve the issue of service discovery using environment variables?**

- a. Kubernetes automatically propagates all environment variables to all pods ensuring the required environment variables are available for containers to leverage.
- b. Controllers are responsible for sharing environment variables between pods.
- c. Kubernetes maintains a list of all environment variables and pods can request them as needed.
- d. Kubernetes automatically injects environment variables for all services in a given namespace into all the pods running on that same namespace.

► SOLUTION

MULTI-CONTAINER APPLICATION CONSIDERATIONS

Choose the correct answer(s) to the following questions:

► 1. Why is it difficult for Docker containers to communicate with each other?

- a. There is no connectivity between containers using the default Docker networking.
- b. The container's firewall must always be disabled in order to enable communication between containers.
- c. Containers use dynamic IP addresses and host names, so it is difficult for one container to reliably find another container's IP address without explicit configuration.
- d. Containers require a VPN client and server in order to connect to each other.

► 2. What are three benefits of using a multi-container architecture? (Choose three.)

- a. Keeping images as simple as possible by only including a single application or service in each container provides for easier deployment, maintenance, and administration.
- b. The more containers used in an application, the better the performance will be.
- c. When using multiple containers for an application, each layer of the application can be scaled independently from the others, thus conserving resources.
- d. Monitoring, troubleshooting, and debugging are simplified when a container only provides a single purpose.
- e. Applications using multiple containers typically contain fewer defects than those built inside a single container.

► 3. How does Kubernetes solve the issue of service discovery using environment variables?

- a. Kubernetes automatically propagates all environment variables to all pods ensuring the required environment variables are available for containers to leverage.
- b. Controllers are responsible for sharing environment variables between pods.
- c. Kubernetes maintains a list of all environment variables and pods can request them as needed.
- d. Kubernetes automatically injects environment variables for all services in a given namespace into all the pods running on that same namespace.

DEPLOYING A MULTI-CONTAINER APPLICATION WITH DOCKER

OBJECTIVES

After completing this section, students should be able to deploy a multi-container application with a user-defined Docker network.

USER-DEFINED DOCKER NETWORK

Standalone containers in an application stack can communicate with each other using IP addresses. However, when a container restarts, there is no guarantee that the IP address of the container is unchanged. This may lead to broken communication between the containers of the application stack, as the destination IP address that the other container uses may no longer be valid.

One way Docker solves this issue is by implementing *user-defined* bridge networks. User-defined bridge networks allow multiple standalone containers to talk to each other using container names. The container names are unlikely to change in the event of a container restart.

All containers attached to the same user-defined bridge have access to the uniform DNS records maintained by the built-in DNS server of the Docker host. These entries are dynamically updated with a containers' IP addresses and host names (container names) as they go in and out of the bridge network. This allows the containers to communicate using static container names.



NOTE

A user-defined bridge network is one method of Docker networking. This course covers the user-defined bridge networks of Docker among other methods, which are beyond the scope of this course. Consult the reference list *References* for more information about Docker networks.

MANAGING DOCKER NETWORKS

To manage Docker networks, use the **docker network** command. Run the **docker network --help** command to get help on command usage.

- To list available networks, use **docker network ls** command.

```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
...output omitted...
d531b5e4dbe   bridge    bridge      local
...output omitted...
```

By default, Docker creates a user-defined bridge network named **bridge**. This bridge can be used for testing temporary container connections. However, using the default bridge network in production is not a recommended practice.

- To examine a network, use **docker inspect network name** command.

```
$ docker inspect do285-bridge
[
```

```
{
    "Name": "do285-bridge①",
    "Id": "7dae997d41f21b7a1b17cd5c3e84ad89c3a38fc05852d44eef9fe4cd6dbe61f2",
    "Created": "2018-06-21T19:09:25.41450065+05:30",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
        "Driver": "default",
        "Options": {},
        "Config": [
            {
                "Subnet": "172.22.0.0/16②",
                "Gateway": "172.22.0.1③"
            }
        ]
    },
    "Internal": false,
    "Attachable": true④,
    "Containers": {}⑤,
    "Options": {},
    "Labels": {}
}
]
```

- ^① Shows the name of the bridge network.
- ^② Shows the subnet for the network that the containers use.
- ^③ Shows the gateway IP for all the containers in the bridge. The IP address is the address of the bridge itself.
- ^④ If set to **true**, containers can dynamically join the bridge.
- ^⑤ List the containers currently connected in the bridge.

- To create a network, use the **docker network create network name** command.

```
$ docker network create --attachable do285-bridge
```

The **--attachable** option allows containers to join the bridge dynamically. The default network type is set to **bridge**. Override this default behavior using **--type** option.

- Use **--network network name** option with **docker run** command to specify the intended Docker network for the container.
- To delete a network, use the **docker network rm network name** command.

```
$ docker network rm do285-bridge
```

SERVICE DISCOVERY WITH USER-DEFINED NETWORKING

Containers within the same user-defined bridge network can connect to each other using the supplied container names. These container names, which can be supplied by the user, are resolvable host names that other containers can resolve. For example, a container named

web_container is able to resolve **db_container**, and the other way around, if both belong to the same user-defined bridge network.

Each container exposes all ports to other containers in the same bridge. As such, services are readily accessible within the same user-defined bridge network. The containers expose ports to external networks only on the basis of explicit configuration.

The user-defined bridge network is particularly relevant in running a multi-container application. Its dynamic name resolution feature helps to resolve the issue of service discoverability between containers in a multi-container application.

The following procedure demonstrates the usage of user-defined networks.

- The following command produces a list of available Docker networks:

```
[student@workstation ~]$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
...output omitted...
1876bcd67a0f    do285-bridge    bridge      local
...output omitted...
```

- The following command runs a database (MySQL) container in the background and attaches it to **do285-bridge** Docker network. The **-e** options specifies the required environment variables to run the database container image.

```
[student@workstation ~]$ docker run -d --name db_container \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
--network do285-bridge \
test/db:1.0
```

- The following command runs another container in the background, using the standard **rhel7** image, with **mysql** client installed in it. This container is attached to the **do285-bridge** Docker network.

```
[student@workstation ~]$ docker run -d --name web_container \
--network do285-bridge test/web:1.0
```

- The following command executes an interactive shell inside the **web_container** container. From this shell, MySQL statements can be executed seamlessly.

```
[student@workstation ~]$ docker exec -it web_container /bin/bash
[root@cf36ce51a8b8 /]# mysql -uuser1 -pmypa55 -hdb_container -e 'show databases;'
+-----+
| Database      |
+-----+
| information_schema |
| items          |
+-----+
```

Notice that the hostname of the database server, **db_container**, resolves to the IP address of the database container. This indicates that the user-defined bridge network allows containers to reach each other using the static container names supplied by the user.



REFERENCES

Further information about Docker networks is available in the *Networking Overview* section of the Docker documentation at
<https://docs.docker.com/network/>

► GUIDED EXERCISE

DEPLOYING THE WEB APPLICATION AND MYSQL CONTAINERS

In this lab, you will create a script that runs and networks a Node.js application container and the MySQL container.

RESOURCES

Files:	/home/student/D0285/labs/networking-containers
Application URL:	http://127.0.0.1:30080/todo/
Resources:	RHSCl MySQL 5.7 image (rhscl/mysql-57-rhel7) Custom MySQL 5.7 image (do285/mysql-57-rhel7) RHEL 7.5 image (rhel7.5) Custom Node.js 4.0 image (do285/nodejs)

OUTCOMES

You should be able to network containers to create a multi-tiered application.

BEFORE YOU BEGIN

The workstation requires the To Do List application source code and lab files. To set up the environment for the exercise, run the following command:

```
[student@workstation ~]$ lab networking-containers setup
```

► 1.



WARNING

Use the **docker ps** command to verify no images are present. If images are displayed, use the command **docker rmi** to remove them prior to beginning this exercise.

Build the MySQL image.

A custom MySQL 5.7 image is used for this exercise. It is configured to automatically run any scripts in the **/var/lib/mysql/init** directory. The scripts load the schema and some sample data into the database for the To Do List application when a container starts.

1.1. Review the Dockerfile.

Using your preferred editor, open and examine the completed Dockerfile located at **/home/student/D0285/labs/networking-containers/images/mysql/Dockerfile**.

- 1.2. Build the MySQL database image.

Examine the **/home/student/D0285/labs/networking-containers/images/mysql/build.sh** script to see how the image is built. To build the base image, run the **build.sh** script.

```
[student@workstation ~]$ cd ~/D0285/labs/networking-containers/images/  
[student@workstation images]$ cd mysql  
[student@workstation mysql]$ ./build.sh
```

- 1.3. Wait for the build to complete, and then run the following command to verify that the image is built successfully:

```
[student@workstation mysql]$ docker images  
REPOSITORY          TAG      IMAGE ID   CREATED       SIZE  
do285/mysql-57-rhel7    latest   b0679ef6  8 seconds ago  405 MB  
registry[...]com/rhscl/mysql-57-rhel7 latest   4ae3a3f4  9 months ago  405 MB
```

- ▶ 2. Build the To Do List application parent image using the Node.js Dockerfile.

- 2.1. Review the Dockerfile.

Using your preferred editor, open and examine the completed Dockerfile located at **/home/student/D0285/labs/networking-containers/images/nodejs/Dockerfile**.

Notice the following instructions defined in the Dockerfile:

- Two environment variables, **NODEJS_VERSION** and **HOME**, are defined using the **ENV** command.
- A custom **yum** repo pointing to the local offline repository is added using the **ADD** command.
- Packages necessary for Node.js are installed with **yum** using the **RUN** command.
- A new user and group is created to run the Node.js application along with the **app-root** directory using the **RUN** command.
- The **enable-rh-nodejs4.sh** script is added to **/etc/profile.d/** to run automatically on login using the **ADD** command.
- The **USER** command is used to switch to the newly created **appuser**.
- The **WORKDIR** command is used to switch to the **\$HOME** directory for application execution.
- The **ONBUILD** command is used to define actions that run when any child container image is built from this image. In this case, the **COPY** command copies the **run.sh** file and the **build** directory into **\$HOME** and the **RUN** command runs **scl enable rh-nodejs4** as well as **npm install**. The **npm install** has a local Node.js module registry specified to override the default behavior of using **http://**

`npmjs.registry.org` to download the dependent node modules, so that the node application can be built without accessing Internet access.

2.2. Build the parent image.

Examine the script located at `/home/student/D0285/labs/networking-containers/images/nodejs/build.sh` to see how the image is built. To build the base image, run the `build.sh` script.

```
[student@workstation mysql]$ cd ../nodejs  
[student@workstation nodejs]$ ./build.sh
```

2.3. Wait for the build to complete, and then run the following command to verify that the image is built successfully.

```
[student@workstation nodejs]$ docker images  
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE  
do285/nodejs        latest   622e8a4d  5 minutes ago  407 MB  
do285/mysql-57-rhel7 latest   b0679ef6  24 minutes ago  405 MB  
registry[...]/rhscl/mysql-57-rhel7 latest   4ae3a3f4  9 months ago   405 MB  
registry.lab.example.com/rhel7    7.3     93bb76dd  12 months ago  193 MB
```

► 3. Build the To Do application child image using the Node.js Dockerfile.

3.1. Review the Dockerfile.

Using your preferred editor, open and examine the completed Dockerfile located at `/home/student/D0285/labs/networking-containers/deploy/nodejs/Dockerfile`.

3.2. Build the child image.

Examine the `/home/student/D0285/labs/networking-containers/deploy/nodejs/build.sh` script to see how the image is built. Run the following commands in order to build the child image.

```
[student@workstation nodejs]$ cd \  
~/D0285/labs/networking-containers/deploy/  
[student@workstation deploy]$ cd nodejs  
[student@workstation nodejs]$ ./build.sh
```



NOTE

The `build.sh` script lowers restriction for write access to the build directory to allow the installation of dependencies by non-root users.

3.3. Wait for the build to complete and then run the following command to verify that the image is built successfully:

```
[student@workstation nodejs]$ docker images  
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE  
do285/todonodejs    latest   21b7cdc6  6 seconds ago  435 MB  
do285/nodejs        latest   622e8a4d  11 minutes ago  407 MB  
do285/mysql-57-rhel7 latest   b0679ef6  29 minutes ago  405 MB
```

```
registry[...]/rhsc1/mysql-57-rhel7    latest  4ae3a3f4  9 months ago  405 MB
registry.lab.example.com/rhel7        7.3     93bb76dd  12 months ago  193 MB
```

► 4. Explore the Environment Variables.

Inspect the environment variables that allow the Node.js REST API container to communicate with the MySQL container.

- 4.1. View the file **/home/student/D0285/labs/networking-containers/ deploy/nodejs/nodejs-source/models/db.js** that holds the database configuration as given below.

```
module.exports.params = {
  dbname: process.env.MYSQL_DATABASE,
  username: process.env.MYSQL_USER,
  password: process.env.MYSQL_PASSWORD,
  params: {
    host: "mysql",
    port: "3306",
    dialect: 'mysql'
  }
};
```

- 4.2. Notice the environment variables being used by the REST API. These variables are exposed to the container using **-e** options with the **docker run** command in this guided exercise. The environment variables are described below.

MYSQL_DATABASE

This is the name of the MySQL database in the **mysql** container.

MYSQL_USER

The name of the database user that the **todoapi** container uses to run MySQL commands.

MYSQL_PASSWORD

The password of the database user that the **todoapi** container uses for authentication to the **mysql** container.



NOTE

The host and port details of the MySQL container are embedded with the REST API application. The host, as shown above the **db.js** file is the resolvable hostname of **mysql** container. Any container is reachable by its name from other containers that belong to the same user-defined Docker network.

► 5. Create and inspect an attachable user-defined bridge network called **do285-bridge**.

- 5.1. Run the following command to create the **do285-bridge** bridge.

```
[student@workstation nodejs]$ docker network create --attachable \
do285-bridge
```

6cf8659f66241bccbbd5cf38cdcb052d0f7846e632c868680221ef6f818e04a9

**NOTE**

The **--attachable** option allows containers to join the bridge dynamically.

- 5.2. Verify the docker network **do285-bridge** using **docker inspect** command.

```
[student@workstation nodejs]$ docker inspect do285-bridge
[
  {
    "Name": "do285-bridge",
    "Id": "6cf8...04a9",
    "Created": "2018-06-20T21:02:48.165825856+05:30",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

- 6. Modify the existing script to create containers in the same bridge, **do285-bridge**, created in the previous step. In this script, the order of commands is given such that it starts the **mysql** container and then starts the **todoapi** container before connecting it to the **mysql** container. After invoking every container, there is a wait time of 9 seconds, giving enough time for each container to start.
- 6.1. Edit the **run.sh** file located at **/home/student/D0285/labs/networking-containers/deploy/nodejs/networked/** to insert the **docker run** command at the appropriate line for invoking **mysql** container. The following screen shows the exact **docker** command to insert into the file.

```
docker run -d --name mysql -e MYSQL_DATABASE=items -e MYSQL_USER=user1 \
-e MYSQL_PASSWORD=mypa55 -e MYSQL_ROOT_PASSWORD=r00tpa55 \
-v $PWD/work/data:/var/lib/mysql/data \
-v $PWD/work/init:/var/lib/mysql/init -p 30306:3306 \
```

```
--network do285-bridge do285/mysql-57-rhel7
```

In the previous command, the **MYSQL_DATABASE**, **MYSQL_USER**, and **MYSQL_PASSWORD** are populated with the credentials to access the MySQL database. These environment variables are required for the **mysql** container to run.

- 6.2. In the same **run.sh** file, insert another **docker run** command at the appropriate line to run the **todoapi** container. The following screen shows the **docker** command to insert into the file.

```
docker run -d --name todoapi -e MYSQL_DATABASE=items \
-e MYSQL_USER=user1 \
-e MYSQL_PASSWORD=mypa55 -p 30080:30080 \
--network do285-bridge do285/todonodejs
```



NOTE

After each **docker run** command inserted into the **run.sh** script, ensure that there is also a **sleep 9** command. If you need to repeat this step, the work directory and its contents must be deleted prior to re-running the **run.sh** script.

- 6.3. Verify that your **run.sh** script matches the solution script located at **/home/student/D0285/solutions/networking-containers/deploy/nodejs/networked/run.sh**.
 - 6.4. Save the file and exit the editor.
- 7. Run the containers.
- 7.1. Use the following command to execute the script that you updated to run the **mysql** and **todoapi** containers, connected to the **do285-bridge** network.

```
[student@workstation nodejs]$ cd \
/home/student/D0285/labs/networking-containers/deploy/nodejs/networked
[student@workstation networked]$ ./run.sh
```

- 7.2. Verify that the containers started successfully.

```
[student@workstation networked]$ docker ps
CONTAINER ID IMAGE COMMAND STATUS PORTS
NAMES
2c61a089105d do285/todonodejs "scl enabl..." Up 7 minutes 0.0.0.0:30080->30080/tcp todoapi
0b9fa2821ba1 do285/mysql-57-rhel7 "container..." Up 7 minutes 0.0.0.0:30306->3306/tcp mysql
```

- 8. Examine the environment variables of the API container.

Run the following command to explore the environment variables that exposed in the API container.

```
[student@workstation networked]$ docker exec -it todoapi env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

```
HOSTNAME=2c61a089105d
TERM=xterm
MYSQL_DATABASE=items
MYSQL_USER=user1
MYSQL_PASSWORD=mypassword
container=oci
NODEJS_VERSION=4.0
HOME=/opt/app-root/src
```

▶ **9.** Test the application.

- 9.1. Run a **curl** command to test the REST API for the To Do List application.

```
[student@workstation networked]$ curl -w "\n" \
http://127.0.0.1:30080/todo/api/items/1
{"description": "Pick up newspaper", "done": false, "id":1}
```

The **-w "\n"** option with **curl** command lets the shell prompt appear at the next line rather than merging with the output in the same line.

- 9.2. Open Firefox on **workstation** and point your browser to <http://127.0.0.1:30080/todo/>. You should see the To Do List application.



NOTE

Make sure to append the trailing slash (/).

- 9.3. Verify that the correct images were built and that the application is running.

```
[student@workstation networked]$ lab networking-containers grade
```

▶ **10.** Clean up.

- 10.1. Navigate to the current user's home directory and stop the running containers:

```
[student@workstation networked]$ cd ~
[student@workstation ~]$ docker stop todoapi mysql
```

- 10.2. Delete the Docker bridge:

```
[student@workstation ~]$ docker network rm do285-bridge
```

- 10.3. Remove all containers and container images from the Docker builds:

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

This concludes the guided exercise.

▶ LAB

DEPLOYING MULTI-CONTAINER APPLICATIONS

In this lab, you will deploy a containerized database application using the Docker daemon on the workstation host.

RESOURCES

Files	/home/student/DO285/labs/deploy-instructor
Application URL:	http://localhost:8080/services/instructors

OUTCOMES

You should be able to:

- Create and configure a Docker container network.
- Create multiple containers with shared network connectivity.
- Coordinate shared configuration between containers.

BEFORE YOU BEGIN

To download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab deploy-instructor setup
```

1. Build the database container image for the instructor application. Tag the image as **instructor/mysql:latest**. The Dockerfile for the container image is located in the **/home/student/DO285/labs/deploy-instructor/mysql** directory.
Examine the **Dockerfile** to determine environment variables that affect the container's runtime behavior.
2. Build the PHP front end container image for the instructor application. Tag the image as **instructor/php:latest**. The Dockerfile for the container image is located in the **/home/student/DO285/labs/deploy-instructor/instructor** directory.
Examine the application's **services/api.php** file to determine environment variables that control connectivity to an external database. The database connection is configured in the **dbConnect()** function.
3. Create an attachable Docker network named **apps** for the two containers.
4. Create a database container from the **instructor/mysql:latest** image with the following properties:
 - The container runs as a detached process.
 - The container connects to the **apps** Docker network.

- The container name is **db**.
- The container creates a new database named **instructor**.
- The container creates a database user called **dbuser1** with a password of **mypa55**.

NOTE

Environment variables used by the **instructor/mysql:latest** image are documented in the **Dockerfile**.

5. Create a container named **webapp** from the **instructor/php:latest** image with the following properties:
 - The container runs as a detached process.
 - The container connects to the **apps** Docker network.
 - The container contains environment variables to configure the database connection to the **db** database container.
 - Local port 8080 requests are forwarded to port 8080 in the container.

NOTE

Environment variables that configure database connectivity are found in the **/home/student/D0285/labs/deploy-instructor/instructor/src/services/api.php** file. The **dbConnect()** function configures the database connection.

6. Verify the functionality of the application endpoint at **http://localhost:8080/services/instructors**. If JSON data is returned, the **webapp** container is able retrieve database records from the **db** container.
7. Run the grading script to verify that all the tasks were completed.

```
[student@workstation ~] lab deploy-instructor grade
```

8. Clean Up.

```
[student@workstation ~] lab deploy-instructor cleanup
```

This concludes the lab.

► SOLUTION

DEPLOYING MULTI-CONTAINER APPLICATIONS

In this lab, you will deploy a containerized database application using the Docker daemon on the workstation host.

RESOURCES	
Files	/home/student/DO285/labs/deploy-instructor
Application URL:	http://localhost:8080/services/instructors

OUTCOMES

You should be able to:

- Create and configure a Docker container network.
- Create multiple containers with shared network connectivity.
- Coordinate shared configuration between containers.

BEFORE YOU BEGIN

To download the files needed by this guided exercise, open a terminal on the **workstation** host and run the following command:

```
[student@workstation ~]$ lab deploy-instructor setup
```

1. Build the database container image for the instructor application. Tag the image as **instructor/mysql:latest**. The Dockerfile for the container image is located in the **/home/student/DO285/labs/deploy-instructor/mysql** directory.
Examine the **Dockerfile** to determine environment variables that affect the container's runtime behavior.
 - 1.1. Open a terminal on **workstation** and change to the **/home/student/DO285/labs/deploy-instructor/mysql** directory. Use the **docker build** command to build the database container with a tag of **instructor/mysql:latest**:

```
[student@workstation ~]$ cd \
/home/student/DO285/labs/deploy-instructor/mysql
[student@workstation mysql]$ docker build -t instructor/mysql:latest .
```

- 1.2. Briefly review the **Dockerfile** for environment variables that affect container behavior:

```
[student@workstation mysql]$ cat Dockerfile
FROM rhel7/mysql-5.7
```

```
# MySQL image
#
# Volumes:
#   * /var/lib/mysql/data - Datastore for MySQL
#   /var/lib/mysql/init - Folder to load *.sql scripts
# Environment:
#   * $MYSQL_USER - Database user name
#   * $MYSQL_PASSWORD - User's password
#   * $MYSQL_DATABASE - Name of the database to create
#   * $MYSQL_ROOT_PASSWORD (Optional) - Password for the 'root' MySQL account
```

2. Build the PHP front end container image for the instructor application. Tag the image as **instructor/php:latest**. The Dockerfile for the container image is located in the **/home/student/D0285/labs/deploy-instructor/instructor** directory.

Examine the application's **services/api.php** file to determine environment variables that control connectivity to an external database. The database connection is configured in the **dbConnect()** function.

- 2.1. Change to the **/home/student/D0285/labs/deploy-instructor/instructor** directory. Use the **docker build** command to build the database container:

```
[student@workstation mysql]$ cd \
/home/student/D0285/labs/deploy-instructor/instructor
[student@workstation instructor]$ docker build -t instructor/php:latest .
```

- 2.2. Briefly review the **src/services/api.php** file. Notice the environment variables used in the **dbConnect()** function:

```
[student@workstation instructor]$ head -n24 src/services/api.php | tail
/*
 * Connect to Database
*/
private function dbConnect(){
    $this->mysqli = new mysqli(getenv('MYSQL_HOST'),
        getenv('MYSQL_USER'),
        getenv('MYSQL_PASSWORD'),
        getenv('MYSQL_DATABASE'));
}
```

If authentication to the database host succeeds, a database connection is established to the database specified by the **MYSQL_DATABASE** variable.

3. Create an attachable Docker network named **apps** for the two containers.

Use the **docker network create** command to create an attachable network named **apps**:

```
[student@workstation instructor]$ docker network create --attachable apps
```

4. Create a database container from the **instructor/mysql:latest** image with the following properties:
- The container runs as a detached process.
 - The container connects to the apps Docker network.
 - The container name is **db**.
 - The container creates a new database named **instructor**.
 - The container creates a database user called **dbuser1** with a password of **mypa55**.

**NOTE**

Environment variables used by the **instructor/mysql:latest** image are documented in the **Dockerfile**.

Use the **docker run** command to start a database container based on the **instructor/mysql:latest** image. Use the **-d** to run the container as a detached process. Use the **--network=apps** option to connect the container to the apps Docker network. Use the **--name=db** option to specify the hostname of the container as db. Use the **-e KEY=VALUE** option to define an environment variable inside the container.

```
[student@workstation instructor]$ docker run -d \
--network=apps \
--name=db \
-e MYSQL_DATABASE=instructor \
-e MYSQL_USER=dbuser1 \
-e MYSQL_PASSWORD=mypa55 \
instructor/mysql:latest
```

5. Create a container named **webapp** from the **instructor/php:latest** image with the following properties:
- The container runs as a detached process.
 - The container connects to the apps Docker network.
 - The container contains environment variables to configure the database connection to the db database container.
 - Local port 8080 requests are forwarded to port 8080 in the container.

**NOTE**

Environment variables that configure database connectivity are found in the **/home/student/D0285/labs/deploy-instructor/instructor/src/services/api.php** file. The **dbConnect()** function configures the database connection.

Use the **docker run** command to start the PHP application container based on the **instructor/php:latest** image. Use the **-d** to run the container as a detached process. Use the **--network=apps** option to connect the container to the apps Docker network. Use the **--name=webapps** option to specify the hostname of the container as db. Use the

-e KEY=VALUE option to define an environment variable inside the container. Use the -p 8080:8080 option to map local port 8080 to port 8080 in the container.

```
[student@workstation instructor]$ docker run -d \
--network=apps \
--name=webapp \
-e MYSQL_HOST=db \
-e MYSQL_USER=dbuser1 \
-e MYSQL_PASSWORD=mypa55 \
-e MYSQL_DATABASE=instructor \
-p 8080:8080 \
instructor/php:latest
```

6. Verify the functionality of the application endpoint at `http://localhost:8080/services/instructors`. If JSON data is returned, the webapp container is able retrieve database records from the db container.

Use the `curl` command to verify the endpoint.

```
[student@workstation instructor]$ curl http://localhost:8080/services/instructors
[{"instructorNumber": "123", "instructorName": "...output omitted..."}
```

7. Run the grading script to verify that all the tasks were completed.

```
[student@workstation ~] lab deploy-instructor grade
```

8. Clean Up.

```
[student@workstation ~] lab deploy-instructor cleanup
```

This concludes the lab.

CHAPTER 7

INSTALLING OPENSHIFT CONTAINER PLATFORM

GOAL

Install OpenShift and configure the cluster.

OBJECTIVES

- Prepare the servers for installation.
- Execute the installation steps to build and configure an OpenShift cluster.
- Execute postinstallation tasks and verify the cluster configuration.

SECTIONS

- Preparing Servers for Installation (and Guided Exercise)
- Installing Red Hat OpenShift Container Platform (and Guided Exercise)
- Executing Postinstallation Tasks (and Guided Exercise)

LAB

- 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 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. Red Hat offers two different methods for installing Red Hat OpenShift Container Platform. The first method, known as Quick Installation, can be used for simple cluster setups. Quick Installation uses answers to a small set of questions to bootstrap installation. The second method, known as Advanced Installation, is designed for more complex installations. Advanced Installation uses *Ansible Playbooks* to automate the process.

Beginning with OpenShift Container Platform 3.9, the Quick Installation method is deprecated. As a result, this course uses the Advanced Installation method for installing Red Hat OpenShift Container Platform.

You must prepare cluster hosts prior to initiating installation. After running the Advanced Installation, you perform a smoke test to verify the cluster's functionality. In the comprehensive review lab at the end of the class, you practice manually performing these tasks. A brief introduction of Ansible follows because the Advanced Installation method relies on Ansible.

WHAT IS ANSIBLE?

Ansible is an open source automation platform which is used to customize and configure multiple servers in a consistent manner. Ansible Playbooks are used to declare the desired configuration of servers. Any server that is already in the declared state is left unchanged. The configuration of all other servers is adjusted to match the declared configuration. As a result, Ansible Playbooks are described as *idempotent*. An idempotent playbook can be executed repeatedly and result in the same final configuration.

OpenShift uses Ansible Playbooks and roles for installation, which are included in the *atomic-openshift-utils* package. The playbooks support a variety of installation and configuration scenarios.

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

- Preparing the hosts for OpenShift installation, such as package installation, disabling services, and configuring the Docker service.
- Installing Red Hat OpenShift.
- Executing smoke tests to verify the Red Hat OpenShift installation.

Installing Ansible

Use the **subscription-manager** command to enable the *rhel-7-server-ansible-2.4-rpms* repository:

```
# subscription-manager repos --enable="rhel-7-server-ansible-2.4-rpms"
```

When the repository is enabled, use the **yum install** command to install Ansible:

```
# yum install ansible
```

Ansible Playbooks Overview

Ansible Playbooks are used to automate configuration tasks. An Ansible Playbook is a YAML file that defines a list of one or more *plays*. Each play defines a set of tasks that are executed on a specified group of hosts. Tasks can be explicitly defined in the **tasks** section of a play. Tasks can also be encapsulated in an Ansible *role*. The following is an example of an Ansible Playbook:

```
---
- name: Install a File①
  hosts: workstations②
  vars:
    sample_content: "Hello World!"
  tasks:
    - name: "Copy a sample file to each workstation."
      copy:③
        content: "{{ sample_content }}"④
        dest: /tmp/sample.txt

- name: Hello OpenShift Enterprise v3.x⑤
  hosts: OSEv3
  roles:
    - hello⑥
```

- ① The first play in the playbook is called "Install a File". A play starts at every leftmost dash character in a playbook.
- ② Tasks in this play are applied to servers in the **workstations** host group.
- ③ The Ansible **copy** module is used in the first task of this play. The module ensures that a **/tmp/sample.txt** file exists on each machine in the **workstations** host group.
- ④ **sample_content** is a task variable. After playbook execution, the content of the **/tmp/sample.txt** file matches the value of this variable.
- ⑤ The second play in the playbook is named "Hello OpenShift Enterprise v3.x". Tasks in the second play are applied to all servers in the **OSEv3** host group. The **hello** role documentation states that the role uses a variable named **hello_message**.
- ⑥ The second play applies tasks found in the **hello** role. Ansible roles make it easier to re-use a set of tasks in other playbooks. Roles often use variables to customize their behavior. In this example, the **hello** role uses a variable called **hello_message**.

Ansible Inventory Files

Ansible Playbooks execute against a set of servers. Ansible inventory files define the groups of machines referenced in a set of playbooks. Any host group referenced in a playbook play must have a corresponding entry in the inventory file. For example, OpenShift playbooks operate on,

among others, a **masters** host group. An inventory file used with OpenShift Playbooks must define which hosts belong to the **masters** group.

In this class, playbook variables are defined in the inventory file. These variables can be defined for an entire group or can be defined on a per-host basis.

Ansible inventory files are **.INI** files. The following inventory file describes the host groups in the classroom environment. This inventory file also includes variable definitions needed for the previous playbook:

```
[workstations]  
workstation.lab.example.com  
  
[nfs]  
services.lab.example.com  
  
[masters]  
master.lab.example.com  
  
[etcd]  
master.lab.example.com  
  
[nodes]  
master.lab.example.com hello_message="I am an OSEv3 master."  
node1.lab.example.com  
node2.lab.example.com  
  
[OSEv3:children]  
masters  
etcd  
nodes  
nfs  
  
[OSEv3:vars]  
hello_message="I am an OSEv3 machine."  
  
[workstations:vars]  
sample_content="This is a workstation machine."
```

- ➊ This statement defines the **workstations** host group. Each line that follows defines one machine in the host group. The **workstations** host group contains one host, **workstation.lab.example.com**.
- ➋ This is an example of a *host variable*. A host variable value takes precedence over a group variable with the same name. For **master.lab.example.com**, the value of the **hello_message** is "**I am an OSEv3 master.**".
- ➌ Host groups can be composed of other host groups. When a host group label contains **:children**, the entries that follow represent other member groups. The **OSEv3** group contains any machine that is also in any of the four listed host groups.
- ➍ When a group label contains **:vars**, the entries that follow represent group variables. Each machine in the group is assigned a variable with this value. The **hello** role in the second play uses a **hello_message** variable. Because the play executes against machines in the **OSEv3** host group, the variable must be defined for all **OSEv3** machines.

Executing Ansible Playbooks

In this class, related playbook artifacts are kept in a project directory. A simple playbook project contains the following artifacts:

- An **inventory** file.
- A **playbooks** directory. The directory may be absent if only a small number of playbooks exist.
- An **ansible.cfg** file. This file customizes the behavior of Ansible command-line utilities. This file often defines the SSH configuration for Ansible commands and specifies the default inventory file for playbooks.

In this class, the **ansible.cfg** file contains the following:

```
[defaults]
remote_user = student ①
inventory = ./inventory ②
roles_path = /home/student/do285-ansible/roles
log_path = ./ansible.log

[privilegeEscalation] ③
become = yes
becomeUser = root
becomeMethod = sudo
```

- ① For each task, Ansible connects to hosts as the `student` user using SSH.
- ② A file named **inventory** is used as the inventory file. This file is located in the same directory as the **ansible.cfg** file.
- ③ After connecting to a host, Ansible becomes the `root` user before executing any task.

This project structure simplifies the execution of playbook. From a terminal, change to the project's root directory. Use the **ansible-playbook** command to execute a playbook:

```
[student@workstation project-dir]$ ansible-playbook <playbook-filename>
```

To execute a playbook against a different inventory file use the **-i** option:

```
[student@workstation project-dir]$ ansible-playbook -i <inventory-file> <playbook-filename>
```

PREPARING THE ENVIRONMENT

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

- Passwordless SSH is configured for a user account on all remote machines. If the remote user is not the root user, passwordless sudo rights must be granted to the remote user. In the classroom environment, the `student` user account satisfies these requirements.
- The master and node hosts need to be able to communicate with each other. The **ping** command provides a way to verify 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. Use the **ping** or **dig** command to call a host name that does not exist in the domain. For example, in the classroom, the wildcard domain `apps.lab.example.com` is used for all applications running on OpenShift:

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

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 OpenShift router needs the wildcard DNS zone. 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.

The OpenShift Advanced Installation method has additional prerequisites. An Ansible Playbook has been developed for the classroom environment to satisfy these requirements. Run this playbook in order to satisfy the prerequisites listed below:

- Each OpenShift Container Platform cluster machine is a Red Hat Enterprise Linux 7.3, 7.4, or 7.5 host.
- Each OpenShift cluster host (that is, masters and nodes) is registered using Red Hat Subscription Management (RHSM), not RHN Classic. To register hosts, use the **subscription-manager register** command.
- Each host is attached to valid OpenShift Container Platform subscriptions. To attach hosts to a subscription, use the **subscription-manager attach** command.
- Only the required repositories are enabled. Besides the standard RHEL repository (*rhel-7-server-rpms*), the *rhel-7-server-extras-rpms*, *rhel-7-fast-datapath-rpms*, and *rhel-7-server-ansible-2.4-rpms* repositories are enabled. The *rhel-7-server-ose-3.9-rpms* repository provides the necessary OpenShift Container Platform packages. To enable the required repositories, use the **subscription-manager repos --enable** command. Enable these repositories on all masters and nodes in the OpenShift cluster.
- Base packages are installed on all OpenShift hosts: *wget*, *git*, *net-tools*, *bind-utils*, *yum-utils*, *iptables-services*, *bridge-utils*, *bash-completion*, *kexec-tools*, *sos*, *psacct*, and *atomic-openshift-utils*. The Advanced Installation method uses playbooks and other installation utilities found in the *atomic-openshift-utils* package.
- *docker* is installed and configured on each OpenShift host. By default, the Docker daemon stores container images using a thin pool on a loopback device. For production Red Hat OpenShift clusters, the Docker daemon must use a thin pool logical volume. Use the **docker-storage-setup** command to set up appropriate storage for the Docker daemon. The Red Hat OpenShift Documentation covers many of the considerations for setting up Docker storage on OpenShift hosts.

EXECUTING HOST PREPARATION TASKS

An Ansible Playbook, **prepare_install.yml**, is provided for the classroom environment to automatically run the preinstallation tasks. Execute this playbook to prepare hosts for installing Red Hat OpenShift Container Platform.

**NOTE**

The **prepare_install.yml** file is a custom playbook written specifically for the classroom environment. This playbook is not included in any official repository or package.

The **prepare_install.yml** playbook ensures that:

- The Docker daemon is installed.
- Storage is configured for the Docker daemon.
- Each node trusts the private registry's certificate.
- Required base packages for OpenShift are installed.

**REFERENCES****DO407: Automation with Ansible I**

<https://www.redhat.com/en/services/training/do407-automation-ansible-i>

OpenShift Container Platform Installation and Configuration Guide

https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/

► WORKSHOP

PREPARING FOR INSTALLATION

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

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` VMs are started, and to download files needed by this exercise, open a terminal and run the following command:

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

Change to the lab working directory:

```
[student@workstation ~]$ cd /home/student/D0285/labs/install-prepare
```

- ▶ 1. As of this writing, the installation playbooks are compatible with Ansible v2.4. Ensure that Ansible v2.4 is available for the installation playbooks.

- 1.1. Install the `ansible` package:

```
[student@workstation install-prepare]$ sudo yum install ansible
```

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

```
[student@workstation install-prepare]$ ansible --version
```

The output indicates that Ansible v2.4 is installed.

The output also shows that the `ansible` command is using the `ansible.cfg` file in `/home/student/D0285/labs/install-prepare/ansible.cfg` file from the `/home/student/D0285/labs/install-prepare` directory.

- 1.3. Briefly review the `ansible.cfg` file.

```
[student@workstation install-prepare]$ cat ansible.cfg
[defaults]
remote_user = student①
inventory = ./inventory②
log_path = ./ansible.log③
roles_path = /home/student/do285-ansible/roles④
```

```
[privilege_escalation]⑤
become = yes
become_user = root
become_method = sudo
```

- ➊ By default, Ansible connects to remote machines as the student user using SSH.
 - ➋ The default inventory file for all Ansible commands is the `./inventory` file.
 - ➌ Playbook output is recorded in the `./ansible.log` file.
 - ➍ Ansible uses the specified path to access roles.
 - ➎ By default, Ansible becomes the root user with the `sudo` command after it connects to a remote machine.
- ▶ 2. The inventory file contains VMs and variables to configure the OpenShift installation. Briefly review the `inventory` file.

```
[student@workstation install-prepare]$ cat inventory
[workstations]
workstation.lab.example.com

[nfs]
services.lab.example.com

[masters]
master.lab.example.com

[etcd]
master.lab.example.com

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

[OSEv3:children]
masters
etcd
nodes
nfs

#Variables needed by the prepare_install.yml playbook.
[nodes:vars]
registry_local=registry.lab.example.com
use_overlay2_driver=true
insecure_registry=false
run_docker_offline=true
docker_storage_device=/dev/vdb
```

The inventory file defines six host groups:

- **workstations**: A group for developer workstations. Playbooks are executed from this machine. The OpenShift Advanced Installation playbooks do not use this host group. This

group is included so that the inventory is representative of the entire set of VMs in the classroom environment.

- **nfs**: The VMs in the environment that provide NFS services for cluster storage.
- **masters**: The group of VMs used as master hosts in the OpenShift cluster.
- **etcd**: The group of VMs used for the OpenShift cluster's etcd service. In the classroom environment, the etcd service is designed to be collocated with master VMs.
- **nodes**: The group of VMs used as node hosts in the OpenShift cluster.
- **OSEv3**: The group of VMs that make up the OpenShift cluster. Any machine in the **masters**, **etcd**, **nodes**, or **nfs** group is also a member of the **OSEv3** group.

By default, **docker** uses online registries to download container images. Because there is no internet access in the classroom environment, **docker** is configured to use a secure private registry. The custom classroom playbook, **prepare_install.yml**, uses the variables defined in the **nodes:vars** section to configure **docker**.

Additionally, the installation configures the **docker** daemon on each host to use the **overlay2** graph driver to store container images. Docker supports a number of different graph driver configurations. See the documentation for the advantages and disadvantages of particular graph drivers.

- ▶ 3. Before initiating the installation process, verify that Ansible can connect to all VMs in the inventory.
- 3.1. Briefly review the custom playbook, **ping.yml**. This playbook is designed to verify Ansible's ability to connect to all machines in the inventory.

```
[student@workstation install-prepare]$ cat ping.yml
---
- name: Verify Connectivity
  hosts: all
  gather_facts: no
  tasks:
    - name: "Test connectivity to machines."
      shell: "whoami"
      changed_when: false
```

This playbook tests that Ansible can connect to each machine using **ssh** and become the **root** user after a connection has been established.

- 3.2. Run the **ping.yml** playbook. Use the **-v** option for verbose output.

```
[student@workstation install-prepare]$ ansible-playbook -v ping.yml
```

The expected output is shown below:

```
Using /home/student/D0285/labs/install-prepare/ansible.cfg as config file
PLAY [Verify Connectivity] ****
TASK [Test connectivity to machines.] ****
...output omitted...
ok: [node2.lab.example.com] => {"changed": false, "cmd": "whoami", ...,
"stderr": "", "stderr_lines": [], "stdout": "root"❶, "stdout_lines": ["root"]}
```

...output omitted...

- ➊ The output of the **whoami** command is expected to be **root**. This indicates that Ansible is able to escalate privileges using the **sudo** command.

- ▶ 4. A custom playbook is provided to prepare the classroom machines for OpenShift installation. Review the custom playbook **prepare_install.yml**.

```
[student@workstation install-prepare]$ cat prepare_install.yml
---
name: "Host Preparation: Docker tasks"
hosts: nodes
roles:
  - docker-storage
  - docker-registry-cert
  - openshift-node

#Tasks below were not handled by the roles above.
tasks:
  - name: Student Account - Docker Access
    user:
      name: student
      groups: docker
      append: yes
```

This playbook applies three roles to each VM in the **nodes** group: **docker-storage**, **docker-registry-cert**, and **openshift-node**.

The **docker-storage** role installs *docker* and configures storage for the *docker* daemon. The **docker-registry-cert** role ensures that the *docker* daemon on each node trusts the private Docker registry certificate. The **openshift-node** role installs the prerequisite base packages.

To see the task list for a particular role, examine the role's **tasks/main.yml** file.

```
[student@workstation install-prepare]$ cat roles/<role-name>/tasks/main.yml
```

After the roles are applied, the **docker** group is added to the **student** user on each node. This allows the **student** user to issue **docker** commands without needing **sudo**.

- ▶ 5. Run the custom classroom playbook to prepare the VMs for OpenShift.

From the workstation VM, run the following command:

```
[student@workstation install-prepare]$ ansible-playbook prepare_install.yml
```

You should see output similar to the following:

```
...output omitted...
PLAY RECAP ****
master.lab.example.com : ok=28   changed=24   unreachable=0   failed=0
node1.lab.example.com : ok=28   changed=24   unreachable=0   failed=0
```

```
node2.lab.example.com      : ok=28    changed=24    unreachable=0    failed=0
```

**NOTE**

The counts in the play recap might be different on your system. Make sure that there are no failed tasks during the execution of the playbook.

- 6. The playbook ensures that:

- Docker is installed and running on each node.
- Docker uses a logical volume for storage on each node.
- Each node trusts the private Docker registry's self-signed certificate.
- Base packages are installed on each node.

Verify that each node is correctly configured.

- 6.1. Verify that the docker service is started and enabled on each node.

```
[student@workstation install-prepare]$ for vm in master node1 node2; do
echo -e "\n$vm"
ssh $vm sudo systemctl status docker | head -n3
done
```

You should see output similar to the following:

```
master
* docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset:
  disabled)
  Active: active (running) since Mon 2018-08-13 04:39:29 PDT; 24min ago

node1
* docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset:
  disabled)
  Active: active (running) since Mon 2018-08-13 04:39:29 PDT; 24min ago

node2
* docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset:
  disabled)
  Active: active (running) since Mon 2018-08-13 04:39:29 PDT; 24min ago
```

- 6.2. Verify that the Docker daemon uses a logical volume named **docker-pool** from the volume group **docker-vg**.

Use the **lvs** command to list the logical volumes on each host. Use the **df -h** command to verify the logical volume is mounted on the **/var/lib/docker** directory.

```
[student@workstation install-prepare]$ for vm in master node1 node2; do
echo -e "\n$vm : lvs"
ssh $vm sudo lvs
```

```
echo -e "\n$vm : df -h"
ssh $vm sudo df -h | grep vg-docker
done
```

You should see output similar to the following:

```
master : lvs
  LV      VG      Attr      LSize  Pool ...output omitted...
  docker-pool① docker-vg -wi-ao---- <20.00g

master : df -h
/dev/mapper/docker--vg-docker--pool② 20G  1.6G   19G   8% /var/lib/docker③

node1 : lvs
  LV      VG      Attr      LSize  Pool ...output omitted...
  docker-pool① docker-vg -wi-ao---- <20.00g

node1 : df -h
/dev/mapper/docker--vg-docker--pool② 20G  1.8G   19G   9% /var/lib/docker③

node2 : lvs
  LV      VG      Attr      LSize  Pool ...output omitted...
  docker-pool① docker-vg -wi-ao---- <20.00g

node2 : df -h
/dev/mapper/docker--vg-docker--pool② 20G  1.8G   19G   9% /var/lib/docker③
```

- ① The logical volume **docker-pool** exists within the **docker-vg** volume group on each node.
- ②③ The **/var/lib/docker** directory is the mount point for the logical volume.

- 6.3. Use the **docker pull** command to verify that images are pulled from the local private registry by default. This also tests if the private registry certificate is trusted on each node.

```
[student@workstation install-prepare]$ for vm in master node1 node2; do
echo -e "\n$vm"
ssh $vm docker pull rhel7:latest
done
```

You should see output similar to the following:

```
master
Trying to pull repository ①registry.lab.example.com/rhel7 ...
latest: Pulling from registry.lab.example.com/rhel7
845d34b6bc6a: Pulling fs layer
...output omitted...
Status: Downloaded newer image② for registry.lab.example.com/rhel7:latest

node1
Trying to pull repository ①registry.lab.example.com/rhel7 ...
```

```

latest: Pulling from registry.lab.example.com/rhel7
845d34b6bc6a: Pulling fs layer
...output omitted...
Status: Downloaded newer image② for registry.lab.example.com/rhel7:latest

node2
Trying to pull repository ①registry.lab.example.com/rhel7 ...
latest: Pulling from registry.lab.example.com/rhel7
845d34b6bc6a: Pulling fs layer
...output omitted...
Status: Downloaded newer image② for registry.lab.example.com/rhel7:latest

```

- ① The rhel7:latest image is pulled from **registry.lab.example.com**, not **registry.access.redhat.com**.
- ② Because the download of the container image succeeded, the private registry certificate is trusted on each node.

6.4. Verify that the packages required for installation are present on each node.

```
[student@workstation install-prepare]$ for vm in master node1 node2; do
echo -e "\n$vm"
ssh $vm rpm -qa wget git net-tools bind-utils \
    yum-utils iptables-services bridge-utils bash-completion \
    kexec-tools sos psacct atomic-openshift-utils
done
```

You should see output similar to the following:

```

master
kexec-tools-2.0.15-13.el7.x86_64
git-1.8.3.1-13.el7.x86_64
net-tools-2.0-0.22.20131004git.el7.x86_64
bridge-utils-1.5-9.el7.x86_64
bash-completion-2.1-6.el7.noarch
atomic-openshift-utils-3.9.14-1.git.3.c62bc34.el7.noarch
sos-3.5-6.el7.noarch
bind-utils-9.9.4-61.el7.x86_64
wget-1.14-15.el7_4.1.x86_64
iptables-services-1.4.21-24.el7.x86_64
yum-utils-1.1.31-45.el7.noarch
psacct-6.6.1-13.el7.x86_64

node1
...output omitted...

node2
...output omitted...

```

This concludes the guided exercise.

INSTALLING RED HAT OPENSIFT CONTAINER PLATFORM

OBJECTIVE

After completing this section, students should be able to install and configure a Red Hat OpenShift Container Platform cluster.

ADVANCED INSTALLATION OVERVIEW

After preparing hosts, the Advanced Installation method involves four steps:

- Compose an inventory file to describe the desired cluster features and architecture.
- Execute the OpenShift **prerequisites.yml** playbook.
- Execute the OpenShift **deploy_cluster.yml** playbook.
- Verify the installation.

COMPOSING AN ADVANCED INSTALLATION INVENTORY FILE

As discussed in the previous section, the inventory file is located in a project directory. The project directory also contains an **ansible.cfg** file:

```
[defaults]
remote_user = student
inventory = ./inventory
log_path = ./ansible.log

[privilegeEscalation]
become = yes
becomeUser = root
becomeMethod = sudo
```

The **ansible.cfg** file defines the default inventory file and SSH configuration for Ansible commands.

The OpenShift installation playbooks expect inventory files to conform to a particular schema. An OpenShift Advanced Installation inventory file defines the following host groups:

masters

This group is mandatory and defines which hosts act as master hosts in the OpenShift cluster.

nodes

This group is mandatory and defines which hosts act as node hosts in the OpenShift cluster. All hosts listed in the **[masters]** section should also be included in this section.

etcd

The group of hosts that run the etcd service for the OpenShift cluster.

nfs

This group is optional and should only contain one host. If particular variables are present in the inventory file, the OpenShift playbooks install and configure NFS on this machine.

OSEv3

This group contains any machine that is a part of the OpenShift cluster. The installation playbooks reference this group to run cluster-wide tasks.

The inventory file from the previous guided exercise conforms to these requirements:

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

[nfs]
services.lab.example.com

[masters]
master.lab.example.com

[etcd]
master.lab.example.com

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

[OSEv3:children]
masters
etcd
nodes
nfs
```

The above inventory serves as a starting point for an OpenShift Advanced Installation inventory file. Group and host variables are added to define the installed cluster's characteristics. In the classroom environment, the inventory file must address the following requirements:

- Install the desired version of OpenShift Container Platform.
- Users authenticate to the cluster using `htpasswd` authentication.
- The wildcard DNS entry `apps.lab.example.com` is used as the subdomain for hosted OpenShift applications.
- NFS storage is used for the OpenShift `etcd` service and the OpenShift internal registry.
- The classroom container registry is used as the external registry, because there is no connectivity to `docker.io` or `registry.access.redhat.com`.

Installation Variables

OpenShift installation variables are documented in the **[OSEv3:vars]** section of the inventory. Installation variables are used to configure a number of OpenShift components, such as:

- A private container registry
- Persistent storage using Gluster, Ceph, or other 3rd party cloud providers
- Cluster metrics
- Cluster logging
- Custom cluster certificates

This section only covers the variables required for the classroom installation.



NOTE

If you are installing a cluster outside of this class, take time to study and understand the available options and variables. Consult the "Advanced Installation" section of the *Installation and Configuration Guide* listed in the References section for more information.

CONFIGURING THE OPENSIFT INSTALLATION VERSION

Red Hat recommends that system administrators decide on the major version of OpenShift to target, and to allow the installation playbook to take the latest minor release of that major version. To specify the OpenShift Container Platform deployment type and version to install, use the `openshift_deployment_type` and `openshift_release` variables respectively within the `[OSEv3:vars]` section.

```
openshift_deployment_type=openshift-enterprise①
openshift_release=v3.9②
```

- ① Specifies the OpenShift deployment type. Possible values are `openshift-enterprise` and `origin`. Use a value of `openshift-enterprise` to install Red Hat OpenShift Container Platform.
- ② Specifies the major version to be installed.

The classroom OpenShift cluster uses two other variables:

```
openshift_image_tag=v3.9.14①
openshift_disable_check=disk_availability,docker_storage,memory_availability②
```

- ① Containerized OpenShift services use images with a tag of "v3.9.14". This prevents the cluster from automatically upgrading to later container images.
- ② The classroom VMs do not conform to recommended system requirements for production use. The OpenShift playbooks are designed to fail early in the installation process when a node does not meet the minimum requirements. For non-production clusters, you can disable checks for the system requirements. A full list of checks is available in the documentation.

CONFIGURING AUTHENTICATION

The OpenShift Container Platform authentication is based on *OAuth*, which provides an HTTP-based API for authenticating both interactive and noninteractive 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. OpenShift supports the following identify providers:

- 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 machine can use OpenShift client commands and APIs.

You must configure another identity provider so that external users can access the OpenShift cluster.

htpasswd Authentication

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

The **htpasswd** utility saves user names and passwords in a colon-delimited plain text file. The password is hashed using MD5. If this file is changed to add or delete a user, or to change a user password, the OpenShift OAuth server rereads it automatically.

To configure an OpenShift cluster to use **HTPasswdPasswordIdentityProvider**, add the `openshift_master_identity_providers` variable in the Ansible inventory:

```
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', ❶
'filename': '/etc/origin/master/htpasswd'}]❷
```

- ❶ The **HTPasswdPasswordIdentityProvider** is the configured type of identity provider.
- ❷ The location of the credentials file on the master VM.

To specify the initial list of users and passwords, add the `openshift_master_htpasswd_users` variable to the inventory file. Refer to the following example:

```
openshift_master_htpasswd_users="{'user1':'$apr1$.NHMsZYc$MdmfWN5DM3q280/W7c51c/',
'user2':'$apr1$.NHMsZYc$MdmfWN5DM3q280/W7c51c/'}
```

Use the **htpasswd** command to generate hashed passwords:

```
[student@workstation ~]$ htpasswd -nb admin redhat
admin:$apr1$.NHMsZYc$MdmfWN5DM3q280/W7c51c/
```

The **openssl** command can also generate a salted hash, without requiring a user name:

```
[student@workstation ~]$ openssl passwd -apr1 redhat
$apr1$A05Inu1A$IXdpq6d/m7mxondHEy6zC1
```

CONFIGURING NETWORK REQUIREMENTS

Wildcard DNS

A wildcard DNS entry for infrastructure nodes enables any newly created routes to be automatically routable to the cluster under the subdomain. The wildcard DNS entry must exist in a unique subdomain, such as `apps.mycluster.com`, and resolve to either the host name or IP address of the infrastructure nodes. The inventory file variable that exposes the wildcard DNS entry is `openshift_master_default_subdomain`.

```
openshift_master_default_subdomain=apps.mycluster.com
```

Master Service Ports

The `openshift_master_api_port` variable defines the listening port for the master API. Although the default is 8443, when using dedicated hosts as masters you can use port 443 and omit the port number from connecting URLs. The master console port is set to the value of the `openshift_master_console_port` variable; the default port is 8443. The master console can also be set to use port 443, and the port number can be omitted from connecting URLs.

Firewalld

The default firewall service on OpenShift nodes is `iptables`. To use `firewalld` as the firewall service on all nodes, set the `os_firewall_use_firewalld` variable to `true`.

```
os_firewall_use_firewalld=true
```

CONFIGURING PERSISTENT STORAGE

Containers are used to provide several OpenShift services, such as the OpenShift Container Registry. By default, container data is ephemeral and is lost when the container is destroyed. The Kubernetes persistent volume framework provides a mechanism for containers to request and use persistent storage. To avoid data loss, these services are configured to use persistent volumes.

OpenShift supports several plug-ins to create persistent volumes using various storage technologies. Persistent volumes can be created using NFS, iSCSI, GlusterFS, Ceph, or other commercial cloud storage.

In this class, the OpenShift Container Registry and OpenShift Ansible Broker services are configured to use NFS persistent storage.



NOTE

NFS persistent storage is not supported for production OpenShift clusters. To allow NFS persistent storage on a non-production cluster, add `openshift_enable_unsupported_configurations=true` to the inventory file.

OpenShift Container Registry

To configure NFS persistent storage for the OpenShift Container Registry, add the following to the inventory file:

```
openshift_hosted_registry_storage_kind=nfs①
openshift_hosted_registry_storage_nfs_directory=/exports②
openshift_hosted_registry_storage_volume_name=registry③
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'④
openshift_hosted_registry_storage_volume_size=40Gi
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
```

- ① The installation playbook uses the NFS plug-in to create a persistent volume for the internal registry. Data written to the registry's persistent volume is written to an NFS export. The NFS export is hosted on the machine listed in the `[nfs]` section of the inventory file.
- ②③ The NFS export is created from the `/exports/registry` directory on the NFS server.
- ④ The recommended NFS mount options for the registry's persistent volume. Options here are identical to those found in an NFS `/etc(exports` file.

OpenShift Ansible Broker

The OpenShift Ansible Broker (OAB) is a containerized OpenShift service that deploys its own `etcd` service. The required variables for persistent Etcd storage are similar to the variables needed for the registry:

```
openshift_hosted_etcd_storage_kind=nfs
openshift_hosted_etcd_storage_nfs_directory=/exports
openshift_hosted_etcd_storage_volume_name=etcd-vol2
openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash,sync,no_wdelay)"
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_access_modes=[ "ReadWriteOnce" ]
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}
```



NOTE

Consult the *Installation and Configuration for Red Hat* guide listed in the References section for examples on how to configure persistent storage for other OpenShift services.

CONFIGURING A DISCONNECTED OPENSHIFT CLUSTER

By default, the OpenShift installation playbooks assume internet connectivity from the cluster. When an RPM or container image is needed, it is downloaded from an external source, such as `access.redhat.com`. A cluster that does not have connectivity to these external resources is called a disconnected cluster, or disconnected installation. The classroom OpenShift cluster is a disconnected installation as there is no internet connectivity.

When installing a disconnected OpenShift cluster, the needed RPMs and container images must be available in the environment.

In the classroom, RPM packages are hosted at `http://content.example.com`. The repositories are appropriately configured in the `/etc/yum.repos.d/training.repo` on all the OpenShift nodes.

Configuring a Different Registry

The container registry at `registry.lab.example.com` provides container images for OpenShift. To configure the cluster to pull images from the private registry, additional variables are needed in the inventory file:

```
#Modifications Needed for a Disconnected Install
oreg_url=registry.lab.example.com/openshift3/ose-${component}:${version}①
openshift_examples_modify_imagestreams=true②
openshift_docker_additional_registries=registry.lab.example.com③
openshift_docker_blocked_registries=registry.access.redhat.com,docker.io④
#Image Prefix Modifications⑤
openshift_web_console_prefix=registry.lab.example.com/openshift3/ose-
openshift_cockpit_deployer_prefix='registry.lab.example.com/openshift3/'
openshift_service_catalog_image_prefix=registry.lab.example.com/openshift3/ose-
template_service_broker_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_image_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_etcd_image_prefix=registry.lab.example.com/rhel7/
```

- ➊ The location of the accessible image registry. The value of this variable must end in **ose-\$component:\$version**.
- ➋ OpenShift installs with templates for deploying example applications. This variable instructs the playbooks to modify the image streams of all the examples to point to the private registry instead of `registry.access.redhat.com`. In previous versions of OpenShift, modifying the image streams for example applications was a postinstallation task.
- ➌ This variable is used to add the locally accessible registry to the **docker** configuration on each node.
- ➍ This variable is used to specify blocked registries in the **docker** configuration on each OpenShift node.
- ➎ Several variables exist to ensure that container images for OpenShift services can be downloaded from the private registry. By prefixing container image names with `registry.lab.example.com`, container images are downloaded from the private registry.

**NOTE**

Additional information regarding disconnected installations is available from the documentation.

CONFIGURING NODE LABELS

Node labels are arbitrary key/value pairs of metadata that are assigned to each node. Node labels are often used to differentiate geographic data centers or identify available resources on a node. Applications can declare in their deployment configuration a node selector in the form of a node label. If present, the application's pods must be deployed on a node with a matching node label. Node labels are set in the inventory file using the host variable `openshift_node_labels`:

```
[nodes]
...output omitted...
nodeX.example.com openshift_node_labels="{'zone':'west', 'gpu':'true'}"
...output omitted...
```

In the above example, the `nodeX.example.com` machine is assigned two node labels: **zone=west** and **gpu=true**.

A common architecture pattern for an OpenShift cluster is to differentiate master nodes, infrastructure nodes, and compute nodes. In this pattern, infrastructure nodes host pods for the OpenShift hosted registry and router, while compute nodes host application pods from user projects. Master nodes do not host application or infrastructure pods. Use node labels to identify the role of a particular node.

By default, all machines in the **[masters]** host group receive a node label of **node-role.kubernetes.io/master=true**.

The default node selector for OpenShift infrastructure services is **region=infra**. Any node that hosts infrastructure pods must have a node label of **region=infra**.

The default node selector for application pods is **node-role.kubernetes.io/compute=true**. Any node that hosts application pods must have this node label. Any node that is not a master or infrastructure node receives this node label during installation.

To configure the classroom environment according to this architecture, the **[nodes]** section of the inventory file becomes:

```
[nodes]
master.lab.example.com
node1.lab.example.com  openshift_node_labels="{'region':'infra'}"
node2.lab.example.com
```

The master VM is labeled with **node-role.kubernetes.io/master=true** because it is also contained in the **[masters]** host group. The node1 VM hosts infrastructure pods because it has a node label of **region=infra**. The node2 VM is labeled with **node-role.kubernetes.io/compute=true** because it is neither a master nor an infrastructure node. Application pods run on the node2 VM.

If a node is designed to host both infrastructure and application pods, both node labels must be explicitly defined:

```
[nodes]
...
nodeX.example.com  openshift_node_labels="{'region':'infra', 'node-
role.kubernetes.io/compute':'true'}"
...
```

EXECUTING THE OPENSHIFT PLAYBOOKS

Two playbooks are executed to install OpenShift: **prerequisites.yml** and **deploy_cluster.yml**. The *atomic-openshift-utils* package provides these playbooks and other Ansible artifacts. Install this package on the machine where you execute playbooks.

In the classroom environment, you execute playbooks from the `workstation` VM. This package is not installed on `workstation` as part of host preparation.

prerequisites.yml

Execute this playbook first to ensure that all the system requirements and prerequisites are satisfied for all of the OpenShift cluster machines. This playbook attempts to modify and fix nodes that do not meet the necessary prerequisites for OpenShift deployment.

```
[student@workstation ocp-project]$ ansible-playbook -i inventory \
> /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
```

When the playbook completes, a play recap is provided:

```
PLAY RECAP ****
master.lab.example.com      : ok=68    changed=12    unreachable=0    failed=0
node1.lab.example.com       : ok=61    changed=12    unreachable=0    failed=0
node2.lab.example.com       : ok=61    changed=12    unreachable=0    failed=0
INSTALLER STATUS ****
Initialization               : Complete (0:00:28)
```

As long as there are no failed tasks you can execute the **deploy_cluster.yml** playbook.

deploy_cluster.yml

This playbook deploys OpenShift Container Platform and assumes that the **prerequisites.yml** playbook has been previously executed.

```
[student@workstation ocp-project]$ ansible-playbook -i inventory \
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

```
...output omitted...
PLAY RECAP ****
master.lab.example.com      : ok=601    changed=251  unreachable=0    failed=0
node1.lab.example.com       : ok=134    changed=51   unreachable=0    failed=0
node2.lab.example.com       : ok=134    changed=51   unreachable=0    failed=0
...output omitted...
```

The deployment succeeded if there are no failed tasks in the play recap.

VERIFYING THE INSTALLATION

When the `deploy_cluster.yml` playbook completes, the OpenShift web console is used to quickly verify the installation. For a cluster with one master, the URL of the web console is of the form: `https://<FQDN of the master>:port`. The port is the value of the `openshift_master_console_port` variable. In the classroom environment, the OpenShift web console URL is `https://master.lab.example.com`.

NOTE

Because the default port for HTTPS is 443, the classroom OpenShift web console URL can be shortened to: `https://master.lab.example.com`.

Enter the OpenShift web console URL in a browser. If the login screen appears, OpenShift is installed and running.

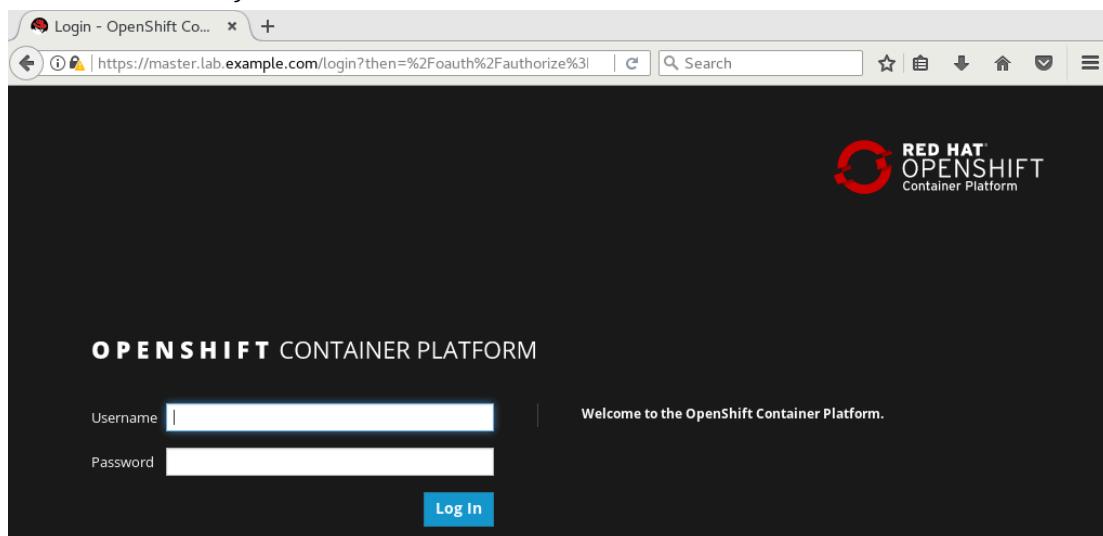


Figure 7.1: Web console login

Further verification is needed to ensure that all OpenShift services are up and running. The next section outlines postinstallation tasks and steps to verify the installation.

REFERENCES

Additional information about the installation process is available in the *Advanced Installation* section of the *Installation and Configuration* document which can be found at
https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/

► GUIDED EXERCISE

INSTALLING RED HAT OPENSHIFT CONTAINER PLATFORM

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

RESOURCES

Files	/home/student/D0285/labs/install-run
-------	--------------------------------------

OUTCOMES

You should be able to:

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

BEFORE YOU BEGIN

The guided exercise from the section called “Workshop: Preparing for Installation” should be completed. 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-prepare setup  
[student@workstation ~]$ sudo yum install ansible  
[student@workstation ~]$ cd /home/student/do285-ansible  
[student@workstation do285-ansible]$ ansible-playbook playbooks/  
prepare_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 and run the following command:

```
[student@workstation ~]$ lab install-run setup  
[student@workstation ~]$ cd /home/student/D0285/labs/install-run  
[student@workstation install-run]$
```

This guided exercise simulates the development of the inventory file needed to install OpenShift Container Platform in the disconnected classroom environment. Groups of related variables are documented in separate files to aid the simulation. These files are then combined to create the Advanced Installation inventory file.

- 1. Install the *atomic-openshift-utils* package.

```
[student@workstation install-run]$ sudo yum install atomic-openshift-utils
```

This package provides the Ansible Playbooks and roles needed to install OpenShift Container Platform. To run the OpenShift Advanced Installation playbooks, an inventory file is needed.

- ▶ **2.** Create the inventory file needed for the OpenShift Advanced Installation method.
 - 2.1. A starting inventory file is available that is based on the inventory file from the previous exercise. Copy the **inventory.initial** file to a file named **inventory**. This enables you to easily start over if a mistake is made in a later step.

```
[student@workstation install-run]$ cp inventory.initial inventory
```

- 2.2. Add a section to the bottom of the **inventory** file for **OSEv3** group variables.

```
[student@workstation install-run]$ echo -e "\n[OSEv3:vars]" >> inventory
```

In the following steps, you will add variables to the inventory to configure:

- General cluster characteristics
- Authentication
- Networking
- Data persistence
- Disconnected installation variables

- ▶ **3.** Configure the OpenShift cluster to have the following high-level characteristics:

- Use default settings for OpenShift Container Platform, not OpenShift Origin.
- OpenShift version 3.9 is installed.
- Each containerized OpenShift service uses an image tagged with "v3.9.14".

- 3.1. Open the **general_vars.txt** file using a text editor.

```
#General Cluster Variables
openshift_deployment_type=
openshift_release=
openshift_image_tag=
openshift_disable_check=disk_availability,docker_storage,memory_availability
```

By default, OpenShift playbooks enforce recommended system requirements. This variable allows individual checks to be disabled. Typically these checks should not

be disabled. The classroom environment needs these checks disabled to allow the installation to proceed.

- 3.2. Add appropriate values for the three variables with missing values.

**IMPORTANT**

Do not copy and paste the solution from this guide for any of the ***_vars.txt** files. This could introduce white space errors that cause the lab grading script to fail. To copy content, use the corresponding file in the **/home/student/D0285/solutions** directory.

- 3.3. The **general_vars.txt** now contains:

```
#General Cluster Variables
openshift_deployment_type=openshift-enterprise
openshift_release=v3.9
openshift_image_tag=v3.9.14
openshift_disable_check=disk_availability,docker_storage,memory_availability
```

Save the **general_vars.txt** file.

- 4. Configure the cluster to use htpasswd authentication. The cluster should have two users, **admin** and **developer**. Both users should have a password of **redhat**.

- 4.1. Review the **authentication_vars.txt** file using a text editor.

```
#Cluster Authentication Variables
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'user1':'htpasswd_hash_here', 'user2':
'htpasswd_hash_here'}
```

- 4.2. Edit the **openshift_master_htpasswd_users** variable. Replace **user1** and **user2** with **admin** and **developer**, respectively.

- 4.3. Password files should not reveal that passwords are repeated across user accounts. Information about the distribution of passwords across users aides attackers/malicious users.

Generate two different MD5 password hashes of **redhat**.

```
[student@workstation install-run]$ openssl passwd -apr1 redhat
$apr1$Vd4/F6nT$xB.UFGvcZewPdMoAXSZJ1
[student@workstation install-run]$ openssl passwd -apr1 redhat
$apr1$jhQYVRa$A6LOPTN0dkSYnsGEhaHr4.
```

- 4.4. Replace each password hash placeholder in the **openshift_master_htpasswd_users** variable with one of the generated hashes.

```
#Cluster Authentication Variables
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
```

```
openshift_master_htpasswd_users={'admin':'$apr1$Vd4/F6nT$xYB.UFGvcZeWPdMoAXSJ1',
'developer': '$apr1$jhQYVWRa$A6L0PTN0dkSYnsGEhaHr4.'}
```

**NOTE**

It is not necessary to create multiple MD5 hashes of the same password. By creating multiple hashes, it is difficult to identify users with the same password.

**WARNING**

Ensure the **openshift_master_htpasswd_users** variable definition is contained on a single line. If the variable's value is split over two lines, neither the **admin** nor the **developer** user will be able to log in after the cluster installation. If this happens, make appropriate changes to the **/etc/origin/master/htpasswd** file on the **master** host.

4.5. Save changes to the **authentication_vars.txt** file.

► 5. Configure the OpenShift cluster's network-related characteristics:

- Use **firewalld** as the firewall.
- The API and Console are both accessible on the standard HTTPS port, 443.
- The wildcard DNS entry, `apps.lab.example.com`, is used as the subdomain for hosted applications.

5.1. Review the **networking_vars.txt** file using a text editor.

```
#OpenShift Networking Variables
firewalld_var=true
API_port_var=443
console_port_var=443
app_subdomain_var=apps.lab.example.com
```

- 5.2. Replace `firewalld_var` with the appropriate OpenShift variable.
- 5.3. Replace `API_port_var` with the appropriate OpenShift variable.
- 5.4. Replace `console_port_var` with the appropriate OpenShift variable.
- 5.5. Replace `app_subdomain_var` with the appropriate OpenShift variable.
- 5.6. The **networking_vars.txt** file now contains:

```
#OpenShift Networking Variables
os_firewall_use_firewalld=true
openshift_master_api_port=443
openshift_master_console_port=443
openshift_master_default_subdomain=apps.lab.example.com
```

Save the **networking_vars.txt** file.

- 6. Configure the OpenShift Container Registry (OCR) and OpenShift Ansible Broker (OAB) to use NFS for persistent storage.
- 6.1. Review the **persistence_vars.txt** file using a text editor.

```
#NFS is an unsupported configuration
allow_nfs_var=true

#OCR configuration variables
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=40Gi

#OAB's etcd configuration variables
openshift_hosted_etcd_storage_kind=nfs
openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash, sync, no_wdelay)"
openshift_hosted_etcd_storage_nfs_directory=/exports
openshift_hosted_etcd_storage_volume_name=etcd-vol2
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}
```

- 6.2. Using NFS for persistent storage is not a supported configuration. Replace `allow_nfs_var` with the OpenShift variable that allows unsupported configurations.
- 6.3. The **persistence_vars.txt** file now contains:

```
#NFS is an unsupported configuration
openshift_enable_unsupported_configurations=true

#OCR configuration variables
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=40Gi

#OAB's etcd configuration variables
openshift_hosted_etcd_storage_kind=nfs
openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash, sync, no_wdelay)"
openshift_hosted_etcd_storage_nfs_directory=/exports
openshift_hosted_etcd_storage_volume_name=etcd-vol2
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}
```

Save the **persistence_vars.txt** file.

▶ 7. Configure OpenShift as a disconnected installation.

- 7.1. Review the **disconnected_vars.txt** file using a text editor.

```
#Modifications Needed for a Disconnected Installation
oreg_url=/openshift3/ose-${component}:${version}
modify_imagestreams_var=true
openshift_docker_additional_registries=
openshift_docker_blocked_registries=

#Image Prefixes
openshift_web_console_prefix=registry.lab.example.com/openshift3/ose-
openshift_cockpit_deployer_prefix='registry.lab.example.com/openshift3/'
openshift_service_catalog_image_prefix=registry.lab.example.com/openshift3/ose-
template_service_broker_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_image_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_etcd_image_prefix=registry.lab.example.com/rhel7/
```

- 7.2. Add the classroom registry to the beginning of the `oreg_url` variable's value.
- 7.3. Replace `modify_imagestreams_var` with the appropriate OpenShift variable to modify the image stream resources.
- 7.4. Add the classroom registry to the `openshift_docker_additional_registries` variable.
- 7.5. Add `registry.access.redhat.com` and `docker.io` as blocked registries, separated by commas.
- 7.6. The **disconnected_vars.txt** file now contains:

```
#Modifications Needed for a Disconnected Install
oreg_url=registry.lab.example.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true
openshift_docker_additional_registries=registry.lab.example.com
openshift_docker_blocked_registries=registry.access.redhat.com,docker.io

#Image Prefixes
openshift_web_console_prefix=registry.lab.example.com/openshift3/ose-
openshift_cockpit_deployer_prefix='registry.lab.example.com/openshift3/'
openshift_service_catalog_image_prefix=registry.lab.example.com/openshift3/ose-
template_service_broker_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_image_prefix=registry.lab.example.com/openshift3/ose-
ansible_service_broker_etcd_image_prefix=registry.lab.example.com/rhel7/
```

Save the **disconnected_vars.txt** file.

The required **OSEv3** group variables have been defined. You now need to define the required host variables.

▶ 8. To maximize node utilization, both nodes host infrastructure and application pods. Set node labels of `region=infra` and `node-role.kubernetes.io/compute=true` on both `node1` and `node2`.

- 8.1. Review the **[nodes]** section of the inventory file using a text editor:

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

```
node2.lab.example.com
```

- 8.2. Add the `openshift_node_labels` host variable to both `node1` and `node2`.
 The variable should have a value of "`{'region':'infra', 'node-role.kubernetes.io/compute':'true'}`"

When finished, the `[nodes]` section of the inventory file should look like:

```
[nodes]
master.lab.example.com
node1.lab.example.com openshift_node_labels="{'region':'infra', 'node-role.kubernetes.io/compute':'true'}"
node2.lab.example.com openshift_node_labels="{'region':'infra', 'node-role.kubernetes.io/compute':'true'}"
```

- 8.3. Save the `inventory` file.

- 9. Add all of the content from the lab text files to the end of the `inventory` file.

```
[student@workstation install-run]$ cat general_vars.txt \
networking_vars.txt authentication_vars.txt persistence_vars.txt \
disconnected_vars.txt >> inventory
```



WARNING

Ensure you use double greater-than (`>>`) symbols to append to the inventory file:

```
>> inventory
```

If you use single greater-than (`>`) symbol, you will overwrite the existing file:

```
> inventory
```

If you do overwrite the `inventory` file, repeat the steps above. You can skip any step that modifies a `*_vars.txt` file.

- 10. Check the inventory file for errors.

The OpenShift Advanced installation takes a long time to complete. Errors in the inventory file require OpenShift to be reinstalled. To prevent repeated lengthy installations due to

an error in the inventory file, an inventory checker is provided. To verify that there are no errors, run the following grading script on the workstation VM:

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

If your **inventory** file is valid, the grading script should pass, and you should see output similar to the following:

```
Overall inventory file check..... PASS
```

If your inventory file has a different variable value than the solution inventory file, expect output similar to:

```
* Checking openshift_hosted_registry_storage_access_mode..... FAIL
1c1
<❶ [ OSEv3:vars ] openshift_hosted_registry_storage_access_modes =
['ReadWriteMany']
---

>❷ [ OSEv3:vars ] openshift_hosted_registry_storage_access_modes =
['ReadWriteMany']
```

- ❶ The variable's value from your inventory file.
- ❷ The variable's value from the solution inventory file.



IMPORTANT

The grading script is sensitive to differences in white space. Check that your variable matches the solution's variable exactly.

If your inventory file has a misspelled variable or the variable is missing, expect output similar to the following:

```
* Checking openshift_hosted_registry_storage_access_mode..... FAIL
1c1
< [ OSEv3:vars ] ❶ openshift_hosted_registry_storage_access_modes =
['ReadWriteMany']
---

>❶
```

- ❶ In this example, the `openshift_hosted_registry_storage_access_modes` variable is not present in your inventory file.

Correct the errors and rerun the grading script to ensure that it passes. You can compare your **inventory** file with the solution file located at [/home/student/do285-ansible/inventory-run](#).

- 11. Run the OpenShift Container Platform **prerequisites.yml** playbook.

```
[student@workstation install-run]$ ansible-playbook \
```

```
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
```

**NOTE**

The **ansible.cfg** file specifies the default inventory file as **./inventory**. To explicitly specify the inventory file, use the **-i inventory file** option.

When the playbook completes you should see the following output in the terminal:

```
PLAY RECAP ****
localhost : ok=12    changed=0    unreachable=0    failed=0
master.lab.example.com : ok=68    changed=12   unreachable=0    failed=0
node1.lab.example.com  : ok=61    changed=12   unreachable=0    failed=0
node2.lab.example.com  : ok=61    changed=12   unreachable=0    failed=0
services.lab.example.com : ok=37    changed=4    unreachable=0    failed=0
workstation.lab.example.com : ok=2    changed=0    unreachable=0    failed=0
INSTALLER STATUS ****
Initialization : Complete (0:00:28)
```

- ▶ 12. Run the OpenShift Container Platform **deploy_cluster.yml** playbook.

```
[student@workstation install-run]$ ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

**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.

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

```
PLAY RECAP ****
localhost : ok=13    changed=0    unreachable=0    failed=0
master.lab.example.com : ok=601   changed=251   unreachable=0    failed=0
node1.lab.example.com  : ok=134   changed=51    unreachable=0    failed=0
node2.lab.example.com  : ok=134   changed=51    unreachable=0    failed=0
services.lab.example.com : ok=32    changed=8     unreachable=0    failed=0
workstation.lab.example.com : ok=21    changed=0    unreachable=0    failed=0

INSTALLER STATUS ****
Initialization : Complete (0:00:28)
Health Check   : Complete (0:00:21)
etcd Install   : Complete (0:01:16)
NFS Install    : Complete (0:00:17)
Master Install  : Complete (0:02:45)
Master Additional Install : Complete (0:01:23)
Node Install    : Complete (0:03:15)
Hosted Install   : Complete (0:03:00)
Web Console Install : Complete (0:00:44)
```

Service Catalog Install : Complete (0:02:44)

Ansible prints a summary of the playbook execution. The counts in the play recap might be different on your system. Make sure that there are no errors during the execution of the installer.

**NOTE**

Detailed information about the installation process is captured in the **ansible.log** file in the **install-run** lab directory. Use this file to troubleshoot issues during the installation or if the installation fails.

- 13. Log in as the **developer** user to the OpenShift web console to verify the installation.
- 13.1. Enter <https://master.lab.example.com> in the browser and trust the self-signed certificate generated by OpenShift. The OpenShift Console Login page should appear:

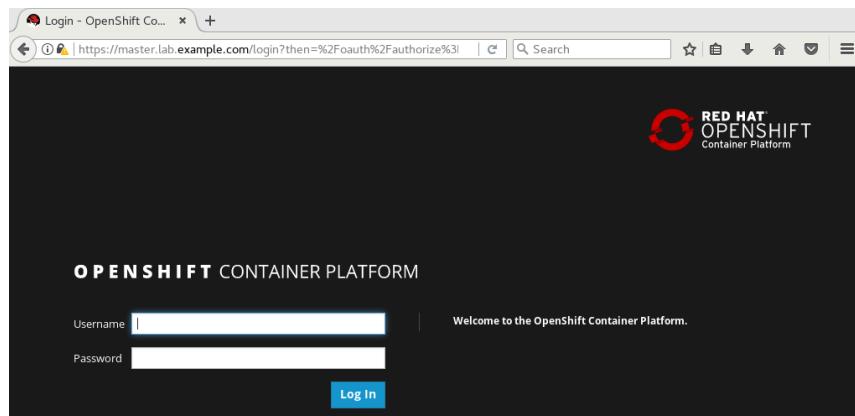


Figure 7.2: Web console login

- 13.2. Log in with **developer** as the user name and **redhat** as the password. After logging in, the OpenShift Catalog should appear:

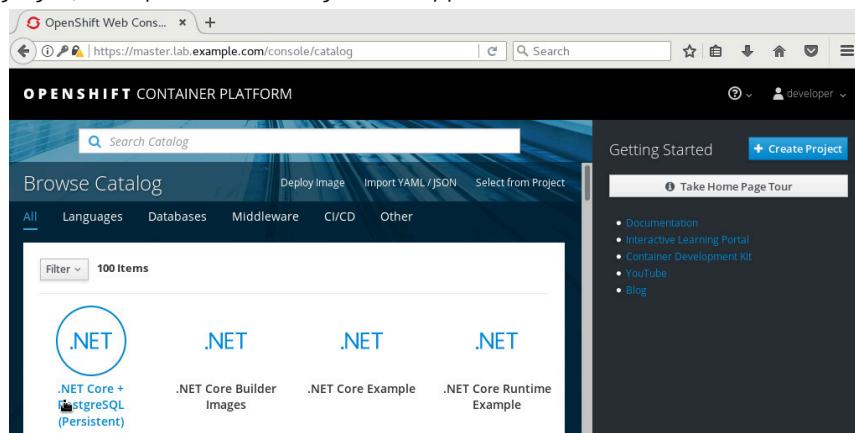


Figure 7.3: OpenShift Container Platform Catalog

This concludes the guided exercise.

EXECUTING POSTINSTALLATION TASKS

OBJECTIVES

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

OVERVIEW

After you have installed Red Hat OpenShift Container Platform, you need to test and verify all of the 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, you:

1. Retrieve the status of all the OpenShift nodes. All of the nodes should in a **Ready** status.
2. Retrieve the status of the OpenShift registry and router pods. All of the pods should in a **Running** status.
3. Build an application from source using the OpenShift cluster. OpenShift generates a container image from the build results and starts a pod from that image. This tests that the cluster can pull from and push to the internal registry. It also tests that applications are correctly scheduled and deployed to OpenShift nodes.
4. Create a route so that the application can be accessed from computers outside the OpenShift cluster's internal network. This tests that the OpenShift router is working and routing external requests to application pods.

You can use the **oc** command or the OpenShift web console to execute the preceding steps. To enable verification, a minimal set of commands is provided in this section. The **oc** command is discussed in detail in Chapter 11, *Executing Commands*.

Some of the above tasks require elevated cluster privileges. Before executing these steps, you need to associate a user account with appropriate cluster privileges.

CONFIGURING A CLUSTER ADMINISTRATOR

After following the installation steps from the previous section, an **admin** user account exists and can authenticate to the OpenShift cluster:

```
[student@workstation ~]$ oc login -u admin -p redhat https://  
master.lab.example.com  
...output omitted...  
  
Login successful.  
  
...output omitted...
```

The **admin** user account, however, does not have cluster administration privileges.

Immediately after installation, the **system:admin** user is the only user with cluster administration privileges. The **root** user on each master VM is authenticated as the **system:admin** user to the OpenShift cluster:

```
[student@workstation ~]$ ssh master
[student@master ~]$ sudo -i
[root@master ~]# oc whoami
system:admin
```

As the system:admin user, add the cluster-admin role to the admin user with the following command:

```
[root@master ~]# oc adm policy add-cluster-role-to-user cluster-admin admin
```

The admin user account can now be used to remotely execute cluster administration commands.



WARNING

The cluster-admin role is very powerful and gives the admin user the ability to destroy and modify cluster resources. If the admin account password is compromised, the entire cluster is at risk. Finely grained access controls should be implemented to mitigate the risk of catastrophic cluster damage. See Chapter 12, *Controlling Access to OpenShift Resources* for more details.

VERIFYING THE INSTALLATION

When the admin user has cluster administration privileges, use the **oc login** command to authenticate to the cluster:

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

If you have not previous logged in, you are prompted to provide a password for the admin user. If prompted, accept the self-signed certificate that was created during the installation process.

Verifying Node Status

To verify that all the OpenShift nodes are in a **Ready** state, execute the **oc get nodes** command:

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

You should see output similar to the following:

NAME	STATUS	ROLES	AGE	VERSION
master.lab.example.com	Ready	master	19m	v1.9.1+a0ce1bc657
node1.lab.example.com	Ready	compute	19m	v1.9.1+a0ce1bc657
node2.lab.example.com	Ready	compute	19m	v1.9.1+a0ce1bc657

Notice that both node1 and node2 are also labeled as **compute** nodes. This is a result of the **node-role.kubernetes.io/compute=true** node label specified in the inventory file.

Verifying Router and Registry Status

Execute the **oc get pods** command to verify that the router and registry pods have a status of **Running**.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
```

docker-registry-1-9ckhk	1/1	Running	0	11m
docker-registry-1-nccxn	1/1	Running	0	11m
registry-console-1-s2tkt	1/1	Running	0	9m
router-1-hmtlx	1/1	Running	0	12m
router-1-qvx6	1/1	Running	0	12m

**NOTE**

The registry deploys two types of pods, one for the docker registry and another that provides a web console for browsing registry images.

Building an Application

OpenShift applications and the associated OpenShift resources are contained within an OpenShift project. You need a new project to build a test application from source code. To create a project named **smoke-test** for the test application, use the **oc new-project** command:

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

The command output guides you to build an application using the **oc new-app** command:

```
Now using project "smoke-test" on server "https://master.lab.example.com:443".
```

```
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
```

The **oc new-app** command requires specification of an application builder image and the application's source code repository URL. For example, to build a python v3.4 application with source code hosted at <http://gitserver.example.com/my-python-app>, use the following command:

```
[student@workstation ~]$ oc new-app python:3.4~http://gitserver.example.com/my-python-app
```

**NOTE**

In the **oc new-app** command, the image specification and the source code repository are separated by a tilde (~) character.

Again, the output guides you on commands to execute next:

```
--> Found image ...output omitted... for "python:3.4"  
...output omitted...  
  
--> Creating resources ...  
imagestream "my-python-app" created  
buildconfig "my-python-app" created  
deploymentconfig "my-python-app" created
```

```
service "my-python-app" created
--> Success

Build scheduled, use 'oc logs -f bc/my-python-app'① to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:

'oc expose svc/my-python-app'②
Run 'oc status' to view your app.
```

- ①** Execute this command after the **oc new-app** command. This command follows the build process logs. This process begins by cloning the source code and ends with the application image being pushed to the OpenShift internal registry.
- ②** The **oc expose svc** command enables access to the application from clients outside of the OpenShift cluster. Run this command after the **oc logs -f** command is finished.

To obtain the FQDN of the exposed application service, execute the **oc get routes** command:

```
[student@workstation ~]$ oc get routes
NAME           HOST/PORT          ...output omitted...
my-python-app  my-python-app-smoke-test.apps.lab.example.com ...output omitted...
```

In this example, the application is available at `http://my-python-app-smoke-test.apps.lab.example.com`. If the application is accessible at this endpoint, the OpenShift cluster is configured correctly.

Failed Verification

If any of the steps above fail, the *Executing Troubleshooting Commands* section of Chapter 11, *Executing Commands* contains detailed troubleshooting procedures.

If you are having difficulty in the classroom environment installing and verifying the OpenShift cluster, first reset the master, node1, and node2 hosts. Then run the following commands on the workstation host to ensure that the environment is set up correctly:

```
[student@workstation ~]$ lab install-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

► WORKSHOP

COMPLETING POSTINSTALLATION TASKS

In this exercise, you test the basic functionality of the OpenShift Container Platform cluster installation.

OUTCOMES

You should be able to:

- Configure the `admin` user as a cluster administrator.
- Use the OpenShift client (`oc`) to retrieve cluster status.
- Build and deploy a simple application on OpenShift to test the cluster's functionality.

BEFORE YOU BEGIN

The guided exercise *Preparing for Installation* and the guided exercise *Installing Red Hat OpenShift Container Platform* should have been completed. If not, reset the `master`, `node1`, and `node2` hosts, and run the following commands from the `workstation` host:

```
[student@workstation ~]$ lab install-prepare setup
[student@workstation ~]$ sudo yum install ansible atomic-openshift-utils
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ansible-playbook playbooks/
prepare_install.yml
[student@workstation do285-ansible]$ ansible-playbook playbooks/install_ocp.yml
[student@workstation do285-ansible]$ cd
[student@workstation ~]$
```

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
```

- 1. From `workstation`, log in to the OpenShift cluster as the `admin` user.
- From `workstation`, use the `oc` command to log in to the cluster as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://master.lab.example.com --insecure-skip-tls-verify=true
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.

The **--insecure-skip-tls-verify=true** option accepts the self-signed certificate generated during the installation.

- 1.2. Try to retrieve the status of the OpenShift nodes.

```
[student@workstation ~]$ oc get nodes
Error from server (Forbidden): nodes is forbidden: User "admin" cannot list nodes
at the cluster scope: User "admin" cannot list all nodes in the cluster
```

The command is expected to fail because the **admin** user is not associated with the **cluster-admin** role.

- 2. Associate the **admin** user with the **cluster-admin** role.

- 2.1. From **workstation**, use SSH to log in to the **master** host. Use **sudo** to become the **root** user.

```
[student@workstation ~]$ ssh master.lab.example.com
[student@master ~]$ sudo -i
[root@master ~]#
```

- 2.2. Verify that the **system:admin** user is authenticated to the OpenShift API.

```
[root@master ~]# oc whoami
system:admin
```

- 2.3. Associate the **cluster-admin** role to the **admin** user.

```
[root@master ~]# oc adm policy add-cluster-role-to-user cluster-admin \
admin
cluster role "cluster-admin" added: "admin"
```

- 2.4. Log off from the **master** host.

```
[root@master ~]# exit
logout
[student@master ~]$ exit
logout
Connection to master.lab.example.com closed.
[student@workstation ~]$
```

You can now use the **admin** user to remotely perform cluster administration tasks from the **workstation** machine.

- 3. Perform basic verification of the cluster installation using **oc get** commands.

- 3.1. From **workstation**, log in to the OpenShift cluster as the **admin** user. If prompted, enter **redhat** for the password.

```
[student@workstation ~]$ oc login -u admin
Logged into ...output omitted... as "admin" using existing credentials.❶
You have access to the following projects ...output omitted...:
* default
```

```
kube-public
kube-service-catalog
kube-system
logging
management-infra
openshift
openshift-ansible-service-broker
openshift-infra
openshift-node
openshift-template-service-broker
openshift-web-console
```

Using project "default".

- ➊ The **oc** command caches credentials by user and cluster domain name and assumes the new user intends to use the same cluster as the previous user.

3.2. Use the **oc get nodes** command to verify that the hosts all return a **Ready** status:

```
[root@master ~]# oc get nodes
NAME           STATUS    ROLES      AGE       VERSION
master.lab.example.com   Ready     master     14m      v1.9.1+a0ce1bc657
node1.lab.example.com    Ready     compute    14m      v1.9.1+a0ce1bc657
node2.lab.example.com    Ready     compute    14m      v1.9.1+a0ce1bc657
```

Both node1 and node2 have a role of **compute**. This shows that the node labels from the inventory file are applied correctly to each node.

3.3. Use the **oc get pods** command to verify that the internal registry and router pods are in a **Running** status:

```
[root@master ~]# oc get pods
NAME          READY   STATUS    RESTARTS   AGE
docker-registry-1-9ckhk   1/1     Running   0          11m
docker-registry-1-nccxn   1/1     Running   0          11m
registry-console-1-s2tkt  1/1     Running   0          9m
router-1-hmtlx        1/1     Running   0          12m
router-1-qvxv6        1/1     Running   0          12m
```

- ▶ 4. Perform an S2I build as the **developer** user to serve as a test of the cluster's functionality.

4.1. From the workstation VM, log in to the OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer
Authentication required for https://master.lab.example.com:443 (openshift)
Username: developer
Password:
```

Enter **redhat** for the password.

Login successful.

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

```
oc new-project <projectname>
```

- 4.2. Create a new project called **smoke-test**.

```
[student@workstation ~]$ oc new-project smoke-test
Now using project smoke-test on server "https://master.lab.example.com:443".
```

You can add applications ...*output omitted*... 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.

- 4.3. Perform an S2I build to deploy the **php-helloworld** application on the OpenShift cluster with PHP v5.6. The source code for the application is located at <http://services.lab.example.com/php-helloworld>.

```
[student@workstation ~]$ oc new-app \
php:5.6~http://services.lab.example.com/php-helloworld --name hello
```



NOTE

The **--name hello** option shortens the name of resources from **php-helloworld** to just **hello**. This results in shorter FQDNs for the deployed application.

- 4.4. Use the **oc logs -f** command to monitor the build process:

```
[student@workstation ~]$ oc logs -f bc/hello
```

The command does not complete immediately because the build process takes several seconds to complete. You should see output similar to the following:

```
Cloning "http://services.lab.example.com/php-helloworld" ...
Commit: ff80c967857225207864664a28fa078dbda31f33 (Establish remote repository)
Author: root <root@services.lab.example.com>
Date: Mon Aug 6 16:26:23 2018 -0700
---> Installing application source...
=> sourcing 20-copy-config.sh ...
---> 20:23:00      Processing additional arbitrary httpd configuration provided by
s2i ...
=> sourcing 00-documentroot.conf ...
=> sourcing 50-mpm-tuning.conf ...
=> sourcing 40-ssl-certs.sh ...

Pushing image docker-registry.default.svc:5000/smoke-test/hello:latest ...
Pushed 0/6 layers, 1% complete
Pushed 1/6 layers, 23% complete
Pushed 2/6 layers, 38% complete
Pushed 3/6 layers, 53% complete
Pushed 4/6 layers, 78% complete
Pushed 5/6 layers, 100% complete
Pushed 6/6 layers, 100% complete
```

```
Push successful
```

The output indicates that OpenShift is able to clone source code repositories, build images, and push new images to the internal registry.

- 4.5. Expose the `hello` service as a route.

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

- 4.6. Retrieve the external route to the `hello` service.

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT          ...output omitted...
hello     hello-smoke-test.apps.lab.example.com  ...output omitted...
```

- 4.7. Use the `curl` command to verify that the application responds to requests at the published route:

```
[student@workstation ~]$ curl hello-smoke-test.apps.lab.example.com
Hello, World! php version is 5.6.25
```

This concludes the guided exercise.

SUMMARY

In this chapter, you learned how to:

- Prepare the environment for OpenShift Container Platform installation using Ansible Playbooks.
- Configure an OpenShift Advanced Installation inventory file with appropriate host groups, group variables, and host variables.
- Configure master and node servers using the OpenShift Advanced Installation Ansible Playbooks.
- Validate a running OpenShift cluster by creating an application from source code and deploying it to OpenShift.

CHAPTER 8

DESCRIBING AND EXPLORING OPENSHIFT NETWORKING CONCEPTS

GOAL

Describe and explore OpenShift networking concepts.

OBJECTIVES

- Describe how OpenShift implements software-defined networking.
- Describe how OpenShift routing works and create a route.

SECTIONS

- 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 Container Platform 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 Container Platform 3.9, administrators can configure three SDN plug-ins for the pod network:

- The **ovs-subnet** plug-in, which is the default plug-in. **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. By using the VNID, pods from different projects cannot communicate with pod and services from a different project.

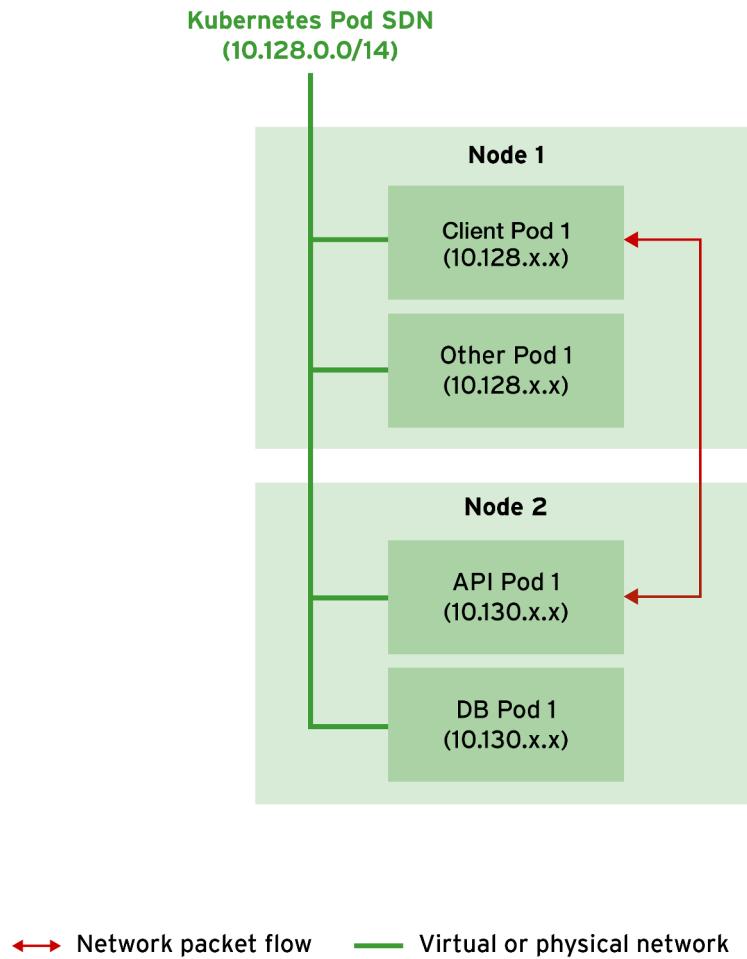


NOTE

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 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 8.1: 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 are 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.

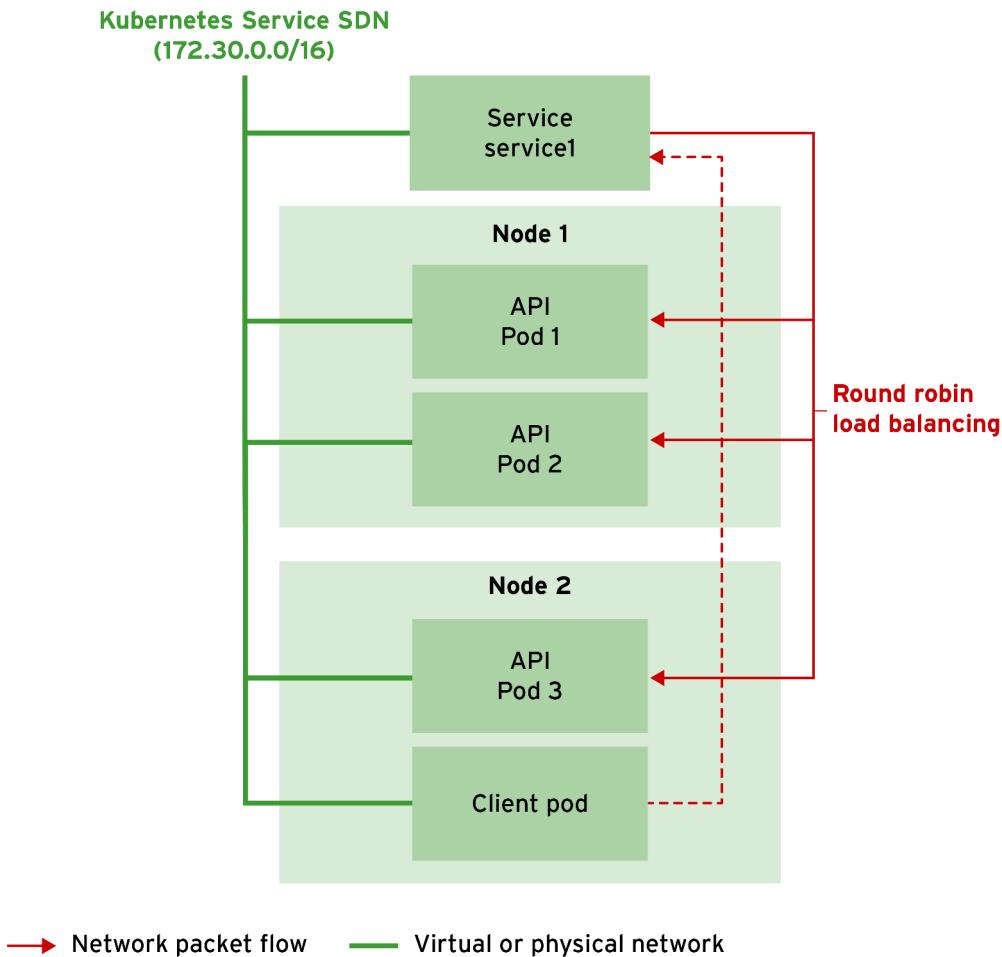


Figure 8.2: Kubernetes services networking

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.

OPENSIFT NETWORK TOPOLOGY

The set of pods running behind a service is managed automatically by OpenShift Container Platform. 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 (**Service** in the template definition).
- 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.



NOTE

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. In this approach, only HTTP, HTTPS, TLS with SNI, and WebSockets are currently supported.



NOTE

OpenShift routes are discussed in more detail in the next section.

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

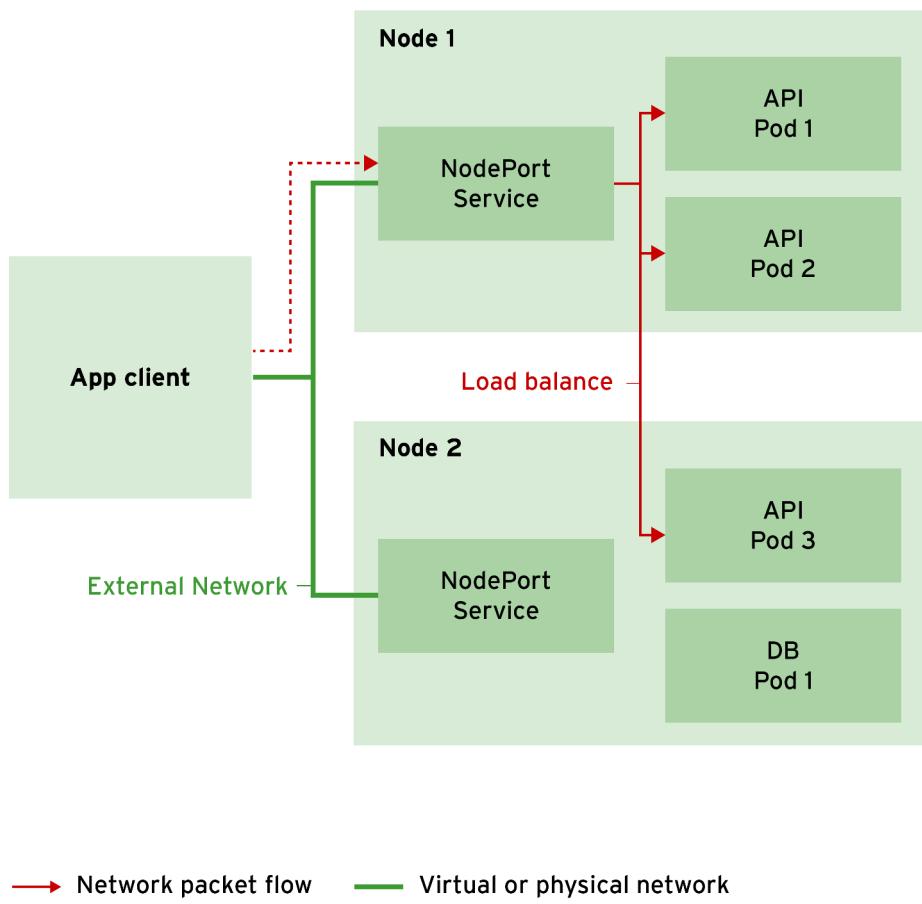


Figure 8.3: Kubernetes NodePort services

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 address of their host. As long as the host can resolve the server that the pod needs to reach, the pods can communicate with the target server using the *network address translation* (NAT) mechanism.



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 review OpenShift's Software-Defined Networking feature.

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

► 1. Create a new project.

- 1.1. From the **workstation** VM, access the OpenShift master at `https://master.lab.example.com` with the OpenShift client.
Log in as **developer** and accept the certificate if prompted.

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://master.lab.example.com
... output omitted ...
Use insecure connections? (y/n): y
```

- 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 private registry.

The application runs on port 8080 and displays the IP address of the host. In the environment, this corresponds to the pod IP address running the application:

```
[student@workstation ~]$ oc new-app --name=hello -i php:7.0 \
http://registry.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 as 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. Run the **oc scale** command to scale the application to two pods.



NOTE

It may take up to 3 minutes to scale the application.

```
[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:

```
[student@workstation ~]$ oc get pods -o wide
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 *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.

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

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

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

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

4.2. 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>
```

- 5. Verify that the application is accessible using the service IP address, which is the cluster IP:

5.1. From the **workstation** VM, 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.

5.2. 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; Connection refused
```

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

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

```
[student@workstation ~]$ ssh root@node1 curl http://172.30.105.51:8080
...
Server IP: 10.129.0.27
...
```

- 5.4. 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.

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

- 5.5. 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
Annotations:    openshift.io/generated-by=OpenShiftNewApp
Selector:        app=hello,deploymentconfig=hello
Type:            ClusterIP
IP:              172.30.171.155
Port:            8080-tcp  8080/TCP
TargetPort:      8080/TCP
Endpoints:      10.128.0.24:8080, 10.129.0.20:8080
Session Affinity: None
Events:          <none>
```

The **Endpoints** attribute displays 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 uses the **selectors** and **labels** that are defined for pods to load balance the application with a given cluster IP. OpenShift routes requests for this service to all pods labeled **app=hello** and **deploymentconfig=hello**.

Display the details one of the pods to ensure that labels are present.

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

- 6. Enable access to the application from outside the cluster.

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

6.1. 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. Update 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.

6.2. 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
Annotations:              openshift.io/generated-by=OpenShiftNewApp
Selector:                 app=hello,deploymentconfig=hello
Type:                     NodePort
IP:                       172.30.171.155
Port:                     8080-tcp  8080/TCP
TargetPort:               8080/TCP
NodePort:                 8080-tcp  30800/TCP
Endpoints:                10.128.0.24:8080,10.129.0.20:8080
Session Affinity:         None
External Traffic Policy: Cluster
Events:                  <none>
```

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.

- ▶ 7. You can see 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.

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

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

- 7.2. Access machines that are outside the OpenShift cluster from the pod. For example, access the Git repository hosted on the **services** VM to verify that pods can reach it.

```
sh-4.2$ curl http://services.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
<kay.sievers@vrfy.org>, Christian Gierke -->
<!-- git core binaries version 1.8.3.1 -->
...
...
```

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

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

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.

DESCRIBING 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.

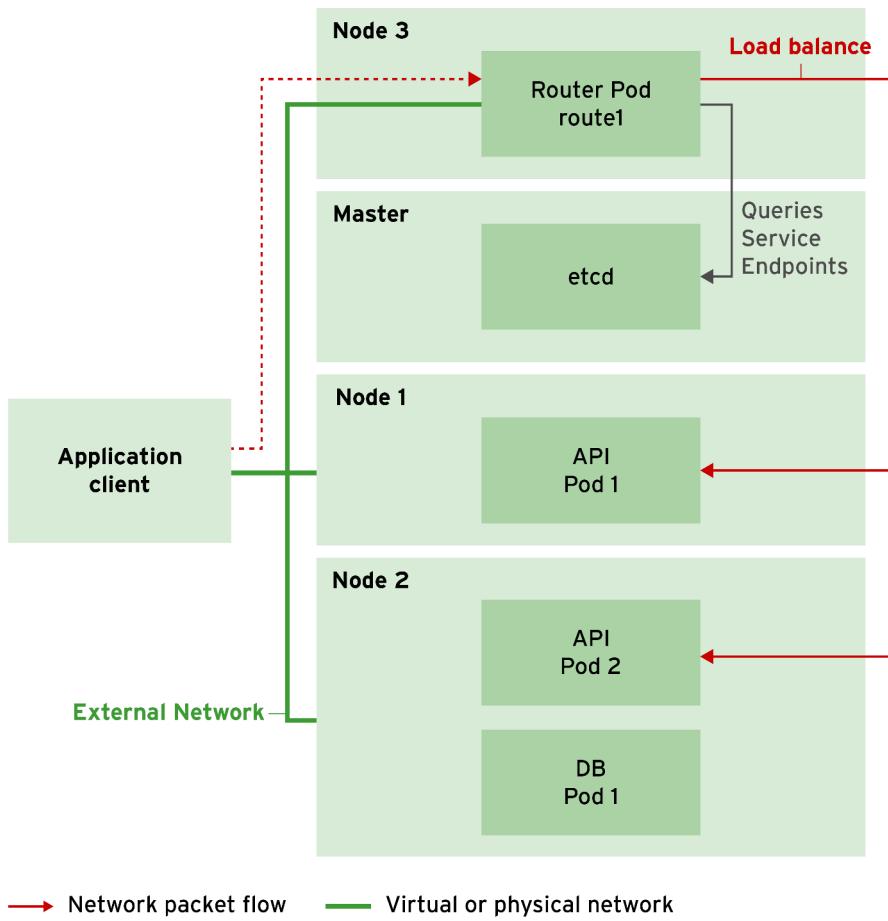


Figure 8.4: OpenShift routes

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.apps.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.apps.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 `apps.example.com` results in the FQDN `quote-test.apps.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: apps.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. The following steps describe how to create a simple self-signed certificate for a route called `test.example.com`.

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
```

4. 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:

1. 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
```

2. 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.

1. 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
```

2. 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 **STATS_PASSWORD** variable was randomly generated when you create the router. The **STATS_USERNAME** and **STATS_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 **firewall-cmd** to open the port specified by the **STATS_PORT** variable.

```
[root@node ~]# firewall-cmd --permanent --zone=public --add-port=1936  
[root@node ~]# firewall-cmd --reload
```

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

Type the value from **STATS_USERNAME** into the User Name field and from **STATS_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.

OUTCOMES

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

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

► 1. Create a new project.

- 1.1. From the **workstation** VM, connect to the OpenShift master server accessible at <https://master.lab.example.com> with the OpenShift client. Log in as **developer**. If prompted, accept the certificate.

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

- 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 private registry. The application runs on port 8080 and displays a simple text message.

**NOTE**

The command is available at **/home/student/D0285/labs/secure-route/commands.txt** to minimize typing errors.

```
[student@workstation ~]$ oc new-app \
--docker-image=registry.lab.example.com/openshift/hello-openshift \
--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/D0285/labs/secure-route** directory.

```
[student@workstation ~]$ cat \
/home/student/D0285/labs/secure-route/create-cert.sh
echo "Generating a private key..."
openssl genrsa -out hello.apps.lab.example.com.key 2048
...
echo "Generating a CSR..."
openssl req -new -key hello.apps.lab.example.com.key \
-out hello.apps.lab.example.com.csr \
-subj "/C=US/ST=NC/L=Raleigh/O=RedHat/OU=RHT/CN=hello.apps.lab.example.com"
...
echo "Generating a certificate..."
openssl x509 -req -days 366 -in \
hello.apps.lab.example.com.csr -signkey \
hello.apps.lab.example.com.key \
-out hello.apps.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/D0285/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.apps.lab.example.com  
Getting Private key  
  
DONE.
```

Verify that three files are created in the same folder:

- **hello.apps.lab.example.com.crt**
- **hello.apps.lab.example.com.csr**
- **hello.apps.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/D0285/labs/secure-route/commands.txt** file to minimize typing errors.

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

- 4.2. Ensure that the route is created.

```
[student@workstation secure-route]$ oc get routes  
NAME      HOST/PORT          SERVICES   PORT      TERMINATION ..  
hello     hello.apps.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.apps.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.apps.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 URL of the route.

```
[student@workstation secure-route]$ curl -k -vvv \
https://hello.apps.lab.example.com
* About to connect() to hello.apps.lab.example.com port 443 (#0)
*   Trying 172.25.250.11...
* Connected to hello.apps.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.apps.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.apps.lab.example.com
*   issuer: CN=hello.apps.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. To do so, use the IP address you noted from the **oc get pods -o wide** command.

Open a new terminal on the **workstation** VM and run the following command.

```
[student@workstation secure-route]$ ssh 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!
...
```

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:	http://hello.apps.lab.example.com:8080
------------------	---

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 the chapter *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-prepare setup  
[student@workstation ~]$ cd /home/student/do285-ansible  
[student@workstation do285-ansible]$ ./install.sh
```

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
```

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 the 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 but 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. From the **workstation** VM, view details about the service using the **oc describe svc** command. Review the endpoints that are 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.

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:

```
... output omitted ...
selector:
  app: hello-openshift
  deploymentconfig: hello-openshift
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

- 4.6. From the **master** VM, 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.
- Run the **oc describe route hello-openshift** command again. The service name and endpoints should now be displayed as follows.
- 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.

Clean up

Delete the **network-review** project.

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

This concludes the lab.

► SOLUTION

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:	http://hello.apps.lab.example.com:8080
-------------------------	---

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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

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 the user.

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

```
[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.apps.lab.example.com ...
```

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

```
[student@workstation ~]$ curl http://hello.apps.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
... output omitted ...
Failed connect to 172.30.162.216:8080; Connection refused
```

- 4.3. From the **workstation** VM, view details about the service using the **oc describe svc** command. Review the endpoints that are registered for this service.

```
[student@workstation ~]$ oc describe svc hello-openshift \
-n network-review
Name:           hello-openshift
Namespace:      network-review
Labels:         app=hello-openshift
Annotations:    openshift.io/generated-by=OpenShiftNewApp
Selector:       app=hello_openshift,deploymentconfig=hello-openshift
```

```
Type: ClusterIP
IP: 172.30.162.216
Port: 8080-tcp 8080/TCP
TargetPort: 8080/TCP
Endpoints: <none>
Port: 8888-tcp 8888/TCP
TargetPort: 8888/TCP
Endpoints: <none>
Session Affinity: None
Events: <none>
```

Notice that there are no endpoints for this service. This is the reason requests to the service IP returned a connection refused error. Remember 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.

```
[student@workstation ~]$ oc describe pod hello-openshift-1-2lxcg
Name: hello-openshift-1-2lxcg
Namespace: network-review
Node: node1.lab.example.com/172.25.250.11
Start Time: Wed, 18 Jul 2018 13:31:03 -0700
Labels: app=hello-openshift
        deployment=hello-openshift-1
        deploymentconfig=hello-openshift
Annotations: kubernetes.io/created-
by={"kind":"SerializedReference","apiVersion":"v1","reference":-
{"kind":"ReplicationController","namespace":"network-review","name":"hello-
openshift-1","uid":"5e91df96-57e1-11e7-9...
        openshift.io/deployment-config.latest-version=1
        openshift.io/deployment-config.name=hello-openshift
        openshift.io/deployment.name=hello-openshift-1
        openshift.io/generated-by=OpenShiftNewApp
        openshift.io/scc=restricted
Status: Running
IP: 10.130.0.16
Containers:
  hello-openshift:
    ...
    ... output omitted ...
```

Notice 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.
Edit the service configuration using the **oc edit svc** command from 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:

```
... output omitted ...
selector:
  app: hello-openshift
  deploymentconfig: hello-openshift
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

- 4.6. From the **master** VM, 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.apps.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.

```
[student@workstation ~]$ oc describe route hello-openshift
Name:      hello-openshift
Namespace: network-review
Created:   19 minutes ago
Labels:    app=hello-openshift
Annotations: <none>
Requested Host: hello.apps.lab.example.com
              exposed on router router 19 minutes ago
Path:      <none>
TLS Termination: <none>
Insecure Policy: <none>
Endpoint Port:  8080-tcp

Service:  hello-opensift
Weight:   100 (100%)
Endpoints: <error: endpoints "hello-opensift" not found>
```

Notice the error about no endpoints for this route. This is the reason requests to the route URL did not show the application home page. Remember 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.

```
... output omitted ...
spec:
  host: hello.apps.lab.example.com
  port:
    targetPort: 8080-tcp
  to:
    kind: Service
    name: hello-openshift
    weight: 100
  wildcardPolicy: None
... output omitted ...
```

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

```
[student@workstation ~]$ curl http://hello.apps.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.

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 9

DEPLOYING CONTAINERIZED APPLICATIONS ON OPENSHIFT

GOAL

Deploy single container applications on OpenShift Container Platform.

OBJECTIVES

- Create standard Kubernetes resources.
- Build an application using the Source-to-Image facility of OpenShift.
- Create a route to a service.
- Create an application using the OpenShift web console.

SECTIONS

- Creating Kubernetes Resources (and Guided Exercise)
- Creating Applications with the Source-to-Image Facility (and Guided Exercise)
- Creating Routes (and Guided Exercise)
- Creating Applications with the OpenShift Web Console (and Guided Exercise)

LAB

- Deploying Containerized Applications on OpenShift

CREATING KUBERNETES RESOURCES

OBJECTIVES

After completing this section, students should be able to create standard Kubernetes resources.

OPENShift CONTAINER PLATFORM (OCP) RESOURCES

The OpenShift Container Platform organizes entities in the OpenShift cluster as objects stored on the master node. These are collectively known as *resources*. The most common ones are:

Pod

A set of one or more containers that run in 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.

Persistent Volume (PV)

Containers' data are ephemeral. Their contents are lost when they are removed. Persistent Volumes represent storage that can be accessed by pods to persist data and are mounted as a file system inside a pod container.

Regular OpenShift users cannot create persistent volumes. They need to be created and provisioned by a cluster administrator.

Persistent Volume Claim (PVC)

A Persistent Volume Claim is a request for storage from a project. The claim specifies desired characteristics of the storage, such as size, and the OpenShift cluster matches it to one of the available Persistent Volumes created by the administrator.

If the claim is satisfied, any pod in the same project that references the claim by name gets the associated PV mounted as a volume by containers inside the pod.

Pods can, alternatively, reference volumes of type **EmptyDir**, which is a temporary directory on the node machine and its contents are lost when the pod is stopped.

Service (SVC)

A name representing a set of pods (or external servers) that are accessed by other pods. A service is assigned an IP address and a DNS name, and can be exposed externally to the cluster via a port or a route. It is also easy to consume services from pods, because an environment variable with the name **SERVICE_HOST** is automatically injected into other pods.

Route

A route is an external DNS entry (either a top-level domain or a dynamically allocated name) that is created to point to a service so that it can be accessed outside the OpenShift cluster. Administrators configure one or more routers to handle those routes.

Replication Controller (RC)

A replication controller ensures that a specific number of pods (or replicas) are running. These pods are created from a template, which is part of the replication controller definition. If pods are lost by any reason, for example, a cluster node failure, then the controller creates new pods to replace the lost ones.

Deployment Configuration (DC)

Manages replication controllers to keep a set of pods updated regarding container image changes. A single deployment configuration is usually analogous to a single microservice. A DC can support many different deployment patterns, including full restart, customizable rolling updates, and fully custom behaviors, as well as hooks for integration with external Continuous Integration (CI) and Continuous Development (CD) systems.

Build Configuration (BC)

Manages building a container image from source code stored in a Git server. Builds can be based on Source-to-Image (S2I) process or Dockerfile. Build configurations also support hooks for integration with external CI and CD systems.

Project

Projects have a list of members and their roles. Most of the previous terms in this list exist inside of an OpenShift project, in Kubernetes terminology, *namespace*. Projects have a list of members and their roles, such as viewer, editor, 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 projects to be created, but administrators control the resource quotas allocated to projects.

Regardless of the type of resource that the administrator is managing, the OpenShift command-line tool (**oc**) provides a unified and consistent way to update, modify, delete, and otherwise administer these resources, as well as helpers for working with the most frequently used resource types.

The **oc types** command provides a quick refresher on all the resource types available in OpenShift.

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 single file would contain many pod definitions, service definitions to connect the pods, replication controllers or **DeploymentConfigs** to horizontally scale the application pods, **PersistentVolumeClaims** to persist application data, and anything else needed that can be managed by OpenShift.

The **oc new-app** command can be used, with the option **-o json** or **-o yaml**, 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.

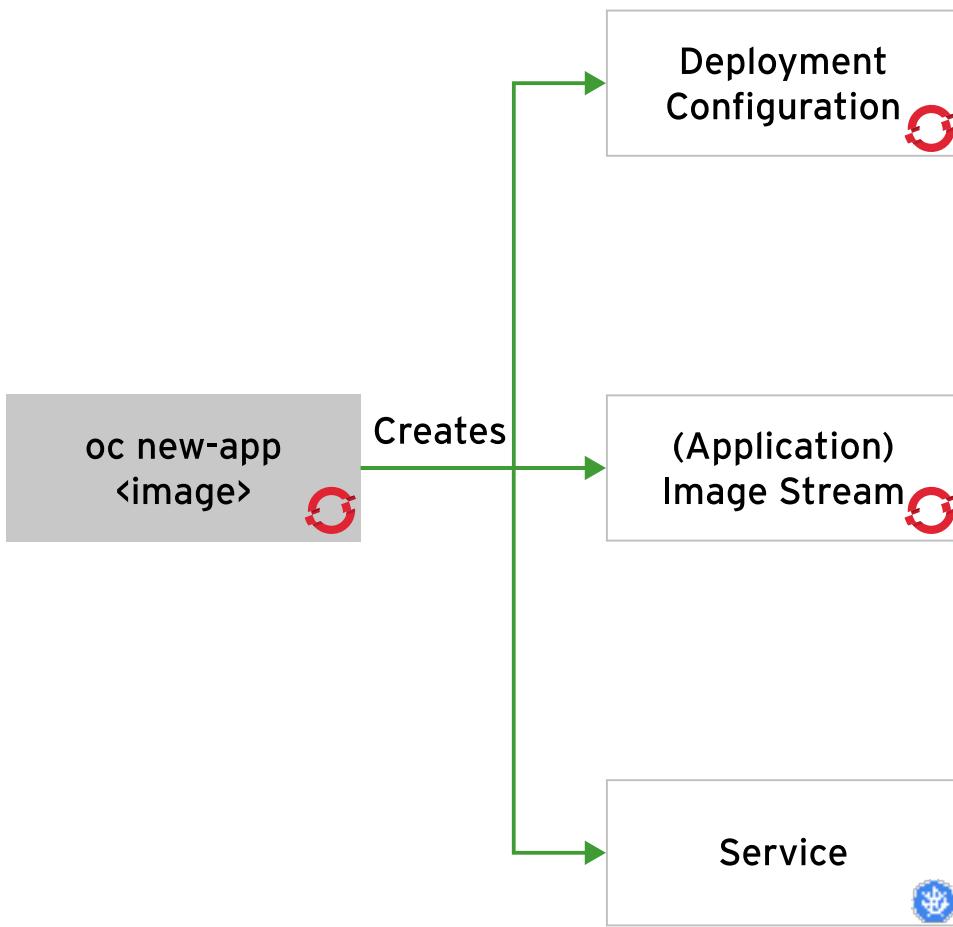
The **oc new-app** command can create application pods to run on OpenShift in many different ways. It can create pods from existing docker images, from Dockerfiles, and from raw source code using the Source-to-Image (S2I) process.

Run the **oc new-app -h** command to briefly understand all the different options available for creating new applications on OpenShift.

The following command creates an application based on an image, **mysql**, from Docker Hub, with the label set to **db=mysql**:

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

The following figure shows the Kubernetes and OpenShift resources created by the **oc new-app** command when the argument is a container image:



→ created by `oc new-app`

Figure 9.1: Resources created by the `oc new-app` command

The following command creates an application based on an image from a private Docker image registry:

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

The following command creates an application based on source code stored in a Git repository:

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

You will learn more about the Source-to-Image (S2I) process, its associated concepts, and more advanced ways to use `oc new-app` to build applications for OpenShift in the next section.

USEFUL COMMANDS TO MANAGE OPENSHIFT RESOURCES

There are several essential commands used to manage OpenShift resources as described below.

Typically, as an administrator, you will most likely use **oc get** command. This retrieves information about resources in the cluster. Generally, this command outputs 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 of an execution of **oc get pods** command.

NAME	READY	STATUS	RESTARTS	AGE
nginx-1-5r583	1/1	Running	0	1h
myapp-1-l44m7	1/1	Running	0	1h

oc get all

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

NAME	DOCKER REPO	TAGS	UPDATED	
is/nginx	172.30.1.1:5000/basic-kubernetes/nginx	latest	About an hour ago	
<hr/>				
NAME	REVISION	DESIRED	CURRENT	
dc/nginx	1	1	1	
<hr/>				
NAME	DESIRED	CURRENT	READY	
rc/nginx-1	1	1	1	
<hr/>				
NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/nginx	172.30.72.75	<none>	80/TCP, 443/TCP	1h
<hr/>				
NAME	READY	STATUS	RESTARTS	AGE
po/nginx-1-ypp8t	1/1	Running	0	1h

oc describe RESOURCE_TYPE 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 simply iterate through all 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
...	
No events	

oc export

This command can be used to export the a resource definition. Typical use cases include creating a backup, or to aid in modification of 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 allows the user to create resources from a resource definition. Typically, this is paired with the **oc export** command for editing definitions.

oc edit

This command allows the user to edit resources of a resource definition. By default, this command opens up a **vi** buffer for editing the resource definition.

oc delete RESOURCE_TYPE name

The **oc delete** command allows the user to remove a resource from an OpenShift cluster. Note that a fundamental understanding of the OpenShift architecture is needed here, because deletion of managed resources like pods result in newer instances of those resources being automatically recreated. When a project is deleted, it deletes all of the resources and applications contained within it.

oc exec CONTAINER_ID options command

The **oc exec** command allows the user to execute commands inside a container. You can use this command to run interactive as well as noninteractive batch commands as part of a script.

DEMONSTRATION: CREATING BASIC KUBERNETES RESOURCES

1. Log in to OpenShift as the **developer** user on the **workstation** VM:

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

2. Create a new project for the resources you will create during this demonstration:

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

3. Relax the default cluster security policy.

The **nginx** image from Docker Hub runs as root, but this is not allowed by the default OpenShift security policy.

As the **admin** user, change the default security policy to allow containers to run as root:

```
[student@workstation ~]$ oc login -u admin -p redhat https://
master.lab.example.com
Login successful.
...
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z default
scc "anyuid" added to: ["system:serviceaccount:basic-kubernetes:default"]
```

4. Log in back to OpenShift as the **developer** user and create a new application from the **nginx** container image using the **oc new-app** command.

Use the **--docker-image** option with **oc new-app** to specify the classroom private registry URI so that OpenShift does not try and pull the image from the Internet:

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

...
[student@workstation ~]$ oc new-app \
--docker-image=registry.lab.example.com/nginx:latest \
--name=nginx
--> Found Docker image ae513a4 (7 weeks old) from registry.lab.example.com for
"registry.lab.example.com/nginx:latest"

* An image stream will be created as "nginx:latest" that will track this image
* This image will be deployed in deployment config "nginx"
* Port 80/tcp will be load balanced by service "nginx"
  * Other containers can access this service through the hostname "nginx"
* WARNING: Image "registry.lab.example.com/nginx:latest" runs as the 'root'
user which may not be permitted by your cluster administrator

--> Creating resources ...
imagestream "nginx" created
deploymentconfig "nginx" created
service "nginx" created
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/nginx'
Run 'oc status' to view your app.
```

5. Run the **oc status** command to view the status of the new application, and verify if the deployment of the Nginx image was successful:

```
[student@workstation ~]$ oc status
In project basic-kubernetes on server https://master.lab.example.com:443

svc/nginx - 172.30.149.93:80
dc/nginx deploys istag/nginx:latest
  deployment #1 deployed 50 seconds ago - 1 pod

2 infos identified, use 'oc status -v' to see details.
```

6. List the pods in this project to verify if the Nginx pod is ready and running:

```
[student@workstation ~]$ oc get pods
NAME        READY     STATUS    RESTARTS   AGE
nginx-1-ypp8t  1/1      Running   0          25m
```

7. Use the **oc describe** command to view more details about the pod:

```
[student@workstation ~]$ oc describe pod nginx-1-ypp8t
Name:           nginx-1-ypp8t
```

```
Namespace: basic-kubernetes
Node: node2.lab.example.com/172.25.250.12
Start Time: Tue, 19 Jun 2018 10:46:50 +0000
Labels: app=nginx
        deployment=nginx-1
        deploymentconfig=nginx
Annotations: openshift.io/deployment-config.latest-version=1
            openshift.io/deployment-config.name=nginx
            openshift.io/deployment.name=nginx-1
            openshift.io/generated-by=OpenShiftNewApp
            openshift.io/scc=anyuid
Status: Running
IP: 10.129.0.44
...
```

8. List the services in this project and verify if a service to access the Nginx pod was created:

```
[student@workstation ~]$ oc get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx    ClusterIP  172.30.149.93  <none>          80/TCP       3m
```

9. Retrieve the details of **nginx** service and note the Service IP through which the Nginx pod can be accessed:

```
[student@workstation ~]$ oc describe service nginx
Name:           nginx
Namespace:      basic-kubernetes
Labels:         app=nginx
Annotations:    openshift.io/generated-by=OpenShiftNewApp
Selector:       app=nginx,deploymentconfig=nginx
Type:          ClusterIP
IP:            172.30.149.93
Port:          80-tcp  80/TCP
TargetPort:     80/TCP
Endpoints:     10.129.0.44:80
Session Affinity: None
Events:        <none>
```

10. Open an SSH session to the **master** machine and access the default Nginx home page using the service IP:

```
[student@workstation ~]$ ssh master curl -s http://172.30.149.93
<!DOCTYPE html>
<html>
...
<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>
...
```

Accessing the Service IP and port directly works in this scenario because **master** is the machine that manages the entire OpenShift cluster in the classroom environment.

11. Delete the project, to remove all the resources in the project:

```
[student@workstation ~]$ oc delete project basic-kubernetes
```

This concludes the demonstration.



REFERENCES

Additional information about pods and services is available in the *Pods and Services* section of the OpenShift Container Platform documentation:

Architecture

https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/architecture/

Additional information about creating images is available in the OpenShift Container Platform documentation:

Creating Images

https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/creating_images/

► GUIDED EXERCISE

DEPLOYING A DATABASE SERVER ON OPENSHIFT

In this exercise, you will create and deploy a MySQL database pod on OpenShift using the **oc new-app** command.

OUTCOMES

You should be able to create and deploy a MySQL database pod on OpenShift.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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 mysql-openshift setup
```

- 1. Log in to OpenShift as a developer user and create a new project for this exercise.

- 1.1. From the **workstation** VM, log in to OpenShift as the **developer** user with **redhat** as password:

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

If the **oc login** command prompts about using insecure connections, answer **y** (yes).



WARNING

Be careful with the user name. If a different user is used to run the following steps, the grading script fails.

- 1.2. Create a new project for the resources you will create during this exercise:

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

- ▶ 2. Create a new application from the **rhscl/mysql-57-rhel7** container image using the **oc new-app** command.

This image requires several environment variables (**MYSQL_USER**, **MYSQL_PASSWORD**, **MYSQL_DATABASE**, and **MYSQL_ROOT_PASSWORD**) to be fed using multiple instances of **-e** option.

Use the **--docker-image** option with **oc new-app** command to specify the classroom private registry URI so that OpenShift does not try and pull the image from the Internet:

```
[student@workstation ~]$ oc new-app \
--docker-image=registry.lab.example.com/rhscl/mysql-57-rhel7:latest \
--name=mysql-openshift \
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 -e MYSQL_DATABASE=testdb \
-e MYSQL_ROOT_PASSWORD=r00tpa55
--> Found Docker image 63d6bb0 (2 months old) from registry.lab.example.com for
"registry.lab.example.com/rhscl/mysql-57-rhel7:latest"
...
--> Creating resources ...
  imagestream "mysql-openshift" created
  deploymentconfig "mysql-openshift" created
  service "mysql-openshift" created
--> Success
  Application is not exposed. You can expose services to the outside world by
  executing one or more of the commands below:
    'oc expose svc/mysql-openshift'
  Run 'oc status' to view your app.
```

- ▶ 3. Verify if the MySQL pod was created successfully and view details about the pod and it's service.

- 3.1. Run the **oc status** command to view the status of the new application and verify if the deployment of the MySQL image was successful:

```
[student@workstation ~]$ oc status
In project mysql-openshift on server https://master.lab.example.com:443

svc/mysql-openshift - 172.30.205.27:3306
dc/mysql-openshift deploys istag/mysql-openshift:latest
  deployment #1 running for 11 seconds - 0/1 pods
...
```

- 3.2. List the pods in this project to verify if the MySQL pod is ready and running:

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE					

```
mysql-openshift-1-f1knk  1/1      Running   0       1m      10.129.0.25
node2.lab.example.com
```

**NOTE**

Notice the node on which the pod is running. You need this information to be able to log in to the MySQL database server later.

- 3.3. Use the **oc describe** command to view more details about the pod:

```
[student@workstation ~]$ oc describe pod mysql-openshift-1-ct78z
Name:           mysql-openshift-1-ct78z
Namespace:      mysql-openshift
Node:           node2.lab.example.com/172.25.250.12
Start Time:    Thu, 14 Jun 2018 09:01:19 +0000
Labels:         app=mysql-openshift
                deployment=mysql-openshift-1
                deploymentconfig=mysql-openshift
Annotations:   openshift.io/deployment-config.latest-version=1
                openshift.io/deployment-config.name=mysql-openshift
                openshift.io/deployment.name=mysql-openshift-1
                openshift.io/generated-by=OpenShiftNewApp
                openshift.io/scc=restricted
Status:        Running
IP:            10.129.0.23
...
...
```

- 3.4. List the services in this project and verify if a service to access the MySQL pod was created:

```
[student@workstation ~]$ oc get svc
NAME          TYPE      CLUSTER-IP     EXTERNAL-IP   PORT(S)      AGE
mysql-openshift  ClusterIP  172.30.205.27 <none>        3306/TCP   6m
```

- 3.5. Retrieve the details of **mysql-openshift** service using **oc describe** command and note that the Service type is **ClusterIP** by default:

```
[student@workstation ~]$ oc describe service mysql-openshift
Name:           mysql-openshift
Namespace:      mysql-openshift
Labels:         app=mysql-openshift
Annotations:   openshift.io/generated-by=OpenShiftNewApp
Selector:       app=mysql-openshift,deploymentconfig=mysql-openshift
Type:          ClusterIP
IP:            172.30.205.27
Port:          3306-tcp  3306/TCP
TargetPort:    3306/TCP
Endpoints:    10.129.0.23:3306
Session Affinity: None
Events:        <none>
```

- 3.6. View details about the deployment configuration (**dc**) for this application:

```
[student@workstation ~]$ oc describe dc mysql-openshift
Name:           mysql-openshift
```

```

Namespace:      mysql-openshift
Labels:         app=mysql-openshift
...
Deployment #1 (latest):
  Name:  mysql-openshift-1
  Created: 10 minutes ago
  Status:  Complete
  Replicas: 1 current / 1 desired
  Selector: app=mysql-openshift,deployment=mysql-
  openshift-1,deploymentconfig=mysql-openshift
  Labels:  app=mysql-openshift,openshift.io/deployment-config.name=mysql-openshift
  Pods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 Failed
...

```

- ▶ 4. Connect to the MySQL database server and verify if the database was created successfully.
- 4.1. From the **master** machine, connect to the MySQL server using the MySQL client.
Use **mysql-openshift.mysql-openshift.svc.cluster.local** as the Database server's hostname:

```

[student@workstation ~]$ ssh master

[student@master ~]$ mysql -P3306 -uuser1 -pmypa55 \
-hmysql-openshift.mysql-openshift.svc.cluster.local

Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.34 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>

```



NOTE

The services can only be accessed on the cluster servers themselves.

- 4.2. Verify if the **testdb** database has been created:

```

MySQL [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| testdb        |
+-----+
2 rows in set (0.00 sec)

```

- 4.3. Exit from the MySQL prompt:

```
MySQL [(none)]> exit
```

Bye

4.4. Exit from **master** node.

```
[student@master ~]$ exit
```

- 5. Verify that the database was correctly set up. Run the following command from a terminal window:

```
[student@workstation ~]$ lab mysql-openshift grade
```

- 6. Delete the project which in turn all the resources in the project:

```
[student@workstation ~]$ oc delete project mysql-openshift
```

This concludes the exercise.

CREATING APPLICATIONS WITH SOURCE-TO-IMAGE

OBJECTIVES

After completing this section, students should be able to deploy an application using the **Source-to-Image (S2I)** facility of OpenShift Container Platform.

THE SOURCE-TO-IMAGE (S2I) PROCESS

Source-to-Image (S2I) is a facility that makes it easy to build a container image from application source code. This facility takes an application's source code from a Git server, injects the source code into a base container based on the language and framework desired, and produces a new container image that runs the assembled application.

The following figure shows the resources created by the `oc new-app <source>` command when the argument is an application source code repository. Notice that a Deployment Configuration and all its dependent resources are also created.

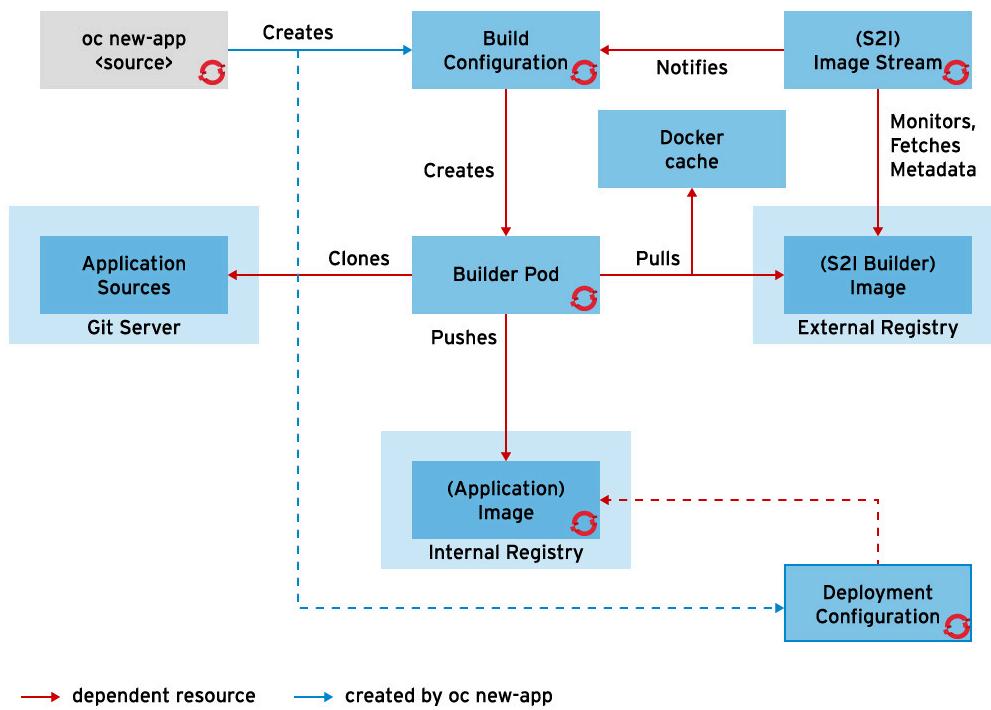


Figure 9.2: Deployment Configuration and dependent resources

S2I is the major strategy used for building applications in OpenShift Container Platform. The main reasons for using source builds are:

- **User efficiency:** Developers do not need to understand Dockerfiles and operating system commands such as `yum install`. They work using their standard programming language tools.
- **Patching:** S2I allows for rebuilding all the applications consistently if a base image needs a patch due to a security issue. For example, if a security issue is found in a PHP base image, then updating this image with security patches updates all applications that use this image as a base.

- **Speed:** With S2I, the assembly process can perform a large number of complex operations without creating a new layer at each step, resulting in faster builds.
- **Ecosystem:** S2I encourages a shared ecosystem of images where base images and scripts can be customized and reused across multiple types of applications.

IMAGE STREAMS

OpenShift deploys new versions of user applications into pods quickly. To create a new application, in addition to the application source code, a base image (the S2I builder image) is required. If either of these two components gets 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 should the builder image change.

The *image stream resource* is a configuration that names specific container images associated with *image stream tags*, an alias for these container images. An application is built against an image stream. The OpenShift installer populates several image streams by default during installation. To check available image streams, use the **oc get** command, as follows:

\$ oc get is -n openshift		
NAME	DOCKER REPO	TAGS
jenkins	172.30.0.103:5000/openshift/jenkins	2,1
mariadb	172.30.0.103:5000/openshift/mariadb	10.1
mongodb	172.30.0.103:5000/openshift/mongodb	3.2,2.6,2.4
mysql	172.30.0.103:5000/openshift/mysql	5.5,5.6
nodejs	172.30.0.103:5000/openshift/nodejs	0.10,4
perl	172.30.0.103:5000/openshift/perl	5.20,5.16
php	172.30.0.103:5000/openshift/php	5.5,5.6
postgresql	172.30.0.103:5000/openshift/postgresql	9.5,9.4,9.2
python	172.30.0.103:5000/openshift/python	3.5,3.4,3.3
ruby	172.30.0.103:5000/openshift/ruby	2.3,2.2,2.0



NOTE

Your OpenShift instance may have more or fewer image streams depending on local additions and OpenShift point releases.

OpenShift has the ability to detect when an image stream changes and to take action based on that change. If a security issue is found in the **nodejs-010-rhel7** image, it can be updated in the image repository and OpenShift can automatically trigger a new build of the application code.

An organization will likely choose several supported base S2I images from Red Hat, but may also create their own base images.

BUILDING AN APPLICATION WITH S2I AND THE CLI

Building an application with S2I can be accomplished using the OpenShift CLI.

An application can be created using the S2I process with the **oc new-app** command from the CLI.

```
$ oc new-app ①php~http://services.lab.example.com/app② --name=myapp③
```

- ➊ The image stream used in the process appears to the left of the tilde (~).
- ➋ The URL after the tilde indicates the location of the source code's Git repository.
- ➌ Sets the application name.

The **oc new-app** command allows creating applications using source code from a local or remote Git repository. If only a source repository is specified, **oc new-app** tries to identify the correct image stream to use for building the application. In addition to application code, S2I can also identify and process Dockerfiles to create a new image.

The following example creates an application using the Git repository at the current directory.

```
$ oc new-app .
```

**NOTE**

When using a local Git repository, the repository must have a remote origin that points to a URL accessible by the OpenShift instance.

It is also possible to create an application using a remote Git repository and a context subdirectory:

```
$ oc new-app https://github.com/openshift/sti-ruby.git \
--context-dir=2.0/test/puma-test-app
```

Finally, it is possible to create an application using a remote Git repository with a specific branch reference:

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git#beta4
```

If an image stream is not specified in the command, **new-app** attempts to determine which language builder to use based on the presence of certain files in the root of the repository:

LANGUAGE	FILES
ruby	Rakefile, Gemfile, config.ru
Java EE	pom.xml
nodejs	app.json, package.json
php	index.php, composer.json
python	requirements.txt, config.py
perl	index.pl, cpanfile

After a language is detected, **new-app** searches for image stream tags that have support for the detected language, or an image stream that matches the name of the detected language.

A JSON resource definition file can be created using the **-o json** parameter and output redirection:

```
$ oc -o json new-app php~http://services.lab.example.com/app --name=myapp >
s2i.json
```

A list of resources will be created by this JSON definition file. The first resource is the image stream:

```
...
{
    "kind": "ImageStream", ①
    "apiVersion": "v1",
    "metadata": {
        "name": "myapp", ②
        "creationTimestamp": null
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {},
    "status": {
        "dockerImageRepository": ""
    }
},
...

```

- ①** Define a resource type of image stream.
- ②** Name the image stream as **myapp**.

The build configuration (bc) is responsible for defining input parameters and triggers that are executed to transform the source code into a runnable image. The **BuildConfig** (BC) is the second resource and the following example provides an overview of the parameters that are used by OpenShift to create a runnable image.

```
...
{
    "kind": "BuildConfig", ①
    "apiVersion": "v1",
    "metadata": {
        "name": "myapp", ②
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "triggers": [
            {
                "type": "GitHub",
                "github": {
                    "secret": "S5_4BZpPabM6KrIuPBvI"
                }
            }
        ]
    }
}
```

```

        },
        {
            "type": "Generic",
            "generic": {
                "secret": "3q8K8JND0Rzhjoz1KgMz"
            }
        },
        {
            "type": "ConfigChange"
        },
        {
            "type": "ImageChange",
            "imageChange": {}
        }
    ],
    "source": {
        "type": "Git",
        "git": {
            "uri": "http://services.lab.example.com/app" 3
        }
    },
    "strategy": {
        "type": "Source", 4
        "sourceStrategy": {
            "from": {
                "kind": "ImageStreamTag",
                "namespace": "openshift",
                "name": "php:5.5" 5
            }
        }
    },
    "output": {
        "to": {
            "kind": "ImageStreamTag",
            "name": "myapp:latest" 6
        }
    },
    "resources": {},
    "postCommit": {},
    "nodeSelector": null
},
"status": {
    "lastVersion": 0
}
},
...

```

- 1** Define a resource type of **BuildConfig**.
- 2** Name the **BuildConfig** as **myapp**.
- 3** Define the address to the source code Git repository.
- 4** Define the strategy to use S2I.
- 5** Define the builder image as the **php:5.5** image stream.
- 6** Name the output image stream as **myapp:latest**.

The third resource is the deployment configuration that is responsible for customizing the deployment process in OpenShift. It may include parameters and triggers that are necessary to create new container instances, and are translated into a replication controller from Kubernetes. Some of the features provided by DeploymentConfigs are:

- User customizable strategies to transition from the existing deployments to new deployments.
- Rollbacks to a previous deployment.
- Manual replication scaling.

```

...
{
    "kind": "DeploymentConfig", ❶
    "apiVersion": "v1",
    "metadata": {
        "name": "myapp", ❷
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "strategy": {
            "resources": {}
        },
        "triggers": [
            {
                "type": "ConfigChange" ❸
            },
            {
                "type": "ImageChange", ❹
                "imageChangeParams": {
                    "automatic": true,
                    "containerNames": [
                        "myapp"
                    ],
                    "from": {
                        "kind": "ImageStreamTag",
                        "name": "myapp:latest"
                    }
                }
            }
        ],
        "replicas": 1,
        "test": false,
        "selector": {
            "app": "myapp",
            "deploymentconfig": "myapp"
        },
        "template": {
            "metadata": {

```

```

        "creationTimestamp": null,
        "labels": {
            "app": "myapp",
            "deploymentconfig": "myapp"
        },
        "annotations": {
            "openshift.io/container.myapp.image.entrypoint":
            "[\"container-entrypoint\", \"/bin/sh\", \"-c\", \"$STI_SCRIPTS_PATH/usage\"]",
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "containers": [
            {
                "name": "myapp",
                "image": "myapp:latest", ⑤
                "ports": [ ⑥
                    {
                        "containerPort": 8080,
                        "protocol": "TCP"
                    }
                ],
                "resources": {}
            }
        ]
    },
    "status": {}
},
...

```

- ①** Define a resource type of **DeploymentConfig**.
- ②** Name the **DeploymentConfig** as **myapp**.
- ③** A configuration change trigger causes a new deployment to be created any time the replication controller template changes.
- ④** An image change trigger causes a new deployment to be created each time a new version of the **myapp:latest** image is available in the repository.
- ⑤** Define that the **library/myapp:latest** container image should be deployed.
- ⑥** Specifies the container ports.

The last item is the service, already covered in previous chapters:

```

...
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "myapp",
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {

```

```
        "openshift.io/generated-by": "OpenShiftNewApp"
    }
},
"spec": {
    "ports": [
        {
            "name": "8080-tcp",
            "protocol": "TCP",
            "port": 8080,
            "targetPort": 8080
        }
    ],
    "selector": {
        "app": "myapp",
        "deploymentconfig": "myapp"
    }
},
"status": {
    "loadBalancer": {}
}
}
]
```

**NOTE**

By default, no route is created by the **oc new-app** command. A route can be created after the application creation. However, a route is automatically created when using the web console because it uses a template.

After creating a new application, the build process starts. See a list of application builds with **oc get builds**:

```
$ oc get builds
NAME      TYPE      FROM      STATUS      STARTED      DURATION
myapp-1   Source    Git@59d3066  Complete   3 days ago  2m13s
```

OpenShift allows viewing of the build logs. The following command shows the last few lines of the build log:

```
$ oc logs build/myapp-1
```

**NOTE**

If the build is not **Running** yet, or the **s2i-build** pod has not been deployed yet, the above command throws an error. Just wait a few moments and retry it.

Trigger a new builder with the **oc start-build build_config_name** command:

```
$ oc get buildconfig
NAME      TYPE      FROM      LATEST
myapp    Source    Git       1
```

```
$ oc start-build myapp  
build "myapp-2" started
```

RELATIONSHIP BETWEEN BUILDCONFIG AND DEPLOYMENTCONFIG

The BuildConfig pod is responsible for creating the images in OpenShift and pushing them to the internal Docker registry. Any source code or content update normally requires a new build to guarantee the image is updated.

The DeploymentConfig pod is responsible for deploying pods into OpenShift. The outcome from a DeploymentConfig pod execution is the creation of pods with the images deployed in the internal docker registry. Any existing running pod may be destroyed, depending on how the DeploymentConfig is set.

The BuildConfig and DeploymentConfig resources do not interact directly. The BuildConfig creates or updates a container image. The DeploymentConfig reacts to this new image or updated image event and creates pods from the container image.



REFERENCES

Additional information about S2I builds is available in the *Core Concepts* section of the OpenShift Container Platform documentation:

Architecture

<https://access.redhat.com/documentation/en-us/>

Additional information about the S2I build process is available in the OpenShift Container Platform documentation:

Developer Guide

<https://access.redhat.com/documentation/en-us/>

► GUIDED EXERCISE

CREATING A CONTAINERIZED APPLICATION WITH SOURCE-TO-IMAGE

In this exercise, you will explore a Source-to-Image container, build an application from source code, and deploy the application on an OpenShift cluster.

OUTCOMES

You should be able to:

- Describe the layout of a Source-to-Image container and the scripts used to build and run an application within the container.
- Build an application from source code using the OpenShift command line interface.
- Verify the successful deployment of the application using the OpenShift command line interface.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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 s2i setup
[student@workstation ~]$ cd ~/D0285/labs/s2i
```

- 1. Examine the source code for the PHP version 5.6 Source-to-Image container.
- 1.1. Use the **tree** command to review the files that make up the container image:

```
[student@workstation s2i]$ tree s2i-php-container
s2i-php-container/
├── 5.6
│   ├── cccp.yml
│   ├── contrib
│   │   └── etc
│   │       └── conf.d
│   │           ├── 00-documentroot.conf.template
│   │           └── 50-mpm-tuning.conf.template
```

```

    └── httpdconf.sed
    └── php.d
        └── 10-opcache.ini.template
    └── php.ini.template
    └── scl_enable
    └── Dockerfile
    └── Dockerfile.rhel7
    └── README.md
    └── s2i
        └── bin
            └── assemble
            └── run
            └── usage
        └── test
            └── run
        └── test-app
            └── composer.json
            └── index.php
    └── hack
        └── build.sh
    └── common.mk
    └── LICENSE
    └── Makefile
    └── README.md

```

- 1.2. Review the **s2i-php-container/5.6/s2i/bin/assemble** script. Note how it moves the PHP source code from the `/tmp/src/` directory to the container working directory near the top of the script. The OpenShift Source-to-Image process executes the `git clone` command on the Git repository that is provided when the application is built using the `oc new-app` command or the web console. The remainder of the script supports retrieving PHP packages that your application names as requirements, if any.
- 1.3. Review the **s2i-php-container/5.6/s2i/bin/run** script. This script is executed as the command (**CMD**) for the PHP container built by the Source-to-Image process. It is responsible for setting up and running the Apache HTTP service which executes the PHP code in response to HTTP requests.
- 1.4. Review the **s2i-php-container/5.6/Dockerfile.rhel7** file. This **Dockerfile** builds the base PHP Source-to-Image container. It installs PHP and Apache HTTP Server from the Red Hat Software Collections Library, copies the Source-to-Image scripts you examined in earlier steps to their expected location, and modifies files and file permissions as needed to run on an OpenShift cluster.

▶ 2. Login to OpenShift:

```
[student@workstation s2i]$ oc login -u developer -p redhat \
https://master.lab.example.com
Login successful.
```

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

```
oc new-project <projectname>
```

- 3. Create a new project named **s2i**:

```
[student@workstation s2i]$ oc new-project s2i
Now using project "s2i" on server "https://master.lab.example.com:443".
...
```

Your current project may differ. You may also be asked to login. If so, the password is **redhat**.

- 4. Create a new PHP application using Source-to-Image from the Git repository at <http://services.lab.example.com/php-helloworld>

- 4.1. Use the **oc new-app** command to create the PHP application.



IMPORTANT

The following example uses backslash (\) to indicate that the second line is a continuation of the first line. If you wish to ignore the backslash, you can type the entire command in one line.

```
[student@workstation s2i]$ oc new-app \
php:5.6~http://services.lab.example.com/php-helloworld
```

- 4.2. Wait for the build to complete and the application to deploy. Review the resources that were built for you by the **oc new-app** command. Examine the **BuildConfig** resource created using **oc describe**:

```
[student@workstation s2i]$ oc get pods -w
...
Ctrl+C
[student@workstation s2i]$ oc describe bc/php-helloworld
Name:  php-helloworld
Namespace: s2i
Created: 2 minutes ago
Labels:  app=php-helloworld
Annotations: openshift.io/generated-by=OpenShiftNewApp
Latest Version: 1

Strategy: Source
URL:  http://services.lab.example.com/php-helloworld
From Image: ImageStreamTag openshift/php:5.6
Output to: ImageStreamTag php-helloworld:latest
...

Build Run Policy: Serial
Triggered by: Config, ImageChange
Webhook GitHub:
  URL: https://master.lab.example.com:443/apis/build.openshift.io/v1/namespaces/
s2i/buildconfigs/php-helloworld/webhooks/<secret>/github
Webhook Generic:
  URL: https://master.lab.example.com:443/apis/build.openshift.io/v1/namespaces/
s2i/buildconfigs/php-helloworld/webhooks/<secret>/generic
  AllowEnv: false
```

```
Build Status Duration Creation Time
php-helloworld-1 complete 1m46s 2018-06-15 10:34:10 +0000 UTC
```

The last part of the display gives the build history for this application. So far, there has been only one build and that build completed successfully.

- 4.3. Examine the build log for this build. Use the build name given in the listing above and add **-build** to the name to create the pod name for this build, **php-helloworld-1-build**.

```
[student@workstation s2i]$ oc logs php-helloworld-1-build
--> Installing application source...
=> sourcing 20-copy-config.sh ...
--> 10:35:12      Processing additional arbitrary httpd configuration provided by
s2i ...
=> sourcing 00-documentroot.conf ...
=> sourcing 50-mpm-tuning.conf ...
=> sourcing 40-ssl-certs.sh ...

Pushing image docker-registry.default.svc:5000/s2i/php-helloworld:latest ...
Pushed 0/6 layers, 1% complete
Pushed 1/6 layers, 20% complete
Pushed 2/6 layers, 40% complete
Pushed 3/6 layers, 59% complete
Pushed 4/6 layers, 81% complete
Pushed 5/6 layers, 99% complete
Pushed 6/6 layers, 100% complete
Push successful
```

Notice the clone of the Git repository as the first step of the build. Next, the Source-to-Image process built a new container called **s2i/php-helloworld:latest**. The last step in the build process is to push this container to the OpenShift private registry.

- 4.4. Review the **DeploymentConfig** for this application:

```
[student@workstation s2i]$ oc describe dc/php-helloworld
Name:    php-helloworld
Namespace:        s2i
Created:          12 minutes ago
Labels:           app=php-helloworld
Annotations:      openshift.io/generated-by=OpenShiftNewApp
Latest Version:   1
Selector:         app=php-helloworld,deploymentconfig=php-helloworld
Replicas:         1
Triggers:         Config, Image(phi-helloworld@latest, auto=true)
Strategy:         Rolling
Template:
  Labels:          app=php-helloworld
                    deploymentconfig=php-helloworld
...
  Containers:
    php-helloworld:
      Image:  docker-registry.default.svc:5000/s2i/php-
helloworld@sha256:6d274e9d6e3d4ba11e5dcb3fc25e1326c8170b1562c2e3330595ed21c7dfb983
```

```
Ports: 8443/TCP, 8080/TCP
Environment: <none>
Mounts: <none>
Volumes: <none>

Deployment #1 (latest):
Name: php-helloworld-1
Created: 5 minutes ago
Status: Complete
Replicas: 1 current / 1 desired
Selector: app=php-helloworld,deployment=php-helloworld-1,deploymentconfig=php-helloworld
Labels: app=php-helloworld,openshift.io/deployment-config.name=php-helloworld
Pods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 Failed
...
```

- 4.5. Review the **service** for this application:

```
[student@workstation s2i]$ oc describe svc/php-helloworld
Name:          php-helloworld
Namespace:     s2i
Labels:        app=php-helloworld
Annotations:   openshift.io/generated-by=OpenShiftNewApp
Selector:      app=php-helloworld,deploymentconfig=php-helloworld
Type:          ClusterIP
IP:            172.30.18.13
Port:          8080-tcp  8080/TCP
TargetPort:    8080/TCP
Endpoints:    10.129.0.26:8080
Port:          8443-tcp  8443/TCP
TargetPort:    8443/TCP
Endpoints:    10.129.0.26:8443
Session Affinity: None
Events:        <none>
```

- 4.6. Test the application by sending it an HTTP GET request (replace this IP address with the one shown in your service listing):

```
[student@workstation s2i]$ ssh student@master curl -s 172.30.18.13:8080
Hello, World! php version is 5.6.25
```

- 5. Explore starting application builds by changing the application in its Git repository and executing the proper commands to start a new Source-to-Image build.

- 5.1. Clone the project locally using **git**:

```
[student@workstation s2i]$ git clone \
http://services.lab.example.com/php-helloworld
Cloning into 'php-helloworld'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (9/9), done.
```

```
[student@workstation s2i]$ cd php-helloworld
```

- 5.2. Edit the **index.php** file and make the contents look like this:

```
<?php  
print "Hello, World! php version is " . PHP_VERSION . "\n";  
print "A change is a coming!\n";  
?>
```

Save the file.

- 5.3. Commit the changes and push the code back to the remote Git repository:

```
[student@workstation php-helloworld]$ git add .  
[student@workstation php-helloworld]$ git commit -m \  
'Changed index page contents.'  
[master ff807f3] Changed index page contents.  
Committer: Student User <student@workstation.lab.example.com>  
Your name and email address were configured automatically based  
on your username and hostname. Please check that they are accurate.  
You can suppress this message by setting them explicitly:  
  
git config --global user.name "Your Name"  
git config --global user.email you@example.com
```

After doing this, you may fix the identity used for this commit with:

```
git commit --amend --reset-author  
  
1 file changed, 1 insertion(+)  
[student@workstation php-helloworld]$ git push origin master  
...  
Counting objects: 5, done.  
Delta compression using up to 2 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 287 bytes | 0 bytes/s, done.  
Total 3 (delta 1), reused 0 (delta 0)  
To http://services.lab.example.com/php-helloworld  
  ecb93d1..eb438ba  master -> master  
[student@workstation php-helloworld]$ cd ..
```

- 5.4. Start a new Source-to-Image build process and wait for it to build and deploy:

```
[student@workstation s2i]$ oc start-build php-helloworld  
build "php-helloworld-2" started  
[student@workstation s2i]$ oc get pods -w  
...  
NAME          READY   STATUS    RESTARTS   AGE  
php-helloworld-1-build  0/1     Completed   0          33m  
php-helloworld-2-2n70q  1/1     Running    0          1m  
php-helloworld-2-build  0/1     Completed   0          1m
```

^C

- 5.5. Test that your changes are served by the application:

```
[student@workstation s2i]$ ssh student@master curl -s 172.30.18.13:8080
Hello, World! php version is 5.6.25
A change is a coming!
```

- 6. Grade the lab:

```
[student@workstation s2i]$ lab s2i grade
Accessing PHP web application..... SUCCESS
```

- 7. Clean up the lab by deleting the OpenShift project, which in turn deletes all the Kubernetes and OpenShift resources:

```
[student@workstation s2i]$ oc delete project s2i
project "s2i" deleted
```

This concludes this guided exercise.

CREATING ROUTES

OBJECTIVES

After completing this section, students should be able to create a route to a service.

WORKING WITH ROUTES

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

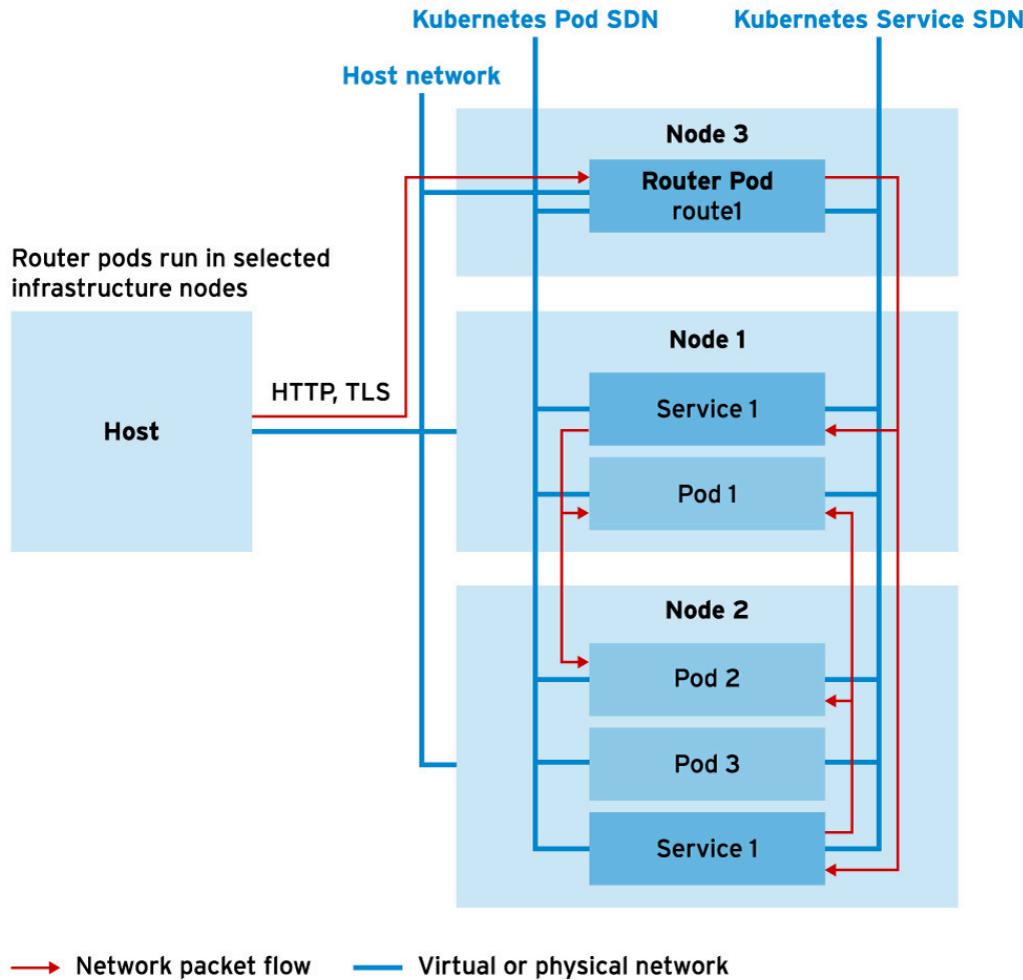


Figure 9.3: OpenShift routes and Kubernetes services

A route connects a public-facing IP address and DNS host name to an internal-facing service IP. At least, this is the concept. In practice, to improve performance and reduce latency, the OpenShift router connects directly to the pods using the internal pod software-defined network (SDN). Use the service to find the end points, that is, the pods exposed by the service.

OpenShift routes are implemented by a shared router service, which runs as pods 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 the OpenShift administrator 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 SDN.

The following listings shows a minimal route defined using JSON syntax:

```
{  
    "apiVersion": "v1",  
    "kind": "Route",  
    "metadata": {  
        "name": "quoteapp"  
    },  
    "spec": {  
        "host": "quoteapp.apps.example.com",  
        "to": {  
            "kind": "Service",  
            "name": "quoteapp"  
        }  
    }  
}
```

Starting the route resource, there are the standard, `apiVersion`, `kind`, and `metadata` attributes. The `Route` value for `kind` shows that this is a resource attribute, and the `metadata.name` attribute gives this particular route, the identifier, `quoteapp`.

As with pods and services, the interesting part is the `spec` attribute, which is an object containing the following attributes:

- `host` is a string containing the FQDN name associated with the route. It has to be preconfigured to resolve to the OpenShift router IP address.
- `to` is an object stating the `kind` of resource this route points to, which in this case is an OpenShift Service with the `name` set to `quoteapp`.



NOTE

Remember 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 uses selectors to link to pod resources containing specific labels, a route links directly to the service resource name.

CREATING ROUTES

Route resources can 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. After all, `oc new-app` does not know if the pod is intended to be accessible from outside the OpenShift instance or not. When `oc new-app` 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.

Another 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. For example:

```
$ oc expose service quotedb --name quote
```

Routes created from templates or from `oc expose` generate DNS names of the 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 prerequisite for installing OpenShift.

For example, creating route `quote` in project `test` from an OpenShift instance where the wildcard domain is `cloudapps.example.com` results in the FQDN `quote-test.cloudapps.example.com`.



NOTE

The DNS server that hosts the wildcard domain knows nothing about route host names. It simply resolves any name to the configured IP addresses. 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, will be blocked by the OpenShift router and result in an HTTP 404 error.

Finding the Default Domain

The subdomain or, default domain, is defined in the OpenShift configuration file `master-config.yaml` in the `routingConfig` section with the keyword `subdomain`. For example:

```
routingConfig:  
  subdomain: 172.25.250.254.xip.io
```



REFERENCES

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

Architecture

<https://access.redhat.com/documentation/en-us/>

Additional developer information about routes is available in the `Routes` section of the OpenShift Container Platform documentation:

Developer Guide

<https://access.redhat.com/documentation/en-us/>

► GUIDED EXERCISE

EXPOSING A SERVICE AS A ROUTE

In this exercise, you will create, build, and deploy an application on an OpenShift cluster and expose its service as a route.

OUTCOMES

You should be able to expose a service as a route for a deployed OpenShift application.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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 route setup
```

- 1. On **workstation** open a terminal and login to OpenShift.

```
[student@workstation ~]$ oc login -u developer https://master.lab.example.com
Logged into "https://master.lab.example.com:443" as "developer" using existing
credentials.
...
```

You may have to enter credentials for the **developer** account. The password is **redhat**. The current project in your command output may differ from the listing above.

- 2. Create a new project named **route**:

```
[student@workstation ~]$ oc new-project route
Now using project "route" on server "https://master.lab.example.com:443".
```

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.

- 3. Create a new PHP application using Source-to-Image from the Git repository at `http://services.lab.example.com/php-helloworld`
- 3.1. Use the `oc new-app` command to create the PHP application.

**IMPORTANT**

The following example uses backslash (\) to indicate that the second line is a continuation of the first line. If you wish to ignore the backslash, you can type the entire command in one line.

```
[student@workstation ~]$ oc new-app \
php:5.6~http://services.lab.example.com/php-helloworld
```

- 3.2. Wait until the application finishes building and deploying by monitoring the progress with the `oc get pods -w` command:

```
[student@workstation ~]$ oc get pods -w
...
^C
```

- 3.3. Review the `service` for this application using `oc describe`:

```
[student@workstation ~]$ oc describe svc/php-helloworld
Name:                  php-helloworld
Namespace:             route
Labels:                app=php-helloworld
Annotations:           openshift.io/generated-by=OpenShiftNewApp
Selector:              app=php-helloworld,deploymentconfig=php-helloworld
Type:                 ClusterIP
IP:                   172.30.200.65
Port:                 8080-tcp  8080/TCP
TargetPort:            8080/TCP
Endpoints:            10.129.0.31:8080
Port:                 8443-tcp  8443/TCP
TargetPort:            8443/TCP
Endpoints:            10.129.0.31:8443
Session Affinity:     None
Events:               <none>
```

The IP address displayed in the output of the command may differ.

- 4. Expose the service creating a route with a default name and fully qualified domain name (FQDN):

```
[student@workstation ~]$ oc expose svc/php-helloworld
route "php-helloworld" exposed
[student@workstation ~]$ oc get route
NAME          HOST/PORT                               PATH      SERVICES
PORT
php-helloworld   php-helloworld-route.apps.lab.example.com      php-
helloworld   8080-tcp
[student@workstation ~]$ curl php-helloworld-route.apps.lab.example.com
```

```
Hello, World! php version is 5.6.25
```

Notice the FQDN is comprised of the application name and project name by default. The remainder of the FQDN, the subdomain, is defined when OpenShift is installed. Use **curl** command to verify that the application can be accessed with its route address.

- 5. Replace this route with a route named **xyz**.

5.1. Delete the current route:

```
[student@workstation ~]$ oc delete route/php-helloworld
route "php-helloworld" deleted
```

5.2. Expose the service creating a route named **xyz**:

```
[student@workstation ~]$ oc expose svc/php-helloworld --name=xyz
route "xyz" exposed
[student@workstation ~]$ oc get route
NAME          HOST/PORT           PATH      SERVICES
PORT          TERMINATION
xyz           xyz-route.apps.lab.example.com   php-helloworld
8080-tcp
```

Note the new FQDN that was generated.

5.3. Make an HTTP request using the FQDN on port 80:

```
[student@workstation ~]$ curl xyz-route.apps.lab.example.com
Hello, World! php version is 5.6.25
A change is a coming!
```



NOTE

The output of the PHP application will be different if you did not complete the previous exercise in this chapter.

- 6. Grade your lab progress:

```
[student@workstation ~]$ lab route grade
Accessing PHP web application..... SUCCESS
```

- 7. Clean up the lab environment by deleting the project:

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

This concludes the guided exercise.

CREATING APPLICATIONS WITH THE OPENSHIFT WEB CONSOLE

OBJECTIVES

After completing this section, students should be able to:

- Create an application with the OpenShift web console.
- Examine resources for an application.
- Manage and monitor the build cycle of an application.

ACCESSING THE OPENSHIFT WEB CONSOLE

The OpenShift web console allows a user to execute many of the same tasks as the OpenShift command line. Projects can be created, applications can be created within those projects, and application resources can be examined and manipulated as needed. The OpenShift web console runs as a pod on the master.

Accessing the Web Console

The web console runs in a web browser. The URL is of the format `https://{{hostname of OpenShift master}}/console`. By default, OpenShift generates a self-signed certificate for the web console. The user must trust this certificate in order to gain access. The console requires authentication.

Managing Projects

Upon successful login, the user may select, edit, delete, and create projects on the home page. Once a project is selected, the user is taken to the Overview page which shows all of the applications created within that project space.

Application Overview Page

The application overview page is the heart of the web console.

The screenshot shows the OpenShift Application Overview page for the 'php-helloworld' application. At the top, it displays the application name 'php-helloworld' and its route 'http://php-helloworld-console.apps.lab.example.com'. Below this, under 'DEPLOYMENT CONFIG', it shows 'php-helloworld, #1'. In the 'CONTAINERS' section, there is one pod listed. The 'Build and Deployment' link is highlighted with a red box. In the 'NETWORKING' section, both the internal 'Service' and external 'Routes' are shown, with their respective URLs highlighted with red boxes. To the right, a 'Scale tool' with up and down arrows is also highlighted.

Figure 9.4: Application overview page

From this page, the user can view the route, build, service, and deployment information. The scale tool (arrows) can be used to increase and decrease the number of replicas of the application that are running in the cluster. All of the hyperlinks lead to detailed information about that particular application resource including the ability to manipulate that resource. For example, clicking on the link for the build allows the user to start a new build.

CREATING NEW APPLICATIONS

The user can select the Add to Project link to create a new application. The user can create an application using a template (Source-to-Image), deploy an existing image, and define a new application by importing YAML or JSON formatted resources. Once an application has been created with one of these three methods, it can be managed on the overview page.

OTHER WEB CONSOLE FEATURES

The web console allows the user to:

- Manage resources such as project quotas, user membership, secrets, and other advanced resources.
- Create persistent volume claims.
- Monitor builds, deployments, pods, and system events.
- Create continuous integration and deployment pipelines with Jenkins.

► GUIDED EXERCISE

CREATING AN APPLICATION WITH THE WEB CONSOLE

In this exercise, you will create, build, and deploy an application on an OpenShift cluster using the OpenShift Web Console.

OUTCOMES

You should be able to create, build, and deploy an application on an OpenShift cluster using the web console.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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 console setup
```

- 1. Go to the `https://master.lab.example.com` address to access the OpenShift Web Console via a browser. Log in and create a new project.
 - 1.1. Enter the web console URL in the browser and trust the self-signed certificate generated by OpenShift.
 - 1.2. Login with **developer** as the user name and **redhat** as the password.

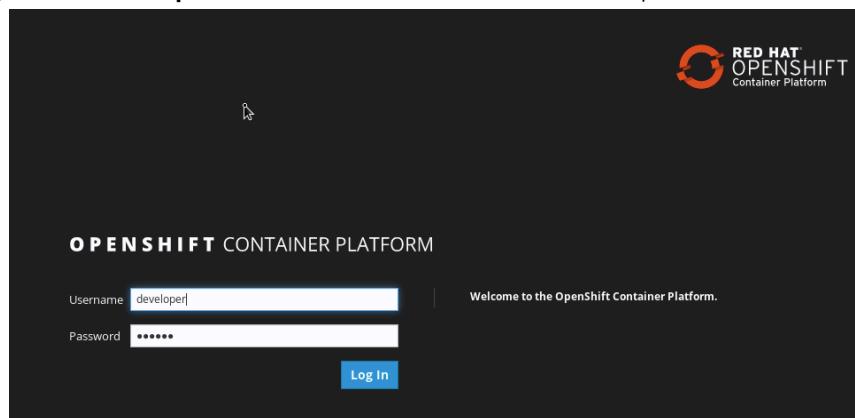


Figure 9.5: Web console login

- 1.3. Create a new project named **console**. You may type any values you wish in the other fields.

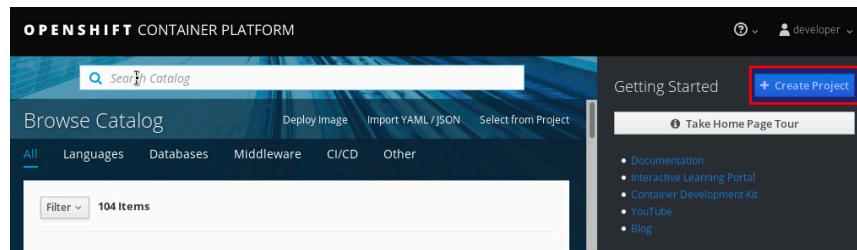


Figure 9.6: Create a new project - step 1

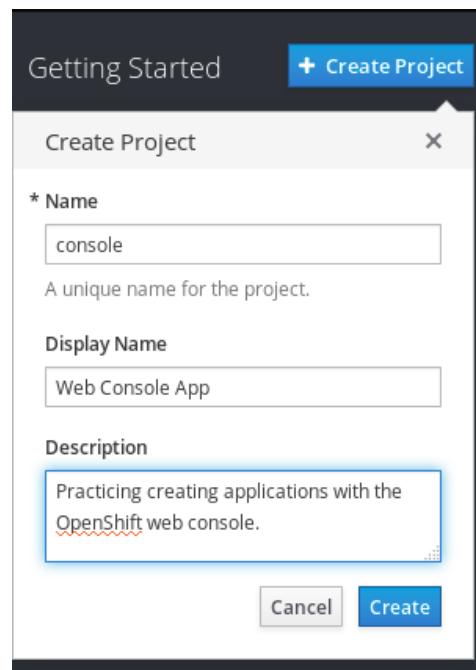


Figure 9.7: Create a new project - step 2

- 1.4. After filling in the details of the appropriate fields, click on the **Create** button of the **Create Project** dialog box.

► 2. Create the new **php-helloworld** application with a PHP template.

2.1. Select the PHP template from the catalog.

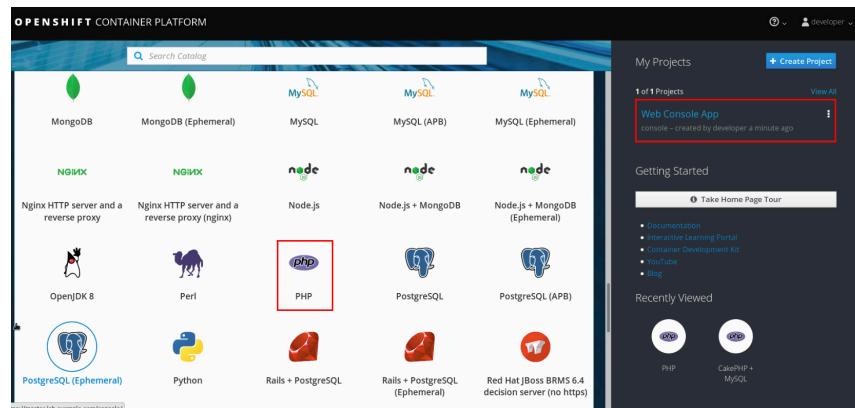


Figure 9.8: Select the PHP template

- 2.2. A PHP dialog box appears. Click on the **Next >** button. Then, select PHP version 7.0 from the drop-down menu Version. Select the console project in the drop-down menu Add to Project

Enter **php-helloworld** as the name for the application and the location of the source code git repository: <http://services.lab.example.com/php-helloworld>

Click on the **Create** button of the **PHP** dialog box.

The screenshot shows the 'PHP' dialog box with three tabs: Information (1), Configuration (2), and Results (3). The 'Information' tab is active. It contains fields for 'Add to Project' (set to 'Web Console App'), 'Version' (set to '7.0'), 'Application Name' (set to 'php-helloworld'), and 'Git Repository' (set to 'http://services.lab.example.com/php-helloworld'). A note at the bottom says 'If you have a private Git repository or need to change application defaults, view advanced options.' At the bottom right, there are 'Cancel', '< Back', and a blue 'Create' button, which is highlighted with a red box.

Figure 9.9: PHP application details

- 2.3. On the confirmation page, click on the Continue to the project overview link.

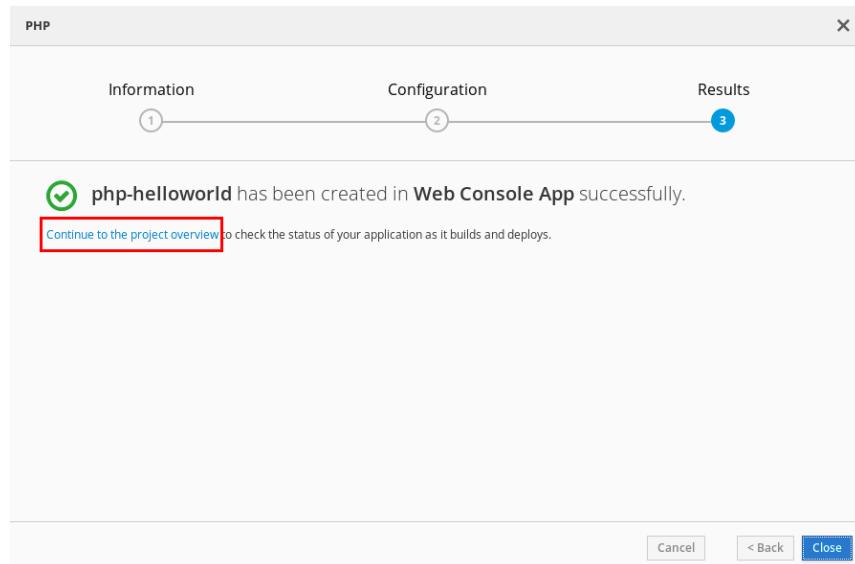


Figure 9.10: Application created confirmation page

- 3. Explore application components from the Overview page. The build may still be running when you reach this page, so the build section may look slightly different from the image below.

Figure 9.11: Application overview page

- 3.1. Identify the components of the application and their corresponding OpenShift and Kubernetes resources.
 - Route URL
Clicking on this link opens a browser tab to view your application.
 - Build
Clicking on the relevant links show the build configuration, specific build information, and build log.
 - Service
Clicking on the relevant link shows the service configuration.
 - Deployment Configuration
Clicking on the relevant links show the deployment configuration and current deployment information.
 - Scale Tool
Clicking on the up arrow will increase the number of running pods. Clicking on the down arrow decreases the number of running pods.
- 3.2. Examine the build log. Under the BUILDS section of the **Overview** page, click on the **php-helloworld** link. Click the **View Log** link to view the build log. Return to the **Overview** page by clicking **Overview** in the left-hand menu.
- 3.3. Examine the deployment configuration. Click on the **php-helloworld** link just below the label **DEPLOYMENT CONFIG** in the **Overview** page. Examine the information and features on this page and return to the **Overview** page.

- 3.4. Examine the service configuration. Click on the **php-helloworld** link in the **NETWORKING** section of the **Overview** page. Examine the service configuration page and return to the **Overview** page.
- 3.5. Click on the route link to view the application output in a new browser tab. This is the URL that appears on the right side at the same line as the title of the application (near the top).
- 4. Modify the application code, commit the change, push the code to the remote Git repository, and trigger a new application build.

4.1. Clone the Git repository:

```
[student@workstation ~]$ git clone \
http://services.lab.example.com/php-helloworld
Cloning into 'php-helloworld'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (9/9), done.
[student@workstation ~]$ cd php-helloworld
```

- 4.2. Add the second print line statement in the **index.php** page to read "A change is in the air!" and save the file. Add the change to the Git index, commit the change, and push the changes to the remote Git repository.

```
[student@workstation php-helloworld]$ vim index.php
[student@workstation php-helloworld]$ cat index.php
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
print "A change is in the air!\n";
?>
[student@workstation php-helloworld]$ git add index.php
[student@workstation php-helloworld]$ git commit -m 'updated app'
[master d198fb5] updated app
 1 file changed, 1 insertion(+), 1 deletion(-)
[student@workstation php-helloworld]$ git push origin master
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 286 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To http://services.lab.example.com/php-helloworld
 eb438ba..d198fb5  master -> master
```

- 4.3. Trigger an application build manually from the web console.

Navigate to the build page from the **Overview** page by clicking on **php-helloworld** link in the **BUILDS** section. Click the **Start Build** button on the upper-right of the

page. Wait for the build to complete. Examine the build log on the build page or the **Overview** page to determine when the build completes.

The screenshot shows the OpenShift web console interface for the 'php-helloworld' application. At the top, it displays the application name and its URL: <http://php-helloworld-console.apps.lab.example.com>. Below this, under 'DEPLOYMENT CONFIG', the entry 'php-helloworld, #1' is shown. A context menu is open next to this entry, with the 'Start Build' option highlighted and circled in blue. Other options in the menu include 'Deploy', 'Edit', 'View Logs', and 'pod'. The 'CONTAINERS' section lists the 'php-helloworld' container with details like image, build, source, and ports. The 'NETWORKING' section shows a service named 'php-helloworld' with an external route at <http://php-helloworld-console.apps.lab.example.com>.

Figure 9.12: Start a new application build

- 4.4. Use the route link on the **Overview** page to verify that your code change was deployed.

► 5. Grade your work:

```
[student@workstation php-helloworld]$ lab console grade
Accessing PHP web application..... SUCCESS
```

- 6. Delete the project. Click the OPENSHIFT CONTAINER PLATFORM icon in the web console to go back to the list of projects. Click the menu icon, next to your project name. From the list choose **Delete Project** and enter the name of the project to confirm its deletion.

The screenshot shows the 'My Projects' page in the OpenShift Container Platform. It lists a single project named 'Web Console App'. A context menu is open over this project, with the 'Delete Project' option highlighted and circled in red. Other options in the menu include 'View Membership' and 'Edit Project'. The page also includes a search bar, a sort dropdown, and a 'Create Project' button.

Figure 9.13: Delete the project

This concludes the guided exercise.

► LAB

DEPLOYING CONTAINERIZED APPLICATIONS ON OPENSIFT

In this lab, you will create an application using the OpenShift Source-to-Image facility.

OUTCOMES

You should be able to create an OpenShift application and access it through a web browser.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup  
[student@workstation ~]$ cd /home/student/do285-ansible  
[student@workstation do285-ansible]$ ./install.sh
```

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 setup
```

1. Login as **developer** and create the project **ocp**.
If the **oc login** command prompts about using insecure connections, answer **y** (yes).
2. Create a temperature converter application written in PHP using the **php:5.6** image stream tag. The source code is in the Git repository at <http://services.lab.example.com/temp>. You may use the OpenShift command line interface or the web console to create the application. Make sure a route is created so that you can access the application from a web browser.
3. Test the application in a web browser using the route URL.
4. Grade your work:

```
[student@workstation ~]$ lab openshift grade  
Accessing PHP web application..... SUCCESS
```

5. Delete the project.

This concludes the lab.

► SOLUTION

DEPLOYING CONTAINERIZED APPLICATIONS ON OPENSIFT

In this lab, you will create an application using the OpenShift Source-to-Image facility.

OUTCOMES

You should be able to create an OpenShift application and access it through a web browser.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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 setup
```

1. Login as **developer** and create the project **ocp**.
If the **oc login** command prompts about using insecure connections, answer **y** (yes).
Issue the following commands:

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

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

```
oc new-project <projectname>
```

```
[student@workstation ~]$ oc new-project ocp
Now using project "ocp" on server "https://master.lab.example.com:443".
```

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. Create a temperature converter application written in PHP using the **php:5.6** image stream tag. The source code is in the Git repository at <http://services.lab.example.com/temps>. You may use the OpenShift command line interface or the web console to create the application. Make sure a route is created so that you can access the application from a web browser.

If using the command line interface, issue the following commands:

```
[student@workstation ~]$ oc new-app \
  php:5.6~http://services.lab.example.com/temps
[student@workstation ~]$ oc logs -f bc/temps
Cloning "http://services.lab.example.com/temps" ...
Commit: 350b6ca43ff05d1c395a658083f74a92c53fc7e9 (Establish remote repository)
Author: root <root@services.lab.example.com>
Date: Tue Jun 26 21:59:41 2018 +0000
--> Installing application source...
...output omitted...
Pushed 6/6 layers, 100% complete
Push successful
[student@workstation ~]$ oc get pods -w
NAME        READY     STATUS    RESTARTS   AGE
temps-1-build  0/1      Completed  0          5m
temps-1-h76lt  1/1      Running   0          5m
Ctrl+C
[student@workstation ~]$ oc expose svc/temps
route "temps" exposed
```

3. Test the application in a web browser using the route URL.

3.1. Discover the URL for the route.

```
[student@workstation ~]$ oc get route/temps
NAME      HOST/PORT           PATH      SERVICES    PORT
TERMINATION WILDCARD
temps     temps-ocp.apps.lab.example.com      temps      8080-tcp
None
```

3.2. Open Firefox with <http://temps-ocp.apps.lab.example.com> URL to verify that the temperature converter application works.

4. Grade your work:

```
[student@workstation ~]$ lab openshift grade
Accessing PHP web application..... SUCCESS
```

5. Delete the project.

Delete the project by running the **oc delete project** command.

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

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The OpenShift command line client **oc** is used to perform the following tasks in an OpenShift cluster:
 - Logging in and out of an OpenShift cluster.
 - Creating, changing, and deleting projects.
 - Creating applications inside a project, including creating a deployment configuration from a container image, or a build configuration from application source 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.
- The OpenShift Container Platform organizes entities in the OpenShift cluster as objects stored on the master node. These are collectively known as **resources**. The most common ones are:
 - Pod
 - Label
 - Persistent Volume (PV)
 - Persistent Volume Claim (PVC)
 - Service
 - Route
 - Replication Controller (RC)
 - Deployment Configuration (DC)
 - Build Configuration (BC)
 - Project
- 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.
- **Source-to-Image (S2I)** is a facility that makes it easy to build a container image from application source code. This facility takes an application's source code from a Git server, injects the source code into a base container based on the language and framework desired, and produces a new container image that runs the assembled application.

- A **Route** connects a public-facing IP address and DNS host name to an internal-facing service IP. While services allow for network access between pods inside an OpenShift instance, routes allow for network access to pods from users and applications outside the OpenShift instance.
- You can create, build, deploy and monitor applications using the OpenShift web console.

CHAPTER 10

DEPLOYING A MULTI-CONTAINER APPLICATION ON OPENSHIFT

GOAL

Deploy applications that are containerized using multiple container images.

OBJECTIVES

- Deploy a multi-container application on OpenShift using a template.

SECTIONS

- Deploying a Multi-Container Application on OpenShift (and Guided Exercise)

LAB

- Deploying Multi-Container Applications

DEPLOYING A MULTI-CONTAINER APPLICATION ON OPENSHIFT

OBJECTIVE

After completing this section, students should be able to deploy a multi-container application on OpenShift using a template.

EXAMINING THE SKELETON OF A TEMPLATE

OpenShift Container Platform contains a facility to deploy applications using *templates*. A template can include any OpenShift resource, assuming users have permission to create them within a project.

A template describes a set of related resource definitions to be created together, as well as a set of parameters for those objects.

For example, an application might consist of a front-end web application and a database server. Each consists of a service resource and a deployment configuration resource. They share a set of credentials (parameters) for the front end to authenticate to the back end. The template can be processed by specifying parameters or by allowing them to be automatically generated (for example, for a unique database password) in order to instantiate the list of resources in the template as a cohesive application.

The OpenShift installer creates several templates by default in the **openshift** namespace. Run the **oc get templates** command with the **-n openshift** option to list these preinstalled templates:

```
$ oc get templates -n openshift
NAME                                     DESCRIPTION
cakephp-mysql-persistent                 An example CakePHP application...
dancer-mysql-persistent                  An example Dancer application...
django-psql-persistent                   An example Django application...
jenkins-ephemeral                         Jenkins service, without persistent
                                         storage....
jenkins-persistent                        Jenkins service, with persistent storage....
jenkins-pipeline-example                 This example showcases the new Jenkins...
logging-deployer-account-template        Template for creating the deployer...
logging-deployer-template                Template for running the aggregated...
mariadb-persistent                        MariaDB database service, with persistent
                                         storage...
mongodb-persistent                       MongoDB database service, with persistent
                                         storage...
mysql-persistent                          MySQL database service, with persistent
                                         storage...
nodejs-mongo-persistent                  An example Node.js application with a
                                         MongoDB...
postgresql-persistent                   PostgreSQL database service...
rails-pgsql-persistent                  An example Rails application...
```

The following listing shows a template definition:

```
{
```

```

"kind": "Template",
"apiVersion": "v1",
"metadata": {
    "name": "mysql-persistent", ①
    "creationTimestamp": null,
    "annotations": {
        "description": "MySQL database service, with persistent storage...",
        "iconClass": "icon-mysql-database",
        "openshift.io/display-name": "MySQL (Persistent)",
        "tags": "database,mysql" ②
    }
},
"message": "The following service(s) have been created ...",
"objects": [
{
    "apiVersion": "v1",
    "kind": "Service",
    "metadata": {
        "name": "${DATABASE_SERVICE_NAME}" ③
    },
    ...
    Service attributes omitted ...
},
{
    "apiVersion": "v1",
    "kind": "PersistentVolumeClaim",
    "metadata": {
        "name": "${DATABASE_SERVICE_NAME}"
    },
    ...
    PVC attributes omitted ...
},
{
    "apiVersion": "v1",
    "kind": "DeploymentConfig",
    "metadata": {
        "name": "${DATABASE_SERVICE_NAME}"
    },
    "spec": {
        "replicas": 1,
        "selector": {
            "name": "${DATABASE_SERVICE_NAME}"
        },
        "strategy": {
            "type": "Recreate"
        },
        "template": {
            "metadata": {
                "labels": {
                    "name": "${DATABASE_SERVICE_NAME}"
                }
            },
            ...
            Other pod and Deployment attributes omitted
        }
    }
},
"parameters": [
]
]
}

```

```

...
{
    "name": "DATABASE_SERVICE_NAME",
    "displayName": "Database Service Name",
    "description": "The name of the OpenShift Service exposed for the
database.",
    "value": "mysql", ❸
    "required": true
},
{
    "name": "MYSQL_USER",
    "displayName": "MySQL Connection Username",
    "description": "Username for MySQL user that will be used for
accessing the database.",
    "generate": "expression",
    "from": "user[A-Z0-9]{3}",
    "required": true
},
{
    "name": "MYSQL_PASSWORD",
    "displayName": "MySQL Connection Password",
    "description": "Password for the MySQL connection user.",
    "generate": "expression",
    "from": "[a-zA-Z0-9]{16}", ❹
    "required": true
},
{
    "name": "MYSQL_DATABASE",
    "displayName": "MySQL Database Name",
    "description": "Name of the MySQL database accessed.",
    "value": "sampledb",
    "required": true
},
...
}

```

- ❶ Defines the template name.
- ❷ Defines a list of arbitrary tags that this template will have in the UI.
- ❸ Defines an entry for using the value assigned by the parameter **DATABASE_SERVICE_NAME**.
- ❹ Defines the default value for the parameter **DATABASE_SERVICE_NAME**.
- ❺ Defines an expression used to generate a random password if one is not specified.

It is also possible to upload new templates from a file to the OpenShift cluster so that other developers can build applications from the template. This can be done using the **oc create** command, as shown in the following example:

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.json
template "todonodejs-persistent" created
```

By default, the template is created under the current project, unless you specify a different one using the **-n** option, as shown in the following example:

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.json -n openshift
```



IMPORTANT

Any template created under the **openshift** namespace (OpenShift project) is available in the web console under the dialog box accessible via Add to Project → Select from Project menu item. Moreover, any template that gets created under the current project is accessible from that project.

Parameters

Templates define a set of *parameters*, which are assigned values. OpenShift resources defined in the template can get their configuration values by referencing *named parameters*. Parameters in a template can have default values, but they are optional. Any default values can be replaced when processing the template.

Each parameter value can be set either explicitly by using the **oc process** command, or generated by OpenShift according to the parameter configuration.

There are two ways to list the available parameters from a template. The first one is using the **oc describe** command:

```
$ oc describe template mysql-persistent -n openshift
Name: mysql-persistent
Namespace: openshift
Created: 2 weeks ago
Labels: <none>
Description: MySQL database service, with persistent storage. For more
information ...
Annotations: iconClass=icon-mysql-database
            openshift.io/display-name=MySQL (Persistent)
            tags=database,mysql

Parameters:
  Name: MEMORY_LIMIT
  Display Name: Memory Limit
  Description: Maximum amount of memory the container can use.
  Required: true
  Value: 512Mi
  Name: NAMESPACE
  Display Name: Namespace
  Description: The OpenShift Namespace where the ImageStream resides.
  Required: false
  Value: openshift
  ... SOME OUTPUT OMITTED ...
  Name: MYSQL_VERSION
  Display Name: Version of MySQL Image
  Description: Version of MySQL image to be used (5.5, 5.6 or latest).
  Required: true
  Value: 5.6

Object Labels: template=mysql-persistent-template
```

```
Message: The following service(s) have been created in your project:  
 ${DATABASE_SERVICE_NAME}.

Username: ${MYSQL_USER}  
 Password: ${MYSQL_PASSWORD}  
 Database Name: ${MYSQL_DATABASE}  
 Connection URL: mysql://${DATABASE_SERVICE_NAME}:3306/  
  
For more information about using this template, including OpenShift  
considerations, see https://github.com/sclorg/mysql-container/blob/master/5.6/README.md.  
  
Objects:  
 Service ${DATABASE_SERVICE_NAME}  
 ...
```

The second way is by using the **oc process** with the **--parameters** option:

```
$ oc process --parameters mysql-persistent -n openshift  
NAME DESCRIPTION  
GENERATOR VALUE  
MEMORY_LIMIT Maximum amount of memory the container can use.  
512Mi  
NAMESPACE The OpenShift Namespace where the ImageStream resides.  
openshift  
DATABASE_SERVICE_NAME The name of the OpenShift Service exposed for the  
database.  
mysql  
MYSQL_USER Username for MySQL user that will be used for accessing  
the database.  
expression user[A-Z0-9]{3}  
MYSQL_PASSWORD Password for the MySQL connection user.  
expression [a-zA-Z0-9]{16}  
MYSQL_DATABASE Name of the MySQL database accessed.  
sampledb  
VOLUME_CAPACITY Volume space available for data, e.g. 512Mi, 2Gi, 1Gi  
MYSQL_VERSION Version of MySQL image to be used (5.5, 5.6 or latest).  
5.6
```

Persistent Volume (PV) and Persistent Volume Claim (PVC)

Any data getting into a container while it is running, is lost with the deletion of the container. The reason behind this is the volatile storage space that the containers operate in. In many circumstances, a persistent storage is desired for the containers to avoid losing data. For example, often while running a database container, having the data entries of the database persist independently of the lifetime of the container is paramount. Both Docker and OpenShift Container Platform address the storage issue and provide solutions in different ways.

Docker uses its bind mount feature to define persistent storage for a container. Bind mounts may create an empty directory on the Docker host and map it to the container, or map an existing directory to the container. Bind-mounting an existing directory requires **svirt_sandbox_file_t** SELinux context to be set on it.

OpenShift Container Platform implements *Persistent Volumes* (PVs) and *Persistent Volume Claims* (PVCs) to offer persistent storage to pod containers. A *Persistent Volume* is a cluster-wide resource in an OpenShift cluster, usually backed by an external storage array. It supports various storage back-ends such as NFS, iSCSI, Red Hat Gluster Storage, Red Hat Ceph Storage through the use of plug-ins. As persistent volumes are cluster-wide objects, only users with *cluster-admin* role can

manage these. A *Persistent Volume Claim* is a user's request from the scope of an OpenShift project or Kubernetes namespace to use persistent volumes. It binds a persistent volume to a pod.

PROCESSING A TEMPLATE USING THE CLI

A template should be processed to generate a list of resources to create a new application. The **oc process** command is responsible for processing a template:

```
$ oc process -f filename
```

The previous command processes a template from a JSON or YAML resource definition file and returns the list of resources to standard output.

Another option is to process a template from the current project or the **openshift** project:

```
$ oc process uploaded-template-name
```



NOTE

The **oc process** command returns the list of resources to standard output. This output can be redirected to a file:

```
$ oc process -f filename -o json > myapp.json
```

Templates can generate different values based on the parameters. To override a parameter, use the **-p** option followed by a **<name>=<value>** pair.

```
$ oc process -f mysql.json \
-p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
-p VOLUME_CAPACITY=10Gi > mysqlProcessed.json
```

To create the application, use the generated JSON resource definition file:

```
$ oc create -f mysqlProcessed.json
```

Alternatively, it is possible to process the template and create the application without saving a resource definition file by using a UNIX pipe:

```
$ oc process -f mysql.json \
-p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank -p \
VOLUME_CAPACITY=10Gi \
| oc create -f -
```

It is not possible to process a template from the **openshift** project as a regular user using the **oc process** command.

```
$ oc process mysql-persistent -n openshift \
-p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
-p VOLUME_CAPACITY=10Gi | oc create -f -
```

The previous command returns an error:

```
error processing the template "mysql-persistent": User "regularUser" cannot create
processed templates in project "openshift"
```

One way to solve this problem is to export the template to a file and then process the template using that file:

```
$ oc -o json export template mysql-persistent \
-n openshift > mysql-persistent-template.json
```

```
$ oc process -f mysql-persistent-template.json \
-p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
-p VOLUME_CAPACITY=10Gi | oc create -f -
```

Another way to solve this problem is to use two slashes (>//) to provide the namespace as part of the template name:

```
$ oc process openshift//mysql-persistent-template \
-p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
-p VOLUME_CAPACITY=10Gi | oc create -f -
```

Alternatively, it is possible to create an application using the **oc new-app** command passing the template name as the **--template** option argument:

```
$ oc new-app --template=mysql-persistent -n openshift \
-p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
-p VOLUME_CAPACITY=10Gi
```



REFERENCES

Further information about templates can be found in the *Templates* section of the OpenShift Container Platform documentation:

Architecture

https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/architecture/

Developer information about templates can be found in the *Templates* section of the OpenShift Container Platform documentation:

Developer Guide

https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/developer_guide/

► GUIDED EXERCISE

CREATING AN APPLICATION WITH A TEMPLATE

In this exercise, you will deploy the To Do List application in OpenShift Container Platform using a template to define the resources the application needs to run.

OUTCOMES

You should be able to build and deploy an application in OpenShift Container Platform using a provided JSON template.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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-template setup
```

- 1. Build the database container image and publish it to the private registry.

- 1.1. Build the MySQL database image.

Examine the **/home/student/do285/labs/openshift-template/images/mysql/build.sh** script to see how the image is built. To build the base image, run the **build.sh** script.

```
[student@workstation ~]$ cd ~/do285/labs/openshift-template/images/mysql
[student@workstation mysql]$ ./build.sh
```

- 1.2. Push the image to the private registry running in the **services** VM.

To make the image available as a template for OpenShift, tag it and push it up to the private registry. To do so, run the following commands in the terminal window.

```
[student@workstation mysql]$ docker tag \
do285/mysql-57-rhel7 \
registry.lab.example.com/do285/mysql-57-rhel7
[student@workstation mysql]$ docker push \
registry.lab.example.com/do285/mysql-57-rhel7
```

```
The push refers to a repository [registry.lab.example.com/do285/mysql-57-rhel7]
b3838c109ba6: Pushed
a72cf1d969d: Mounted from rhscl/mysql-57-rhel7
9ca8c628d8e7: Mounted from rhscl/mysql-57-rhel7
827264d42df6: Mounted from rhscl/mysql-57-rhel7
latest: digest:
sha256:170e2546270690fded13f3ced0d575a90cef58028abcef8d37bd62a166ba436b size:
1156
```

**NOTE**

The output for the layers might differ.

- 2. Build the parent image for the To Do List application using the Node.js Dockerfile.

To build the base image, run the **build.sh** script located at **~/D0285/labs/openshift-template/images/nodejs**.

```
[student@workstation mysql]$ cd ~/D0285/labs/openshift-template/images/nodejs
[student@workstation nodejs]$ ./build.sh
```

- 3. Build the To Do List application child image using the Node.js Dockerfile.

- 3.1. Go to the **~/D0285/labs/openshift-template/deploy/nodejs** directory and run the **build.sh** command to build the child image.

```
[student@workstation nodejs]$ cd ~/D0285/labs/openshift-template/deploy/nodejs
[student@workstation nodejs]$ ./build.sh
```

- 3.2. Push the image to the private registry.

In order to make the image available for OpenShift to use in the template, tag it and push it to the private registry. To do so, run the following commands in the terminal window.

```
[student@workstation nodejs]$ docker tag do285/todonodejs \
registry.lab.example.com/do285/todonodejs
[student@workstation nodejs]$ docker push \
registry.lab.example.com/do285/todonodejs
The push refers to a repository [registry.lab.example.com/do285/todonodejs]
c024668a8c8e: Pushed
b26e8fcbb95bc: Pushed
dbca12c3540e: Pushed
140887c9341b: Pushed
3935c38159a7: Pushed
44b08d0727ff: Pushed
86888f0aea6d: Layer already exists
dda6e8dfdcf7: Layer already exists
latest: digest:
sha256:c2c0639d09b9e12d3236da9905bb57978f7548b7d1cc60025c76bb8b82fddd47 size:
1993
```

► 4. Create the persistent volume.

- 4.1. Log in to OpenShift Container Platform as administrator to create a persistent volume.

Similar to the volumes used with plain Docker containers, OpenShift uses the concept of persistent volumes to provide persistent storage for pods.

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

If the **oc login** command prompts about using insecure connections, answer **y** (yes).

- 4.2. A script is provided for you to create the persistent volumes needed for this exercise. Run the following commands in the terminal window to create the persistent volume.

```
[student@workstation nodejs]$ cd ~/D0285/labs/openshift-template
[student@workstation openshift-template]$ ./create-pv.sh
```



NOTE

If you have an issue with one of the later steps in the lab, you can delete the **template** project using the **oc delete project** to remove any existing resources and restart the exercise from this step.

► 5. Create the To Do List application from the provided JSON template.

- 5.1. Create a new project *template* in OpenShift to use for this exercise. Run the following command to create the **template** project.

```
[student@workstation openshift-template]$ oc new-project template
Now using project "template" on server "https://master.lab.example.com:443".
```

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.

5.2. Set the security policy for the project.

To allow the container to run with the adequate privileges, set the security policy using the provided script. Run the following command to set the policy.

```
[student@workstation openshift-template]$ ./setpolicy.sh
```

5.3. Review the template.

Using your preferred editor, open and examine the template located at **/home/student/D0285/labs/openshift-template/todo-template.json**. Notice the following resources defined in the template and review their configurations.

- The **todoapi** pod definition defines the Node.js application.
- The **mysql** pod definition defines the MySQL database.
- The **todoapi** service provides connectivity to the Node.js application pod.
- The **mysql** service provides connectivity to the MySQL database pod.
- The **dbinit** persistent volume claim definition defines the MySQL **/var/lib/mysql/init** volume.
- The **db-volume** persistent volume claim definition defines the MySQL **/var/lib/mysql/data** volume.

5.4. Process the template and create the application resources.

Use the **oc process** command to process the template file and then use the **pipe** command to send the result to the **oc create** command. This creates an application from the template.

Run the following command in the terminal window:

```
[student@workstation openshift-template]$ oc process -f todo-template.json \
| oc create -f -
pod "mysql" created
pod "todoapi" created
service "todoapi" created
service "mysql" created
persistentvolumeclaim "dbinit" created
persistentvolumeclaim "dbclaim" created
```

5.5. Review the deployment.

Review the status of the deployment using the **oc get pods** command with the **-w** option to continue to monitor the pod status. Wait until both the containers are running. It may take some time for both pods to start.

```
[student@workstation openshift-template]$ oc get pods -w
NAME      READY     STATUS            RESTARTS   AGE
mysql     0/1      ContainerCreating   0          9s
todoapi   1/1      Running           0          9s
NAME      READY     STATUS            RESTARTS   AGE
mysql     1/1      Running           0          2m
^C
```

Press **Ctrl+C** to exit the command.

► 6. Expose the Service

To allow the To Do List application to be accessible through the OpenShift router and to be available as a public FQDN, use the **oc expose** command to expose the **todoapi** service.

Run the following command in the terminal window.

```
[student@workstation openshift-template]$ oc expose service todoapi
route "todoapi" exposed
```

► 7. Test the application.

7.1. Find the FQDN of the application by running the **oc status** command and note the FQDN for the app.

Run the following command in the terminal window.

```
[student@workstation openshift-template]$ oc status
In project template on server https://master.lab.example.com:443

svc/mysql - 172.30.105.104:3306
pod/mysql runs registry.lab.example.com/do285/mysql-57-rhel7

http://todoapi-template.apps.lab.example.com to pod port 30080 (svc/todoapi)
pod/todoapi runs registry.lab.example.com/do285/todonodejs

2 infos identified, use 'oc status -v' to see details.
```

7.2. Use **curl** to test the REST API for the To Do List application.

```
[student@workstation openshift-template]$ curl -w "\n" \
http://todoapi-template.apps.lab.example.com/todo/api/items/1
{"description": "Pick up newspaper", "done": false, "id":1}
```

The **-w "\n"** option with **curl** command lets the shell prompt appear at the next line rather than merging with the output in the same line.

7.3. Open Firefox on workstation and point your browser to <http://todoapi-template.apps.lab.example.com/todo/> and you should see the To Do List application.



NOTE

The trailing slash in the URL mentioned above is necessary. If you miss to include that in the URL, you may encounter issues with the application.

7.4. Verify that the correct images were built, and that the application is running correctly:

```
[student@workstation openshift-template]$ lab openshift-template grade
```

► 8. Clean up.

- 8.1. Delete the project used by this exercise by running the following commands in the terminal window.

```
[student@workstation openshift-template]$ oc delete project template
```

- 8.2. Delete the persistent volumes using the provided shell script by running the following command in your terminal window.

```
[student@workstation openshift-template]$ ./delete-pv.sh
persistentvolume "pv0001" deleted
persistentvolume "pv0002" deleted
```

- 8.3. Delete the container images generated during the Dockerfile builds:

```
[student@workstation openshift-template]$ docker rmi -f $(docker images -q)
```

This concludes the guided exercise.

► LAB

DEPLOYING MULTI-CONTAINER APPLICATIONS

In this lab, you will deploy a PHP Application with a MySQL database using an OpenShift template to define the resources needed for the application.

OUTCOMES

You should be able to create a OpenShift application comprised of multiple containers and access it through a web browser.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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-multicontainer setup
[student@workstation ~]$ cd ~/DO285/labs/deploy-multicontainer
```

1. Log in to OpenShift cluster as the **admin** user and create a new project for this exercise.
 - 1.1. From **workstation** VM, log in as the **admin** user.
 - 1.2. Create a new project in OpenShift, named **deploy**, for this lab.
 - 1.3. Relax the default cluster security policy.
To allow the container, run with the appropriate privileges, set the security policy using the provided **setpolicy.sh** shell script.
2. Build the Database container image and publish it to the private registry.
 - 2.1. Build the MySQL Database image using the provided Dockerfile and build script in the **images/mysql** directory.
 - 2.2. Push the MySQL Image to the Private Registry
In order to make the image available for OpenShift to use in the template, give it a tag of **registry.lab.example.com/do285/mysql-57-rhel7** and push it to the private registry.
3. Build the PHP container image and publish it to the private registry.

- 3.1. Build the PHP image using the provided Dockerfile and build script in the **images/quote-php** directory.
- 3.2. Tag and push the PHP Image to the private registry.

In order to make the image available for OpenShift to use in the template, give it a tag of **registry.lab.example.com/do285/quote-php** and push it to the private registry.

```
[student@workstation quote-php]$ docker tag do285/quote-php \
registry.lab.example.com/do285/quote-php
[student@workstation quote-php]$ docker push \
registry.lab.example.com/do285/quote-php
The push refers to a repository [registry.lab.example.com/do285/quote-php]
4a01dd82afa9: Pushed
40e133ec4a04: Pushed
b32755a27787: Pushed
4cd1411a1153: Pushed
... output omitted ...
latest: digest:
sha256:a60a9c7a9523b4e6674f2cdbfb437107782084828e4a18dc4dfa62ad4939fd6a size:
2194
```

4. Review the provided template file **/home/student/D0285/labs/deploy-multicontainer/quote-php-template.json**. Note the definitions and configuration of the pods, services, and persistent volume claims defined in the template.
5. Create the persistent volumes needed for the application using the provided **create-pv.sh** script.



NOTE

If you have an issue with one of the later steps in the lab, you can delete the **template** project using the **oc delete project** command to remove any existing resources and restart the exercise from this step.

6. Upload the PHP application template so that it can be used by any developer with access to your project.
7. Process the uploaded template and create the application resources.
 - 7.1. Use the **oc process** command to process the template file and use the **pipe** command to send the result to the **oc create** command to create an application from the template.
 - 7.2. Verify the deployment. Verify the status of the deployment using the **oc get pods** command with the **-w** option to monitor the deployment status. Wait until both the pods are running. It may take some time for both pods to start up.
8. Expose the Service.

To allow the PHP Quote application to be accessible through the OpenShift router and reachable from an external network, use the **oc expose** command to expose the **quote-php** service.

9. Test the application.
 - 9.1. Find the FQDN where the application is available using the **oc get route** command and note the FQDN for the app.
 - 9.2. Use **curl** command to test the REST API for the PHP Quote application.

**NOTE**

The text displayed in the output above may differ. But the **curl** command should run successfully.

- 9.3. Verify that the correct images were built and that the application is running without errors.

```
[student@workstation deploy-multicontainer]$ lab deploy-multicontainer grade
```

10. Clean up.

- 10.1. Delete the project used in this exercise.
- 10.2. Delete the persistent volumes using the provided shell script, available at **/home/student/D0285/labs/deploy-multicontainer/delete-pv.sh**.
- 10.3. Delete the container images generated during the Dockerfile builds.

**WARNING**

You may see an error deleting one of the images if there are multiple tags for a single image. This can be safely ignored.

This concludes the lab.

► SOLUTION

DEPLOYING MULTI-CONTAINER APPLICATIONS

In this lab, you will deploy a PHP Application with a MySQL database using an OpenShift template to define the resources needed for the application.

OUTCOMES

You should be able to create a OpenShift application comprised of multiple containers and access it through a web browser.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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-multicontainer setup
[student@workstation ~]$ cd ~/D0285/labs/deploy-multicontainer
```

1. Log in to OpenShift cluster as the **admin** user and create a new project for this exercise.
 - 1.1. From **workstation** VM, log in as the **admin** user.
Run the following command in your terminal window.

```
[student@workstation deploy-multicontainer]$ oc login -u admin \
-p redhat https://master.lab.example.com
Login successful.
...output omitted...
```

- If the **oc login** command prompts about using insecure connections, answer **y** (yes).
- 1.2. Create a new project in OpenShift, named **deploy**, for this lab.
Run the following command to create the project.

```
[student@workstation deploy-multicontainer]$ oc new-project deploy
Now using project "deploy" on server "https://master.lab.example.com:443".
```

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.

- 1.3. Relax the default cluster security policy.

To allow the container, run with the appropriate privileges, set the security policy using the provided **setpolicy.sh** shell script.

Run the following command.

```
[student@workstation deploy-multicontainer]$ ./setpolicy.sh
```

2. Build the Database container image and publish it to the private registry.

- 2.1. Build the MySQL Database image using the provided Dockerfile and build script in the **images/mysql** directory.

To build the base MySQL image, run the **build.sh** script.

```
[student@workstation ~]$ cd ~/D0285/labs/deploy-multicontainer/images/mysql  
[student@workstation mysql]$ ./build.sh
```

- 2.2. Push the MySQL Image to the Private Registry

In order to make the image available for OpenShift to use in the template, give it a tag of **registry.lab.example.com/do285/mysql-57-rhel7** and push it to the private registry.

To tag and push the image run the following commands in the terminal window.

```
[student@workstation mysql]$ docker tag do285/mysql-57-rhel7 \  
registry.lab.example.com/do285/mysql-57-rhel7  
[student@workstation mysql]$ docker push \  
registry.lab.example.com/do285/mysql-57-rhel7  
The push refers to a repository [registry.lab.example.com/do285/mysql-57-rhel7]  
...output omitted...  
latest: digest:  
sha256:170e2546270690fded13f3ced0d575a90cef58028abcef8d37bd62a166ba436b size:  
1156
```

3. Build the PHP container image and publish it to the private registry.

- 3.1. Build the PHP image using the provided Dockerfile and build script in the **images/quote-php** directory.

To build the base PHP image, run the **build.sh** script.

```
[student@workstation ~]$ cd ~/D0285/labs/deploy-multicontainer/images/quote-php
```

```
[student@workstation quote-php]$ ./build.sh
```

3.2. Tag and push the PHP Image to the private registry.

In order to make the image available for OpenShift to use in the template, give it a tag of **registry.lab.example.com/do285/quote-php** and push it to the private registry.

To tag and push the image run the following commands in the terminal window.

```
[student@workstation quote-php]$ docker tag do285/quote-php \
registry.lab.example.com/do285/quote-php
[student@workstation quote-php]$ docker push \
registry.lab.example.com/do285/quote-php
The push refers to a repository [registry.lab.example.com/do285/quote-php]
4a01dd82afa9: Pushed
40e133ec4a04: Pushed
b32755a27787: Pushed
4cd1411a1153: Pushed
... output omitted ...
latest: digest:
sha256:a60a9c7a9523b4e6674f2cdbfb437107782084828e4a18dc4dfa62ad4939fd6a size:
2194
```

4. Review the provided template file **/home/student/D0285/labs/deploy-multicontainer/quote-php-template.json**.

Note the definitions and configuration of the pods, services, and persistent volume claims defined in the template.

5. Create the persistent volumes needed for the application using the provided **create-pv.sh** script.

Run the following commands in the terminal window to create the persistent volume.

```
[student@workstation quote-php]$ cd ~/D0285/labs/deploy-multicontainer
[student@workstation deploy-multicontainer]$ ./create-pv.sh
```



NOTE

If you have an issue with one of the later steps in the lab, you can delete the **template** project using the **oc delete project** command to remove any existing resources and restart the exercise from this step.

6. Upload the PHP application template so that it can be used by any developer with access to your project.

Use the **oc create -f** command to upload the template file to the project.

```
[student@workstation deploy-multicontainer]$ oc create -f quote-php-template.json
template "quote-php-persistent" created
```

7. Process the uploaded template and create the application resources.

- 7.1. Use the **oc process** command to process the template file and use the **pipe** command to send the result to the **oc create** command to create an application from the template.

Run the following command in the terminal window.

```
[student@workstation deploy-multicontainer]$ oc process quote-php-persistent \
| oc create -f -
pod "mysql" created
pod "quote-php" created
service "quote-php" created
service "mysql" created
persistentvolumeclaim "dbinit" created
persistentvolumeclaim "dbclaim" created
```

7.2. Verify the deployment.

Verify the status of the deployment using the **oc get pods** command with the **-w** option to monitor the deployment status. Wait until both the pods are running. It may take some time for both pods to start up.

Run the following command in the terminal window.

```
[student@workstation deploy-multicontainer]$ oc get pods -w
NAME      READY     STATUS            RESTARTS   AGE
mysql     0/1      ContainerCreating   0          8s
quote-php 1/1      Running           0          8s
NAME      READY     STATUS            RESTARTS   AGE
mysql     1/1      Running           0          2m
^C
```

Press **Ctrl+C** to exit the command.

8. Expose the Service.

To allow the PHP Quote application to be accessible through the OpenShift router and reachable from an external network, use the **oc expose** command to expose the **quote-php** service.

Run the following command in the terminal window.

```
[student@workstation deploy-multicontainer]$ oc expose svc quote-php
route "quote-php" exposed
```

9. Test the application.

- 9.1. Find the FQDN where the application is available using the **oc get route** command and note the FQDN for the app.

Run the following command in the terminal window.

```
[student@workstation deploy-multicontainer]$ oc get route
NAME      HOST/PORT            PATH      SERVICES      PORT
TERMINATION  WILDCARD
```

quote-php	quote-php-deploy.apps.lab.example.com	quote-php	8080
	None		

- 9.2. Use **curl** command to test the REST API for the PHP Quote application.

```
[student@workstation ~]$ curl http://quote-php-deploy.apps.lab.example.com
Always remember that you are absolutely unique. Just like everyone else.
```



NOTE

The text displayed in the output above may differ. But the **curl** command should run successfully.

- 9.3. Verify that the correct images were built and that the application is running without errors.

```
[student@workstation deploy-multicontainer]$ lab deploy-multicontainer grade
```

10. Clean up.

- 10.1. Delete the project used in this exercise.

Run the following command in your terminal window.

```
[student@workstation deploy-multicontainer]$ oc delete project deploy
```

- 10.2. Delete the persistent volumes using the provided shell script, available at **/home/student/D0285/labs/deploy-multicontainer/delete-pv.sh**.

Run the following commands in your terminal window.

```
[student@workstation deploy-multicontainer]$ ./delete-pv.sh
persistentvolume "pv0001" deleted
persistentvolume "pv0002" deleted
```

- 10.3. Delete the container images generated during the Dockerfile builds.

```
[student@workstation deploy-multicontainer]$ docker rmi -f $(docker images -q)
```



WARNING

You may see an error deleting one of the images if there are multiple tags for a single image. This can be safely ignored.

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Containerized applications cannot rely on fixed IP addresses or host names to find services. Docker and Kubernetes provide mechanisms that define environment variables with network connection parameters.
- The user-defined docker networks allow containers to contact other containers by names. To be able to communicate with each other, containers must be attached to the same user-defined docker network, using **--network** option with the **docker run** command.
- Kubernetes services define environment variables injected into all pods from the same project.
- Kubernetes templates automate creating applications consisting of multiple pods interconnected by services.
- Template parameters define environment variables for multiple pods.

CHAPTER 11

EXECUTING COMMANDS

GOAL

Execute commands using the command-line interface.

OBJECTIVES

- Configure OpenShift resources using the command-line interface.
- Execute commands that assist in troubleshooting common problems.

SECTIONS

- 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 (namespaces)
- Deployment Configuration
- 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.

Red Hat OpenShift Container Platform 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 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 \
-u developer -p redhat
```



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 reenter 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

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"
```

It is possible to log in as the OpenShift cluster administrator from any master node without a password by connecting via **ssh** to the master node.

```
[root@master ~]# oc whoami
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	
<hr/>				
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
<hr/>				
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
<hr/>				
NAME	HOST/PORT	PATH		
SERVICES	PORT	TERMINATION		
routes/docker-registry	5000-tcp	docker-registry-default.apps.lab.example.com passthrough		
routes/registry-console		registry-console-default.apps.lab.example.com registry-console passthrough		
<hr/>				
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
<hr/>				
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

Use the **oc export** command 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

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

oc delete RESOURCE_TYPE name

Use the **oc delete** command 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 recreated.

oc exec

Use the **oc exec** command to execute commands inside a container. You can use this command to run interactive as well as noninteractive batch commands as part of a script.

oc rsh POD

The **oc rsh pod** 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, run the following command.

```
[student@workstation ~]$ oc rsh <pod>
```

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. As an administrator, you 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 life-cycle 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.



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
------------------------	---

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 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**. If prompted, access the security certificate.

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

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

```
[student@workstation ~]$ oc project default
Already on project "default" on server "https://master.lab.example.com:443".
```

- 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 **Ready** as part of their status descriptions. Applications (pods) are deployed on such nodes.

NAME	STATUS	ROLES	AGE	VERSION
master.lab.example.com	Ready	master	8m	v1.9.1+a0ce1bc657
node1.lab.example.com	Ready	compute	8m	v1.9.1+a0ce1bc657
node2.lab.example.com	Ready	compute	8m	v1.9.1+a0ce1bc657

- 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
                node-role.kubernetes.io/master=true
                openshift-infra=apiserver
                region=master

Taints:        <none>
... output omitted ...
System Info:
... output omitted ...
Kernel Version:      3.10.0-862.el7.x86_64
OS Image:          Red Hat Enterprise Linux Server 7.5 (Maipo)
Operating System:   linux
Architecture:       amd64
Container Runtime Version: docker://1.13.1
Kubelet Version:    v1.9.1+a0ce1bc657
Kube-Proxy Version: v1.9.1+a0ce1bc657
ExternalID:        master.lab.example.com
... output omitted ...
Events:
... output omitted ...
Normal  Starting  Starting kubelet.
... output omitted ...
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
Roles:          compute
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
```

```

kubernetes.io/hostname=node1.lab.example.com
node-role.kubernetes.io/compute=true
region=infra
Annotations:      volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:           <none>
CreationTimestamp: Wed, 25 Jul 2018 14:18:36 -0700
... output omitted ...
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-pnt4r  1/1   Running  10.128.0.12  node2.lab.example.com
docker-registry-1-q8hrl  1/1   Running  10.129.0.7   node1.lab.example.com
registry-console-1-ch4gp 1/1   Running  10.128.0.11  node2.lab.example.com
router-1-9dq65        1/1   Running  172.25.250.11 node1.lab.example.com
router-1-vsnb9        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-pnt4r
Name:         docker-registry-1-pnt4r
Namespace:    default
Node:         node2.lab.example.com/172.25.250.12
Start Time:   Wed, 25 Jul 2018 14:21:13 -0700
Labels:       deployment=docker-registry-1
              deploymentconfig=docker-registry
              docker-registry=default
Annotations:  openshift.io/deployment-config.latest-version=1
              openshift.io/deployment-config.name=docker-registry
              openshift.io/deployment.name=docker-registry-1
              openshift.io/scc=restricted
Status:       Running
... output omitted ...
Events:
  Type  Reason          Age           From
  Message
  ----  ----
  Normal Scheduled      9m           default-scheduler
  Successfully assigned docker-registry-1-pnt4r to node2.lab.example.com
  ... output omitted ...
  Normal Created        5m (x3 over 9m)  kubelet, node2.lab.example.com
  Created container
```

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-pnt4r hostname  
docker-registry-1-pnt4r
```

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

```
[student@workstation ~]$ oc exec router-1-9dq65 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 reading files, contents, and processes from within the container itself. Inspect the **/etc/resolv.conf** file.

```
[student@workstation ~]$ oc exec docker-registry-1-pnt4r cat /etc/resolv.conf  
nameserver 172.25.250.12  
search default.svc.cluster.local svc.cluster.local cluster.local lab.example.com  
example.com  
options ndots:5
```

- 2.3. Use the **oc rsh** command to initiate a remote shell connection to the router pod, which is useful for more in-depth troubleshooting sessions. On the **master** node, launch a remote shell in the pod:

```
[student@workstation ~]$ oc rsh docker-registry-1-pnt4r  
sh-4.2$
```

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

```
bash-4.2$ ls /  
bin config.yml 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:443

https://docker-registry-default.apps.lab.example.com (passthrough) (svc/docker-registry)
dc/docker-registry deploys registry.lab.example.com/openshift3/ose-docker-registry:v3.9.14
    deployment #1 deployed 15 minutes ago - 2 pods

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

https://registry-console-default.apps.lab.example.com (passthrough) (svc/registry-console)
dc/registry-console deploys docker.io/openshift3/registry-console:v3.9
    deployment #1 deployed 14 minutes ago - 1 pod

svc/router - 172.30.149.232 ports 80, 443, 1936
dc/router deploys registry.lab.example.com/openshift3/ose-haproxy-router:v3.9.14
    deployment #1 deployed 16 minutes ago - 2 pods

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.
```

The output on your **master** node may 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. Import and export resources.

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

```
[student@workstation ~]$ oc get all
NAME                                REVISION DESIRED  CURRENT   TRIGGERED BY
deploymentconfigs/docker-registry   1         2          2          config
deploymentconfigs/registry-console  1         1          1          config
deploymentconfigs/router            1         2          2          config

NAME                               DOCKER REPO                      TAGS UPDATED
imagestreams/registry-console      docker-registry.default.svc:
                                         5000/default/\registry-console v3.9

... output omitted ...

NAME        READY  STATUS  RESTARTS  AGE
po/docker-registry-1-pnt4r        1/1    Running  2          16m
po/docker-registry-1-q8hrl        1/1    Running  1          16m
po/registry-console-1-ch4gp      1/1    Running  2          15m
```

```
po/router-1-9dq65          1/1      Running  1      16m
po/router-1-vsnb9          1/1      Running  2      16m

NAME            DESIRED  CURRENT  READY   AGE
rc/docker-registry-1    2        2        2      17m
rc/registry-console-1  1        1        1      15m
rc/router-1         2        2        2      17m
```

... output omitted ...

The output on your system may 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 recreating resources elsewhere in the cluster.

Export the *docker-registry-1-pnt4r* pod resource in the default YAML format. Replace the pod name with one of the available registry pods in your cluster.

```
[student@workstation ~]$ oc export pod docker-registry-1-pnt4r
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/deployment-config.latest-version: "1"
    openshift.io/deployment-config.name: docker-registry
    openshift.io/deployment.name: docker-registry-1
    openshift.io/scc: restricted
  creationTimestamp: null
  generateName: docker-registry-1-
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  ownerReferences:
  - apiVersion: v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicationController
    name: docker-registry-1
... output omitted ...
```



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

Export the service and deployment configuration definition as a single OpenShift template by running the following command:

```
[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
  metadata:
    creationTimestamp: null
    labels:
      docker-registry: default
    name: docker-registry
  spec:
    ports:
    - name: 5000-tcp
      port: 5000
      protocol: TCP
      targetPort: 5000
    selector:
      docker-registry: default
    sessionAffinity: ClientIP
    sessionAffinityConfig:
      clientIP:
        timeoutSeconds: 10800
    type: ClusterIP
  status:
    loadBalancer: {}
... output omitted ...

```

The previous command exports both the service definition and the deployment configuration as a template. The output of this command can be sent as input to the **oc create** command to recreate the resource in a cluster.

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

```

[student@workstation ~]$ oc export --help
... output omitted ...
Examples:
# export the services and deployment configurations labeled name=test
oc export svc,dc -l name=test

# export all services to a template
oc export service --as-template=test

# export to JSON
oc export service -o json

```

```
... output omitted ...
```

**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 Red Hat OpenShift Container Platform 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 3.5)

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

...
output omitted ...
Running plugins.
Please wait ...

...
Running 60/93: openvswitch...
Running 61/93: origin...
...

Creating compressed archive...

Your sosreport has been generated and saved in:
/var/tmp/sosreport-master.lab.example.com-20180725145249.tar.xz

The checksum is: a544e79319d08538ecfef07687f77e54

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 Red Hat 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. You can get high-level information about failures and issues in the cluster, and 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:

Monitoring » Events

Events

Filter by keyword			Sort by	Time
Time	Kind and Name	Reason and Message		
3:10:51 PM	Cluster Service Broker ansible-service-broker	Error Fetching Catalog	⚠	Error getting broker catalog: Get https://asb.openshift-ansible-service-broker.svc:1338/ansible-service-broker/v2/catalog: dial tcp 172.30.153.16:1338: getsockopt: no route to host 161 times in the last 49 minutes
3:01:46 PM	Cluster Service Broker template-service-broker	Fetched Catalog		Successfully fetched catalog entries from broker. 3 times in the last 48 minutes
2:54:21 PM	Pod pod-diagnostic-test-6zkxs	Sandbox Changed		Pod sandbox changed, it will be killed and re-created. 21 times in the last 21 minutes

Figure 11.1: Viewing events in the web console.

A comprehensive list of events in the OpenShift Container Platform 3.9 is available at https://docs.openshift.com/container-platform/3.9/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
```

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 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

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 the following command:

```
[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 configuration 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 configuration 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 the issue.

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

Run 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**, the **atomic-openshift-master-controllers**, and **atomic-openshift-master-api** services by editing the **--loglevel** variable in the respective configuration files, and then restarting the associated service.

For example, to set the OpenShift master controller log level to debug, edit the following line in the **/etc/sysconfig/atomic-openshift-master-controllers** file:

```
OPTIONS=--loglevel=4 --listen=https://0.0.0.0:8444
```



NOTE

Red Hat OpenShift Container Platform 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.

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

► 1. Create a new project.

- 1.1. On the **workstation** host, access the OpenShift master located at `https://master.lab.example.com` with the OpenShift client.
Log in as **developer** and accept the security certificate.

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

- 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:443".
...
```

► 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 Git repository running on the `services` VM.

```
[student@workstation ~]$ oc new-app --name=hello -i php:5.4 \
http://services.lab.example.com/php-helloworld
error: multiple images or templates matched "php:5.4": 2
```

The argument "php:5.4" could apply to the following Docker images, OpenShift image streams, or templates:

```
* Image stream "php" (tag "7.0") in project "openshift"
  Use --image-stream="openshift/php:7.0" to specify this image or template

* Image stream "php" (tag "5.6") in project "openshift"
  Use --image-stream="openshift/php:5.6" 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
Name:      php
Namespace:  openshift
Created:   About an hour ago
Labels:    <none>
Annotations:  openshift.io/display-name=PHP
              openshift.io/image.dockerRepositoryCheck=2018-07-25T21:16:14Z
Docker Pull Spec: docker-registry.default.svc:5000/openshift/php
Image Lookup:  local=false
Unique Images:  2
Tags:      5

7.1 (latest)
tagged from registry.lab.example.com/rhscl/php-71-rhel7:latest

Build and run PHP 7.1 applications on RHEL 7. For more information about using
this builder image, including OpenShift considerations, see https://github.com/
sclorg/s2i-php-container/blob/master/7.1/README.md.

Tags: builder, php
Supports: php:7.1, php
Example Repo: https://github.com/openshift/cakephp-ex.git

! error: Import failed (NotFound): dockerimage.image.openshift.io
"registry.lab.example.com/rhscl/php-71-rhel7:latest" not found

7.0
tagged from registry.lab.example.com/rhscl/php-70-rhel7:latest

Build and run PHP 7.0 applications on RHEL 7. For more information about using
this builder image, including OpenShift considerations, see https://github.com/
sclorg/s2i-php-container/blob/master/7.0/README.md.

Tags: builder, php
Supports: php:7.0, php
Example Repo: https://github.com/openshift/cakephp-ex.git
```

```
* registry.lab.example.com/rhscl/php-70-
rhel7@sha256:23765e00df8d0a934ce4f2e22802bc0211a6d450bfbb69144b18cb0b51008ddd
      5 days ago

5.6
tagged from registry.lab.example.com/rhscl/php-56-rhel7:latest

Build and run PHP 5.6 applications on RHEL 7. For more information about using
this builder image, including OpenShift considerations, see https://github.com/
sclorg/s2i-php-container/blob/master/5.6/README.md.
Tags: builder, php
Supports: php:5.6, php
Example Repo: https://github.com/openshift/cakephp-ex.git

* registry.lab.example.com/rhscl/php-56-
rhel7@sha256:920c2cf85b5da5d0701898f0ec9ee567473fa4b9af6f3ac5b2b3f863796bbd68

5.5
tagged from registry.lab.example.com/openshift3/php-55-rhel7:latest

Build and run PHP 5.5 applications on RHEL 7. For more information about using
this builder image, including OpenShift considerations, see https://github.com/
sclorg/s2i-php-container/blob/master/5.5/README.md.
Tags: hidden, builder, php
Supports: php:5.5, php
Example Repo: https://github.com/openshift/cakephp-ex.git

! error: Import failed (NotFound): dockerimage.image.openshift.io
"registry.lab.example.com/openshift3/php-55-rhel7:latest" not found
About an hour ago
```

The output of the command shows that **php:7.0** and **php-5.6** are valid tags, whereas **php-7.1** and **php-5.5** are invalid, because those images are 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://services.lab.example.com/php-helloworld
--> Found image c101534 (10 months old) in image stream "openshift/php" under tag
"7.0" for "php:7.0"
... output omitted ...
--> Success
Build scheduled, use 'oc logs -f bc/hello' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/hello'
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       41s    <none>    <none>
```

The **hello-1-build** pod is in the **Pending** state and the application pod is not starting. Investigate why the deployment is in the **Pending** state and fix the issue.

- ▶ 3. Check the logs of the builder pod by using the **oc logs** command.

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

This command does 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 as follows:

```
[student@workstation ~]$ oc get events
LAST SEEN   FIRST SEEN   COUNT   NAME                                     KIND
5m          5m           6        hello-1-build.1544bd6f20c095e9   Pod

TYPE        REASON          SOURCE
Warning     FailedScheduling default-scheduler

MESSAGE
0/3 nodes are available: 1 MatchNodeSelector, 2 NodeNotReady.
```

Use the **oc describe** command to see if the output gives some hints on why the pod is failing:

```
[student@workstation ~]$ oc describe pod hello-1-build
Name:         hello-1-build
Namespace:    common-troubleshoot
Node:         <none>
Labels:       openshift.io/build.name=hello-1
Annotations:  openshift.io/build.name=hello-1
              openshift.io/scc=privileged
Status:       Pending
... output omitted ...
Events:
  Type      Reason          Age            From           Message
  ----      ----          --            ----          -----
  Warning   FailedScheduling 2s (x18 over 4m)  default-scheduler 0/3 nodes are
available: 1 MatchNodeSelector, 2 NodeNotReady.
```

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    ROLES   AGE     VERSION
master.lab.example.com  Ready    master   1h      v1.9.1+a0ce1bc657
node1.lab.example.com   NotReady  compute  1h      v1.9.1+a0ce1bc657
```

```
node2.lab.example.com    NotReady    compute    1h          v1.9.1+a0ce1bc657
```

The **STATUS** column indicates that both **node1** and **node2** are in the **NotReady** state. Kubernetes 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 the following command. Then proceed with running the **oc get nodes** command.

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

- ▶ 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:

```
... output omitted ...
Jul 25 15:46:08 node2.lab.example.com atomic-openshift-node[23635]: E0725
15:46:08.480373 23635 generic.go:197] GenericPLEG: Unable to retrieve pods: rpc
error: code = Unknown desc = Cannot connect to the Docker daemon at unix:///var/
run/docker.sock. Is the docker daemon running?
... output omitted ...
```

- ▶ 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:

```
... output omitted ...
Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset:
disabled)
```

```
Active: inactive (dead) since Wed 2018-07-25 15:32:35 PDT; 14min ago
... output omitted ...
Jul 25 15:32:35 node2.lab.example.com 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    ROLES      AGE      VERSION
master.lab.example.com   Ready     master     1h      v1.9.1+a0ce1bc657
node1.lab.example.com    Ready     compute    1h      v1.9.1+a0ce1bc657
node2.lab.example.com    Ready     compute    1h      v1.9.1+a0ce1bc657
```

**NOTE**

It may take several minutes before the command reports that the nodes are in the **Ready** state.

- 10. From the **workstation** VM, verify that the pod is now in the **Running** state:

```
[student@workstation ~]$ oc get pods
NAME        READY    STATUS    RESTARTS   AGE
```

```
hello-1-build 1/1 Running 0 11m
```

You should see the application pod in the **Running** state.



NOTE

It might take some time for the application to build and deploy on the cluster. 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://services.lab.example.com/php-helloworld" ...
Commit: e9f757edc9ab596aea20e1f1a44df739005b1453 (Establish remote
repository)
Author: root <root@services.lab.example.com>
Date: Tue Jul 17 18:52:26 2018 -0700
---> Installing application source...
Pushing image docker-registry.default.svc: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.

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: 13 minutes ago
Labels: app=hello
Annotations: openshift.io/generated-by=OpenShiftNewApp
Docker Pull Spec: docker-registry.default.svc:5000/common-troubleshoot/hello
Image Lookup: local=false
Tags: <none>

latest
no spec tag

* docker-registry.default.svc:5000/common-troubleshoot/
hello@sha256:1aad0df1a216b6b070ea3ecfd8cadfdee6dd10b451b8e252dbc835148fc9faf0
About a minute ago
```

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 on OpenShift using the **oc** command line tool.

RESOURCES

Docker image URL:	http://services.lab.example.com/node-hello
Application URL:	http://hello.apps.lab.example.com

OUTCOMES

You should be able to troubleshoot and fix errors related to a custom application.

BEFORE YOU BEGIN

All the labs from the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- The lab setup script creates a new project called **execute-review** for the **developer**. From the **workstation** VM, go to the **/home/student/D0285/labs/execute-review** directory and use the **git clone** command to retrieve the custom **node-hello** application located at <http://services.lab.example.com/node-hello>. Use Docker to build the application with a name of **node-hello** and a tag of **latest**. List the images on **workstation** then tag the **node-hello** image with a repository value of **registry.lab.example.com** and a tag value of **latest**. Push the **node-hello** image to the private Docker repository available at <https://registry.lab.example.com>. After pushing the image, go to the home directory of the **student** user.
- 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. Run the **oc new-app** command to create a new application from the latest **node-hello** image in the **execute-review** project. Inspect and list the resources in the **execute-review** project.

3. Check the logs for the **deploy** pod and determine if it provides any useful information for troubleshooting the issue.
Check the event log for the project and locate the entry that indicates an error.
4. Investigate the cause of this error located in the previous step. Inspect the deployment configuration for the application and review the image settings. The application uses the `registry.lab.example.com/node-hello` image from the classroom registry running on the **services** VM.
Identify the node on which the pod is scheduled, and try to manually pull the Docker image for the application from the node. If the pulling fails, review the Docker settings on the two nodes to identify and resolve the issue.
5. After fixing the issue, connect to the **workstation** VM and roll out a new version of the deployment configuration by running the following command:

```
[student@workstation ~]$ oc rollout latest hello
```

- Review the status of the application pod and access pod logs.
6. Expose the **hello** service as a route. Use `hello.apps.lab.example.com` as the hostname for the route.
Access the route URL `hello.apps.lab.example.com` using the `curl` command, or a browser from the **workstation** VM. **Hi! I am running on host -> hello-2-25bts** should be displayed.

7. 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.
8. **Clean up**
Delete the **execute-review** project.
This concludes the lab.

► SOLUTION

EXECUTING COMMANDS

In this lab, you will troubleshoot and fix issues related to an application on OpenShift using the **oc** command line tool.

RESOURCES

Docker image URL:	http://services.lab.example.com/node-hello
Application URL:	http://hello.apps.lab.example.com

OUTCOMES

You should be able to troubleshoot and fix errors related to a custom application.

BEFORE YOU BEGIN

All the labs from the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- The lab setup script creates a new project called **execute-review** for the **developer**. From the **workstation** VM, go to the **/home/student/D0285/labs/execute-review** directory and use the **git clone** command to retrieve the custom **node-hello** application located at <http://services.lab.example.com/node-hello>. Use Docker to build the application with a name of **node-hello** and a tag of **latest**. List the images on **workstation** then tag the **node-hello** image with a repository value of **registry.lab.example.com** and a tag value of **latest**. Push the **node-hello** image to the private Docker repository available at <https://registry.lab.example.com>. After pushing the image, go to the home directory of the **student** user.
 - From the **workstation** VM, open a new terminal and go to the **/home/student/D0285/labs/execute-review** directory. Run the **git clone** command to retrieve the custom **node-hello** application.

```
[student@workstation ~]$ cd /home/student/D0285/labs/execute-review
[student@workstation execute-review]$ git clone \
```

```
http://services.lab.example.com/node-hello  
Cloning into 'node-hello'...  
remote: Counting objects: 5, done.  
remote: Compressing objects: 100% (5/5), done.  
remote: Total 5 (delta 0), reused 0 (delta 0)  
Unpacking objects: 100% (5/5), done.
```

- 1.2. Go to the **node-hello** directory and run the **docker build** command to build the application. Give the image a name of **node-hello** and a tag of **latest**.

```
[student@workstation execute-review]$ cd node-hello  
[student@workstation node-hello]$ docker build -t node-hello:latest .  
Sending build context to Docker daemon 54.27 kB  
Step 1/6 : FROM registry.lab.example.com/rhscl/nodejs-6-rhel7:latest  
--> fba56b5381b7  
Step 2/6 : MAINTAINER username "username@example.com"  
--> Using cache  
--> 594c1b383445  
Step 3/6 : EXPOSE 3000  
--> Running in abbbbee82f23f  
--> 9697c5a46218  
Removing intermediate container abbbbee82f23f  
Step 4/6 : COPY . /opt/app-root/src  
--> a6da56146c15  
Removing intermediate container bfa6669d33bd  
Step 5/6 : RUN source scl_source enable rh-nodejs6 &&      npm install \  
--registry=http://services.lab.example.com:8081/nexus/content/groups/nodejs/  
--> Running in 8403f9741cbc  
... output omitted ...  
Removing intermediate container c5a774d306fe  
Successfully built a9861ee36be4
```

- 1.3. List the Docker images on **workstation** VM to confirm that the image has been created.

```
[student@workstation node-hello]$ docker images  
REPOSITORY                      TAG      IMAGE ID      ...  
node-hello                      latest   a9861ee36be4 ...  
registry.lab.example.com/openshift/hello-openshift  latest   7af3297a3fb4 ...  
registry.lab.example.com/rhscl/nodejs-6-rhel7        latest   fba56b5381b7 ...
```

- 1.4. Use the **docker tag** command to tag the image. Give the repository value of **registry.lab.example.com** and a tag value of **latest**. To tag the image, copy the image ID of the **node-hello** image.

Rerun the **docker images** command to ensure that the tag is properly applied.

```
[student@workstation node-hello]$ docker tag a9861ee36be4 \  
registry.lab.example.com/node-hello:latest  
[student@workstation node-hello]$ docker images  
REPOSITORY                      TAG      IMAGE ID      ...  
node-hello                        latest   a9861ee36be4 ...  
registry.lab.example.com/node-hello          latest   1362bf635aa4...  
registry.lab.example.com/openshift/hello-openshift  latest   7af3297a3fb4 ...
```

```
registry.lab.example.com/rhscl/nodejs-6-rhel7      latest fba56b5381b7 ...
```

- 1.5. Push the **node-hello** image to the private Docker registry.

```
[student@workstation node-hello]$ docker push \
registry.lab.example.com/node-hello:latest
The push refers to a repository [registry.lab.example.com/node-hello]
29ed16eb80e5: Pushed
82dfac496b77: Mounted from rhscl/nodejs-6-rhel7
aa29c7023a3c: Mounted from rhscl/nodejs-6-rhel7
45f0d85c3257: Mounted from rhscl/nodejs-6-rhel7
5444fe2e6b50: Mounted from rhscl/nodejs-6-rhel7
d4d408077555: Mounted from rhscl/nodejs-6-rhel7
latest: digest:
sha256:d1a9ee136434268efc3029a43fa91e576fc0e3dd89b72784bb6cf16808be7d91 size:
1579
```

- 1.6. Go to the home directory.

```
[student@workstation node-hello]$ cd
[student@workstation ~]$
```

2. 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.

Run the **oc new-app** command to create a new application from the latest **node-hello** image in the **execute-review** project.

Inspect and list the resources in the **execute-review** project.

- 2.1. Log in to the cluster as the **developer** user and list the projects.

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://master.lab.example.com
[student@workstation ~]$ oc projects
[student@workstation ~]$ oc project execute-review
Now using project "execute-review" on server "https://master.lab.example.com:443".
```

- 2.2. Run the **oc new-app** command to create a new application named **hello** in the **execute-review** project.

```
[student@workstation ~]$ oc new-app \
registry.lab.example.com/node-hello \
--name hello
--> Found Docker image 94ca56d (9 minutes old) from registry.lab.example.com for
"registry.lab.example.com/node-hello"
... output omitted ...
--> Creating resources ...
    imagestream "hello" created
    deploymentconfig "hello" created
    service "hello" created
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/hello'
```

Run 'oc status' to view your app.

2.3. Inspect and list the resources in the **execute-review** project.

```
[student@workstation ~]$ oc get all
NAME           REVISION  DESIRED  CURRENT  TRIGGERED BY
deploymentconfigs/hello  1          1          1        config,image(hello:latest)

NAME           DOCKER REPO
imagestreams/hello  docker-registry.default.svc:5000/execute-review/hello

NAME           READY  STATUS      RESTARTS  AGE
po/hello-1-deploy  1/1    Running     0         36s
po/hello-1-jq4lf   0/1    Terminating  2         2m
po/hello-1-qz7v2   0/1    ImagePullBackOff  0         33s

NAME  DESIRED  CURRENT  READY  AGE
rc/hello-1  1        1        0      36s

NAME  TYPE      CLUSTER-IP      EXTERNAL-IP  PORT(S)      AGE
svc/hello  ClusterIP  172.30.40.167  <none>       3000/TCP, 8080/TCP  36s
```

The **hello-1-qz7v2** pod is be stuck in the **ImagePullBackOff** state and the application pod is not starting. Investigate why the application pod is not in *Running* state and fix the issue.

3. Check the logs for the **deploy** pod and determine if it provides any useful information for troubleshooting the issue.

Check the event log for the project and locate the entry that indicates an error.

- 3.1. Run the **oc logs hello-1-deploy** command to retrieve the logs for the pod responsible for deploying the application. The logs do not provide any useful information that can help you troubleshoot the issue.

```
[student@workstation ~]$ oc logs hello-1-qz7v2
Error from server (BadRequest): container "hello" in pod "hello-1-qz7v2" is
waiting to start: trying and failing to pull image
```

- 3.2. Use the **oc describe** against the deployment pod command to see if it gives some hints on why the application pod is failing. Unfortunately, the output does not provide any useful information.

```
[student@workstation ~]$ oc describe pod hello-1-deploy
Name:           hello-1-deploy
Namespace:      execute-review
Node:           node2.lab.example.com/172.25.250.12
Start Time:    Thu, 26 Jul 2018 11:13:16 -0700
Labels:         openshift.io/deployer-pod-for.name=hello-1
Annotations:   openshift.io/deployment-config.name=hello
                  openshift.io/deployment.name=hello-1
                  openshift.io/scc=restricted
Status:        Running
```

IP: 10.128.1.119

- 3.3. Check the event log for the project and locate the entry that indicates an error.

The event log shows that OpenShift is unable to pull images defined in the application's deployment configuration. Locate the line that indicates that the endpoints are blocked.

```
[student@workstation ~]$ oc get events \
--sort-by='.metadata.creationTimestamp'
... output omitted ...
g Failed kubelet, node1.lab.example.com \
Failed to pull image "registry.lab.example.com/node-hello@\nsha256:02071b47f0385e8e94893fcfa74943ec6ec96fce5eb53eac6691dea57ae18a98": \
rpc error: code = Unknown desc = All endpoints blocked.
1m 1m 2 hello-1-qz7v2.154724ea481d96b3 \
Pod spec.containers{hello} Normal \
Pulling kubelet, node1.lab.example.com \
pulling image "registry.lab.example.com/node-hello@\nsha256:02071b47f0385e8e94893fcfa74943ec6ec96fce5eb53eac6691dea57ae18a98"
1m 1m 7 hello-1-qz7v2.154724ea6c66941e \
Pod spec.containers{hello} Normal \
SandboxChanged kubelet, node1.lab.example.com \
Pod sandbox changed, it will be killed and re-created.
1m 1m 5 hello-1-qz7v2.154724eaedb1ee14 \
Pod spec.containers{hello} Normal \
BackOff kubelet, node1.lab.example.com \
Back-off pulling image "registry.lab.example.com/node-hello@\nsha256:02071b47f0385e8e94893fcfa74943ec6ec96fce5eb53eac6691dea57ae18a98"
1m 1m 6 hello-1-qz7v2.154724eaedb20987 \
Pod spec.containers{hello} Warning \
Failed kubelet, node1.lab.example.com \
Error: ImagePullBackOff
```

4. Investigate the cause of this error located in the previous step. Inspect the deployment configuration for the application and review the image settings. The application uses the `registry.lab.example.com/node-hello` image from the classroom registry running on the `services` VM.

Identify the node on which the pod is scheduled, and try to manually pull the Docker image for the application from the node. If the pulling fails, review the Docker settings on the two nodes to identify and resolve the issue.

- 4.1. Use the `oc get dc` command to retrieve the deployment configuration. Locate the section that defines the location of the Docker image.

```
[student@workstation ~]$ oc get dc hello -o yaml
... output omitted ...
spec:
  containers:
    - image: registry.lab.example.com/node-
hello@sha256:d1a9ee136434268efc3029a43fa91e576fc0e3dd89b72784bb6cf16808be7d91
      imagePullPolicy: Always
      name: hello
      ports:
        - containerPort: 8080
          protocol: TCP
        - containerPort: 3000
```

```
protocol: TCP
resources: {}
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
... output omitted ...
```

The deployment configuration should point to the **registry.lab.example.com/node-hello** image and should have a **sha256** hash suffix.

- 4.2. Identify the node on which the pod is scheduled, and try to manually pull the docker image for the application (**registry.lab.example.com/node-hello**) from the node.

```
[student@workstation ~]$ oc get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP
NODE
hello-1-deploy 1/1     Running   0          33s    10.128.1.139
node1.lab.example.com
hello-1-qz7v2   0/1     ImagePullBackOff 0          29s    <none>
node1.lab.example.com
```

Open a new terminal on **workstation** and log in to the node where the pod is scheduled using the **ssh** command, then pull the **node-hello** docker image.

```
[student@workstation ~]$ ssh root@node1
[root@node1 ~]# docker pull registry.lab.example.com/node-hello
Using default tag: latest
Trying to pull repository registry.lab.example.com/node-hello ...
All endpoints blocked.
[root@node1 ~]# exit
```

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.

- 4.3. Review the Docker settings on both nodes and correct them. Ensure that the settings for **ADD_REGISTRY** and **BLOCK_REGISTRY** variables in the Docker configuration file are correct.

From the **workstation** VM, log in to **node1** and **node2** using the **ssh** command:

```
[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 of the file.

```
... output omitted ...
# Added by the 'lab execute-review setup' script
BLOCK_REGISTRY='--block-registry registry.access.redhat.com \
```

```
--block-registry docker.io --block-registry registry.lab.example.com'
```

You are seeing errors about blocked endpoints because the classroom private registry is added to the **BLOCK_REGISTRY** entry. Remove the entry from the **BLOCK_REGISTRY** directive. The line should read as follows:

```
# Added by the 'lab execute-review setup' script
BLOCK_REGISTRY='--block-registry registry.access.redhat.com \
--block-registry docker.io'
```

Repeat the steps and correct the Docker configuration on the other node VM.

- 4.4. Restart the **docker** service on both nodes.

```
[root@node1 ~]# systemctl restart docker.service
[root@node2 ~]# systemctl restart docker.service
```

5. After fixing the issue, connect to the **workstation** VM and roll out a new version of the deployment configuration by running the following command:

```
[student@workstation ~]$ oc rollout latest hello
```

Review the status of the application pod and access pod logs.

- 5.1. Deploy a new version of the application by running the **oc rollout latest hello** command.

```
[student@workstation ~]$ oc rollout latest hello
deploymentconfig "hello" rolled out
```

- 5.2. Run the **oc get pods** command to retrieve the list of the running pods. There should be a pod with a state of **Running**



NOTE

It may take a couple minutes for the pod to transition to **Running** state.

```
[student@workstation ~]$ oc get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE      IP           NODE
hello-1-deploy  0/1     Error     0          11m     10.128.1.139
node2.lab.example.com
hello-2-xk4nn   1/1     Running   0          2m      10.128.1.146
node2.lab.example.com
```

- 5.3. Use the **oc logs** command to ensure that the pod is properly running.

```
[student@workstation ~]$ oc logs hello-2-xk4nn
nodejs server running on http://0.0.0.0:3000
```

6. Expose the **hello** service as a route. Use `hello.apps.lab.example.com` as the hostname for the route.

Access the route URL `hello.apps.lab.example.com` using the **curl** command, or a browser from the **workstation** VM. **Hi! I am running on host -> hello-2-25bts** should be displayed.

- 6.1. Expose the **hello** service as a route. Give the route a hostname of `hello.apps.lab.example.com`.

```
[student@workstation ~]$ oc expose svc/hello \
--hostname=hello.apps.lab.example.com
route "node-hello" exposed
```

- 6.2. Access the route URL `hello.apps.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.apps.lab.example.com
Hi! I am running on host -> hello-2-25bts
```

7. 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.

8. 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 command may create a service, a deployment configuration, and a build configuration if source code is used.
- 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 12

CONTROLLING ACCESS TO OPENSHIFT RESOURCES

GOAL

Control access to OpenShift resources.

OBJECTIVES

- 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

- 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 Red Hat 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"
```

INTRODUCING ROLES IN RED HAT OPENSHIFT CONTAINER PLATFORM

Roles apply various levels of access and policies, which includes both cluster and local policies. Users and groups can be associated with multiple roles at the same time. Run the **oc describe** command to view details about the roles and their bindings.

Users with the **cluster-admin** default role in the cluster policy can view cluster policy and all local policies. Users with the **admin** default role in a given local policy can view the policy on a per-project basis.

To view the current set of cluster bindings, which show the users and groups that are bound to various roles, run the following command:

```
[root@demo ~]# oc describe clusterPolicyBindings :default
Name:                      :default
Created:                  7 days ago
Labels:                    <none>
Annotations:              <none>
Last Modified:            2017-11-14 16:46:42 -0800 PST
Policy:                   <none>
RoleBinding[admin]:
    Role:      admin
    Users:    <none>
    Groups:   <none>
```

```
ServiceAccounts: openshift-infra/template-instance-
controller Subjects: <none>
...
...
```

Reading Local Policies

Although the list of local roles and their associated rule sets are not viewable within a local policy, all of the default roles are still applicable and can be added to users or groups, except the **cluster-admin** default role. However, the local bindings are viewable.

To view the current set of local bindings, which shows the users and groups that are bound to various roles, run the following command:

```
[root@demo ~]# oc describe policyBindings :default
Name: :default
Namespace: project-review1
Created: 3 days ago
Labels: <none>
Annotations: <none>
Last Modified: 2017-11-13 11:52:47 -0800 PST
Policy: <none>
RoleBinding[admin]:
  Role: admin
  Users: developer
  Groups: <none>
  ServiceAccounts: <none>
  Subjects: <none>
RoleBinding[system:deployers]:
  Role: system:deployer
  Users: <none>
  Groups: <none>
  ServiceAccounts: deployer
  Subjects: <none>
...
...
```



NOTE

By default, in a local policy, only the binding for the **admin** role is immediately listed. However, if other default roles are added to users and groups within a local policy, they are listed as well.

Managing Role Bindings

Adding, or *binding*, a role to users or groups gives the user or group the relevant access granted by the role. You can add and remove roles to and from users and groups using **oc adm policy** commands.

When managing the user and group roles for a local policy using the following operations, a project may be specified with the **-n** option. If it is not specified, the current project is used.

The following table shows some of the operations for managing local policies:

COMMAND	DESCRIPTION
<code>oc adm policy who-can <i>verb resource</i></code>	Indicates which users can perform an action on a resource.
<code>oc adm policy add-role-to-user <i>role username</i></code>	Binds a given role to specified users.
<code>oc adm policy remove-role-from-user <i>role username</i></code>	Removes a given role from specified users.
<code>oc adm policy remove-user <i>username</i></code>	Removes specified users and all of their roles.
<code>oc adm policy add-role-to-group <i>role groupname</i></code>	Binds a given role to specified groups.
<code>oc adm policy remove-role-from-group <i>role groupname</i></code>	Removes a given role from specified groups.
<code>oc adm policy remove-group <i>groupname</i></code>	Removes specified groups and all of their roles.

You can also manage role bindings for the cluster policy using the operations described below. The `-n` option is not used for these operations because the cluster policy does not operate at the namespace level.

The following table shows some of the operations for managing cluster policies:

COMMAND	DESCRIPTION
<code>oc adm policy add-cluster-role-to-user <i>role username</i></code>	Binds a given role to specified users for all projects in the cluster.
<code>oc adm policy remove-cluster-role-from-user <i>role username</i></code>	Removes a given role from specified users for all projects in the cluster.
<code>oc adm policy add-cluster-role-to-group <i>role groupname</i></code>	Binds a given role to specified groups for all projects in the cluster.
<code>oc adm policy remove-cluster-role-from-group <i>role groupname</i></code>	Removes a given role from specified groups for all projects in the cluster.



NOTE

The `oc policy` command applies to the current project, whereas the `oc adm policy` command applies to cluster-wide operations, even if there are some overlaps.

For example, to give the `admin` role to the `developer` user in the `example` project, run the following command:

```
[root@demo ~]# oc adm policy add-role-to-user admin developer -n example
```

Run the `oc describe policyBindings :default -n project` command to review the binding:

```
[root@demo ~]# oc describe policyBindings :default -n example
Name:          :default
Created:       5 minutes ago
Labels:        <none>
Last Modified: 2015-06-10 22:00:44 +0000 UTC
Policy:        <none>
RoleBinding[admins]:
    Role:      admin
        Users: [developer] ①
        Groups: []
RoleBinding[system:deployers]:
    Role:      system:deployer
        Users: [system:serviceaccount:developer:deployer]
        Groups: []
RoleBinding[system:image-builders]:
    Role:      system:image-builder
        Users: [system:serviceaccount:developer-project:builder]
        Groups: []
RoleBinding[system:image-pullers]:
    Role:      system:image-puller
        Users: []
        Groups: [system:serviceaccounts:developer-project]
```

- ① The `alice` user has the `admin` role binding.

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.



NOTE

SCCs are covered in depth in later sections of this chapter

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, OpenShift creates the user object.

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**.

Figure 12.1: 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 Red Hat OpenShift Container Platform is an entity that can make requests to the OpenShift API. Typically, this represents the account of a developer or administrator that is interacting with OpenShift.

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.

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 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, **user1**.

In the existing terminal window, run the following command:

```
[root@master ~]# htpasswd -b /etc/origin/master/htpasswd user1 redhat
```

- 1.3. Create the developer user, **user2**.

In the terminal window, run the following command:

```
[root@master ~]# htpasswd -b /etc/origin/master/htpasswd 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
```

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/D0285/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:oauth
cluster role "self-provisioner" removed: "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
Login successful.
```

You don't have any projects. Contact your system administrator to request a project.

- 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 \
```

```
https://master.lab.example.com
```

- 4.2. Create a new project.

Run the following command:

```
[student@workstation ~]$ oc new-project project-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 project-user2
```

- 5. Associate **user1** with project-user1 and **user2** with both project-user1 and project-user2.

- 5.1. Add **user1** as an administrator for the project-user1 project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc project project-user1
Now using project "project-user1" on server "https://master.lab.example.com:443".
[student@workstation ~]$ oc policy add-role-to-user admin user1
role "admin" added: "user1"
```

- 5.2. Add **user2** as a developer for the project-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 project-user2 project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc project project-user2
Now using project "project-user2" on server "https://master.lab.example.com:443".
[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 project-user1 project.

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
```

- 6.2. Enter the project-user1 project.

From the terminal window, run the following command. It should work:

```
[student@workstation ~]$ oc project project-user1
```

- 6.3. Enter the project-user2 project.

From the terminal window, run the following command. It should fail:

```
[student@workstation ~]$ oc project project-user2
error: You are not a member of project "project-user2".
You have one project on this server: project-user1
```

- 6.4. Verify that **user2** can access both project-user1 and project-user2 projects.

Log in to OpenShift as **user2**.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u user2 -p redhat
Login successful.
```

You have access to the following projects and can switch between them with 'oc project <projectname>':

```
* project-user1
  project-user2
```

Using project "project-user1".

- 6.5. Enter the project-user1 project.

From the terminal window, run the following command. It should work:

```
[student@workstation ~]$ oc project project-user1
Already on project "project-user1" on server "https://master.lab.example.com:443".
```

- 6.6. Enter the project-user2 project

From the terminal window, run the following command. It should work:

```
[student@workstation ~]$ oc project project-user2
Now using project "project-user2" on server "https://master.lab.example.com:443".
```

- 7. As a developer user, deploy to project-user1 an application that requires elevated privileges.

- 7.1. Log in as **user2** and enter the project-user1 project:

```
[student@workstation ~]$ oc login -u user2 -p redhat
[student@workstation ~]$ oc project project-user1
```

- 7.2. Run the following command:

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

```
... output omitted ...
* WARNING: Image "registry.lab.example.com/nginx:latest" runs as the 'root'
user which may not be permitted by your cluster administrator
... output omitted ...
```

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/D0285/labs/secure-resources** folder.

8.1. Log in as the **user1** user and select the **project-user1** project.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc login -u user1 -p redhat
Login successful.
```

```
You have one project on this server: "project-user1"
```

```
Using project "project-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
serviceaccount "userroot" created
```

8.3. Log in as the **admin** user and select the **project-user1** project.

From the terminal window, run the following commands:

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

```
[student@workstation ~]$ oc project project-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  
scc "anyuid" added to: ["system:serviceaccount:project-user1:userroot"]
```

- 8.5. Log in as the **user2** user and select the **project-user1** project.

From the terminal window, run the following commands:

```
[student@workstation ~]$ oc login -u user2 -p redhat  
[student@workstation ~]$ oc project project-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/D0285/labs/secure-resources** folder.

```
[student@workstation ~]$ oc patch dc/nginx --patch \  
'{"spec": {"template": {"spec": {"serviceAccountName": "userroot"}}}}'  
deploymentconfig "nginx" patched
```

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 pod is created using the updated deployment configuration.

Wait until the new pod status is 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
```

```
route "nginx" exposed
```

9.2. Connect to the **workstation** VM.

From the terminal window, run the following command:

```
[student@workstation ~]$ curl -s \
http://nginx-project-user1.apps.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/D0285/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
cluster role "self-provisioner" added: ["system:authenticated"
"system:authenticated:oauth"]
```

10.2. Delete the projects.

Run the following commands:

```
[student@workstation ~]$ oc delete project project-user1
[student@workstation ~]$ oc delete project project-user2
```

10.3. Delete the users.

Run the following commands:

```
[student@workstation ~]$ ssh root@master
[root@master ~]$ htpasswd -D /etc/origin/master/htpasswd user1
[root@master ~]$ htpasswd -D /etc/origin/master/htpasswd 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 12.2: 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 objects 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. **ConfigMap** 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 web console interface. At the top, there's a header with a home icon, the project name "demo_project", an "Add to project" button, a help icon, and a user profile "developer". The left sidebar has navigation links for "Overview", "Applications", "Builds", and "Resources", with "Resources" being the active tab. The main content area is titled "Config Maps" with a "Learn More" link and a "Create Config Map" button. It includes a search/filter bar labeled "Filter by label" and an "Add" button. A table header with columns "Name", "Created", and "Labels" is shown, followed by a message "No config maps to show".

Figure 12.3: Managing ConfigMaps using the web console



REFERENCES

Further information is available in the Developer Guide of the OpenShift Container Platform 3.9 at
https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9

Further information is also available at
OpenShift Container Platform 3.9 Documentation
<https://docs.openshift.com/container-platform/3.9/>

► GUIDED EXERCISE

PROTECTING A DATABASE PASSWORD

In this exercise, you will use a secret to encrypt the credentials to access a database container.

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

► 1. Create a new project.

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

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

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

- 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 ~/D0285/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='do285-admin'
secret "mysql" created
```

- 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: ZG8y0DATYWRTaw4=
  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
```

Forwarding from 127.0.0.1:3306 -> 3306



IMPORTANT

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 -pdo285-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 **do285-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.

RED HAT OPENSHIFT CONTAINER PLATFORM AUTHORIZATION

Red Hat OpenShift Container Platform 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 are 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 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
```

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 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 a 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 security context constraints (SCCs) 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
```

OPENSFIFT 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: restricted❶
  priority: null
  readOnlyRootFilesystem: false
  requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT
- SETUID
- SETGID
  runAsUser:
    type: MustRunAsRange
  seLinuxContext:❷
    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

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

► 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.

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 the chapter *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-prepare setup  
[student@workstation ~]$ cd /home/student/do285-ansible  
[student@workstation do285-ansible]$ ./install.sh
```

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
```

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 **do285-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 and list the databases.
 - 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 **registry.lab.example.com** registry which is a secure 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/D0285/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.apps.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 and from OpenShift by using the **oc delete user** command.

This concludes the lab.

► SOLUTION

CONTROLLING ACCESS TO OPENSHIFT RESOURCES

In this lab, you will enable users to access a project and deploy a MySQL database container using secrets.

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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

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/master/htpasswd 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
```

If prompted, accept the insecure connection.

- 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
role "edit" added: "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 ~/D0285/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 **do285-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='do285-admin'
secret "mysql" created
```

- 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
```

```
Forwarding from 127.0.0.1:3306 -> 3306
```

- 9.2. Access the container using the **mysql** command as the **mysql** user with the **redhat** password and list the databases.

Open another command line window to execute the following command, that connect to the database and list the databases:

```
[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 **registry.lab.example.com** registry which is a secure 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=registry.lab.example.com/phpmyadmin/phpmyadmin:4.7 \
-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  
serviceaccount "phpmyadmin-account" created
```

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  
scc "anyuid" added to: ["system:serviceaccount:secure-review: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/D0285/labs/secure-review** folder.

Run the following command:

```
[student@workstation ~]$ oc patch dc/phpmyadmin --patch \  
'{"spec": {"template": {"spec": {"serviceAccountName": "phpmyadmin-account"}}}}'  
deploymentconfig "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.apps.lab.example.com.

From the terminal window, run the following command:

```
[student@workstation ~]$ oc expose svc/phpmyadmin \
```

```
--hostname=phpmyadmin.apps.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.apps.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 and from OpenShift by using the **oc delete user** command.

Run the following command:

```
[student@workstation ~]$ ssh root@master htpasswd -D \
/etc/origin/master/htpasswd user-review
[student@workstation ~]$ oc delete user 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 13

ALLOCATING PERSISTENT STORAGE

GOAL

Implement persistent storage.

OBJECTIVES

- Provision persistent storage for use by applications.
- Describe how persistence is configured for the internal container registry.

SECTIONS

- Provisioning Persistent Storage (and Guided Exercise)
- Describing Persistence for the Internal Registry (and Quiz)

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
- Azure Disk and Azure File
- FlexVolume (allows for the extension of storage back-ends that do not have a built-in plug-in)
- VMWare vSphere
- Dynamic Provisioning and Creating Storage Classes
- Volume Security
- Selector-Label Volume Binding

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 persistent volume'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 persistent volume'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 a node VM:

```
[root@node ~]# showmount -e
Export list for master.lab.example.com:
/var/export/nfs-demo *
```

From the NFS server on the **services** VM:

```
[root@services ~]# cat /etc/exports.d/nfs-demo.conf
/var/export/nfs-demo
...
[root@services ~]# 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 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 preallocated 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.

OUTCOMES

You should be able to configure an NFS share on the OpenShift **services** VM to provide storage for OpenShift nodes, and OpenShift persistent volumes bound to a database pod.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 1. Configure an NFS share on the **services** VM. This share is used as an OpenShift Container Platform persistent volume.
 - 1.1. Log in to the **services** VM as the **root** user:

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

- 1.2. The **config-nfs.sh** script is available to automate the configuration of the NFS share for the persistent volume. The script is located at **/root/D0285/labs/deploy-volume/config-nfs.sh**. Examine the script to become familiar with the process for configuring the NFS share:

```
[root@services ~]$ less -FiX /root/D0285/labs/deploy-volume/config-nfs.sh
... output omitted ...
```

- 1.3. Run the script:

```
[root@services ~]$ /root/D0285/labs/deploy-volume/config-nfs.sh
```

- 1.4. Verify that the **/var/export/dbvol** share is included in the export list on the OpenShift **services** VM.

```
[root@services ~]# showmount -e
Export list for services.lab.example.com:
... output omitted ...
/var/export/dbvol *
```

1.5. Log out of the **services** VM:

```
[root@services ~]# exit
[student@workstation ~]$
```

- 2. Verify that both the **node1** and **node2** hosts can access the NFS exported volume from the **services** 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 **services** VM:

```
[root@node1 ~]# mount -t nfs services.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 31 11:03 .
dr-xr-xr-x. 17 root      root    224 Jul 26 14:24 ..
services.lab.example.com:/var/export/dbvol on /mnt \
type nfs4 (rw,relatime,vers=4.1,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=172.25.250.13)
```

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 the **root** user:

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

- 2.7. Confirm that the **node2** host can access the NFS share on the OpenShift **services** VM:

```
[root@node2 ~]# mount -t nfs services.lab.example.com:/var/export/dbvol /mnt
```

- 2.8. Verify that the file system has the correct permissions from the **node2** host:

```
[root@node2 ~]# ls -la /mnt ; mount | grep /mnt
total 0
drwx----- 2 nfsnobody nfsnobody 6 Jul 31 11:03 .
dr-xr-xr-x. 17 root      root    224 Jul 26 14:27 ..
services.lab.example.com:/var/export/dbvol on /mnt \
type nfs4 (rw,relatime,vers=4.1,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=172.25.250.13)
```

- 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. If prompted, accept the insecure connection.

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

- 3.2. Inspect the persistent volume definition:

```
[student@workstation ~]$ less -FiX \
~/DO285/labs/deploy-volume/mysqlDb-volume.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysqlDb-volume
spec:
  capacity:
    storage: 3Gi
```

```

accessModes:
- ReadWriteMany
nfs:
  path: /var/export/dbvol
  server: services.lab.example.com
persistentVolumeReclaimPolicy: Recycle

```

- 3.3. Create the persistent volume using the provided YAML resource definition file:

```
[student@workstation ~]$ oc create -f \
~/D0285/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          Recycle        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 **redhat** as the password.

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

- 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=registry.lab.example.com/rhscl/mysql-57-rhel7 \
-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
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/mysqldb'
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:443

svc/mysqldb - 172.30.183.123:3306
dc/mysqldb deploys istag/mysqldb:latest
    deployment #1 deployed 22 seconds ago - 1 pod

2 infos identified, use 'oc status -v' to see details.
```

- 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:  EmptyDir (a temporary directory that shares a pod's lifetime)
  -
Volumes:
  mysqldb-volume-1:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
the same namespace)
    ClaimName:  mysqldb-pvclaim
    ReadOnly:   false
    default-token-h2dp4:
  -
Volumes:
  deployer-token-6wwsd:
```

Type: Secret (a volume populated by a Secret)

- 7. On the **workstation** host, verify that the persistent volume claim **mysqldb-pvclaim** is bound to the persistent volume named **mysqldb-volume** by using the **oc get pvc** command.

```
[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/D0285/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
mysqldb-2-k2v1h  1/1     Running   0          1h
```

```
[student@workstation ~]$ oc port-forward mysqldb-2-k2v1h 3306:3306
Forwarding from 127.0.0.1:3306 -> 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/D0285/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 should be:

```
+-----+
| count(*) |
+-----+
|      3 |
+-----+
```

- 8.4. Verify that the MySQL server has initialized a database in the exported folder.

```
[student@workstation ~]$ ssh root@services ls -la /var/export/dbvol/
total 41036
drwx----- 6 nfsnobody nfsnobody    4096 Jul 31 11:11 .
drwxr-xr-x  3 root      root        19 Jul 31 11:03 ..
-rw-r----- 1 nfsnobody nfsnobody     56 Jul 31 11:10 auto.cnf
```

```
-rw----- 1 nfsnobody nfsnobody      1676 Jul 31 11:10 ca-key.pem
-rw-r--r-- 1 nfsnobody nfsnobody     1075 Jul 31 11:10 ca.pem
-rw-r--r-- 1 nfsnobody nfsnobody     1079 Jul 31 11:10 client-cert.pem
-rw----- 1 nfsnobody nfsnobody     1680 Jul 31 11:10 client-key.pem
-rw-r----- 1 nfsnobody nfsnobody      352 Jul 31 11:11 ib_buffer_pool
-rw-r----- 1 nfsnobody nfsnobody 12582912 Jul 31 11:13 ibdata1
-rw-r----- 1 nfsnobody nfsnobody    8388608 Jul 31 11:13 ib_logfile0
-rw-r----- 1 nfsnobody nfsnobody    8388608 Jul 31 11:10 ib_logfile1
-rw-r----- 1 nfsnobody nfsnobody 12582912 Jul 31 11:11 ibtmp1
drwxr-x--- 2 nfsnobody nfsnobody     4096 Jul 31 11:10 mysql
-rw-r----- 1 nfsnobody nfsnobody        2 Jul 31 11:11 mysql_db-2-4lvs8.pid
drwxr-x--- 2 nfsnobody nfsnobody     8192 Jul 31 11:10 performance_schema
-rw----- 1 nfsnobody nfsnobody     1676 Jul 31 11:10 private_key.pem
-rw-r--r-- 1 nfsnobody nfsnobody     452 Jul 31 11:10 public_key.pem
drwxr-x--- 2 nfsnobody nfsnobody      54 Jul 31 11:13 quotes
-rw-r--r-- 1 nfsnobody nfsnobody     1079 Jul 31 11:10 server-cert.pem
-rw----- 1 nfsnobody nfsnobody     1676 Jul 31 11:10 server-key.pem
drwxr-x--- 2 nfsnobody nfsnobody     8192 Jul 31 11:10 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@services ls -la /var/export/dbvol/quotes
total 212
drwxr-x--- 2 nfsnobody nfsnobody      54 Jul 31 11:13 .
drwx----- 6 nfsnobody nfsnobody    4096 Jul 31 11:11 ..
-rw-r----- 1 nfsnobody nfsnobody     65 Jul 31 11:10 db.opt
-rw-r----- 1 nfsnobody nfsnobody    8584 Jul 31 11:13 quote.frm
-rw-r----- 1 nfsnobody nfsnobody   98304 Jul 31 11:13 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, terminate 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** host, 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
```

```
[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 **services** VM. This shows that the files remain even after the PV has been deleted.

```
[student@workstation ~]$ ssh root@services 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-----. 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 **services** VM, and then delete the contents of the NFS shared **/var/export/dbvol/** directory:

```
[student@workstation ~]$ ssh root@services rm -rf /var/export/dbvol/*
```

Verify that the files have been deleted:

```
[student@workstation ~]$ ssh root@services 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 ..
```

- 9.5. From the **workstation** VM, run the **lab deploy-volume cleanup** command to remove the NFS share.

```
[student@workstation ~]$ lab deploy-volume cleanup
```

This concludes the guided exercise.

DESCRIBING PERSISTENCE FOR THE INTERNAL REGISTRY

OBJECTIVE

After completing this section, students should be able to describe how persistence is configured for the internal container registry.

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, which can then 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 persistent registry, which uses NFS shares as defined by the set of `openshift_hosted_registry_storage_*` variables in the inventory file. 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 advanced installer configures the NFS sever to use persistent storage on an external NFS server, as defined in the inventory file.

The NFS server defined in the `[nfs]` group contains the list of NFS servers. The server is used in conjunction with the set of `openshift_hosted_registry_storage*` variables to configure the NFS server.

```
[OSEv3:vars]
openshift_hosted_registry_storage_kind=nfs①
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']②
openshift_hosted_registry_storage_nfs_directory=/exports③
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'④
openshift_hosted_registry_storage_volume_name=registry⑤
openshift_hosted_registry_storage_volume_size=40Gi⑥
... output omitted ...

[nfs]
services.lab.example.com
```

- ① Defines the storage back-end used by Red Hat OpenShift Container Platform. In this case, NFS.
- ② Defines the access mode for the volume. Uses **ReadWriteMany** by default, which allows the volume to be mounted as read and write by many nodes.
- ③ Defines the NFS storage directory on the NFS server.
- ④ Defines the NFS options for the storage volume. The options are added to the `/etc/exports.d/openshift-ansible.exports` file. The `rw` option allows read and write

access to the NFS volume. The **root_squash** option prevents **root** users connected remotely from having **root** privileges and assigns them the user ID for the **nfsnobody**.

- ⑤ Defines the name of the NFS directory to use for the persistent registry.
- ⑥ Defines the size for the persistent volume.

After the installation and configuration of the storage for the persistent registry, OpenShift creates a persistent volume in the **openshift** project named **registry-volume**. The persistent volume has a capacity of 40 GB and a policy of **Retain**, as set by the definition.

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY
registry-volume	40Gi	RWX	Retain
STATUS	CLAIM		STORAGECLASS REASON AGE
Bound	default/registry-claim		

A **registry-claim** persistent volume claim in the **default** project claims the persistent volume.

NAME	STATUS	VOLUME	CAPACITY
registry-claim	Bound	registry-volume	40Gi
ACCESS MODES	STORAGECLASS	AGE	
RWX		16h	

The following listing illustrates the persistent volume definition that the persistent registry uses.

```
Name: registry-volume①
Labels: <none>
Annotations: pv.kubernetes.io/bound-by-controller=yes
StorageClass:
Status: Bound
Claim: default/registry-claim②
Reclaim Policy: Retain③
Access Modes: RWX④
Capacity: 40Gi⑤
Message:
Source: ⑥
Type: NFS (an NFS mount that lasts the lifetime of a pod)
Server: services.lab.example.com
Path: /exports/registry
ReadOnly: false
Events: <none>
```

- ① Defines persistent volume name. This value can be used to identify the persistent volume when using commands such as **oc get pv volume-name**.
- ② Defines the claim that utilizes the persistent volume.
- ③ **Retain** is the default persistent volumes policy. Volumes with a policy of **Retain** are not scrubbed after being released from their claim (that is, when the persistent volume claim bound is deleted).
- ④ Defines the access mode for the persistent volume, as defined by the **openshift_hosted_registry_storage_access_modes=['ReadWriteMany']** variable of the Ansible inventory file.

- ⑤ Defines the size of the persistent volume as defined by the **openshift_hosted_registry_storage_volume_size** variable of the Ansible inventory file.
- ⑥ Defines the location and the NFS share of the storage back-end.

Run the following command to confirm that the OpenShift internal registry is configured to use the volume **registry-volume** with the default **PersistentVolumeClaim** volume type:

```
[user@demo ~] oc describe dc/docker-registry | grep -A4 Volumes
Volumes:
  registry-storage:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the
          same namespace)
      ClaimName: registry-claim
      ReadOnly: false
```

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 the classroom environment the NFS share is configured on the **services** VM, therefore **ssh** can be used to remotely access the NFS share to verify that the OpenShift internal registry stores images to the persistent storage.

Assuming there is an application named **hello** running in the default namespace, the following command verifies that images are 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/

► QUIZ

DESCRIBING PERSISTENCE FOR THE INTERNAL REGISTRY

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 of the following Ansible variable defines the storage back-end to use for the integrated registry?**
 - a. openshift_hosted_registry_nfs_backend
 - b. openshift_hosted_registry_storage_kind
 - c. openshift_integrated_registry_storage_type

- ▶ **2. Which of the two following objects get created by the advanced installer for the integrated registry storage? (Choose two.)**
 - a. An image stream.
 - b. A persistent volume claim.
 - c. A storage class.
 - d. A persistent volume.
 - e. A deployment configuration.

- ▶ **3. Which of the following Ansible variables creates the persistent volume with an access mode of RWX?**
 - a. openshift_set_hosted_rwx
 - b. openshift_integrated_registry_nfs_option
 - c. openshift_hosted_registry_storage_access_modes
 - d. openshift_hosted_registry_storage_nfs_options

- ▶ **4. Which of the following command allows you to verify proper usage of a storage back-end for the registry persistence?**
 - a. oc describe dc/docker-registry | grep -A4 Volumes
 - b. oc describe pvc storage-registry | grep nfs
 - c. oc describe sc/docker-registry
 - d. oc describe pv docker-persistent

► SOLUTION

DESCRIBING PERSISTENCE FOR THE INTERNAL REGISTRY

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 of the following Ansible variable defines the storage back-end to use for the integrated registry?**
 - a. openshift_hosted_registry_nfs_backend
 - b. openshift_hosted_registry_storage_kind
 - c. openshift_integrated_registry_storage_type

- ▶ **2. Which of the two following objects get created by the advanced installer for the integrated registry storage? (Choose two.)**
 - a. An image stream.
 - b. A persistent volume claim.
 - c. A storage class.
 - d. A persistent volume.
 - e. A deployment configuration.

- ▶ **3. Which of the following Ansible variables creates the persistent volume with an access mode of RWX?**
 - a. openshift_set_hosted_rwx
 - b. openshift_integrated_registry_nfs_option
 - c. openshift_hosted_registry_storage_access_modes
 - d. openshift_hosted_registry_storage_nfs_options

- ▶ **4. Which of the following command allows you to verify proper usage of a storage back-end for the registry persistence?**
 - a. oc describe dc/docker-registry | grep -A4 Volumes
 - b. oc describe pvc storage-registry | grep nfs
 - c. oc describe sc/docker-registry
 - d. oc describe pv docker-persistent

▶ 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.

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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- From the **services** VM, run the script located at `/root/D0285/labs/storage-review/config-review-nfs.sh` to configure the NFS share `/var/export/review-dbvol` that is used in this lab for the OpenShift 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/D0285/labs/storage-review/review-volume-pv.yaml` file.
- Make sure you are logged in as the OpenShift **admin** user in the **openshift** namespace. Import the instructor application template from `/home/student/D0285/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.

- On the **workstation** host, log in to OpenShift as the **developer** user, using **redhat** as the password, then create a new project named **instructor**.
- From **workstation**, access the OpenShift web console at `https://master.lab.example.com`. Log in to OpenShift as the **developer** user with a password

of **redhat**. Select the instructor project, then browse the catalog of OpenShift templates. Select The Instructor Application Template PHP template.

Select the **The Instructor Application Template** application template, then add `instructor.apps.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/D0285/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/D0285/labs/storage-review/instructor.sql
```

7. Access the application, available at `http://instructor.apps.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

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.

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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- From the **services** VM, run the script located at **/root/D0285/labs/storage-review/config-review-nfs.sh** to configure the NFS share **/var/export/review-dbvol** that is used in this lab for the OpenShift persistent volume.
 - Log in to the **services** VM as the **root** user:

```
[student@workstation ~]$ ssh root@services
[root@services ~]#
```

- View the script at **/root/D0285/labs/storage-review/config-review-nfs.sh** to become familiar with the process of configuring the NFS share before running the script:

```
[root@services ~]# less -FiX /root/D0285/labs/storage-review/config-review-nfs.sh
... output omitted ...
```

- Run the script to configure the NFS share:

```
[root@services ~]# /root/D0285/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@services ~]# showmount -e  
Export list for services.lab.example.com:  
... output omitted ...  
/var/export/review-dbvol *  
... output omitted ...
```

- 1.5. Log out of the **services** VM.

```
[root@services ~]# 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/D0285/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
```

- 2.2. Inspect the contents of the **/home/student/D0285/labs/storage-review/review-volume-pv.yaml** for correct entries of PV attributes and the NFS share:

```
[student@workstation ~]$ less -FiX \  
/home/student/D0285/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: services.lab.example.com  
  persistentVolumeReclaimPolicy: Recycle
```

- 2.3. Use **oc create** command against the persistent volume definition to create a persistent volume named **review-pv**:

```
[student@workstation ~]$ oc create -f \  
/home/student/D0285/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. Import the instructor application template from **/home/student/D0285/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/D0285/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/D0285/labs/storage-review/instructor-template.yaml** instructor template file as the **admin** user.

```
[student@workstation ~]$ oc create -n openshift -f \
/home/student/D0285/labs/storage-review/instructor-template.yaml
template "instructor" created
```

4. On the **workstation** host, log in to OpenShift as the **developer** user, using **redhat** 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 redhat \
https://master.lab.example.com
```

- 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`. Log in to OpenShift as the **developer** user with a password

of **redhat**. Select the instructor project, then browse the catalog of OpenShift templates. Select The Instructor Application Template PHP template.

Select the **The Instructor Application Template** application template, then add `instructor.apps.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`. If prompted, accept the security certificate) and log in as the **developer** user, using **redhat** as the password.

Click the instructor project to access the project.

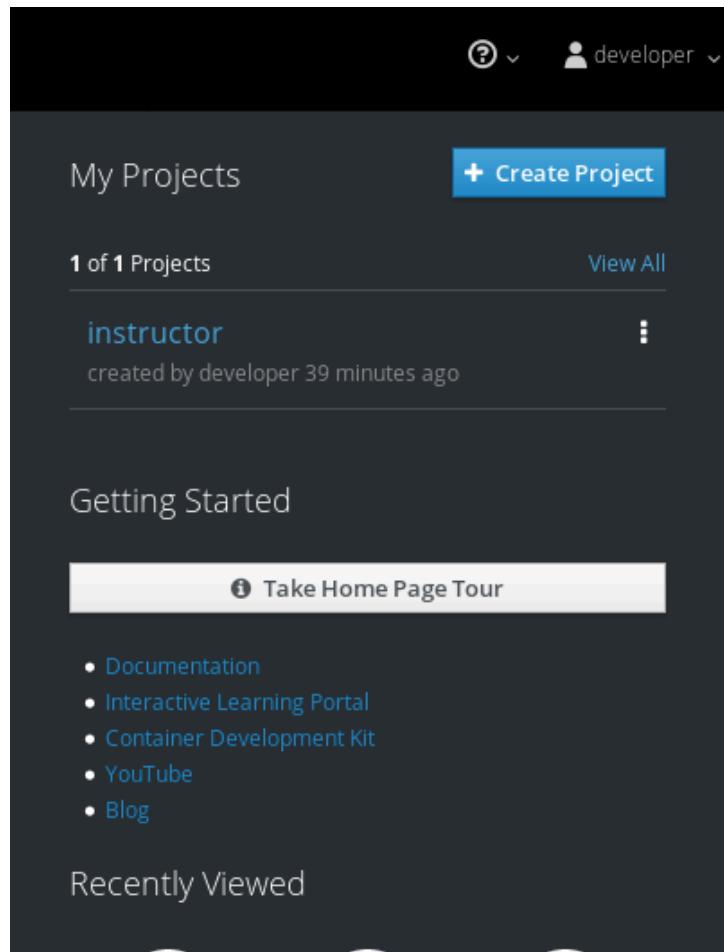


Figure Error.1: Selecting projects

- 5.2. Click Browse Catalog to view the available components.

Get started with your project.

Add content to your project from the catalog of web frameworks, databases, and other components. You may also deploy an existing image, create or replace resources from their YAML or JSON definitions, or select an item shared from another project.



Figure Error.2: Selecting templates

- 5.3. From the catalog, click the Languages tab and select PHP.

Click the The Instructor Application Template template to start editing the template properties and then click Next to move to the next stage

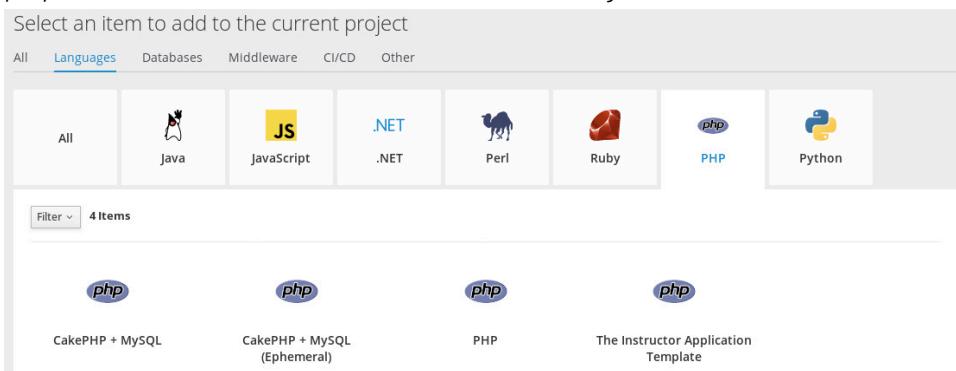


Figure Error.3: Selecting application templates



NOTE

During the testing of this lab, we found that sometimes the template is not visible to the **developer** user. If this is the case, login as the **admin** user and try again. It is safe to proceed with the following lab steps as **admin**.

- 5.4. All the default values populated by the template are used, however, the Application Hostname field must be updated. Enter `instructor.apps.lab.example.com` as the value, and then click Next to move to the next stage.

On the next screen, leave the default value and click Create to create the application.

The screenshot shows the 'The Instructor Application Template' configuration dialog. The 'Information' tab is selected. It contains fields for 'Application Hostname' (set to 'instructor.apps.lab.example.com') and 'GitHub Webhook Secret'. At the bottom right, there are 'Cancel', '< Back', and 'Next >' buttons. The 'Next >' button is highlighted in blue.

Figure Error.4: Setting the application hostname

- 5.5. Click Continue to project overview to monitor the application as it builds. From the Provisioned Services frame, click `instructor`.

Click the drop-down arrow next to the **instructor**, #1 entry of the deployment configuration to open the deployment panel. When the build completes, a green check mark next to Complete should appear for the Builds section. The application route,

<http://instructor.apps.lab.example.com>, displays in the Networking section.

Other Resources

Figure Error.5: Project overview

Figure Error.6: Project overview

- 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/D0285/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/D0285/labs/storage-review/instructor.sql
```

- Make sure you are logged in to OpenShift as the **developer** user:

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

- 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
```

- 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/D0285/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. Press **Ctrl+C** to close the port forward connection in the first terminal.

7. Access the application, available at <http://instructor.apps.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
```

- 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 **services** VM to delete the database files.

From **workstation**, log in to the **services** VM as the **root** user:

```
[student@workstation ~]$ ssh root@services
[root@services ~]#
```

Delete the database and NFS files, and then verify that the files have been removed:

```
[root@services ~]# rm -rf /var/export/review-dbvol
[root@services ~]# ls -la /var/export/review-dbvol/
ls: cannot access /var/export/review-dbvol/: No such file or directory
[root@services ~]# rm -f /etc/exports.d/review-dbvol.exports
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Red Hat OpenShift Container Platform 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.
- A set of Ansible variables allow the configuration of an external NFS storage for the OpenShift default registry. This creates a persistent volume and an persistent volume claim.

CHAPTER 14

MANAGING APPLICATION DEPLOYMENTS

GOAL

Manipulate resources to manage deployed applications.

OBJECTIVES

- 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

- 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.
- ❸** A template for pods that the controller creates.
- ❹** 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
frontend  DeploymentConfig/myapp/frontend/scale  80%      59%      1        10
8d
```

```
$ oc describe hpa/frontend
Name:                      frontend
Namespace:                 myapp
Labels:                    <none>
CreationTimestamp:         Mon, 26 Jul 2018 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.

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://registry.lab.example.com/scaling
Application URL	http://scaling.apps.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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

► 1. Create a new project.

- 1.1. Log in to OpenShift as the **developer** user:

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

- 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://registry.lab.example.com/scaling > ~/scaling.yml
```

- 2.2. Open the YAML resource definition file in a text editor:

```
[student@workstation ~]$ vi ~/scaling.yml
```

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-1-0l8wr  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.apps.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>. Use **developer** as the user name and **redhat** as the password.
 - 4.2. Under My Projects on the right side of the page, click **scaling**.

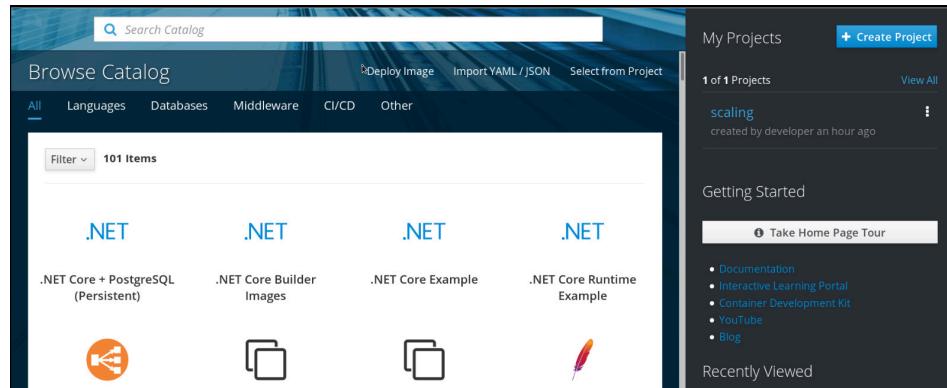


Figure 14.1: The scaling project

- 4.3. Select the drop-down menu to the left of scaling, #1 to access the three application pods

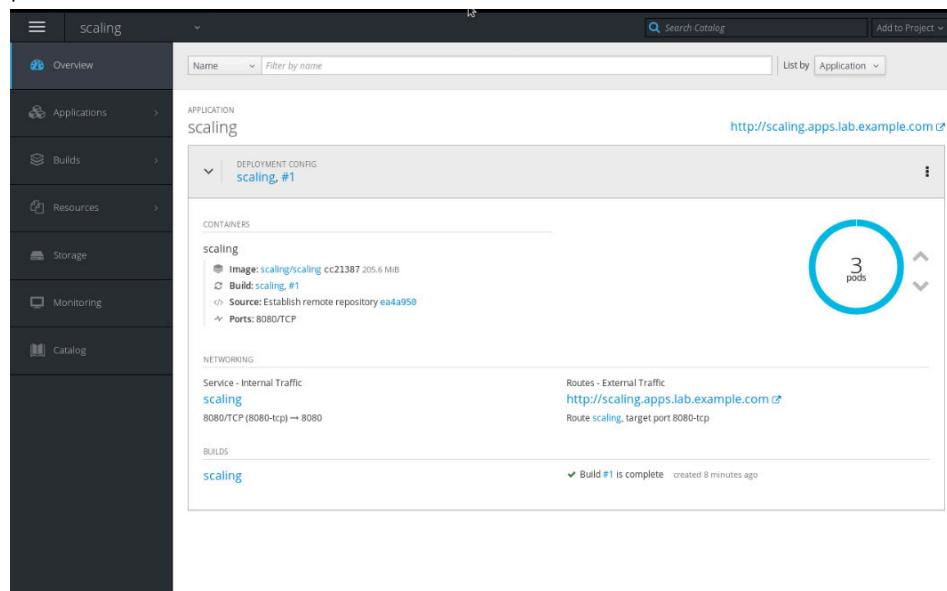


Figure 14.2: Project overview page

- 4.4. Navigate to Applications → Pods in the left navigation pane to view the pods.

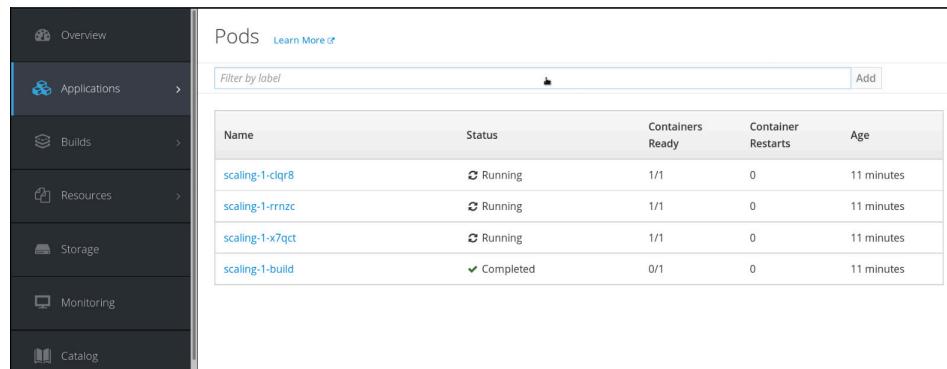


Figure 14.3: Application pods

Click one of the pod names to display details about the pod, including its internal IP address.

Figure 14.4: Pod details

- 4.5. 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
```

- 5. 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.apps.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.

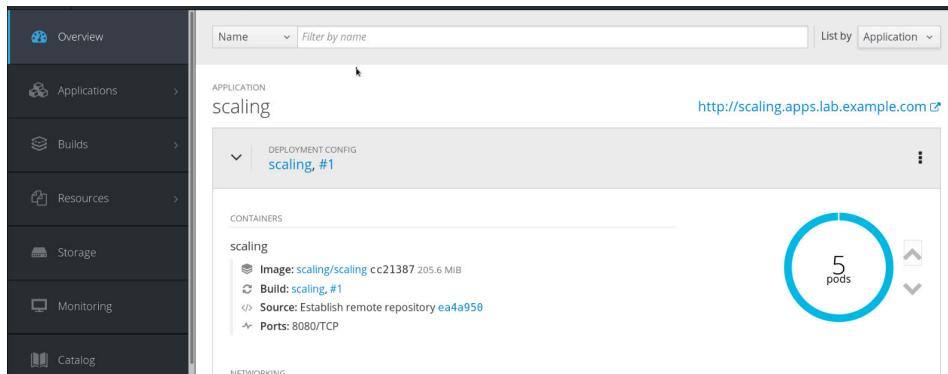


Figure 14.5: Scaling pods



NOTE

As the number of pods scales up or down, you should see the blue donut in the Overview page change accordingly.

- 6.4. 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
```

- 6.5. 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.apps.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 3.9 supports the common data center concepts of *zones* and *regions* by using node labels, affinity rules, and anti-affinity rules.

In previous releases of Red Hat OpenShift Container Platform the installer marked the master hosts as unscheduled which would not allow new pods to be placed on the hosts. However, with the release of Red Hat OpenShift Container Platform 3.9 masters are marked schedulable automatically during installation and upgrade. This results in the web console being able to run as a pod deployed to the master instead of being run as a component of the master. The default node selector is set by default during installations and upgrades. It is set to **node-role.kubernetes.io/compute=true** unless overridden using the **osm_default_node_selector** Ansible variable.

The following automatic labeling occurs for hosts defined in your inventory file during installations and upgrades regardless of the **osm_default_node_selector** configuration.

- The compute node role is assigned by non-master, non-dedicated infrastructure nodes hosts (by default, nodes with a region=infra label) are labeled with **node-role.kubernetes.io/compute=true**.
- The master nodes are labeled with **node-role.kubernetes.io/master=true**, which assigns the master node role.

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.

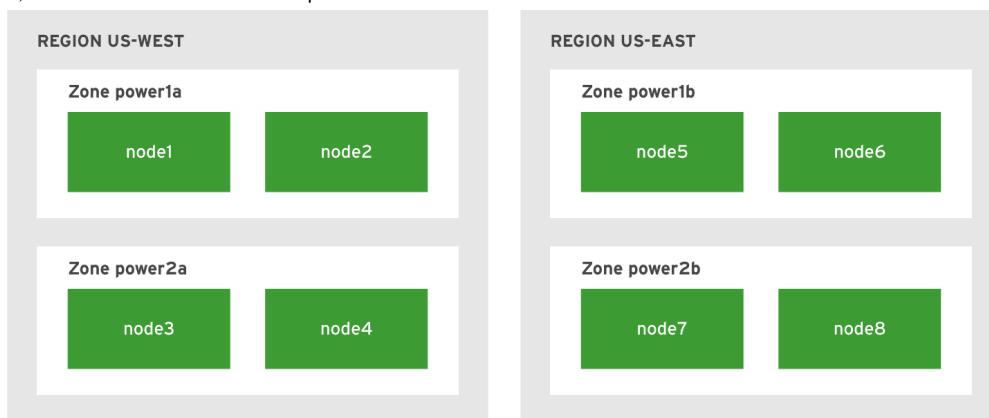


Figure 14.6: Sample cluster topology using regions and zones

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 node 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 3.9 Advanced Installation method configures 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 3.9 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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 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
```

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.

```
[student@workstation ~]$ oc get nodes -L region
NAME           STATUS     ...   REGION
master.lab.example.com Ready
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:443".
[student@workstation ~]$ oc new-app --name=hello \
--docker-image=registry.lab.example.com/openshift/hello-openshift
```

- 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  ...  10.128.0.36  node2.lab.example.com
hello-1-gmpmt  1/1    Running  ...  10.129.0.48  node1.lab.example.com
hello-1-k7sr3   1/1    Running  ...  10.128.0.35  node2.lab.example.com
hello-1-rd661   1/1    Running  ...  10.128.0.50  node2.lab.example.com
hello-1-wfx33   1/1    Running  ...  10.129.0.59  node1.lab.example.com
```

There should be 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
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.yaml** 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:



NOTE

Use spaces instead of tabs when editing the YAML file.

```
...
spec:
  nodeSelector:
    region: apps
  containers:
...
...
```

- 3.3. Apply the changes to the deployment configuration:

```
[student@workstation ~]$ oc apply -f dc.yaml
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  ...  10.128.0.36  node2.lab.example.com
hello-2-dj136  1/1    Running  ...  10.128.0.48  node2.lab.example.com
hello-2-g91kb  1/1    Running  ...  10.128.0.35  node2.lab.example.com
hello-2-mdjbg  1/1    Running  ...  10.128.0.50  node2.lab.example.com
hello-2-v22hv  1/1    Running  ...  10.128.0.59  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   ...
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 \
--delete-local-data
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 recreated on **node1**:

```
[student@workstation ~]$ oc get pods -o wide
NAME      READY   STATUS    ...   IP          NODE
hello-2-dtbp9  1/1    Running  ...  10.128.0.36  node1.lab.example.com
hello-2-g502k  1/1    Running  ...  10.128.0.48  node1.lab.example.com
hello-2-tr4cz  1/1    Running  ...  10.128.0.35  node1.lab.example.com
hello-2-x3nh5  1/1    Running  ...  10.128.0.50  node1.lab.example.com
hello-2-z3w7w  1/1    Running  ...  10.128.0.59  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   ... 
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-2018`, 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 tabs for 'Browse Catalog', 'Deploy Image', and 'Import YAML / JSON'. Below this is a search bar with the placeholder 'Filter by name or description'. The main area is divided into sections for 'Languages' and 'Technologies'.

Languages:

- Java (represented by a Java icon)
- JS (JavaScript) (represented by a JS icon)
- .NET (represented by a .NET icon)
- Perl (represented by a Perl icon)
- PHP (represented by a PHP icon)
- Python (represented by a Python icon)
- Ruby (represented by a Ruby 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 14.7: 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 'Rails + PostgreSQL (Persistent)'. Under the 'Images' section, it lists three components: 'openshift/ruby:2.3' from parameter Namespace, 'rails-pgsql-persistent:latest' from parameter Name, and 'postgres:9.5'. In the 'Parameters' section, there are two fields: 'Name' (set to 'rails-pgsql-persistent') and 'Namespace' (set to 'openshift').

Figure 14.8: 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/D0285/labs/schedule-is
-------	--------------------------------------

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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

```
[student@workstation ~]$ lab schedule-is setup
```



WARNING

If the setup script fails, ensure you have re-enabled project creation for all users with the following command: **oc adm policy add-cluster-role-to-group self-provisioner system:authenticated:oauth**

- 1. Deploy the phpmyadmin application in a new project named **schedule-is**.

- 1.1. On the **workstation** host, log in as the **developer** user:

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

- 1.2. Create a project called **schedule-is**:

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

- 1.3. Create a new application using the **phpmyadmin/phpmyadmin:4.7** image from the classroom registry:

```
[student@workstation ~]$ oc new-app --name=phpmyadmin \
--docker-image=registry.lab.example.com/phpmyadmin/phpmyadmin:4.7
```

- 2. Create a service account with root support to enable the container to run with root privileges.

- 2.1. From the terminal window, run the following command:

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

- 2.2. Connect to the **schedule-is** project:

```
[student@workstation ~]$ oc project schedule-is
```

- 2.3. Create a service account named **phpmyadmin-account**.

From the existing terminal window, run the following command:

```
[student@workstation ~]$ oc create serviceaccount phpmyadmin-account
serviceaccount "phpmyadmin-account" created
```

- 2.4. 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
scc "anyuid" added to: ["system:serviceaccount:secure-review:phpmyadmin-account"]
```

- 3. As the **developer** user, update the deployment configuration to use the newly created service account. This configuration change triggers a new phpmyadmin deployment resulting in a running pod.

- 3.1. On the **workstation** host, log in as the **developer** user:

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

- 3.2. 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/D0285/labs/secure-review** folder.

Run the following command:

```
[student@workstation ~]$ oc patch dc/phpmyadmin --patch \
'{"spec": {"template": {"spec": {"serviceAccountName": "phpmyadmin-account"}}}}'
deploymentconfig "phpmyadmin" patched
```

- 3.3. Use the **oc get pods** command to see that phpmyadmin is redeployed:

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

NAME	READY	STATUS	RESTARTS	AGE
phpmyadmin-2-r47m3	1/1	Running	0	27m

This output indicates there is a single pod running. The number "2" in its name indicates that this is the second deployment of the application.

- ▶ 4. Update the image in the internal image registry to trigger a new deployment of the application.

You will use the local Docker daemon on the **workstation** host to load a new container image and push the image into the OpenShift internal registry.

- 4.1. On the **workstation** host, a new docker image for **phpmyadmin** is available in the **/home/student/D0285/labs/schedule-is** folder. Load it to the local docker daemon.

Run the following command:

```
[student@workstation ~]$ cd /home/student/D0285/labs/schedule-is
[student@workstation schedule-is]$ docker load -i phpmyadmin-latest.tar
cd7100a72410: Loading layer [=====] 4.403 MB/4.403 MB
f06b58790eeb: Loading layer [=====] 2.873 MB/2.873 MB
730b09e0430c: Loading layer [=====] 11.78 kB/11.78 kB
931398d7728c: Loading layer [=====] 3.584 kB/3.584 kB
...output omitted...
Loaded image ID: sha256:93d0d7db5...output omitted...
```

- 4.2. Retrieve the image ID from the image loaded to the local docker daemon cache.

Run the following command:

```
[student@workstation schedule-is]$ docker images
REPOSITORY          TAG           IMAGE ID ...
none               none          93d0d7db5ce2 ...
```

- 4.3. 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:

```
[student@workstation schedule-is]$ docker tag 93d0d7db5ce2 \
docker-registry-default.apps.lab.example.com/schedule-is/phpmyadmin:4.7
```

- 4.4. Verify that the image tag is set:

```
[student@workstation schedule-is]$ docker images
REPOSITORY          TAG           IMAGE ID ...
TAG    IMAGE ID ...
```

```
docker-registry-default.[...]/schedule-is/phpmyadmin 4.7 93d0d7db5ce2 ...
```

- 4.5. Get the authentication token to access the OpenShift APIs.

From the terminal window, run the following command:

```
[student@workstation schedule-is]$ TOKEN=$(oc whoami -t)
```

- 4.6. Log in to the internal image registry using the token.

From the terminal window, run the following command:

```
[student@workstation schedule-is]$ docker login -u developer -p ${TOKEN} \
docker-registry-default.apps.lab.example.com
Error ...output omitted... certificate signed by unknown authority
```

The docker daemon considers the internal registry insecure as it uses a self-signed certificate.

- 4.7. Ensure the docker daemon on **workstation** trusts the OpenShift internal registry.

A script is provided. Run the script.

```
[student@workstation schedule-is]$ ./trust_internal_registry.sh
Fetching the OpenShift internal registry certificate.
done.
```

```
Copying certificate to the correct directory.
done.
```

```
System trust updated.
```

```
Restarting docker.
done.
```

- 4.8. Now that the internal registry is trusted, try to log in to the internal registry again.

```
[student@workstation schedule-is]$ docker login -u developer -p ${TOKEN} \
docker-registry-default.apps.lab.example.com
Login Succeeded
```

- 4.9. 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:

```
[student@workstation schedule-is]$ docker push \
docker-registry-default.apps.lab.example.com/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
```

- 5. Verify that the new image triggered a new deploy process.

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-3-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 3

- 6. 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.apps.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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

1. Update the **region** label on the **node1** host to **services**, and on the **node2** host to **applications**.
2. As the OpenShift **admin** user, create a new project named **manage-review**.
3. Deploy the new **version** application scaled to three pods. The application source code is available at <http://registry.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** host to **applications**, in preparation for maintenance of the **node2** host.
7. Prepare the **node2** host for maintenance, by setting it to *unschedulable* and then draining the node. Delete all of its pods and recreate them on the **node1** host.
8. Create a route to allow external communication with the **version** application. The route must be accessible using the host name **version.apps.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

MANAGING APPLICATION DEPLOYMENTS

In this lab, you will manage pods to run ordinary maintenance tasks on an OpenShift cluster.

RESOURCES

Application URL:	http://version.apps.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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

1. Update the **region** label on the **node1** host to **services**, and on the **node2** host 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
```

- 1.2. Verify the current node labels. They should be as configured by the OpenShift installer:

```
[student@workstation ~]$ oc get nodes -L region
```

The expected output is:

NAME	STATUS	...	REGION
master.lab.example.com	Ready	...	
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	...
node1.lab.example.com	Ready	... services
node2.lab.example.com	Ready	... applications

2. As the OpenShift **admin** user, create a new project named **manage-review**.

Run the following command to create a new project:

```
[student@workstation ~]$ oc new-project manage-review
Now using project "manage-review" on server "https://master.lab.example.com:443".
```

3. Deploy the new **version** application scaled to three pods. The application source code is available at <http://registry.lab.example.com/version>.

- 3.1. Run the following command in the terminal window:

```
[student@workstation ~]$ oc new-app -i php:7.0 \
http://registry.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
deploymentconfig "version" scaled
```

- 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** host.

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 export dc/version -o yaml > version-dc.yaml
```

- 4.2. Update the YAML file to include a node selector.

Open the **version-dc.yaml** 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 replace -f version-dc.yaml
deploymentconfig "version" 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** host to **applications**, in preparation for maintenance of the **node2** host.

- 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** host.

```
[student@workstation ~]$ oc get nodes -L region
```

The expected output is as follows:

NAME	STATUS	... REGION
master.lab.example.com	Ready	...
node1.lab.example.com	Ready	... applications
node2.lab.example.com	Ready	... applications

7. Prepare the **node2** host for maintenance, by setting it to *unschedulable* and then draining the node. Delete all of its pods and recreate them on the **node1** host.

- 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 \
--delete-local-data
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.apps.lab.example.com**.

```
[student@workstation ~]$ oc expose service version \
--hostname=version.apps.lab.example.com
route "version" exposed
```

9. Test the application using the **curl** command.

```
[student@workstation ~]$ curl http://version.apps.lab.example.com
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<p>Version v1</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	...	
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 15

INSTALLING AND CONFIGURING THE METRICS SUBSYSTEM

GOAL

Install and configure the metrics gathering subsystem.

OBJECTIVES

- Describe the architecture and operation of the metrics subsystem.
- Install the metrics subsystem.

SECTIONS

- 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.

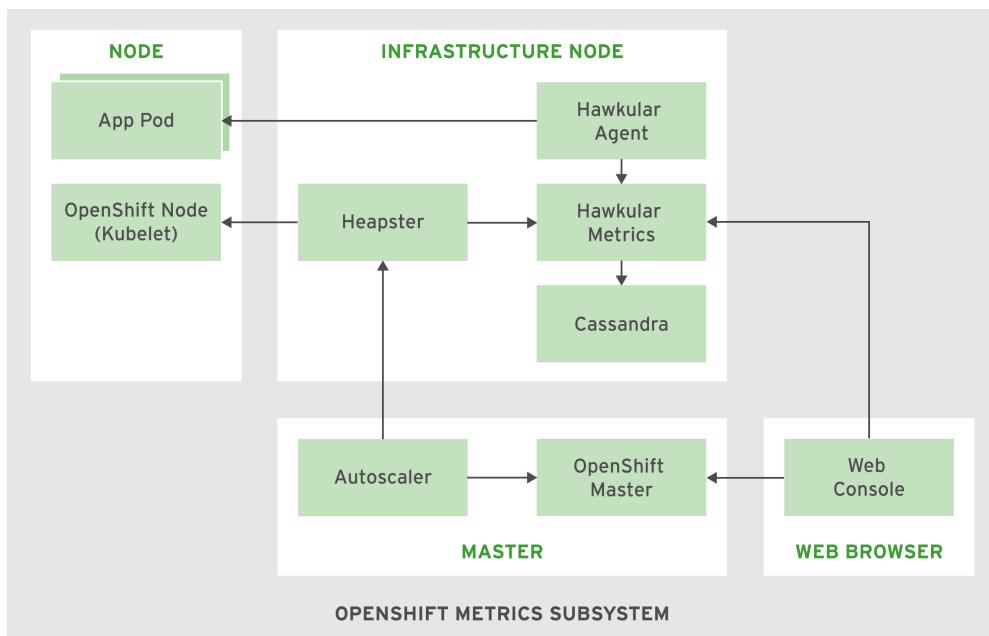


Figure 15.1: OpenShift metrics subsystem architecture

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. Red Hat OpenShift Container Platform uses Hawkular as the storage engine for Heapster.

The Heapster project was incubated 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 provides metrics to the Hawkular Agent.

The Hawkular OpenShift Agent (HOSA) is currently a technology preview features and is not installed by default Red Hat does not provide support for technology preview features and does not recommend to use them for production.

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 require the metrics subsystem in order to provide some optional features:

- The web console invokes the Hawkular Metrics API to fetch data to render performance graphics about pods in a project. If the metrics subsystem is not deployed, the charts are not displayed. Notice that the calls are made from the user web browser, not from the OpenShift master node.
- 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 invokes the Heapster API to fetch data about the current state of all pods from a deployment in order to make decisions about scaling the deployment controller.

Red Hat OpenShift Container Platform does not force an organization to deploy 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 documentation* link in the *References* 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 3.9, only shows information related to resource requests declared by pods. 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 displays 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 3.9, shows metrics only for pods and projects. The web console does not show node metrics.

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:/api/v1/model②

TOKEN=$(oc whoami -t)③

curl -k -H "Authorization: Bearer $TOKEN" \
-X GET $APIPROXY/$HEAPSTER/$NODE/metrics/memory/working_set?start=$START④
⑤
```

- 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.

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 EAP 7 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

DESCRIBING THE ARCHITECTURE OF 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. Access 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

DESCRIBING THE ARCHITECTURE OF 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. Access 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 Ansible Playbooks, and you can choose to use the base playbook or the metrics-specific playbook for the deployment.

Most of the metrics subsystem configuration is performed using Ansible variables in the main inventory file used for the advanced installation method. Although it is possible to use the `-e` option to override or define the values for some variables, Red Hat recommends defining the metrics variable in the main inventory file. If you need to make changes to the metrics subsystem configuration, update the variables in the inventory and rerun the installation playbook.

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.

The following command expects a default `ansible.cfg` configuration file that instructs Ansible how to execute the playbook, such as the following Ansible configuration file.

```
[defaults]
remote_user = student
inventory = ./inventory
log_path = ./ansible.log

[privilegeEscalation]
become = yes
becomeUser = root
becomeMethod = sudo
```

```
# ansible-playbook \①
/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml \②
-e openshift_metrics_install_metrics=True ③
```

- ② The OpenShift metrics installation playbook. It is provided by the `openshift-ansible-playbooks` package which is installed as a dependency of the `atomic-openshift-utils` package.
The metrics playbook is located at `/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/`.
- 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 variable can also be defined in the inventory file that has been used to deploy the cluster.

The following example deploys the OpenShift metrics subsystem using the Ansible inventory that was used to deploy the cluster. The inventory file contains the `openshift_metrics_install_metrics=True` variable in the `[OSEv3:vars]` group.

```
# ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml
```

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. The `-e` option overrides the value defined in the inventory file.

```
# ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.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. Consult 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 does not create any 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 should 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` and `oc logs` commands 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.

**NOTE**

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 as the one which is configured in the OpenShift master service, and used as default domain for new routes.

The playbook gets the *master-wildcard-domain* value from the Ansible hosts file, as defined by the **openshift_master_default_subdomain** variable. If the OpenShift master service configuration was changed then they will not match. In this case, provide the new value to the **openshift_metrics_hawkular_hostname** variable to the metrics 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**
- 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**

Consult the Red Hat OpenShift Container Platform *Install and Configuration* guide for the definition, default values, and syntax of each of these Ansible variables.

The following example installs the metric subsystem with a custom configuration. It overrides the Cassandra configuration that is defined in the inventory file.

The inventory file contains the following instructions:

```
[OSEv3:vars]
...output omitted...
openshift_metrics_cassandra_replicas=2
openshift_metrics_cassandra_requests_memory=2Gi
openshift_metrics_cassandra_pvc_size=50Gi
```

The following command overrides the properties defined for Cassandra. It assumes an Ansible configuration file, **ansible.cfg**, which instructs Ansible to escalate privileges.

```
# ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml \
-e openshift_metrics_cassandra_replicas=3 \①
-e openshift_metrics_cassandra_requests_memory=4Gi \②
-e openshift_metrics_cassandra_pvc_size=25Gi \③
```

① Defines three Cassandra nodes for the metrics stack instead of two. This value dictates the number of Cassandra replication controllers.

- ❷ Defines 4 Gb of memory for the Cassandra pod instead of 2 Gb.
- ❸ Defines a size of 25 Gb for each of the Cassandra nodes instead of 50 Gb.

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.

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.

BEFORE YOU BEGIN

All the labs from Chapter 7, *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-prepare setup  
[student@workstation ~]$ cd /home/student/do285-ansible  
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 1. 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 registry.lab.example.com \  
search metrics-cassandra ssl  
available options:-  
  
-----  
1) Name: openshift3/ose-metrics-cassandra  
Tags: v3.9  
  
1 images found !  
[student@workstation ~]$ docker-registry-cli registry.lab.example.com \  
search metrics-hawkular-metrics ssl  
available options:-
```

```
-----
1) Name: openshift3/ose-metrics-hawkular-metrics
Tags: v3.9

1 images found !
[student@workstation ~]$ docker-registry-cli registry.lab.example.com \
search metrics-heapster ssl
available options:-

-----
1) Name: openshift3/ose-metrics-heapster
Tags: v3.9

1 images found !
```

Notice from the search output that all image names have the prefix **openshift3/ose-** and the **v3.9** tag. This information is required by the metrics subsystem installation playbook.

- ▶ 2. The Atomic OpenShift volume recycler image, **ose-recycler** is used to prepare persistent volumes for reuse after they are deleted. Query the Docker registry for the image.

```
[student@workstation ~]$ docker-registry-cli registry.lab.example.com \
search ose-recycler ssl
available options:-

-----
1) Name: openshift3/ose-recycler
Tags: v3.9

1 images found !
```

- ▶ 3. Review the NFS configuration on the **services** VM.

- 3.1. The advanced installer configured an NFS share on the **services** VM. Use the **ssh** command to connect to the **services** host, and from there inspect the NFS configuration.

Review the permissions for the **/exports/metrics** folder and the **/etc/exports.d/openshift-ansible.exports** configuration file.

```
[student@workstation ~]$ ssh root@services
[root@services ~]# 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@services ~]# 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)
"/exports/etcd-vol2" *(rw,root_squash,sync,no_wdelay)
"/exports/prometheus" *(rw,root_squash)
"/exports/prometheus-alertmanager" *(rw,root_squash)
```

```
"/exports/prometheus-alertbuffer" *(rw,root_squash)
```

3.2. Exit the server.

```
[root@services ~]# exit
```

► 4. Create a persistent volume (PV) definition file for the NFS share.

- 4.1. Use the following parameters so that the PV uses the NFS share created by the 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
  5 GiB

accessModes
  ReadWriteOnce

nfs.path
  /exports/metrics

nfs.server
  services.lab.example.com

persistentVolumeClaimPolicy
  Recycle (for a production cluster, many system administrators prefer Retain)
```

The PV definition is ready to use in the **metrics-pv.yaml** file of the **/home/student/D0285/labs/install-metrics** directory on the **workstation** VM. 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: services.lab.example.com
  persistentVolumeReclaimPolicy: Recycle
```

- 4.2. Log in as a cluster administrator and create the persistent volume using the YAML file provided in this exercise labs folder by running the two following commands. If prompted, acknowledge the insecure connection.

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://master.lab.example.com
[student@workstation ~]$ oc create -f D0285/labs/install-metrics/metrics-pv.yaml
persistentvolume "metrics" created
```

- 4.3. Verify that the persistent volume was created and is available to be claimed.

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

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS
<i>... output omitted ...</i>				
metrics	5Gi	RWO	Recycle	Available

- 5. Install the metrics subsystem using the playbook provided by the Red Hat OpenShift Container Platform installer.

- 5.1. 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 **services** 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 the images with a tag of **v3.9**.

openshift_metrics_heapster_requests_memory

300 MB is sufficient for this environment.

openshift_metrics_hawkular_requests_memory

750 MB is sufficient for this environment.

openshift_metrics_cassandra_requests_memory

750 MB is sufficient for this environment.

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 for this environment.

openshift_metrics_cassandra_pvc_prefix

Use **metrics** as the prefix for the persistent volume claim name.



NOTE

You need 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 will fail to start.

More information about these variables is available in the *Enabling Cluster Metrics* chapter of the Red Hat OpenShift Container Platform 3.9 *Installation and*

Configuration guide. The variable definitions are provided by the *Ansible Variables* table.

- 5.2. Go to the `/home/student/DO285/labs/install-metrics` directory and open the inventory file located at `/home/student/DO285/labs/install-metrics`. Add the variables for the deployment of the metrics subsystem.

Use your favorite text editor to edit the `inventory` file. Locate the `[OSEv3:vars]` group and append the following values:



NOTE

If needed, copy and paste the content of the `metrics-vars.txt` file in the `install-metrics` directory to the `[OSEv3:vars]` group.

```
[OSEv3:vars]
... output omitted ...
# Metrics Variables
openshift_metrics_install_metrics=True
openshift_metrics_image_prefix=registry.lab.example.com/openshift3/ose-
openshift_metrics_image_version=v3.9
openshift_metrics_heapster_requests_memory=300M
openshift_metrics_hawkular_requests_memory=750M
openshift_metrics_cassandra_requests_memory=750M
openshift_metrics_cassandra_storage_type=pv
openshift_metrics_cassandra_pvc_size=5Gi
openshift_metrics_cassandra_pvc_prefix=metrics
```

- 5.3. Check the inventory file for errors. To prevent repeated lengthy installations due to an error in the inventory file, an inventory checker is provided. To check that there are no errors, run the following grading script on the workstation VM:

```
[student@workstation install-metrics]$ lab install-metrics grade
```

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

```
Overall inventory file check..... PASS
```

If there are errors in your inventory file and the grading script fails, you can compare your inventory file with the valid answer file located at `/home/student/DO285/solutions/install-metrics/inventory`. Correct the errors and rerun the grading script to ensure that it passes.

- 5.4. Use the `ansible-playbook` command against the metrics playbook located at `/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml` to deploy the metrics subsystem. Ansible detects and uses the `ansible.cfg` configuration file to escalate privileges and use the inventory file in the current directory.

```
[student@workstation ~]$ cd \
/home/student/DO285/labs/install-metrics/
[student@workstation install-metrics]$ ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/openshift-metrics/config.yml
```

```
...
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. 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.

A successful installation should display the following output:

```
INSTALLER STATUS ****
Initialization          : Complete (0:00:14)
Metrics Install        : Complete (0:01:43)
```



IMPORTANT

If something goes wrong during this or the following steps, run the `/home/student/D0285/labs/install-metrics/uninstall-metrics.sh` script to uninstall the metrics subsystem. After the completion of the script, recreate the persistent volume by performing the Step 4.2.

- ▶ 6. Verify that the metrics subsystem was deployed successfully.
 - 6.1. Verify that the persistent volume claim generated by the metrics playbook is bound. To do so, retrieve the list of persistent volume claims in the **openshift-infra** project as the **admin** user.

```
[student@workstation install-metrics]$ oc get pvc -n openshift-infra
NAME      STATUS    VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
metrics-1 Bound     metrics   5Gi       RWO          20m
```

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.

- 6.2. Wait until all metrics subsystem pods are ready and running, as indicated by a STATUS of **1/1**. This may take a few minutes.

```
[student@workstation install-metrics]$ oc get pod -n openshift-infra
```

NAME	READY	STATUS	...
hawkular-cassandra-1-kdhgr	1/1	Running	...
hawkular-metrics-0jmkm	1/1	Running	...
heapster-43pl3	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 inventory file and then follow the instructions in the *Important* box from the previous step.

► 7. Open the Hawkular home page in a web browser to accept its self-signed certificate.

- 7.1. Find the route host name for the Hawkular pod:

```
[student@workstation install-metrics]$ oc get route -n openshift-infra
NAME           HOST/PORT   ...
hawkular-metrics  hawkular-metrics.apps.lab.example.com ...
```

- 7.2. Open a web browser on the **workstation** VM and open the Hawkular URL. Make sure to use the HTTPS protocol.

<https://hawkular-metrics.apps.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.

► 8. Deploy a test application to generate some load in the OpenShift cluster.

- 8.1. Open another terminal on the **workstation** VM. From **workstation**, go to the home directory of the **student** user and log in as the **developer** user, create a new project, and a **Hello World** application from the **hello-openshift** container image:

```
[student@workstation install-metrics]$ cd
[student@workstation ~]$ oc login -u developer -p redhat \
https://master.lab.example.com
[student@workstation ~]$ oc new-project load
[student@workstation ~]$ oc new-app --name=hello \
--docker-image=registry.lab.example.com/openshift/hello-openshift
```

- 8.2. Scale the application and wait until all nine pods are ready and running:

```
[student@workstation ~]$ oc scale --replicas=9 dc/hello
deploymentconfig "hello" scaled
...
[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
```

- 8.3. Expose the application to outside access:

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

- 8.4. If not already present, install the *httpd-tools* package on the **workstation** VM:

```
[student@workstation ~]$ sudo yum install httpd-tools
```

- 8.5. 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.apps.lab.example.com/
...
Benchmarking hello-load.apps.lab.example.com (be patient)
Completed 5000 requests
...
```

Continue to the next step while Apache Bench is running.

- ▶ 9. 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  164m     8%    1276Mi        73%
node1.lab.example.com   1276m    63%   2382Mi        30%
node2.lab.example.com   629m    31%   2167Mi        28%
```

The values you see might be different from this example.

- ▶ 10. Fetch current metrics from the Heapster pod using the **curl** command.

- 10.1. 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.

Run the following command to inspect the contents of the script:

```
[student@workstation ~]$ cat ~/D0285/labs/install-metrics/node-metrics.sh
#!/bin/bash

oc login -u admin -p redhat

TOKEN=$(oc whoami -t)
APIPROXY=https://master.lab.example.com/api/v1/proxy/namespaces/openshift-infra/
services
HEAPSTER=https://heapster:8080/api/v1/model
NODE=nodes/node1.lab.example.com
START=$(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.

10.2. Run the script to see the current node metrics.



NOTE

Your values will differ from the output.

```
[student@workstation ~]$ ./DO285/labs/install-metrics/node-metrics.sh
{
  "metrics": [
    {
      "timestamp": "2018-08-09T02:53:30Z",
      "value": 1856569344 ①
    },
    {
      "timestamp": "2018-08-09T02:54:00Z",
      "value": 1860599808 ②
    }
  ],
  "latestTimestamp": "2018-08-09T02:54:00Z"
}
{
  "metrics": [
    {
      "timestamp": "2018-08-09T02:53:30Z",
      "value": 499 ③
    },
    {
      "timestamp": "2018-08-09T02:54:00Z",
      "value": 1213 ④
    }
  ],
  "latestTimestamp": "2018-08-09T02:54:00Z"
}
```

The sample output shows:

- ① ② 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.

- 11. View the project metrics from Hawkular, using the OpenShift web console.
- 11.1. Open a web browser and navigate to the OpenShift web console at `https://master.lab.example.com`. Log in as the **developer** user, using **redhat** as the password.
 - 11.2. Navigate to the **load** project and visit its overview page. Click the arrow to open the details of the `hello`,#1 deployment configuration. The following image displays several pod metrics:

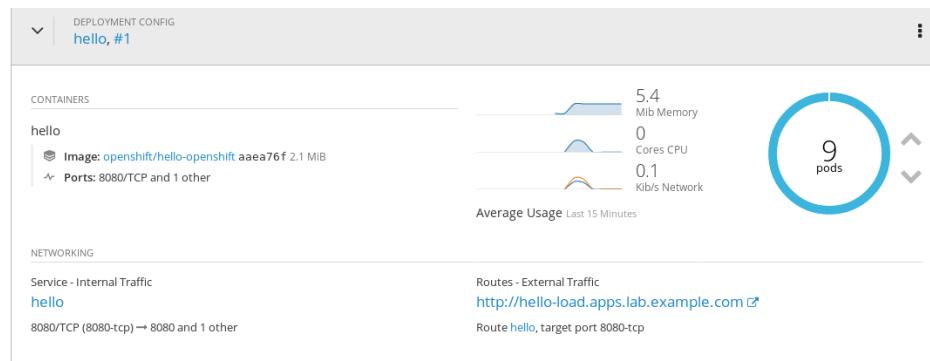


Figure 15.2: Metrics overview

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 7 and refresh the web console page.
- The Hawkular pod is not ready and running. You may need to uninstall the metrics subsystem, updated your installation script, and reinstall.

- 11.3. 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 function.
 - Hawkular Metrics stores the metrics and provides querying capabilities. The OpenShift web console requires 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 the addition of Ansible variables for the metrics subsystem.
- 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 the advanced OpenShift installation method. 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 16

MANAGING AND MONITORING OPENSHIFT CONTAINER PLATFORM

GOAL

Manage and monitor OpenShift resources and software.

OBJECTIVES

- Limit the amount of resources consumed by an application.
- Describe how to upgrade an instance of OpenShift.
- Configure probes to monitor application health.
- Monitor OpenShift resources using data obtained from the web console.

SECTIONS

- 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 Etcd 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

Consult the Red Hat OpenShift Container Platform 3.9 *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 is 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 refuses 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 that is 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** object. See the Red Hat OpenShift Platform 3.9 *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

TYPE	RESOURCE NAME	DESCRIPTION
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 3.9 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. You can also use the **oc get limits** command, which produces the same output.

```
$ 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 is 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 3.9 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 3.9 documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform

► WORKSHOP

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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 1. 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/D0285/labs/monitor-limit** folder. You can copy and paste the commands from that script.

- 1.1. 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
```

- 1.2. 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
----- -----
200m (10%) 0 (0%) 512Mi (6%) 0 (0%)
```

```
[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%) 512Mi (6%) 0 (0%)
```

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 the **hello-openshift** image:

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

- 1.5. Wait for the **hello** pod to be ready and running, and then retrieve the node on which the pod is running.

```
[student@workstation ~]$ oc get pod -o wide  
NAME READY ... NODE  
hello-1-wbt9n 1/1 ... 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 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  
-----  
200m (10%) 0 (0%) 512Mi (6%) 0 (0%)
```



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, which includes the deployment configuration, the image stream, the pod, and the service.

```
[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/D0285/labs/monitor-limit** folder.

- 2.1. Inspect the limit range definition, which specifies default resource requests for CPU only.

```
[student@workstation ~]$ cat ~/D0285/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 3.9 documentation provides a sample file if you want to create this file from scratch. Consult the *Setting Limit Ranges* chapter of the *Cluster Administration* documentation for more details on how to apply limit ranges.

- 2.2. Add the limit range to the project:

```
[student@workstation ~]$ oc create -f ~/D0285/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 ~/D0285/labs/monitor-limit/quota.yml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: project-quota
spec:
  hard:
```

```
cpu: "900m"
```

The Red Hat OpenShift Container Platform 3.9 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 more details on how to apply quotas.

2.5. Add the quota to the project:

```
[student@workstation ~]$ oc create -f ~/D0285/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/D0285/labs/monitor-limit** directory.

3.1. Log in to OpenShift as the **developer** user:

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

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 4m
[student@workstation ~]$ oc delete limits project-limits
Error from server (Forbidden): limitranges "project-limits" is forbidden: \
User "developer" cannot delete limitranges in the namespace "resources": \
User "developer" cannot delete limitranges in project "resources"
```

```
[student@workstation ~]$ oc get quota
NAME          AGE
project-quota 1m
```

```
[student@workstation ~]$ oc delete quota project-quota
Error from server (Forbidden): resourcequotas "project-quota" is forbidden: \
User "developer" cannot delete resourcequotas in the namespace "resources": \
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 for the project.

You can also use the **oc get limitranges** command, which produces the same output.

- 3.4. Create pods from the **hello-openshift** image:

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

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
```



NOTE

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/D0285/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
```

- 4.2. Retrieve the node on which the **hello** pod is running.

```
[student@workstation ~]$ oc get pod -o wide -n resources
NAME          READY  ...  NODE
hello-1-454cc 1/1   ...  node1.lab.example.com
```

Make note of the node name as 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
-----        -----      -----           -----
450m (22%)    250m (12%)  512Mi (6%)     0 (0%)
```

- 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 -A2 Requests
Requests:
  cpu:        250m
Environment: <none>
```

- 4.5. Log back in as the **developer** user before moving to the next step:

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

- 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/D0285/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-1-454cc 1/1   Running   0         6m
```

```
hello-1-99rkb 1/1 Running 0 6s
```

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
deploymentconfig "hello" scaled
```

- 5.5. Wait a couple of minutes and verify that a third hello pod is created, however, the 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-1-454cc 1/1 Running 0 7m
hello-1-99rkb 1/1 Running 0 1m
hello-1-t2vnm 1/1 Running 0 41s
```

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 list of events 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
... output omitted ...
... Error creating: pods "hello-1-4s45n" 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. Ensure that after a few moments only one **hello** pod is running.

```
[student@workstation ~]$ oc get pod
NAME READY STATUS RESTARTS AGE
```

```
hello-1-454cc 1/1      Running  0      20mm
```

- 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/D0285/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
  Environment: <none>
```

- 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/D0285/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-f3b21	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 meantime, 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:443

svc/hello - 172.30.207.52 ports 8080, 8888
dc/hello deploys istag/hello:latest
      deployment #3 failed 10 minutes ago: config change
      deployment #2 deployed 12 minutes ago - 1 pod
      deployment #1 deployed 34 minutes ago

3 infos identified, use 'oc status -v' to see details.
```

- 7.3. Get the list of events 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 hello-3.*Failed
9m       15m       26       hello-3-9nw8n.15495845f8ab7b98 Pod \
Warning   FailedScheduling           default-scheduler \
0/3 nodes are available: 1 MatchNodeSelector, 3 Insufficient memory.
... output omitted ...
```

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

This concludes the guided exercise.

UPGRADING THE OPENSHIFT CONTAINER PLATFORM

OBJECTIVE

After completing this section, students should be able to describe how to upgrade an instance of OpenShift.

UPGRADING OPENSIFT

When new versions of OpenShift Container Platform are released, you can upgrade an existing cluster to apply the latest enhancements and bug fixes. This includes upgrading from previous minor versions, such as an upgrade from 3.7 to 3.9, and applying updates to a minor version (3.7.z releases).



IMPORTANT

Red Hat OpenShift Container Platform 3.9 includes a merge of features and fixes from Kubernetes 1.8 and 1.9. If you follow the upgrade path from OpenShift Container Platform 3.7, your cluster will be upgraded to a 3.9 release.



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, you should not run mismatched versions longer than is necessary to upgrade the entire cluster. In addition, using the quick installer to upgrade from version 3.7 to 3.9 is not supported.

UPGRADE METHODS

There are two methods for performing OpenShift Container Platform cluster upgrades. You can either do in-place upgrades (which can be 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.



NOTE

Automated in-place upgrades are available for clusters that were installed using the quick and advanced installation methods.

**IMPORTANT**

As detailed in Chapter 7, *Installing OpenShift Container Platform*, the quick installation method is now deprecated. The quick installer is capable of deploying Red Hat OpenShift Container Platform, however this method cannot be used to upgrade a cluster from 3.7 to 3.9.

When the cluster is installed with the advanced installation method, you can perform either automated or manual in-place upgrades by reusing their inventory file.

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. The OpenShift upgrade method marks a subset of nodes unschedulable and drains the pods to the available nodes. After a successful upgrade, the nodes are put back in a schedulable state.

PERFORMING AN AUTOMATED CLUSTER UPGRADE

With the advanced installation method, you can use Ansible Playbooks to automate the OpenShift cluster upgrade process. Playbooks for upgrade are available in `/usr/share/ansible/openshift-ansible/playbooks/common/openshift-cluster/upgrades/`. The directory contains a set of subdirectories for upgrading the cluster, for example, `v3_9`.

To perform an upgrade, run the `ansible-playbook` command against the upgrade Playbooks. You can use the `v3_9` playbooks to upgrade an existing OpenShift cluster running the 3.7 version to 3.9 or to apply the latest asynchronous errata updates.

The automated upgrade performs the following tasks:

- Applies the latest configuration changes
- Saves the Etcd data
- Updates the APIs from 3.7 to 3.8, then from 3.8 to 3.9
- If present, updates the default router from 3.7 to 3.9
- If present, updates the default registry from 3.7 to 3.9
- 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.

If you are using containerized GlusterFS, the nodes will not evacuate the pods because GlusterFS pods are running as part of a `daemonset`. To properly upgrade a cluster that runs containerized GlusterFS, you need to:

1. Upgrade the master, Etcd, and infrastructure services (router, registry, logging, and metrics).
2. Upgrade the nodes that run application containers.
3. Upgrade the node that run GlusterFS one at a time.



IMPORTANT

Before upgrading the cluster to OpenShift Container Platform 3.9, the cluster must already be upgraded to the latest asynchronous release of version 3.7. 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 3.6, you must first upgrade incrementally, for example, 3.5 to 3.6, then 3.6 to 3.7.

Failure to do so may result in upgrade failure.

To update your cluster incrementally from earlier versions, use the set of playbooks located at **/usr/share/ansible/openshift-ansible/playbooks/common/openshift-cluster/upgrades/**. The directory contains the following structure:

```
v3_6
├── roles -> ../../../../../../roles/
├── upgrade_control_plane.yml
├── upgrade_nodes.yml
└── upgrade.yml
└── validator.yml

v3_7
├── roles -> ../../../../../../roles/
├── upgrade_control_plane.yml
├── upgrade_nodes.yml
├── upgrade.yml
└── validator.yml

v3_9
├── label_nodes.yml
├── master_config_upgrade.yml
├── roles -> ../../../../../../roles/
├── upgrade_control_plane.yml
├── upgrade_nodes.yml
└── upgrade.yml

3 directories, 13 files
```

Each release directory contains the same set of files that allow for upgrading masters and nodes in a single phase or for upgrading the masters and nodes in separate phases, as discussed in the following section *Upgrading the Cluster in Multiple Phases*.



NOTE

Before attempting the upgrade, 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.

Preparing for an Automated Upgrade

The following procedure shows how to prepare your environment for an automated upgrade. Before performing an upgrade, Red Hat recommends to review your inventory file to ensure that manual updates to the inventory file have been applied. If the changes are not applied, the changes may be overwritten with default values.

1. If this is an upgrade from OpenShift Container Platform 3.7 to 3.9, manually disable the 3.7 repository and enable the 3.8 and 3.9 repositories on each master and node host:

```
[root@demo ~]# subscription-manager repos \
--disable="rhel-7-server-ose-3.7-rpms" \
--enable="rhel-7-server-ose-3.9-rpms" \
--enable="rhel-7-server-ose-3.8-rpms" \
--enable="rhel-7-server-rpms" \
--enable="rhel-7-server-extras-rpms" \
--enable="rhel-7-server-ansible-2.4-rpms" \
--enable="rhel-7-fast-datapath-rpms"
[root@demo ~]# 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. In previous versions of OpenShift Container Platform, master hosts were marked unschedulable by default by the installer. However, starting with Red Hat OpenShift Container Platform 3.9, the master nodes must be marked as schedulable, which is done automatically during the upgrade process.

If you did not set the default node selectors, as shown in the following screen, they will be added during the upgrade process. Master nodes will also be labeled with the master node role. All other nodes will be labeled with the **compute** node role.

```
openshift_node_labels="{'region':'infra', 'node-role.kubernetes.io/
compute':'true'}"
```

4. If you added the **openshift_disable_swap=false** variable to your Ansible inventory, or if you manually configured swap on your nodes, disable swap memory before running the upgrade.

Upgrading Master and Application Nodes

After satisfying the prerequisites, as defined in the *Preparing for an Automated Upgrade* section, you can upgrade your environment. The following procedure details the required steps for upgrading an OpenShift cluster.

1. Set the **openshift_deployment_type=openshift-enterprise** variable in the inventory file.
2. If you are using a custom Docker registry, you must explicitly specify the address of your registry to the **openshift_web_console_prefix** and **template_service_broker_prefix** variables. These values are used by Ansible during the upgrade process.

```
openshift_web_console_prefix=registry.demo.example.com/openshift3/ose-
template_service_broker_prefix=registry.demo.example.com/openshift3/ose-
```

3. If you want to enable a restart of the services or a reboot of the nodes, set the **openshift_rolling_restart_mode=system** option in your inventory file. If the option is not set, the default value instructs the upgrade process to perform restarts of services on the master nodes, but does not reboot the systems.

4. You can either update all the nodes in your environment by running a single Ansible Playbook (**upgrade.yml**), or roll-out the upgrade in multiple phases by using separate playbooks. This is discussed in the following section, *Upgrading the Cluster in Multiple Phases*.
5. Reboot all hosts. After rebooting, if you did not deploy any extra features, you can verify the upgrade.

Upgrading the Cluster in Multiple Phases

If you decide to upgrade your environment in multiple phases, the first phase, as determined by the Ansible Playbook (**upgrade_control_plane.yml**), upgrades the following components:

- The master nodes.
- The node services that are running the master nodes.
- The Docker services on the master nodes and on any stand-alone Etcd hosts.

The second phase, managed by the **upgrade_nodes.yml** playbook, upgrades the following components. Before running this second phase, the master nodes must already be upgraded.

- The node services.
- The Docker services running on stand-alone nodes.

The two phases upgrade process allows you to customize the way the upgrade runs by specifying custom variables. For example, to upgrade 50 percent of the total nodes, run the following command:

```
[root@demo ~]# ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/common/openshift-cluster/upgrades/
v3_9/upgrade_nodes.yml \
-e openshift_upgrade_nodes_serial="50%"
```

To upgrade two nodes at a time in the **HA** region, run the following command:

```
[root@demo ~]# ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/common/openshift-cluster/upgrades/
v3_9/upgrade_nodes.yml \
-e openshift_upgrade_nodes_serial="2"
-e openshift_upgrade_nodes_label="region=HA"
```

To specify how many nodes may fail in each update batch, use the **openshift_upgrade_nodes_max_fail_percentage** option. When the percentage of failure exceeds the value that you defined, Ansible aborts the upgrade. Use the **openshift_upgrade_nodes_drain_timeout** option to specify the length of time to wait before aborting the play.

The following example shows how to upgrade ten nodes at a time, and how to abort the play if more than 20 percent of the nodes (two nodes) fail. The **openshift_upgrade_nodes_drain_timeout** option defines a 600 seconds wait time to drain the nodes.

```
[root@demo ~]# ansible-playbook \
/usr/share/ansible/openshift-ansible/playbooks/common/openshift-cluster/upgrades/
v3_9/upgrade_nodes.yml \
-e openshift_upgrade_nodes_serial=10 \
-e openshift_upgrade_nodes_max_fail_percentage="20" \
-e openshift_upgrade_nodes_drain_timeout="600"
```

```
-e openshift_upgrade_nodes_max_fail_percentage=20 \
-e openshift_upgrade_nodes_drain_timeout=600
```

Using Ansible Hooks

You can execute custom tasks for specific operations through hooks. Hooks allow you to extend the default behavior of the upgrade process by defining tasks to execute before or after specific points during the upgrade process. You can for example validate or update custom infrastructure components when upgrading your cluster.



IMPORTANT

Hooks do not have any error handling mechanism, therefore, any error in a hook halts the upgrade process. You will need to fix the hook and rerun the upgrade process.

Use the **[OSEv3:vars]** section of your inventory file to define your hooks. Each hook must point to a **.YAML** file which defines Ansible tasks. The file is integrates as part of an **include** statement, which requires you to define a set of tasks rather than a Playbook. Red Hat recommends the usage of absolute paths to avoid any ambiguity.

The following hooks are available for customizing the upgrade process:

1. **openshift_master_upgrade_pre_hook**: the hook runs before each master node is updated.
2. **openshift_master_upgrade_hook**: the hook runs after each master node is upgraded, and before the master services or the nodes reboot.
3. **openshift_master_upgrade_post_hook**: the hook runs after each master node is upgraded and its service or the system restarted.

The following example shows the integration of a hook in an inventory file.

```
[OSEv3:vars]
openshift_master_upgrade_pre_hook=/usr/share/custom/pre_master.yml
openshift_master_upgrade_hook=/usr/share/custom/master.yml
openshift_master_upgrade_post_hook=/usr/share/custom/post_master.yml
```

In this example, the **pre_master.yml** file contains the following tasks.

```
---
- name: note the start of a master upgrade
  debug:
    msg: "Master upgrade of {{ inventory_hostname }} is about to start"

- name: require an operator agree to start an upgrade pause:
  prompt: "Hit enter to start the master upgrade"
```

Verifying the Upgrade

After the upgrade finishes, there are steps that you 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  ROLES   AGE    VERSION
master.lab.example.com  Ready   master   1h    v1.9.1+a0ce1bc657
node1.lab.example.com   Ready   compute  1h    v1.9.1+a0ce1bc657
node2.lab.example.com   Ready   compute  1h    v1.9.1+a0ce1bc657
```

- Verify the versions of the **docker-registry** and **router** images. The **tag** should match a 3.9 release.

```
[root@demo ~]# oc get -n default dc/docker-registry -o json | grep \"image\
\"image\": \"registry.lab.example.com/openshift3/ose-docker-registry:v3.9.14\",
```

```
[root@demo ~]# oc get -n default dc/router -o json | grep \"image\
\"image\": \"registry.lab.example.com/openshift3/ose-haproxy-router:v3.9.14\",
```

- Use the diagnostics tool on the master node to look for common issues:

```
[root@demo ~]# oc adm diagnostics
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/home/student/.kube/config'
Info: Using context for cluster-admin access: '/master:443/admin'
... output omitted ...

[Note] Summary of diagnostics execution (version v3.9.14):
[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 OPENSIFT

The steps to automatically upgrade an OpenShift cluster are shown below. Indicate the order in which the steps should be run.

1. Ensure that you have the latest version of the `atomic-openshift-utils` package on each Red Hat Enterprise Linux 7 system.
2. Optionally, if you are using a custom Docker registry, specify the address of your registry to the `openshift_web_console_prefix` and `template_service_broker_prefix` variables.
3. Disable swap memory on all the nodes.
4. Reboot all hosts. After rebooting, review the upgrade.
5. Optionally, review the node selectors in your inventory file.
6. Disable the 3.7 repository and enable the 3.8 and 3.9 repositories on each master and node host.
7. Update with a single or multiple phases strategy by using the appropriate set of Ansible Playbooks.
8. Set the `openshift_deployment_type=openshift-enterprise` variable in the inventory file.

► SOLUTION

UPGRADING OPENSIFT

The steps to automatically upgrade an OpenShift cluster are shown below. Indicate the order in which the steps should be run.

2. 1. Ensure that you have the latest version of the `atomic-openshift-utils` package on each Red Hat Enterprise Linux 7 system.
6. 2. Optionally, if you are using a custom Docker registry, specify the address of your registry to the `openshift_web_console_prefix` and `template_service_broker_prefix` variables.
4. 3. Disable swap memory on all the nodes.
8. 4. Reboot all hosts. After rebooting, review the upgrade.
3. 5. Optionally, review the node selectors in your inventory file.
1. 6. Disable the 3.7 repository and enable the 3.8 and 3.9 repositories on each master and node host.
7. 7. Update with a single or multiple phases strategy by using the appropriate set of Ansible Playbooks.
5. 8. Set the `openshift_deployment_type=openshift-enterprise` variable in the inventory file.

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 redeploy 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 from the Actions drop-down list.

The screenshot shows the OpenShift web console interface. At the top, there's a navigation bar with tabs for 'History', 'Configuration', 'Environment', and 'Events'. Below this, a message says 'Deployment #4 is active. View Log'. On the right, there's a 'Actions' dropdown menu with options like 'Edit', 'Pause Rollouts', 'Add Storage', 'Add Autoscaler', 'Edit Resource Limits', 'Edit Health Checks' (which is highlighted in blue), 'Edit YAML', and 'Delete'. Below the actions menu is a table listing four deployments. The columns are 'Deployment', 'Status', 'Created', and 'Trigger'. The data is as follows:

Deployment	Status	Created	Trigger
#4 (latest)	Active, 2 replicas	12 minutes ago	Manual
#3	✓ Complete	13 minutes ago	Manual
#2	✓ Complete	13 minutes ago	Manual
#1	✓ Complete	15 minutes ago	Config change

Figure 16.1: 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 16.2 shows the management of readiness and liveness probes.

The screenshot shows the configuration page for a deployment. It starts with a section titled 'Readiness Probe'. The 'Type' is set to 'HTTP GET'. There's a checkbox for 'Use HTTPS' which is unchecked. The 'Path' is set to '/ready'. The 'Port' is set to '3000'. Below these, there's a 'Initial Delay' field with a spin control and a 'seconds' unit indicator. A note below it says 'How long to wait after the container starts before checking its health.' At the bottom, there's a 'Timeout' field with a spin control and a 'seconds' unit indicator, currently set to '1'.

Figure 16.2: Managing readiness probes using the web console

Liveness Probe

A liveness probe checks if the container is still running. If the liveness probe fails, the container is killed.

* Type
HTTP GET

Use HTTPS

Path
/health

* Port
3000

Initial Delay
10 seconds

How long to wait after the container starts before checking its health.

Timeout
5 seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.

Figure 16.3: 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 the creation of a probe, a new entry is added to the configuration file for the deployment configuration. You can review or edit a probe by using the deployment configuration editor. The live editor allows you to edit the **periodSeconds**, **successThreshold**, and **failureThreshold** options. The following example shows the live editor for a deployment configuration.

hello-5 > Edit YAML

Edit Replication Controller hello-5

```

58   imagePullPolicy: Always
59   livenessProbe:
60     failureThreshold: 3
61     httpGet:
62       path: /health
63       port: 3000
64       scheme: HTTPS
65     initialDelaySeconds: 10
66     periodSeconds: 10
67     successThreshold: 1
68     timeoutSeconds: 5
69   name: hello
70   ports:
71     - containerPort: 3000
72       protocol: TCP
73

```

Save Cancel

Figure 16.4: 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:	http://probe.apps.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 7, *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

- 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 redhat \
https://master.lab.example.com
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:443".
```

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. 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 \
http://services.lab.example.com/node-hello
... output omitted ...
--> Success
Build scheduled, use 'oc logs -f bc/probes' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/probes'
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:443

svc/probes - 172.30.67.215:3000
dc/probes deploys istag/probes:latest <-
bc/probes docker builds http://registry.lab.example.com/node-hello on istag/
nodejs-6-rhel7:latest
build #1 running for 16 seconds - bf7e7e9: Establish remote repository (root
<root@services.lab.example.com>)
deployment #1 pending less than a second ago
```

```
3 infos identified, use 'oc status -v' to see details.
```

- 2.3. Run the **oc get pods** command until there is one pod in the *Running* state.

**NOTE**

It may take up to 3 minutes for the pod to be deployed.

```
[student@workstation ~]$ oc get pods -w
NAME        READY     STATUS    RESTARTS   AGE
probes-1-build  0/1      Completed   0          42s
probes-1-ttzb6  1/1      Running    0          23s
```

- 3. Expose a route for the service that allows external clients to access the application.

```
[student@workstation ~]$ oc expose svc probes \
--hostname=probe.apps.lab.example.com
route "probes" exposed
```

- 4. Use the **curl** command to access the application.

```
[student@workstation ~]$ curl http://probe.apps.lab.example.com
Hi! I am running on host -> probes-1-ttzb6
```

- 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.apps.lab.example.com/health
OK
[student@workstation ~]$ curl http://probe.apps.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>. Use **developer** as the user name, and **redhat** as the password. Click **probes** to access the project.
 - 6.2. Navigate to Applications → Deployments to list the deployments. Click the current version, #1, next to the **probes** deployment. Notice the following banner:

Container probes does not have health checks to ensure your application is running correctly. Add Health Checks

To add a probe, click **Add Health Checks**.

probes-1 created 5 minutes ago

app probes openshift.io/deployment-config.name probes

Details Environment Metrics Logs Events

Status: Active
Deployment Config: probes
Status Reason: config change
Selectors: app=probes
Replicas: deployment=probes-1 deploymentconfig=probes 1 current / 1 desired

Template

i Container probes does not have health checks to ensure your application is running correctly. [Add Health Checks](#)

Figure 16.5: 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
Path	/ready
Port	3000
Initial Delay	3

FIELD	VALUE
Timeout	2

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: HTTP GET

Path: /ready

Port: 3000

Initial Delay: 3 seconds

How long to wait after the container starts before checking its health.

Timeout: 2 seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.

Figure 16.6: Adding a readiness probe

Click Save to create the readiness probe.

- On the next page, click the latest deployment from the table to access the latest deployment configuration.



NOTE

The change causes a new deployment to trigger because the probe updates the deployment configuration.

- 7. Create a liveness probe by navigating to Actions → Edit 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
Port	3000
Initial Delay	3

FIELD	VALUE
Timeout	3

Click Save to create the liveness probe.

Liveness Probe

A liveness probe checks if the container is still running. If the liveness probe fails, the container is killed.

* Type
HTTP GET

Use HTTPS

Path
/healtz

* Port
3000

Initial Delay
3 seconds

How long to wait after the container starts before checking its health.

Timeout
3 seconds

How long to wait for the probe to finish. If the time is exceeded, the probe is considered failed.

[Remove Liveness Probe](#)

Figure 16.7: Adding liveness probes



NOTE

The change causes a new deployment to trigger because the probe updates the deployment configuration.

- 8. 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.

Monitoring

All Filter by name Hide older resources

Events ▲ 2 warnings View Details

pods/probes-3-fvh7b Unhealthy
Liveness probe failed: HTTP probe failed w... a few seconds ago 5 times in the last minute

pods/probes-3-fvh7b Pulling
pulling Image "docker-registry.default.svc... a few seconds ago 2 times in the last minute

Pods

probes-3-fvh7b created 2 minutes ago Running - 1/1 ready probes/probes d5aab3f

Figure 16.8: Viewing monitoring events

- 9. Click the **View Details** link on the upper-right corner to access the monitoring events. Review the events.

Events				
<input type="text"/> Filter by keyword				
Time	Name	Kind	Reason	Message
1:17:15 PM	probes-4-jdtzg	Pod	⚠️ Unhealthy	Liveness probe failed: HTTP probe failed with statuscode: 404 5 times in the last 2 minutes
1:17:07 PM	probes-4-jdtzg	Pod	Created	Created container with docker id 689ecca6e128; Security:[seccomp=unconfined]

Figure 16.9: Viewing events

- 10. 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 \
--sort-by=.metadata.creationTimestamp' \
| grep 'probe failed'
... output omitted ...
... Liveness probe failed: HTTP probe failed with statuscode: 404
```

- 11. Edit the liveness probe.

- 11.1. Update the liveness probe to query the correct URL. From the web console, 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 entry, replace **/healtz** with **/health**.

Click Save to save your changes.



NOTE

The change causes a new deployment to trigger because the probe updates the deployment configuration.

- 12. From the terminal, rerun the **oc get events** command. Notice how OpenShift deployed a new version of the deployment configuration, and how the previous pods are killed. Ensure that you do not see any more entries at the bottom of the screen about the pods being unhealthy. The last line should not indicate that the pods are being killed, which indicates that the deployment is healthy.

```
[student@workstation ~]$ oc get events \
--sort-by=.metadata.creationTimestamp'
... output omitted ...
... Successfully pulled image "docker-registry.default.svc:5000...
... Created kubelet, node1.lab.example.com    Created container
... Started kubelet, node1.lab.example.com    Started container
```

Clean up

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.

You can use the web console to monitor critical properties in your 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 and selecting a project, the web console provides an overview of your projects project:

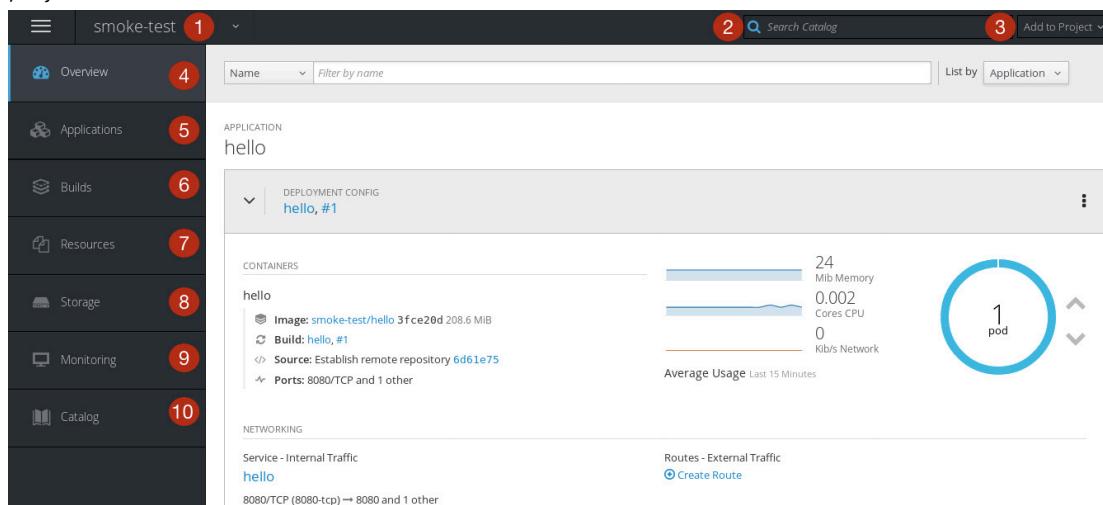


Figure 16.10: The web console overview

1. The project selector allows you to switch between projects that you are authorized to access.
2. The Search Catalog link allows you to browse the catalog of images.
3. The Add to project link allows you to add new resources and applications to the project. You can import resources from files or from existing projects.
4. 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.

5. The Applications tab provides access to deployments, pods, services, and routes. It also gives access to *Stateful Sets*, a Kubernetes feature that provides a unique identity to pods for managing the ordering of deployments.
6. The Builds tab provides access to builds and image streams.
7. The Resources tab provides access to quota management and various resources such as roles and endpoints.
8. The Storage tab provides access to persistent volumes and storage requests.
9. 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.
10. The Catalog tab provides access to the templates that you can use to deploy application bundles.

The web console provides a consolidated view of an application by linking the various elements that it contains. For example, the following figures show a running pod, a readiness prob, its template, and the volumes it uses.

hawkular-cassandra-1-cwmh2 created 18 hours ago 1

Actions 4

Metrics **Environment** **Details** **Logs** **Terminal** **Events**

Status

Status:	Running 3
Replication Controller:	hawkular-cassandra-1
IP:	10.129.0.17
Node:	node2.lab.example.com (172.25.250.12)
Restart Policy:	Always

Container hawkular-cassandra-1

State:	Running since Aug 14, 2018 3:58:44 PM
Ready:	true
Restart Count:	0

Template

Containers

hawkular-cassandra-1 5

- Image: openshift3/ose-metrics-cassandra 6
- Command: /opt/apache-cassandra/bin/cassandra-docker.sh --cluster_name=hawkular-metrics - - ... See All
- Ports: 7000/TCP (tcp-port), 7001/TCP (ssl-port), 9042/TCP (cql-port), 9160/TCP (thrift-port)
- Mount: cassandra-data → /cassandra_data read-write
- Mount: hawkular-cassandra-certs → /hawkular-cassandra-certs read-write
- Mount: cassandra-token-76x98 → /var/run/secrets/kubernetes.io/serviceaccount read-only
- Memory: 750 MB to 2 GB
- Readiness Probe: /opt/apache-cassandra/bin/cassandra-docker-ready.sh 1s timeout

Volumes

cassandra-data

Type:	persistent volume claim (reference to a persistent volume claim)
Claim name:	metrics-1
Mode:	read-write

Figure 16.11: Pods details overview

Figure 16.12: Pods details overview

Annotations:

- `openshift.io/scc`: restricted

Secrets:

- Type: secret (populated by a secret when the pod is created)
- Secret: `hawktl-cassandra-certs`

Annotations:

- `openshift.io/scc`: restricted

Figure 16.13: Pods details overview

1. The name of the pod.
2. Application labels.
3. Overall status of the pod.
4. Available actions for the pod. For example, developers can attach storage to their pod.
5. Template specifications.
6. Monitoring probes declared for the application.

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 your OpenShift cluster, you 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 three images show the various charts that you can consult after the deployment of the metrics subsystem.

Memory

584 Available of 1907.3486328125 MIB

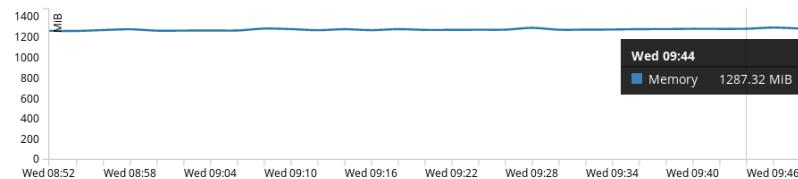


Figure 16.14: Monitoring pods with graphs

Network

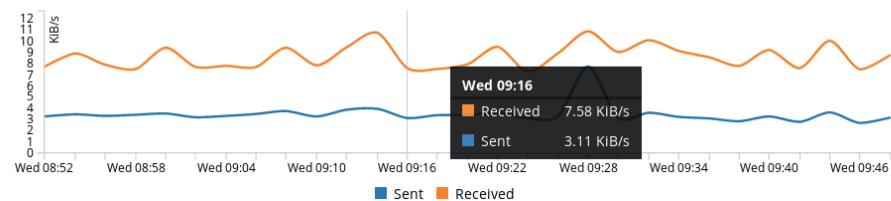


Figure 16.15: Monitoring pods with graphs

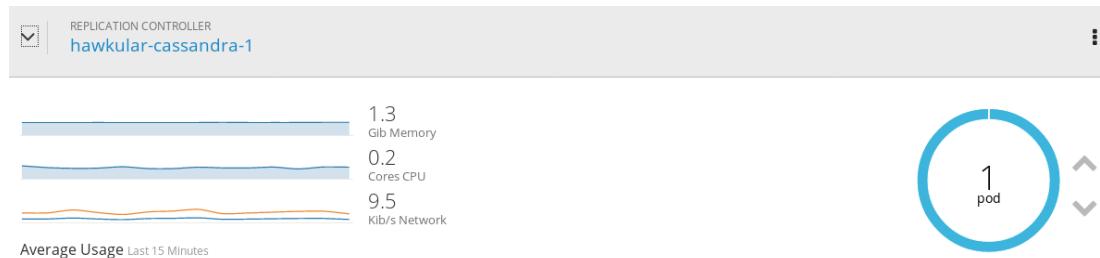


Figure 16.16: Overview of pods

Managing Deployments and Pods

The Actions button, available for pods and deployment, allows you to manage various settings. For example, you can add storage or a health check (which includes readiness and liveness probes) to a deployment. The button also gives access to a YAML editor for updating the configuration in real time through the web console.



Figure 16.17: Managing deployments

Managing Storage

The web console gives you access to storage management. You can use the interface to create volume claims to use volumes that are exposed to the projects. 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 an administrator, you can use the web console to create a claim. The interface supports the use of selectors and labels attributes.

The following image shows how you can manage storage via the web console.

The screenshot shows the 'Create Persistent Volume Claim' form. It includes fields for Name (database), Access Mode (Shared Access (RWX) selected), Size (16 GiB), Labels (datacenter:east), and a checkbox for using label selectors. A 'Labels' section at the bottom right provides a link to 'About Labels'.

Figure 16.18: Creating a persistent volume claim

After a volume claim is defined, the console displays the persistent volume it uses, as defined by the administrator. The following image shows a volume claim bound to the **metrics** volume and one pending persistent volume claim.

The screenshot shows the 'Storage' page with a table of persistent volume claims. The table has columns: Name, Status, Capacity, Access Modes, and Age. It lists two entries: 'database' (Pending, - capacity, RWX access mode, a few seconds old) and 'metrics-1' (Bound to volume metrics, 5 GiB capacity, RWO access mode, 18 hours old).

Name	Status	Capacity	Access Modes	Age
database	Pending	-	RWX (Read-Write-Many)	a few seconds
metrics-1	Bound to volume metrics	5 GiB	RWO (Read-Write-Once)	18 hours

Figure 16.19: Listing available persistent volume claims

Use the Add Storage entry in the Actions menu to add storage to the deployment configuration.

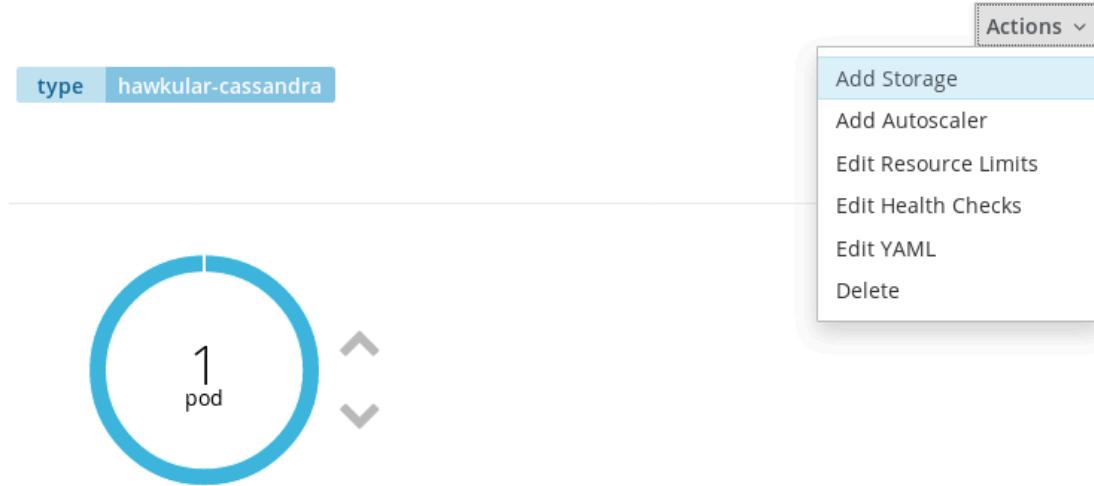


Figure 16.20: Adding storage for a deployment

This option allows you to add an existing persistent volume claim to the template of a deployment configuration. After selecting this option, you 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. You can also specify the volume name, or let the console generate a name.

Replication Controllers » hawkular-cassandra-1 » Add Storage

Add Storage to hawkular-cassandra-1

Add an existing persistent volume claim to the template of replication controller hawkular-cassandra-1.

* Storage

- database 16 GiB (Read-Write-Many) Pending
- metrics-1 5 GiB (Read-Write-Once) Bound to volume **metrics**

Select storage to use or [create storage](#).

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path

/var/lib/data

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Figure 16.21: Storage definition

Adding storage triggers a new deployment for the deployment configuration. The new deployment instructs the pods to mount the volume.

Containers

load

- Image: db-load/load 7cbe001 208.6 MIB
- Build: load, #1
- Source: Establish remote repository bf7e7e9 authored by root
- Ports: 8080/TCP, 8443/TCP
- Mount: database → /data read-write

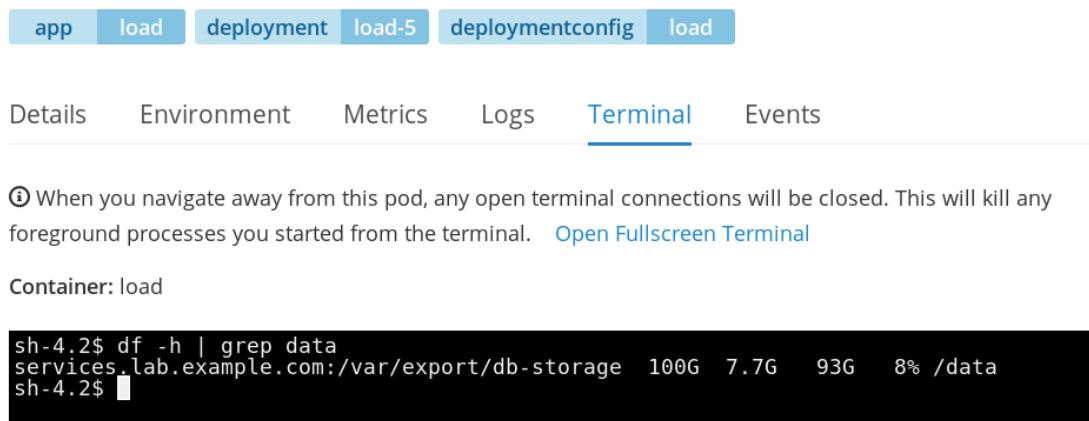
Volumes

database

Type: persistent volume claim (reference to a persistent volume claim)
 Claim name: database
 Mode: read-write

[Add Storage](#) | [Add Config Files](#)

Figure 16.22: Overview of volumes attached to a pod



The screenshot shows the OpenShift web interface for managing pods. At the top, there's a navigation bar with tabs: app, load, deployment, load-5, deploymentconfig, and load. Below the navigation bar, there are tabs for Details, Environment, Metrics, Logs, Terminal (which is underlined in blue), and Events. In the main content area, there's a note: "When you navigate away from this pod, any open terminal connections will be closed. This will kill any foreground processes you started from the terminal." followed by a link to "Open Fullscreen Terminal". Below this note, it says "Container: load" and shows a terminal window with the command "sh-4.2\$ df -h | grep data" and its output: "services.lab.example.com:/var/export/db-storage 100G 7.7G 93G 8% /data".

Figure 16.23: 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 7, *Installing OpenShift Container Platform* and Chapter 15, *Installing and Configuring the Metrics Subsystem* should be completed and you should have an OpenShift Container Platform cluster running with a master, two nodes, and the metrics subsystem. 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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

Upon the installation of the cluster, follow the steps detailed in Chapter 15, *Installing and Configuring the Metrics Subsystem* to deploy the metrics subsystem.

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
```

► 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 redhat \
https://master.lab.example.com
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** project.

```
[student@workstation ~]$ oc new-project load
Now using project "load" on server "https://master.lab.example.com:443".
```

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. Create the load application using the **node-hello** repository located at <http://services.lab.example.com/node-hello>.

```
[student@workstation ~]$ oc new-app --name=load \
http://services.lab.example.com/node-hello
...output omitted...
--> Success
Build scheduled, use 'oc logs -f bc/load' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/load'
Run 'oc status' to view your app.
```

- 2.2. Create a route for the application:

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

- 2.3. Wait until the application pod is ready and running:

```
[student@workstation ~]$ oc get pod
NAME        READY     STATUS    RESTARTS   AGE
load-1-build 0/1      Completed  0          1m
load-1-df4z9 1/1      Running   0          38s
```

- ▶ 3. If not present, install the **httpd-tools** package on **workstation** 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 install httpd-tools
```

- 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.apps.lab.example.com/>.

```
[student@workstation ~]$ ab -n 3000000 -c 20 \
http://load-load.apps.lab.example.com/
```

```
...output omitted...
Benchmarking load-load.apps.lab.example.com (be patient)
Completed 300000 requests
...output omitted...
```

- 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`. If prompted, accept the self-signed certificate.
 - 4.2. Log in to the web console using **developer** as the user name, and **redhat** as the password.
 - 4.3. Click the **load** project to access its overview.
 - 4.4. Review the Overview page. Ensure that there is one pod running, indicated by a blue donut. Click the arrow next to the deployment configuration **load**, #1 and 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.

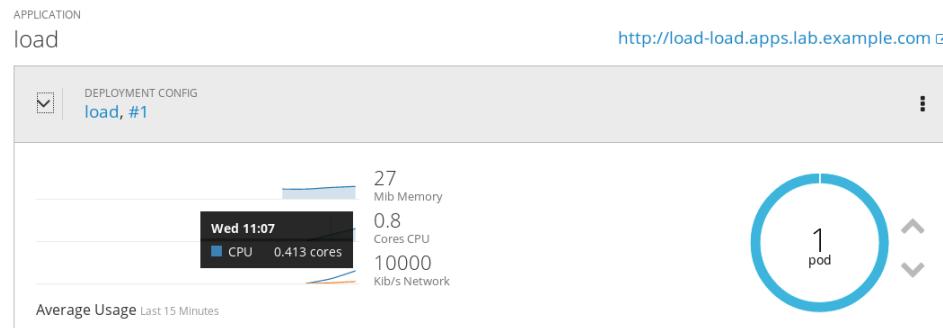


Figure 16.24: The Overview page

- 4.5. Click the upward pointing arrow next to the blue donut to scale up the number of pods for this application to two pods.
- 4.6. Navigate to Applications → Deployments to access the deployments for the project.

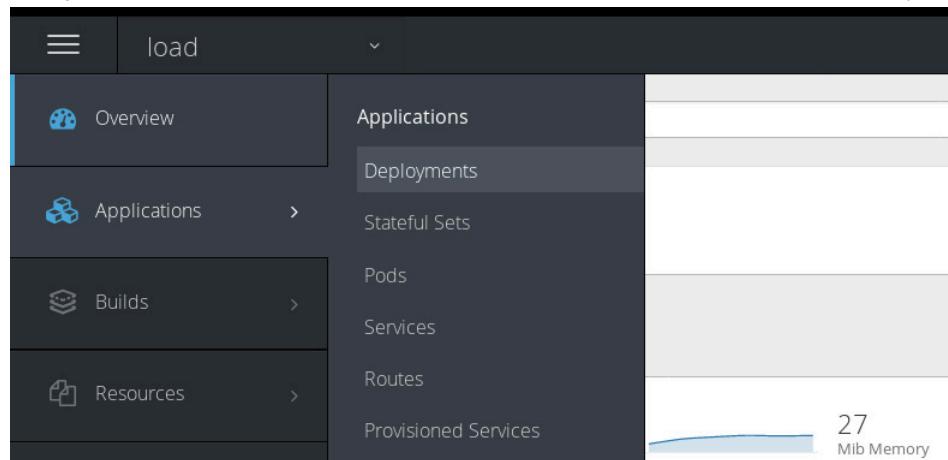


Figure 16.25: The Deployments page

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.

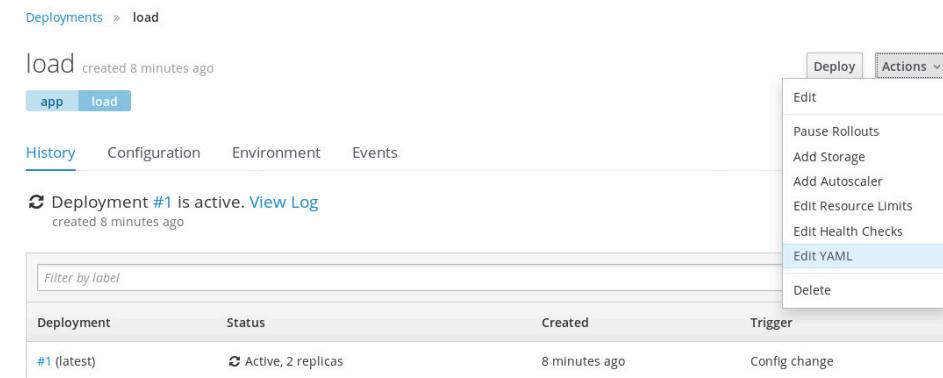


Figure 16.26: Editing YAML

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.



NOTE

If the graphs do not show any value, wait a couple of minutes before retrying.

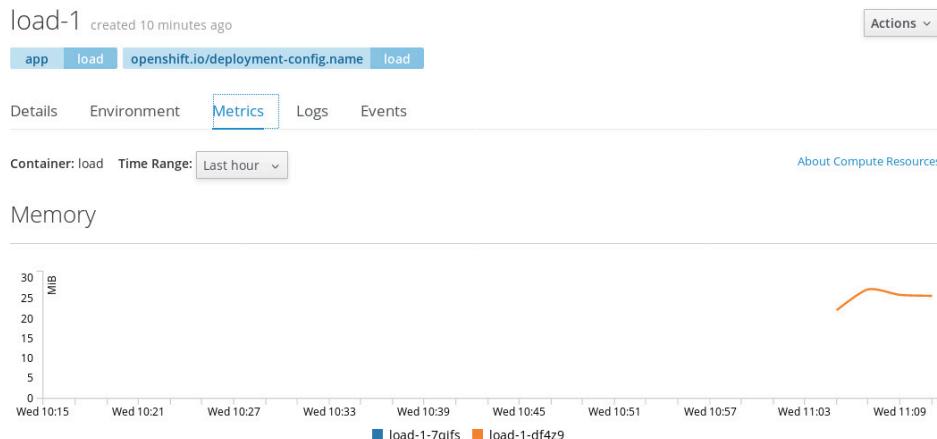


Figure 16.27: Metrics for the application

- 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. Scroll down to access the deployments, and click the arrow next to the deployment name to open the frame. There should be three graphs below the logs: 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 pods use.

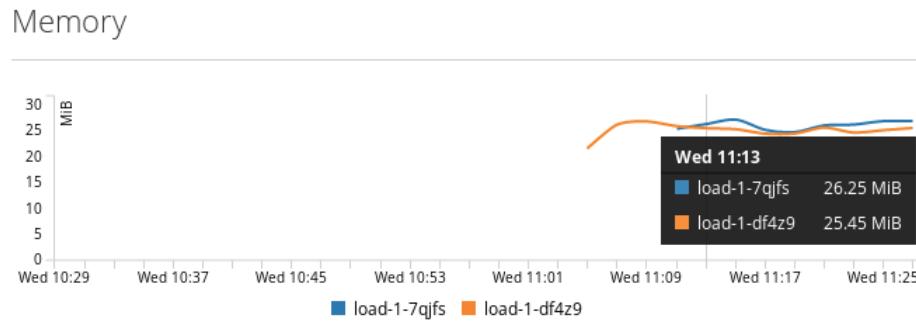


Figure 16.28: Metrics for the deployment

- 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.

Storage > Create Storage

Create Storage

Create a request for an administrator-defined storage asset by specifying size and permissions for a best fit. [Learn More](#)

* Name: web-storage

A unique name for the storage claim within the project.

* Access Mode:

- Single User (RWO)
- Shared Access (RWX)
- Read Only (RO)

Permissions to the mounted volume.

* Size:

1 GiB

Desired storage capacity. [What are GiB?](#)

Use label selectors to request storage. [Learn More](#)

Create **Cancel**

Figure 16.29: Creating a volume claim

Click **Create** to create the persistent volume claim. On the next page, ensure that the claims transitions from **Pending** to **Bound to volume web-storage**.

► 8. Add storage to your application.

- 8.1. Navigate to Applications → Deployments to manage your deployments.
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.

* Storage

web-storage 2 GiB (Read-Write-Many) Bound to volume **web-storage**

Select storage to use or [create storage](#).

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path

/web-storage

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Subpath

example: application/resources

Optional path within the volume from which it will be mounted into the container. Defaults to the volume's root.

Volume Name

web-storage

Figure 16.30: 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, indicated by (**latest**). 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 at the bottom, 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
services.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
```

Clean up

Run the **lab web-console cleanup** command to delete the **load** project and the persistent volume used by the project.

```
[student@workstation ~]$ lab web-console cleanup
```

This concludes the guided exercise.

► LAB

MANAGING AND MONITORING OPENSIFT

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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

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/D0285/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 located at **services.lab.example.com** with an image stream of **PHP:5.6**. 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 and look for the entry that indicates that the request was rejected because of limit violation. This should also prevent the creation of a new version for the deployment. Revert the request to 200 MiB.

6. In the `/home/student/D0285/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.

Liveness Probe Properties

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

MANAGING AND MONITORING OPENSIFT

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 the chapter *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-prepare setup
[student@workstation ~]$ cd /home/student/do285-ansible
[student@workstation do285-ansible]$ ./install.sh
```

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
```

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 redhat \
https://master.lab.example.com
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:443".
```

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/D0285/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/D0285/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.
...output omitted...
```

- 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:443".
```

- 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/D0285/labs/monitor-review/limits.yml
limitrange "review-limits" created
```

- 2.5. Run the **oc describe** command to review the limits created:

```
[student@workstation ~]$ oc describe 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 located at `services.lab.example.com` with an image stream of **PHP:5.6**. 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 redhat \
https://master.lab.example.com
```

3.2. Use the **oc new-app** command to create the **load** application:

```
[student@workstation ~]$ oc new-app --name=load \
http://services.lab.example.com/node-hello
...output omitted...
--> Success
Build scheduled, use 'oc logs -f bc/load' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/load'
Run 'oc status' to view your app.
```

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:



NOTE

It may take up to 2 minutes for the pod to deploy.

```
[student@workstation ~]$ oc get pods
NAME        READY     STATUS    RESTARTS   AGE
load-1-build 0/1      Completed  0          3m
load-1-cgz2x 1/1      Running   0          2m
```

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-cgz2x
...output omitted...
Limits:
  memory: 200Mi
Requests:
  memory: 200Mi
...output omitted...
```

5. Update the limit for the deployment configuration by requesting 350 MiB. Review the events in the project and look for the entry that indicates that the request was rejected because of limit violation. This should also prevent the creation of a new version for 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
...output omitted...
... Error creating: Pod "load-2-q9hm9" is invalid:
spec.containers[0].resources.requests: Invalid value: "350Mi": must be less than
or equal to memory limit
```

- 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:443

svc/load - 172.30.110.135:3000
dc/load deploys istag/load:latest >-
bc/load docker builds http://registry.lab.example.com/node-hello on istag/
nodejs-6-rhel7:latest
deployment #3 deployed 24 seconds ago - 1 pod
deployment #2 failed about a minute ago: newer deployment was found running
deployment #1 deployed 6 minutes ago

2 infos identified, use 'oc status -v' to see details.
NAME        READY     STATUS    RESTARTS   AGE
load-1-build 0/1      Completed  0          6m
load-3-fp788 1/1      Running   0          19s
```

6. In the **/home/student/D0285/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/D0285/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 200Mi  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 redhat
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:

NAME	READY	STATUS	RESTARTS	AGE
load-1-build	0/1	Completed	0	8m
load-3-fp788	1/1	Running	0	2m
load-3-jlkwj	1/1	Running	0	12s
load-3-mdr2g	1/1	Running	0	12s

- 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
...output omitted...
```

```
... Error creating: pods "load-3-qwh7m" 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.apps.lab.example.com** as the host name:

```
[student@workstation ~]$ oc expose svc load \
--hostname=load-review.apps.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.

Liveness Probe Properties

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>. Use **developer** as the user name, and **redhat** 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: automated in-place upgrades via Ansible Playbooks, and upgrading using a blue-green deployment method.
- Cluster upgrades cannot span more than one minor version at a time, so if the cluster is at a version earlier than 3.6, you must first upgrade incrementally. For example, 3.5 to 3.6, then 3.6 to 3.7. Failure to do so may result in upgrade failure.
- 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 17

COMPREHENSIVE REVIEW OF INTRODUCTION TO CONTAINERS, KUBERNETES, AND RED HAT OPENSHIFT

GOAL

Demonstrate how to containerize a software application, test it with Docker, and deploy it on an OpenShift cluster.

OBJECTIVE

Review concepts in the course to assist in completing the comprehensive review lab.

SECTIONS

- Comprehensive Review

LAB

- Deploying a Software Application

COMPREHENSIVE REVIEW

OBJECTIVES

After completing this section, students should be able to demonstrate knowledge and skills learned in *Containers, Kubernetes, and Red Hat OpenShift Administration I*.

REVIEWING CONTAINERS, KUBERNETES, AND RED HAT OPENSHIFT ADMINISTRATION I

Before beginning the comprehensive review lab for this course, students should be comfortable with the topics covered in the following chapters.

Chapter 1, Describing Container Technology

Describe how software can run in containers orchestrated by Red Hat OpenShift Container Platform.

- Describe the architecture of Linux containers.
- Describe how containers are implemented using Docker.
- Describe the architecture of a Kubernetes cluster running on the Red Hat OpenShift Container Platform.

Chapter 2, Creating Containerized Services

Provision a server using container technology.

- Create a database server from a container image stored on Docker Hub.

Chapter 3, Managing Containers

Manipulate pre-built container images to create and manage containerized services.

- Manage the life cycle of a container from creation to deletion.
- Save application data across container restarts through the use of persistent storage.
- Describe how Docker provides network access to containers, and access a container through port forwarding.

Chapter 4, Managing Container Images

Manage the life cycle of a container image from creation to deletion.

- Search for and pull images from remote registries.
- Export, import, and manage container images locally and in a registry.

Chapter 5, Creating Custom Container Images

Design and code a Dockerfile to build a custom container image.

- Describe the approaches for creating custom container images.
- Create a container image using common Dockerfile commands.

Chapter 9, Deploying Containerized Applications on OpenShift

Deploy single container applications on OpenShift Container Platform.

- Create standard Kubernetes resources.
- Build an application using the Source-to-Image facility of OpenShift.
- Create a route to a service.
- Create an application using the OpenShift web console.

Chapter 6, Deploying Multi-Container Applications

Deploy containerized applications using multiple container images.

- Describe the considerations for containerizing applications with multiple container images.
- Deploy a multi-container application with user-defined Docker network.
- Troubleshoot an application build and deployment on OpenShift.
- Implement techniques for troubleshooting and debugging containerized applications.

GENERAL CONTAINER, KUBERNETES, AND OPENSHIFT HINTS

These hints may save some time in completing the comprehensive review lab:

- The **docker** command allows you to build, run, and manage container images. Docker command documentation can be found by issuing the command **man docker**.
- The **oc** command allows you to create and manage OpenShift resources. OpenShift command-line documentation can be found by issuing either of the commands **man oc** or **oc help**. OpenShift commands that are particularly useful include:

oc login -u

Log in to OpenShift as the specified user. In this classroom, there are two user accounts defined: **admin** and **developer**.

oc new-project

Create a new project (*namespace*) to contain OpenShift resources.

oc project

Select the current project (namespace) to which all subsequent commands apply.

oc create -f

Create a resource from a file.

oc process -f

Convert a template into OpenShift resources that can be created with the **oc create** command.

oc get

Display the runtime status and attributes of OpenShift resources.

oc describe

Display detailed information about OpenShift resources.

oc delete

Delete OpenShift resources. The label option, **-l label-value** is helpful with this command to delete multiple resources simultaneously.

- Before mounting any volumes on the Docker and OpenShift host, make sure to apply the correct SELinux context to the directory. The correct context is **svirt_sandbox_file_t**. Also, make sure the ownership and permissions of the directory are set according to the **USER** directive in the **Dockerfile** that was used to build the container being deployed. Most of the time you will have to use the numeric UID and GID rather than the user and group names to adjust ownership and permissions of the volume directory.
- In this classroom, all RPM repositories are defined locally. You must configure the repository definitions in a custom container image (**Dockerfile**) before running **yum** commands.
- When executing commands in a **Dockerfile**, combine as many related commands as possible into one **RUN** directive. This reduces the number of UFS layers in the container image.
- A best practice for designing a **Dockerfile** includes the use of environment variables for specifying repeated constants throughout the file.

► LAB

CONTAINERIZING AND DEPLOYING A SOFTWARE APPLICATION

In this review, you will containerize a Nexus Server, build and test it using Docker, and deploy it on an OpenShift cluster.

OUTCOMES

You should be able to:

- Write a **Dockerfile** that successfully containerizes a Nexus server.
- Build a Nexus server container image that deploys using Docker.
- Deploy the Nexus server container image to an OpenShift cluster.

BEFORE YOU BEGIN

Run the setup script for this comprehensive review.

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

The lab files are located in the `/home/student/D0285/labs/review` directory. The solution files are located in the `/home/student/D0285/solutions/review` directory.

INSTRUCTIONS

Create a Docker container image that starts an instance of a Nexus server:

- The server should run as the **nexus** user and group. They have a UID and GID of **1001**, respectively.
- The server requires that the `java-1.8.0-openjdk-devel` package be installed. The RPM repositories are configured in the provided **training.repo** file. Be sure to add this file to the container in the `/etc/yum.repos.d` directory.
- The server is provided as a compressed tar file: **nexus-2.14.3-02-bundle.tar.gz**. The file can be retrieved from the server at http://content.example.com/ocp3.9/x86_64/installers/{tarball_name}. A script is provided to retrieve the tar bundle before you build the image: **get-nexus-bundle.sh**.
- Run the following script to start the nexus server: **nexus-start.sh**.
- The working directory and home for the nexus installation should be **/opt/nexus**. The version-specific nexus directory should be linked to a directory named **nexus2**.
- Nexus produces persistent data at **/opt/nexus/sonatype-work**. Make sure this can be mounted as a volume. You may want to initially build and test your container image without a persistent volume and add this in a second pass.
- There are two snippet files in the **image** lab directory that provide the commands needed to create the nexus account and install Java. Use these snippets to assist you in writing the **Dockerfile**.

Build and test the container image using Docker with and without a volume mount. In the directory, **/home/student/D0285/labs/deploy/docker**, there is a shell script to assist you in running the container with a volume mount. Remember to inspect the running container to determine its IP address. Use **curl** as well as the container logs to determine if the Nexus server is running properly.

```
[student@workstation review]$ docker logs -f {container-id}
[student@workstation review]$ curl http://{ipaddress}:8081/nexus/
```

Deploy the Nexus server container image into the OpenShift cluster:

- Publish the container image to the classroom private registry at `registry.lab.example.com`.
- The **/home/student/D0285/labs/review/deploy/openshift** directory contains several shell scripts, Kubernetes resource definitions, and an OpenShift template to help complete the lab.

- **create-pv.sh**

This script creates the Kubernetes persistent volume that stores the Nexus server persistent data.

- **delete-pv.sh**

This script deletes the Kubernetes persistent volume.

- **resources/pv.yaml**

This is a Kubernetes resource file that defines the persistent volume. This file is used by the **create-pv.sh** script.

- **resources/nexus-template.json**

This is an OpenShift template to deploy the Nexus server container image.

- Several helpful scripts are located in the **/home/student/D0285/solutions/review** directory for this lab that can help you deploy and undeploy the application if you are unsure how to proceed.
- Remember to push the container image to the classroom private registry at `registry.lab.example.com`. The container must be named **nexus** and have a tag **latest**. The OpenShift template expects this name.
- Use **review** for the OpenShift project name. Execute the **setpolicy.sh** shell script, located at **/home/student/D0285/labs/review/deploy/openshift**, after creating the project. The script sets the right privileges for the container.
- Make sure to create the persistent volume with the provided shell script before deploying the container with the template.
- Expose the Nexus server service as a route using the default route name. Test the server using a browser.
- Run the grading script to evaluate your work:

```
[student@workstation review]$ lab review grade
```

- Clean up your project by deleting the OpenShift project created in this lab and removing the local copy of the Docker container image as well.

Evaluation

After deploying the Nexus server container image into the OpenShift cluster, verify your work by running the lab grading script:

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

Cleanup

Delete the **review** project. Also delete the persistent volume using **delete-pv.sh** script.

```
[student@workstation docker]$ cd ~/student/D0285/labs/review/deploy/openshift  
[student@workstation review]$ oc delete project review  
[student@workstation review]$ ./delete-pv.sh
```

This concludes the lab.

► SOLUTION

CONTAINERIZING AND DEPLOYING A SOFTWARE APPLICATION

In this review, you will containerize a Nexus Server, build and test it using Docker, and deploy it on an OpenShift cluster.

OUTCOMES

You should be able to:

- Write a **Dockerfile** that successfully containerizes a Nexus server.
- Build a Nexus server container image that deploys using Docker.
- Deploy the Nexus server container image to an OpenShift cluster.

BEFORE YOU BEGIN

Run the setup script for this comprehensive review.

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

The lab files are located in the `/home/student/D0285/labs/review` directory. The solution files are located in the `/home/student/D0285/solutions/review` directory.

INSTRUCTIONS

Create a Docker container image that starts an instance of a Nexus server:

- The server should run as the **nexus** user and group. They have a UID and GID of **1001**, respectively.
- The server requires that the `java-1.8.0-openjdk-devel` package be installed. The RPM repositories are configured in the provided **training.repo** file. Be sure to add this file to the container in the `/etc/yum.repos.d` directory.
- The server is provided as a compressed tar file: **nexus-2.14.3-02-bundle.tar.gz**. The file can be retrieved from the server at http://content.example.com/ocp3.9/x86_64/installers/{tarball_name}. A script is provided to retrieve the tar bundle before you build the image: **get-nexus-bundle.sh**.
- Run the following script to start the nexus server: **nexus-start.sh**.
- The working directory and home for the nexus installation should be **/opt/nexus**. The version-specific nexus directory should be linked to a directory named **nexus2**.
- Nexus produces persistent data at **/opt/nexus/sonatype-work**. Make sure this can be mounted as a volume. You may want to initially build and test your container image without a persistent volume and add this in a second pass.
- There are two snippet files in the **image** lab directory that provide the commands needed to create the nexus account and install Java. Use these snippets to assist you in writing the **Dockerfile**.

Build and test the container image using Docker with and without a volume mount. In the directory, **/home/student/D0285/labs/deploy/docker**, there is a shell script to assist you in running the container with a volume mount. Remember to inspect the running container to determine its IP address. Use **curl** as well as the container logs to determine if the Nexus server is running properly.

```
[student@workstation review]$ docker logs -f {container-id}
[student@workstation review]$ curl http://{ipaddress}:8081/nexus/
```

Deploy the Nexus server container image into the OpenShift cluster:

- Publish the container image to the classroom private registry at `registry.lab.example.com`.
- The **/home/student/D0285/labs/review/deploy/openshift** directory contains several shell scripts, Kubernetes resource definitions, and an OpenShift template to help complete the lab.

- **create-pv.sh**

This script creates the Kubernetes persistent volume that stores the Nexus server persistent data.

- **delete-pv.sh**

This script deletes the Kubernetes persistent volume.

- **resources/pv.yaml**

This is a Kubernetes resource file that defines the persistent volume. This file is used by the **create-pv.sh** script.

- **resources/nexus-template.json**

This is an OpenShift template to deploy the Nexus server container image.

- Several helpful scripts are located in the **/home/student/D0285/solutions/review** directory for this lab that can help you deploy and undeploy the application if you are unsure how to proceed.
- Remember to push the container image to the classroom private registry at `registry.lab.example.com`. The container must be named **nexus** and have a tag **latest**. The OpenShift template expects this name.
- Use **review** for the OpenShift project name. Execute the **setpolicy.sh** shell script, located at **/home/student/D0285/labs/review/deploy/openshift**, after creating the project. The script sets the right privileges for the container.
- Make sure to create the persistent volume with the provided shell script before deploying the container with the template.
- Expose the Nexus server service as a route using the default route name. Test the server using a browser.
- Run the grading script to evaluate your work:

```
[student@workstation review]$ lab review grade
```

- Clean up your project by deleting the OpenShift project created in this lab and removing the local copy of the Docker container image as well.

1. Write a **Dockerfile** that containerizes the Nexus server. Go to **/home/student/D0285/labs/review/image** directory and create the **Dockerfile**.

- 1.1. Specify the base image to use:

```
FROM rhel7:7.5
```

- 1.2. Enter arbitrary name and email as the maintainer:

```
FROM rhel7:7.5
MAINTAINER username <username@example.com>
```

- 1.3. Set the environment variables for **NEXUS_VERSION** and **NEXUS_HOME**:

```
FROM rhel7:7.5
MAINTAINER username <username@example.com>

ENV NEXUS_VERSION=2.14.3-02 \
    NEXUS_HOME=/opt/nexus
```

- 1.4. Add the **training.repo** to the **/etc/yum.repos.d** directory. Install the Java package using **yum** command.

```
...
ENV NEXUS_VERSION=2.14.3-02 \
    NEXUS_HOME=/opt/nexus

ADD training.repo /etc/yum.repos.d/training.repo
RUN yum install -y --setopt=tsflags=nodocs \
    java-1.8.0-openjdk-devel && \
    yum clean all -y
```

- 1.5. Create the server home directory and service account/group.

```
...
RUN groupadd -r nexus -f -g 1001 && \
    useradd -u 1001 -r -g nexus -m -d ${NEXUS_HOME} \
        -s /sbin/nologin \
        -c "Nexus User" nexus && \
    chown -R nexus:nexus ${NEXUS_HOME} && \
    chmod -R 755 ${NEXUS_HOME}
```

- 1.6. Install the Nexus server software at **NEXUS_HOME** and add the start-up script. Note that the **ADD** directive will extract the Nexus files. Create the **nexus2** symbolic link pointing to the Nexus server directory.

```
...
ADD nexus-${NEXUS_VERSION}-bundle.tar.gz ${NEXUS_HOME}/
ADD nexus-start.sh ${NEXUS_HOME}/
```

```
RUN ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} \
${NEXUS_HOME}/nexus2 && \
chown -R nexus:nexus ${NEXUS_HOME}
```

- 1.7. Make the container run as the *nexus* user and make the working directory **/opt/nexus**:

```
...
USER nexus
WORKDIR ${NEXUS_HOME}
```

- 1.8. Define a volume mount point to store the Nexus server persistent data:

```
...
VOLUME ["/opt/nexus/sonatype-work"]
```

- 1.9. Execute the Nexus server shell script. The completed **Dockerfile** should read as follows:

```
FROM rhel7:7.5
MAINTAINER New User <user1@myorg.com>

ENV NEXUS_VERSION=2.14.3-02 \
NEXUS_HOME=/opt/nexus

ADD training.repo /etc/yum.repos.d/training.repo
RUN yum --noplugins update -y && \
yum --noplugins install -y --setopt=tsflags=nodocs \
java-1.8.0-openjdk-devel && \
yum --noplugins clean all -y

RUN groupadd -r nexus -f -g 1001 && \
useradd -u 1001 -r -g nexus -m -d ${NEXUS_HOME} \
-s /sbin/nologin \
-c "Nexus User" nexus && \
chown -R nexus:nexus ${NEXUS_HOME} && \
chmod -R 755 ${NEXUS_HOME}

ADD nexus-${NEXUS_VERSION}-bundle.tar.gz ${NEXUS_HOME}/
ADD nexus-start.sh ${NEXUS_HOME}/

RUN ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} \
${NEXUS_HOME}/nexus2 && \
chown -R nexus:nexus ${NEXUS_HOME}

USER nexus
WORKDIR ${NEXUS_HOME}

VOLUME ["/opt/nexus/sonatype-work"]

CMD ["sh", "nexus-start.sh"]
```

2. Retrieve the Nexus server files to the **image** directory by using the script **get-nexus-bundle.sh**. Upon retrieval, build the container image with tag **nexus**:

```
[student@workstation ~]$ cd /home/student/D0285/labs/review/image
[student@workstation image]$ ./get-nexus-bundle.sh
[student@workstation image]$ docker build -t nexus .
```

3. Test the image with Docker using the provided **run-persistent.sh** shell script located under **/home/student/D0285/labs/review/deploy/docker** directory. Replace the container name appropriately as shown in the output of **docker ps** command.

```
[student@workstation images]$ cd /home/student/D0285/labs/review/deploy/docker
[student@workstation docker]$ ./run-persistent.sh
80970007036bbb313d8eeb7621fada0ed3f0b4115529dc50da4dccef0da34533
[student@workstation docker]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
80970007036b        nexus              "sh nexus-start.sh"   5 seconds ago    Up 4 seconds       quizzical_fermat
[student@workstation docker]$ docker logs -f quizzical_fermat
...output omitted...
2018-07-05 12:09:31,851+0000 INFO  [jetty-main-1] *SYSTEM
org.sonatype.nexus.webresources.internal.WebResourceServlet - Max-age: 30 days
(2592000 seconds)
2018-07-05 12:09:31,873+0000 INFO  [jetty-main-1] *SYSTEM
org.sonatype.nexus.bootstrap.jetty.InstrumentedSelectChannelConnector - Metrics
enabled
2018-07-05 12:09:31,878+0000 INFO  [pxpool-1-thread-1] *SYSTEM
org.sonatype.nexus.configuration.application.DefaultNexusConfiguration - Applying
Nexus Configuration due to changes in [Scheduled Tasks] made by *TASK...
2018-07-05 12:09:31,887+0000 INFO  [jetty-main-1] *SYSTEM
org.eclipse.jetty.server.AbstractConnector - Started
InstrumentedSelectChannelConnector@0.0.0:8081
2018-07-05 12:09:31,887+0000 INFO  [jetty-main-1] *SYSTEM
org.sonatype.nexus.bootstrap.jetty.JettyServer - Running
2018-07-05 12:09:31,887+0000 INFO  [main] *SYSTEM
org.sonatype.nexus.bootstrap.jetty.JettyServer - Started
Ctrl+C
[student@workstation docker]$ docker inspect \
-f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' \
quizzical_fermat
172.17.0.2
[student@workstation docker]$ curl -v 172.17.0.2:8081/nexus/
* About to connect() to 172.17.0.2 port 8081 (#0)
*   Trying 172.17.0.2...
* Connected to 172.17.0.2 (172.17.0.2) port 8081 (#0)
> GET /nexus HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 172.17.0.2:8081
> Accept: */*
>
< HTTP/1.1 302 Found
< Date: Fri, 15 Jun 2018 22:59:27 GMT
< Location: http://172.17.0.2:8081/nexus/
< Content-Length: 0
```

```
< Server: Jetty(8.1.16.v20140903)
<
* Connection #0 to host 172.17.0.2 left intact
[student@workstation docker]$ docker kill quizzical_fermat
quizzical_fermat
```

4. Publish the Nexus server container image to the classroom private registry:

```
[student@workstation docker]$ docker tag nexus:latest \
registry.lab.example.com/nexus:latest
[student@workstation docker]$ docker push \
registry.lab.example.com/nexus:latest
The push refers to a repository [registry.lab.example.com/nexus]
92ee09aa3f02: Pushed
8759f7e05003: Pushed
fa7c74031ba6: Pushed
91ddbb7de6ee: Pushed
a8011fed33d7: Pushed
86d3cd59a99c: Pushed
d6a4dd6ace1f: Mounted from rhel7
f4fa6c253d2f: Mounted from rhel7
latest: digest:
sha256:b71aff6921e524fa6395b2843b2fcdc1a96ba6a455d1c3f59a783b51a358efff size:
1995
```

5. Deploy the Nexus server container image into the OpenShift cluster using the resources located beneath **/home/student/D0285/labs/review/deploy/openshift** directory.
- 5.1. Create the OpenShift project and set the security context policy by running the **setpolicy.sh** script located beneath **/home/student/D0285/labs/review/deploy/openshift** directory.

```
[student@workstation docker]$ cd ~student/D0285/labs/review/deploy/openshift
```

```
[student@workstation openshift]$ oc login -u developer -p redhat \
https://master.lab.example.com
Login successful.
...output omitted...
[student@workstation openshift]$ oc new-project review
Now using project "review" on server "https://master.lab.example.com:443".
```

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.

```
[student@workstation openshift]$ ./setpolicy.sh
scc "anyuid" added to: ["system:serviceaccount:review=default"]
```

- 5.2. Create the persistent volume that will hold the Nexus server's persistent data:

```
[student@workstation openshift]$ ./create-pv.sh
Warning: Permanently added 'node1,172.25.250.11' (ECDSA) to the list of known
hosts.
```

```
Warning: Permanently added 'node2,172.25.250.12' (ECDSA) to the list of known hosts.
persistentvolume "pv-nexus" created
```

- 5.3. Process the template and create the Kubernetes resources:

```
[student@workstation openshift]$ oc process -f resources/nexus-template.json \
| oc create -f -
service "nexus" created
persistentvolumeclaim "nexus" created
deploymentconfig "nexus" created
[student@workstation openshift]$ oc get pods
NAME          READY     STATUS    RESTARTS   AGE
nexus-1-wk8rv 1/1      Running   1          1m
```

- 5.4. Expose the service by creating a route:

```
[student@workstation openshift]$ oc expose svc/nexus
route "nexus" exposed
[student@workstation openshift]$ oc get route
NAME        HOST/PORT           PATH      SERVICES   PORT
TERMINATION  WILDCARD
nexus       nexus-review.apps.lab.example.com      nexus      nexus
None
```

- 5.5. Use a browser to connect to the Nexus server web application using <http://nexus-review.apps.lab.example.com/nexus/> URL.

Evaluation

After deploying the Nexus server container image into the OpenShift cluster, verify your work by running the lab grading script:

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

Cleanup

Delete the **review** project. Also delete the persistent volume using **delete-pv.sh** script.

```
[student@workstation docker]$ cd ~student/D0285/labs/review/deploy/openshift
[student@workstation review]$ oc delete project review
[student@workstation review]$ ./delete-pv.sh
```

This concludes the lab.

COMPREHENSIVE REVIEW

OBJECTIVES

After completing this section, students should be able to review and refresh knowledge and skills learned in *Containers, Kubernetes, and Red Hat OpenShift Administration I*.

REVIEWING CONTAINERS, KUBERNETES, AND 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 7, *Installing OpenShift Container Platform*

Install OpenShift and configure the cluster.

- Prepare the servers for installation.
- Execute the installation steps to build and configure an OpenShift cluster.
- Execute postinstallation tasks and verify the cluster configuration.

Chapter 8, *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 11, *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 12, *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 13, *Allocating Persistent Storage*

Implement persistent storage.

- Provision persistent storage for use by applications.

- Describe how persistence is configured for the internal container registry.

Chapter 14, 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 15, 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 16, Managing and Monitoring OpenShift Container Platform

Manage and monitor OpenShift resources and software.

- Limit the amount of resources consumed by an application.
- Describe how to upgrade an instance of OpenShift.
- Configure probes to monitor application health.
- Monitor OpenShift resources using data obtained from the web console.

GENERAL OPENSIFT 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. In most cases, **root** is already logged into the cluster.



NOTE

Previous chapter labs and demonstrations contain troubleshooting hints that will be useful if problems occur during the completion of this lab.

► LAB

DEPLOYING 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.
- Build a custom container image for use in the multi-container application.

BEFORE YOU BEGIN

If you do not have an OpenShift cluster running with a master and two nodes, you must complete one of the options in the comprehensive review lab *Installing OpenShift* in the previous section.

Run the following command to verify that your environment is ready to start this review lab and also to download files required for this lab:

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

The **/home/student/D0285/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.

The Red Hat OpenShift Container Platform 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 (**todoui**) 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 **redhat**. These users were created when the OpenShift cluster was installed and configured.

- 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.
- 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 as follows:

- The quota is named **todoapp-quota**.
- The quota places a hard limit of 1 pod.

If you do not wish to enter the commands manually, you may execute the script at **/home/student/D0285/labs/review-deploy/set-quotas.sh** on the **workstation** VM.

- 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://services.lab.example.com/php-helloworld**. The name of the application should be **hello**.

Verify that the deployment of the **php-helloworld** application is successful. Ensure that the application pod is in the *Running* state.

If the deployment fails, troubleshoot why deployment did not succeed, and fix the issues.

To cancel a failed deployment, you can use the **oc rollout cancel** command. To redeploy an application, use the **oc rollout latest** command.

- Delete all the resources for the **php-helloworld** application. Do not delete the **todoapp** project.
- Provision persistent storage for the MySQL database. Create a new NFS share on the **services** 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 script to automate the NFS share creation is provided for you in the **/home/student/D0285/labs/review-deploy/config-nfs.sh** file on the **workstation** VM.

- 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 **services** VM.

A template YAML configuration file for this persistent volume is provided for you in the **/home/student/D0285/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.

- 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/D0285/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.

Import the template into OpenShift, ensuring that the template appears in the OpenShift web console under the **JavaScript** category. This allows developers to create new applications from this template from the web console.

8. The **Dockerfile** for the **todoapi** application and its associated build artifacts are available in the **/home/student/D0285/labs/review-deploy/todoapi** folder on the **workstation** 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.

Build, tag, and push the image to the private registry, **registry.lab.example.com**.

Import the **registry.lab.example.com/todoapp/todoapi:latest** image from the private docker registry into OpenShift. Ensure that an image stream called **todoapi** is created in the **todoapp** namespace.

9. Create a new application, as the **developer** user, 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://services.lab.example.com/todoapi
Application Hostname	todoapi.apps.lab.example.com
MySQL Username	todoapp
MySQL Password	todoapp
Database name	todoappdb
Database Administrator Password	redhat



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.

Verify that the application is built and deployed. You must see two pods, one each for the todoapi application and the MySQL database.

10. Sample data for the TODO List application is provided for you in the **/home/student/D0285/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/D0285/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.

11. 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.apps.lab.example.com/todo/api/host**, which prints the pod name on which the application is running.
12. Verify that the **todoapi** application fetches data from MySQL by using the **curl** command to access the URL **http://todoapi.apps.lab.example.com/todo/api/items**, which prints the list of TODO items in JSON format.
13. On the **workstation** VM, as the **developer** user, create a new application named **todoui** using the **todoui** image stream imported earlier.
Verify that the deployment is successful.
14. Create a new route for the **todoui** service with a *hostname* of **todo.apps.lab.example.com**.

Use a browser on the **workstation** VM to navigate to the home page of the TODO List application at **http://todo.apps.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

DEPLOYING 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.
- Build a custom container image for use in the multi-container application.

BEFORE YOU BEGIN

If you do not have an OpenShift cluster running with a master and two nodes, you must complete one of the options in the comprehensive review lab *Installing OpenShift* in the previous section.

Run the following command to verify that your environment is ready to start this review lab and also to download files required for this lab:

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

The **/home/student/D0285/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.

The Red Hat OpenShift Container Platform 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 (**todoui**) 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 **redhat**. These users were created when the OpenShift cluster was installed and configured.

- 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.
- 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 as follows:

- The quota is named **todoapp-quota**.
- The quota places a hard limit of 1 pod.

If you do not wish to enter the commands manually, you may execute the script at **/home/student/D0285/labs/review-deploy/set-quotas.sh** on the **workstation** VM.

- 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://services.lab.example.com/php-helloworld**. The name of the application should be **hello**.

Verify that the deployment of the **php-helloworld** application is successful. Ensure that the application pod is in the *Running* state.

If the deployment fails, troubleshoot why deployment did not succeed, and fix the issues.

To cancel a failed deployment, you can use the **oc rollout cancel** command. To redeploy an application, use the **oc rollout latest** command.

- Delete all the resources for the **php-helloworld** application. Do not delete the **todoapp** project.
- Provision persistent storage for the MySQL database. Create a new NFS share on the **services** 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 script to automate the NFS share creation is provided for you in the **/home/student/D0285/labs/review-deploy/config-nfs.sh** file on the **workstation** VM.

- 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 **services** VM.

A template YAML configuration file for this persistent volume is provided for you in the **/home/student/D0285/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.

- 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/D0285/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.

Import the template into OpenShift, ensuring that the template appears in the OpenShift web console under the **JavaScript** category. This allows developers to create new applications from this template from the web console.

8. The **Dockerfile** for the **todoapi** application and its associated build artifacts are available in the **/home/student/D0285/labs/review-deploy/todoapi** folder on the **workstation** 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.

Build, tag, and push the image to the private registry, **registry.lab.example.com**.

Import the **registry.lab.example.com/todoapp/todoapi:latest** image from the private docker registry into OpenShift. Ensure that an image stream called **todoapi** is created in the **todoapp** namespace.

9. Create a new application, as the **developer** user, 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://services.lab.example.com/todoapi
Application Hostname	todoapi.apps.lab.example.com
MySQL Username	todoapp
MySQL Password	todoapp
Database name	todoappdb
Database Administrator Password	redhat



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.

Verify that the application is built and deployed. You must see two pods, one each for the todoapi application and the MySQL database.

10. Sample data for the TODO List application is provided for you in the **/home/student/D0285/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/D0285/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.

11. 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.apps.lab.example.com/todo/api/host**, which prints the pod name on which the application is running.
12. Verify that the **todoapi** application fetches data from MySQL by using the **curl** command to access the URL **http://todoapi.apps.lab.example.com/todo/api/items**, which prints the list of TODO items in JSON format.
13. On the **workstation** VM, as the **developer** user, create a new application named **todoui** using the **todoui** image stream imported earlier.
Verify that the deployment is successful.
14. Create a new route for the **todoui** service with a *hostname* of **todo.apps.lab.example.com**.

Use a browser on the **workstation** VM to navigate to the home page of the TODO List application at **http://todo.apps.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.

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
```

- 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:443".
...
```

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/D0285/labs/review-deploy/set-quotas.sh** on the **workstation** VM:

```
[student@workstation ~]$ /home/student/D0285/labs/review-deploy/set-quotas.sh  
Setting quotas on the todoapp project...  
Already on project "todoapp" on server "https://master.lab.example.com".  
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://services.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>) as **developer** with a password of **redhat**, and acknowledge that you accept insecure connections:

```
[student@workstation ~]$ oc login -u developer -p redhat \  
https://master.lab.example.com  
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 \  
php:5.6-http://services.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.

```
[student@workstation ~]$ oc get pods
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...FailedCreate...deployer-controller
Error creating deployer pod: 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:

```
[student@workstation ~]$ oc get dc
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           1      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/D0285/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.

The pod count was incorrectly typed as 1 instead of 10.

- 4.3. Cancel the existing deployment for the application:

```
[student@workstation ~]$ oc rollout cancel dc/hello
```

```
deploymentconfig "hello" cancelling
```

- 4.4. 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/D0285/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.

- 4.5. Switch back to the OpenShift **developer** user on **workstation**. Verify that the quota for the pod count has been incremented to 10.

```
[student@workstation ~]$ oc login -u developer -p redhat
[student@workstation ~]$ oc describe quota
Name:          todoapp-quota
Namespace:     todoapp
Resource Used Hard
-----
pods           0    10
```

- 4.6. Redeploy the application:

```
[student@workstation ~]$ oc rollout latest dc/hello
deploymentconfig "hello" rolled out
```

- 4.7. 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-2-1xqkr  1/1     Running   0          8s
```

The deployment of the application should now succeed and the pod must be in *Running* state.

- 4.8. 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 **services** VM.

Run the **config-nfs.sh** script located in the **/home/student/D0285/labs/review-deploy** folder on the **services** VM.

```
[student@workstation ~]$ cd /home/student/D0285/labs/review-deploy
[student@workstation review-deploy]$ scp config-nfs.sh \
services:/home/student/config-nfs.sh
Warning: Permanently added 'services' (ECDSA) to the list of known hosts.
config-nfs.sh                                              100%  779   423.2KB/s   00:00
[student@workstation review-deploy]$ ssh services sudo \
/home/student/config-nfs.sh
Export directory /var/export/dbvol created.
Export list for services.lab.example.com:
/exports/prometheus-alertbuffer *
/exports/prometheus-alertmanager *
/exports/prometheus *
/exports/etcdb-vol2 *
/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 and become the **root** user to confirm that the **node1** host can access the NFS share on the **services** VM.

```
[student@workstation review-deploy]$ ssh node1
[student@node1 ~]$ sudo -i
[root@node1 ~]# mount -t nfs services.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 ..
services.lab.example.com:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.1,rsiz=262144,wsiz=262144,namlen=255,hard,
proto=tcp,port=0,timeo=600,retrans=2,sec=sys,
```

```
clientaddr=172.25.250.11,local_lock=none,addr=172.25.250.13)
```

- 5.4. Unmount the NFS share:

```
[root@node1 ~]# umount /mnt
```

- 5.5. Similarly, log in to the **node2** VM and become the **root** user to confirm that the **node2** VM can access the NFS share on **master**:

```
[root@node2 ~]# mount -t nfs services.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 ..
services.lab.example.com:/var/export/dbvol on /mnt type nfs4
(rw,relatime,vers=4.1,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=172.25.250.13)
```

- 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/D0285/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: services.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/D0285/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/D0285/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 apply -n openshift -f \
/home/student/D0285/labs/review-deploy/todoapi/openshift/nodejs-mysql-
template.yaml
template "nodejs-mysql-persistent" created
```

7. Build the **todoui** docker image on the **workstation** 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 **/home/student/D0285/labs/review-deploy/todoui/** folder on the **workstation** VM. Briefly review the provided **Dockerfile**.

- 7.1. Build the **todoui** docker image by running the provided **build.sh** script in the **/home/student/D0285/labs/review-deploy/todoui/** folder on the **workstation** VM:

```
[student@workstation ~]$ cd /home/student/D0285/labs/review-deploy/todoui/
[student@workstation todoui]$ ./build.sh
Sending build context to Docker daemon 647.2 kB
Step 1/11 : FROM rhel7:7.5
Trying to pull repository registry.lab.example.com/rhel7 ...
7.5: Pulling from registry.lab.example.com/rhel7
...
Successfully built 531b63448385
```

- 7.2. Tag the newly built **todoui** docker image:

```
[student@workstation todoui]$ docker tag todoapp/todoui:latest \
registry.lab.example.com/todoapp/todoui:latest
```

- 7.3. Push the newly tagged **todoui** docker image to the classroom private docker registry on the **services** VM:

```
[student@workstation todoui]$ docker push \
registry.lab.example.com/todoapp/todoui:latest
The push refers to a repository [registry.lab.example.com/todoapp/todoui]
```

- ...
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**:

```
[student@workstation todoui]$ oc whoami -c
todoapp/master-lab-example-com:443/admin
[student@workstation todoui]$ oc import-image todoui \
--from=registry.lab.example.com/todoapp/todoui \
--confirm -n todoapp
The import completed successfully.
```

```
Name: todoui
Namespace: todoapp
Created: Less than a second ago
...
```

- 8.2. Verify that the **todoui** image stream has been created:

```
[student@workstation todoui]$ oc get is -n todoapp | grep todoui
todoui docker-registry.default.svc:5000/todoapp/todoui latest 44 seconds ago
```

- 8.3. Describe the **todoui** image stream and verify that the latest **todoui** docker image is referenced:

```
[student@workstation todoui]$ oc describe is todoui -n todoapp
Name: todoui
Namespace: todoapp
Created: 2 minutes ago
...
Unique Images: 1
Tags: 1

latest
tagged from registry.lab.example.com/todoapp/todoui
...
```

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>) as the **developer** user with a

password of **redhat**, and verify that a single project called **todoapp** is visible. If necessary, create an exception to accept the self-signed certificate.

- 9.2. Click **todoapp**, and then click **Add to Project**. Then click **Browse Catalog**. On the **Browse Catalog** page, click **Languages** and then **JavaScript** to get a list of Javascript-based templates. One of the templates should be **Node.js + MySQL (Persistent)**.



NOTE

During the testing of this lab, we found that sometimes the template is not visible to the **developer** user. If this is the case, login as the **admin** user and try again. It is safe to proceed with the following lab steps as **admin**.

- 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://services.lab.example.com/todoapi
Application Hostname	todoapi.apps.lab.example.com
MySQL Username	todoapp
MySQL Password	todoapp
Database name	todoappdb
Database Administrator Password	redhat

Click the **Next** button. Ignore the values on the **Binding** page and click the **Create** button. When the confirmation page is displayed click the **Close** button.

- 9.4. Click on the **Overview** link to see the application being built and deployed.
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>) as **developer**. You should have access to a single project called **todoapp**.
 - 10.2. Sample data for the TODO List application is provided for you in the **/home/student/D0285/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 todoui]$ cd
[student@workstation ~]$ oc port-forward mysql-1-crjnw 3306:3306
Forwarding from 127.0.0.1:3306 -> 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.

- 10.3. Import the SQL data in **db.sql** file into the **todoappdb** database.

Execute the script available at **/home/student/D0285/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/D0285/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	FALSE
2	Buy groceries	TRUE

DONE!

- 10.4. Verify that the application is working correctly by using the **curl** command to access the URL **http://todoapi.apps.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.apps.lab.example.com/todo/api/host \
| python -m json.tool
{
    "hostname": "todoapi-2-gzb54",
    "ip": "10.130.0.20"
}
```

- 10.5. Verify that the application fetches data from the MySQL database correctly by using the **curl** command to access the URL **http://todoapi.apps.lab.example.com/todo/api/items**, which prints the list of tasks stored in the database.

```
[student@workstation ~]$ curl -s \
http://todoapi.apps.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
        }
    ]
}
```

```

        "done": true,
        "id": 2
    },
],
"pageSize": 10,
"sortDirections": "asc",
"sortFields": "id",
"totalResults": 2
}

```

- 10.6. 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.apps.lab.example.com
```

```
route "todoui" exposed
```

12.2. Access the **todooui** application using the URL from the route:

```
[student@workstation ~]$ curl http://todo.apps.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>
</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.apps.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.



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.

To Do List Application

To Do List

ID	Description	Done	
1	Pick up news...	false	
2	Buy groceries	true	

Add Task

Description:

Add Description.

Completed:

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.

