Red Hat
Training and Certification

Student Workbook (ROLE)

OCP 4.2 DO280

Red Hat OpenShift Administration I

Edition 1

Red Hat OpenShift Administration I

OCP 4.2 DO280
Red Hat OpenShift Administration I
Edition 1 20200123

Authors:	Christopher Caillouet, Fernando Lozano, Ivan Chavero, Michael Jarrett, Michael Phillips, Razique Mahroua, Joel Birchler, Dan Kolepp
Editor:	David O'Brien, Dave Sacco, Nicole Muller

Contributors: Jim Rigsbee

Document Conventions vii

Introduction ix
DO280 Red Hat OpenShift Administration I ... ix
Orientation to the Classroom Environment ... x
Internationalization ... xiii

1. Describing Red Hat OpenShift Container Platform 1
Guided Exercise: Creating a Lab Environment ... 2
Describing OpenShift Container Platform Features ... 4
Quiz: Describing OpenShift Container Platform Features ... 8
Describing OpenShift Container Platform Architecture ... 12
Quiz: Describing OpenShift Container Platform Architecture ... 15
Describing Cluster Operators ... 19
Quiz: Describing Cluster Operators ... 22
Summary ... 24

2. Verifying a Cluster 25
Describing Installation Methods ... 26
Quiz: Describing Installation Methods ... 28
Configuring a Lab Environment ... 30
Guided Exercise: Configuring a Lab Environment ... 32
Executing Troubleshooting Commands ... 35
Guided Exercise: Executing Troubleshooting Commands ... 44
Summary ... 52

3. Configuring Authentication 53
Configuring Identity Providers ... 54
Guided Exercise: Configuring Identity Providers ... 61
Summary ... 70

4. Controlling Access to OpenShift Resources 71
Defining and Applying Permissions Using RBAC ... 72
Guided Exercise: Defining and Applying Permissions using RBAC ... 76
Managing Sensitive Information with Secrets ... 82
Guided Exercise: Managing Sensitive Information With Secrets ... 86
Controlling Application Permissions with Security Context Constraints (SCCs) ... 92
Guided Exercise: Controlling Application Permissions with Security Context Constraints (SCC) ... 95
Lab: Controlling access to OpenShift resources ... 99
Summary ... 107

5. Configuring OpenShift Networking Components 109
Troubleshooting OpenShift Software-defined Networking ... 110
Guided Exercise: Troubleshooting OpenShift Software-defined Networking ... 119
Controlling Cluster Network Ingress ... 128
Guided Exercise: Controlling Cluster Network Ingress ... 135
Lab: Configuring OpenShift Networking Components ... 145
Summary ... 157

6. Controlling Pod Scheduling 159
Controlling Pod Scheduling Behavior ... 160
Guided Exercise: Controlling Pod Scheduling Behavior ... 167
Limiting Resource Usage ... 173
Guided Exercise: Limiting Resource Usage ... 182
Scaling an Application ... 190
Guided Exercise: Scaling an Application ... 194
Lab: Controlling Pod Scheduling ... 200

Summary 207

7. Scaling an OpenShift Cluster **209**
Manually Scaling an OpenShift Cluster 210
Guided Exercise: Manually Scaling an OpenShift Cluster 215
Automatically Scaling an OpenShift Cluster 221
Guided Exercise: Automatically Scaling an OpenShift Cluster 225
Summary 230

8. Performing Cluster Updates **231**
Describing the Cluster Update Process 232
Quiz: Describing the Cluster Update Process 241
Summary 245

9. Managing a Cluster with the Web Console **247**
Performing Cluster Administration 248
Guided Exercise: Performing Cluster Administration 252
Managing Workloads 259
Guided Exercise: Managing Workloads 263
Examining Cluster Metrics 270
Guided Exercise: Examining Cluster Metrics 273
Lab: Managing the Cluster with the Web Console 277
Summary 288

10. Comprehensive Review **289**
Comprehensive Review 290
Lab: Install, manage, and troubleshoot an OpenShift cluster 292

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

DO280 Red Hat OpenShift Administration I

Red Hat® OpenShift® Container Platform is a containerized application platform that allows enterprises to manage and scale applications utilizing container deployments. OpenShift provides predefined application environments, based upon Kubernetes, to support DevOps principles such as reduced time to market, infrastructure-as-code, continuous integration (CI), and continuous delivery (CD).

Red Hat OpenShift Administration I (DO280) teaches students how to configure, troubleshoot, and manage the Red Hat® OpenShift® Container Platform. This hands-on, lab-based course shows students how to review the installation of a cluster, configure it, and manage it on a day-to-day basis.

Course Objectives

- Install, configure, manage, and troubleshoot OpenShift clusters.
- This course, together with *Introduction to Containers, Kubernetes, and Red Hat OpenShift* (DO180), prepares the student to take the *Red Hat Certified Specialist in OpenShift Administration* exam (EX280).

Audience

- System and Software Architects
- System Administrators
- Cluster Operators
- Site Reliability Engineers

Prerequisites

- Either complete the *Introduction to Containers, Kubernetes, and Red Hat OpenShift* (DO180) course, or have equivalent knowledge.
- Either attain the *Red Hat Certified System Administrator* certification (RHCSA), or have equivalent knowledge.

Orientation to the Classroom Environment

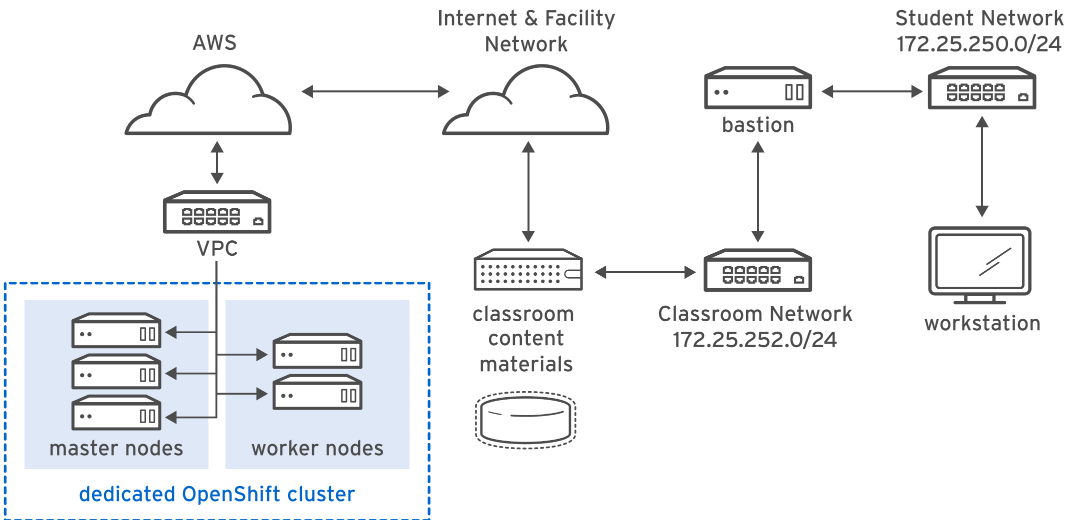


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is **`workstation`**. The system called **`bastion`** must always be running. These two 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`**.

Classroom Machines

Machine name	IP addresses	Role
bastion.lab.example.com	172.25.250.254	Router linking student's VMs to classroom servers
workstation.lab.example.com	172.25.250.9	Graphical workstation used for system administration

The **`bastion`** system acts as a router between the network that connects the student machines and the classroom network. If **`bastion`** is down, other student machines may not function properly or may even hang during boot.

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

Students use the **`workstation`** machine to access a dedicated OpenShift cluster, hosted externally in Amazon Web Services (AWS), for which they have cluster administrator privileges.

Each student has a dedicated cluster that is provisioned at the same time as their classroom environments using the Red Hat On-Line Learning Environment web interface.

Cluster information, such as the API endpoint, cluster ID, and **`kubeadmin`** password, are presented when the student provisions their environment.

Controlling Your Systems

Students are assigned remote computers in a Red Hat Online Learning classroom. They are accessed through a web application hosted at rol.redhat.com [http://rol.redhat.com]. Students should log in to this site using their Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through a web page. The state of each virtual machine in the classroom is displayed on the page under the **Online Lab** tab.

Machine States

Virtual Machine State	Description
STARTING	The virtual machine is in the process of booting.
STARTED	The virtual machine is running and available (or, when booting, soon will be).
STOPPING	The virtual machine is in the process of shutting down.
STOPPED	The virtual machine is completely shut down. Upon starting, the virtual machine boots 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 is available.

Classroom/Machine Actions

Button or Action	Description
PROVISION LAB	Create the ROL classroom. Creates all of the virtual machines needed for the classroom and starts them. Can take several minutes to complete.
DELETE LAB	Delete the ROL classroom. Destroys all virtual machines in the classroom. **Caution: Any work generated on the disks is lost.**
START LAB	Start all virtual machines in the classroom.
SHUTDOWN LAB	Stop all virtual machines in the classroom.

Button or Action	Description
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 virtual machine and run commands. In most cases, students should log in to the **`workstation`** virtual machine and use **`ssh`** to connect to the other virtual machines.
ACTION → Start	Start (power on) the virtual machine.
ACTION → Shutdown	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION → Power Off	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION → Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. **Caution: Any work generated on the disk is lost.**

At the start of an exercise, if instructed to reset a single virtual machine node, click **ACTION → Reset** for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION → Reset**

If you want to return the classroom environment to its original state at the start of the course, you can click **DELETE LAB** to remove the entire classroom environment. After the lab has been deleted, you can click **PROVISION LAB** to provision a new set of classroom systems.

> **Warning**
> The **DELETE LAB** operation cannot be undone. Any work you have completed in the classroom environment up to that point will be lost.

The Autostop Timer

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

To adjust the timer, click **MODIFY** to display the **New Autostop Time** dialog box. Set the number of hours until the classroom should automatically stop. Note that there is a maximum time of ten hours. Click **ADJUST TIME** to apply this change to the timer settings.

Internationalization

Per-user Language Selection

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

Language Settings

In the GNOME desktop environment, the user might 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.

You can start this application in two ways. You can run the command **`gnome-control-center region`** from a terminal window, or on the top bar, from the system menu in the right corner, select the settings button (which has a crossed screwdriver and wrench for an icon) from the bottom left of the menu.

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 you log in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications such as **`gnome-terminal`** that are started inside it. However, by default they do not apply to that account if accessed through an **`ssh`** login from a remote system or a text-based login on a virtual console (such as **`tty5`**).

> **Note**
>
> You can make your shell environment use the same **`LANG`** setting as your graphical environment, even when you log in through a text-based virtual console or over **`ssh`**. One way to do this is to place code similar to the following in your **`~/.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=$(grep 'Language=' /var/lib/AccountsService/users/${USER} \
>   | sed 's/Language=//')
> if [ "$i" != "" ]; then
>     export LANG=$i
> fi
> ```
>
> Japanese, Korean, Chinese, and other languages with a non-Latin character set might not display properly on text-based virtual 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 25 17:55:01 CET 2019
```

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 or later 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 you know the character's Unicode code point. Type **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 default language, run the command **localectl set-locale LANG=*locale***, where *locale* is the appropriate value for the **LANG** environment variable 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 by clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the graphical login screen will also adjust the system-wide default language setting stored in the **/etc/locale.conf** configuration file.

> **Important**
>
> Text-based virtual consoles such as **tty4** are more limited in the fonts they can display than terminals in a virtual console running a graphical environment, or pseudoterminals for **ssh** sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a text-based virtual console. For this reason, you should consider using English or another language with a Latin character set for the system-wide default.
>
> Likewise, text-based virtual 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 text-based virtual consoles and the graphical environment. See the **localectl**(1) and **vconsole.conf**(5) man pages for more information.

Language Packs

Special RPM packages called *langpacks* install language packages that add support for specific languages. These langpacks use dependencies to automatically install additional RPM packages containing localizations, dictionaries, and translations for other software packages on your system.

To list the langpacks that are installed and that may be installed, use **yum list langpacks-***:

```
[root@host ~]# yum list langpacks-*
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Installed Packages
langpacks-en.noarch      1.0-12.el8        @AppStream
Available Packages
langpacks-af.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-am.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-ar.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-as.noarch      1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
langpacks-ast.noarch     1.0-12.el8        rhel-8-for-x86_64-appstream-rpms
...output omitted...
```

To add language support, install the appropriate langpacks package. For example, the following command adds support for French:

```
[root@host ~]# yum install langpacks-fr
```

Use **`yum repoquery --whatsupplements`** to determine what RPM packages may be installed by a langpack:

```
[root@host ~]# yum repoquery --whatsupplements langpacks-fr
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Last metadata expiration check: 0:01:33 ago on Wed 06 Feb 2019 10:47:24 AM CST.
glibc-langpack-fr-0:2.28-18.el8.x86_64
gnome-getting-started-docs-fr-0:3.28.2-1.el8.noarch
hunspell-fr-0:6.2-1.el8.noarch
hyphen-fr-0:3.0-1.el8.noarch
libreoffice-langpack-fr-1:6.0.6.1-9.el8.x86_64
man-pages-fr-0:3.70-16.el8.noarch
mythes-fr-0:2.3-10.el8.noarch
```

Important

Langpacks packages use RPM *weak dependencies* in order to install supplementary packages only when the core package that needs it is also installed.

For example, when installing *langpacks-fr* as shown in the preceding examples, the *mythes-fr* package will only be installed if the *mythes* thesaurus is also installed on the system.

If *mythes* is subsequently installed on that system, the *mythes-fr* package will also automatically be installed due to the weak dependency from the already installed *langpacks-fr* package.

References

`locale`(7), **`localectl`**(1), **`locale.conf`**(5), **`vconsole.conf`**(5), **`unicode`**(7), and **`utf-8`**(7) 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

Note

This table might not reflect all langpacks available on your system. Use **`yum info langpacks-`*`SUFFIX`*** to get more information about any particular langpacks package.

Language Codes

Language	Langpacks Suffix	$LANG value
English (US)	en	en_US.utf8

Language	Langpacks Suffix	$LANG value
Assamese	as	as_IN.utf8
Bengali	bn	bn_IN.utf8
Chinese (Simplified)	zh_CN	zh_CN.utf8
Chinese (Traditional)	zh_TW	zh_TW.utf8
French	fr	fr_FR.utf8
German	de	de_DE.utf8
Gujarati	gu	gu_IN.utf8
Hindi	hi	hi_IN.utf8
Italian	it	it_IT.utf8
Japanese	ja	ja_JP.utf8
Kannada	kn	kn_IN.utf8
Korean	ko	ko_KR.utf8
Malayalam	ml	ml_IN.utf8
Marathi	mr	mr_IN.utf8
Odia	or	or_IN.utf8
Portuguese (Brazilian)	pt_BR	pt_BR.utf8
Punjabi	pa	pa_IN.utf8
Russian	ru	ru_RU.utf8
Spanish	es	es_ES.utf8
Tamil	ta	ta_IN.utf8
Telugu	te	te_IN.utf8

Chapter 1

Describing Red Hat OpenShift Container Platform

Goal	Describe the architecture of Red Hat OpenShift Container Platform (RHOCP).
Objectives	• Create an OpenShift cluster for use as a lab environment. • Describe the typical use of RHOCP and its features. • Describe the architecture of RHOCP. • Describe what a cluster operator is, how it works, and name the major cluster operators.
Sections	• Creating a Lab Environment (Guided Exercise) • Describing OpenShift Container Platform Features (and Quiz) • Describing the OpenShift Container Platform Architecture (and Quiz) • Describing Cluster Operators (and Quiz)

Guided Exercise

Creating a Lab Environment

In this exercise, you will start the provisioning process of a dedicated OpenShift cluster in the cloud using the Red Hat On-line Learning Environment (ROLE).

Outcomes

You should be able to provision a dedicated OpenShift cluster that you will use for all exercises in this course.

Before You Begin

To perform this exercise, ensure you have access to the Red Hat On-line Learning Environment (ROLE).

The following procedure describes how to provision an OpenShift cluster from the Red Hat Online Learning platform. You must complete the following procedure before attempting any of the course activities.

> **Note**
>
> The provisioning of the cluster takes approximately one hour.

1. Open a browser and navigate to the Red Hat Online Learning platform at `https://rol.redhat.com`. Log in with your credentials.

2. Access the *Red Hat OpenShift Administration I – DO280* course and click **ONLINE TRAINING**.

3. Click the **Lab Environment** tab and then click **CREATE LAB** to provision the cluster.

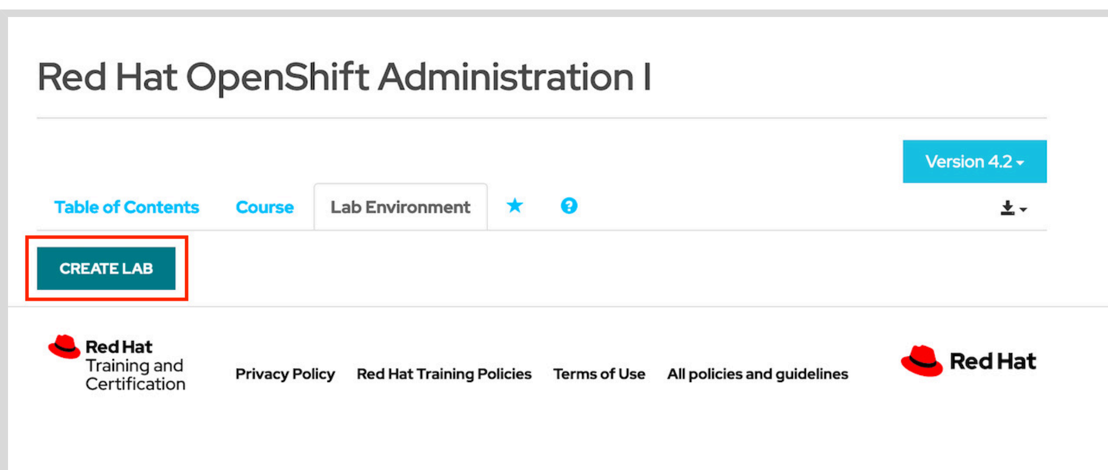


Figure 1.1: Managing the lab environment

4. A page appears that contains information on how to access the cluster; namely, the user name, the password, and the cluster console and API endpoint.

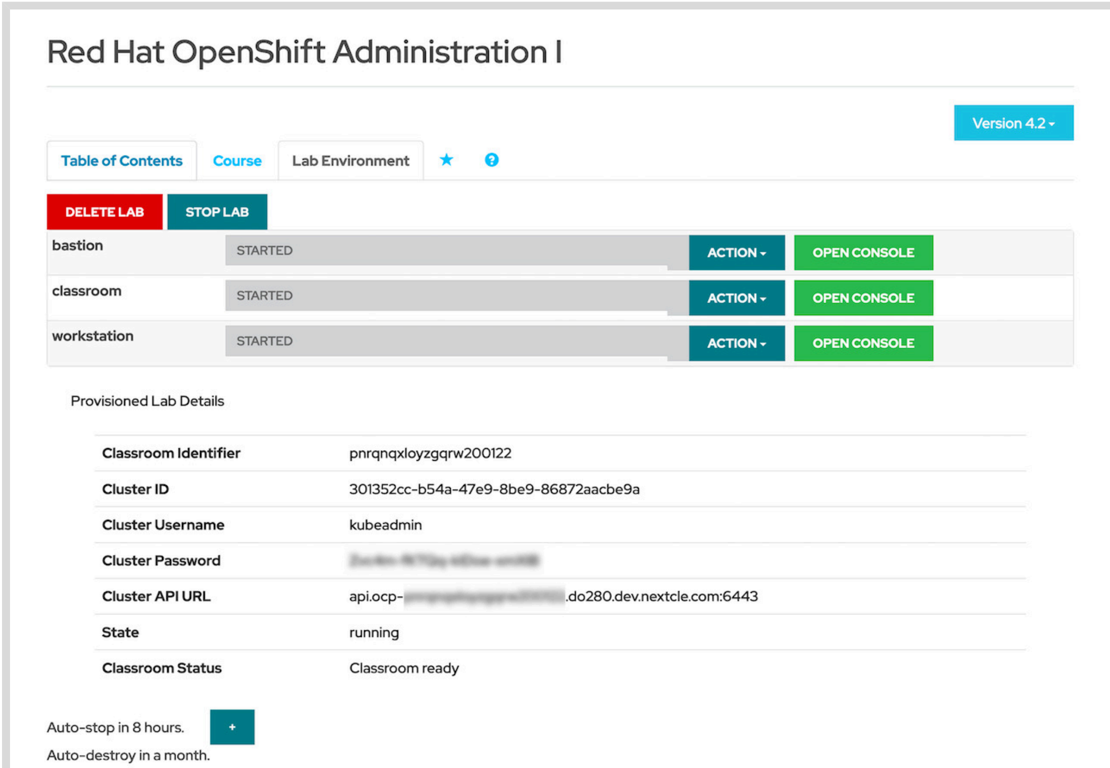


Figure 1.2: Provisioning a cluster

From this page, you can also control the environment, such as stopping or starting the cluster.

Finish

This exercise has no command to finish it. Provisioning the cluster takes some time; while you wait, continue to the next section in this chapter.

This concludes the section.

Describing OpenShift Container Platform Features

Objectives

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

Introducing OpenShift Container Platform

Container orchestration is a fundamental enabler of digital transformation initiatives. However, as monolithic applications transition to containerized services, it can be tedious to manage these applications with legacy infrastructure. Red Hat OpenShift Container Platform (RHOCP) allows developers and IT organizations to better manage application life cycles.

RHOCP is based on the Kubernetes open source project and extends the platform with features that bring a robust, flexible, and scalable container platform to customer data centers, enabling developers to run their workload in a high availability environment.

A container *orchestrator*, such as OpenShift Container Platform, manages a cluster of servers that runs multiple containerized applications. The Red Hat OpenShift product family includes a set of solutions to improve the delivery of business applications in a variety of environments:

Red Hat OpenShift Container Platform
: Provides an enterprise-ready Kubernetes environment for building, deploying, and managing container-based applications in any public or private data center, including bare metal servers. RHOCP is compatible with multiple cloud and virtualization providers, isolating application developers and administrators from differences between these providers. You decide when to update to newer releases and which additional components to enable.

Red Hat OpenShift Dedicated
: Provides a managed OpenShift environment in a public cloud, such as Amazon Web Services (AWS) or Google Cloud Platform (GCP). This product provides all the features that RHOCP provides, however, Red Hat manages the cluster for you. You retain some control of decisions such as when to update to a newer release of OpenShift or to install add-on services.

Red Hat OpenShift Online
: Provides a hosted, public container orchestration platform that offers an application development, build, deployment, and hosting solution in a cloud environment. The solution is shared across multiple customers, and Red Hat manages the cluster life cycle, which includes applying updates or integrating new features.

Red Hat Code Ready Containers
: Provides a minimal installation of OpenShift that you can run on a laptop for local development and experimentation.

Some cloud providers also provide offerings based on RHOCP, which add tight integration with other services from their platforms and which are supported by the provider in a partnership with Red Hat. One example is Microsoft Azure Red Hat OpenShift.

The following figure describes the services and features of OpenShift:

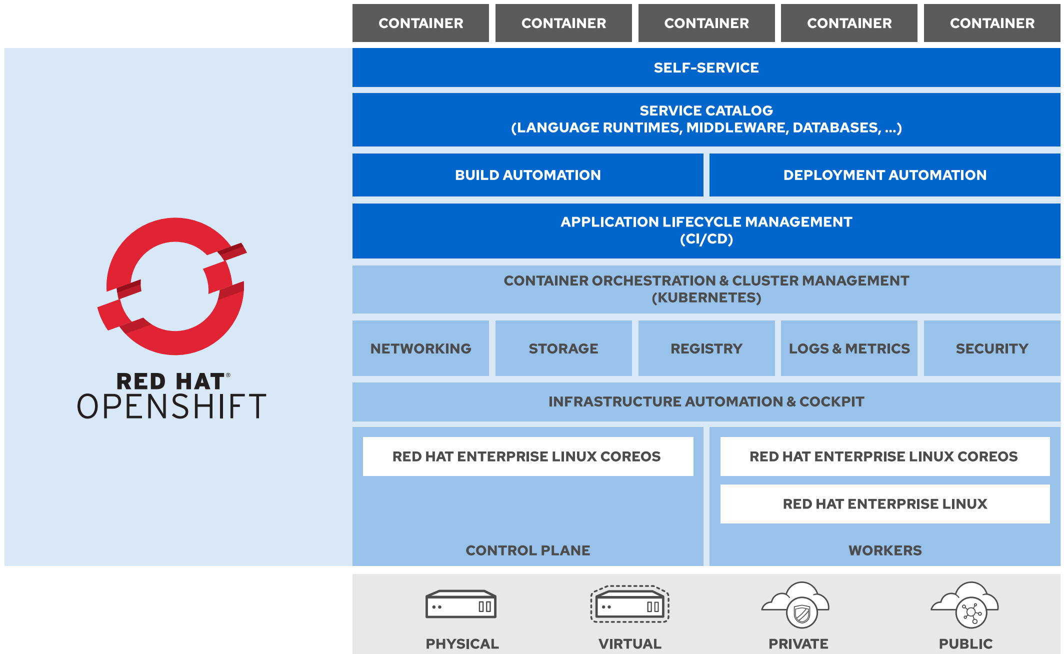


Figure 1.3: OpenShift services and features

The Red Hat OpenShift product family integrates many components:

- The Red Hat Enterprise Linux CoreOS container-optimized, immutable operating system.
- The CRI-O engine, a small footprint, Open Container Initiative (OCI)-compliant container runtime engine with a reduced attack surface.
- Kubernetes, an open source container orchestration platform.
- A self-service web console.
- A number of preinstalled application services, such as an internal container image registry and monitoring framework.
- Certified container images for multiple programming language runtimes, databases, and other software packages.

Introducing OpenShift Features

OpenShift offers many features to automate, scale, and maintain your applications. All of these features are enabled by Kubernetes and most of them require additional components that you need to add and configure on a build-your-own (BYO) Kubernetes setup.

High Availability

Kubernetes has been designed with high availability in mind, for both internal components and user applications. A highly available etcd cluster stores the state of the OpenShift cluster and its applications.

Resources stored in etcd, such as deployment configurations, provide automatic restarting of containers to ensure that your application is always running and that faulty containers are terminated. This applies not only to your applications, but also to containerized services that make up the cluster, such as the web console and the internal image registry.

Lightweight Operating System

RHOCP runs on Red Hat Enterprise Linux CoreOS, Red Hat's lightweight operating system that focuses on agility, portability, and security.

Red Hat Enterprise Linux CoreOS (RHEL CoreOS) is an immutable operating system that is optimized for running containerized applications. The entire operating system is updated as a single image, instead of on a package-by-package basis, and both user applications and system components such as network services run as containers.

RHOCP controls updates to RHEL CoreOS and its configurations, and so managing an OpenShift cluster includes managing the operating system on cluster nodes, freeing system administrators from these tasks and reducing the risk of human error.

Load Balancing

Clusters provide three types of load balancers: an external load balancer, which manages access to the OpenShift API; the HAProxy load balancer, for external access applications; and the internal load balancer, which uses Netfilter rules for internal access to applications and services.

Route resources use HAProxy to manage external access to the cluster. Service resources use Netfilter rules to manage traffic from inside the cluster." The technology that external load balancers use is dependent on the cloud provider that runs your cluster.

Automating Scaling

OpenShift clusters can adapt to increased application traffic in real time by automatically starting new containers, and terminating containers when the load decreases. This features ensures that your application's access time remains optimal regardless of the number of concurrent connections or activity.

OpenShift clusters can also add or remove more worker nodes from the cluster according to the aggregated load from many applications, keeping responsiveness and costs down on public and private clouds.

Logging and Monitoring

RHOCP ships with an advanced monitoring solution, based on Prometheus, which gathers hundreds of metrics about your cluster. This solution interacts with an alerting system that allows you to obtain detailed information about your cluster activity and health.

RHOCP ships with an advanced aggregated logging solution, based on Elasticsearch, which allows long-term retention of logs from cluster nodes and containers.

Services Discovery

RHOCP runs an internal DNS service on the cluster, and configures all containers to use that internal DNS for name resolution. This means that applications can rely on friendly names to find other applications and services, without the overhead of an external services catalog.

Storage

Kubernetes adds an abstraction layer between the storage back end and the storage consumption. As such, applications can consume long-lived, short-lived, block, and file-based storage using unified storage definitions that are independent of the storage back end. This way your applications are not dependent on particular cloud provider storage APIs.

RHOCP embeds a number of storage providers that allow for automatic provisioning of storage on popular cloud providers and virtualization platforms, and so cluster administrators do not need to manage the fine details of proprietary storage arrays.

Application Management

RHOCP empowers developers to automate the development and deployment of their applications. Use the OpenShift *Source-to-Image (S2I)* feature to automatically build containers based on your source code and run them in OpenShift. The internal registry stores application container images, which can be reused. This decreases the time it takes to publish your applications.

The developer catalog, accessible from the web console, is a place for publishing and accessing application templates. It supports many runtime languages, such as Python, Ruby, Java, and Node.js, and also database and messaging servers. You can expand the catalog by installing new *operators*, which are prepackaged applications and services that embed operational intelligence for deploying, updating, and monitoring their applications.

Cluster Extensibility

RHOCP relies on standard extension mechanisms from Kubernetes, such as extension APIs and custom resource definitions, to add features that are otherwise not provided by upstream Kubernetes. OpenShift packages these extensions as operators for ease of installation, update, and management.

OpenShift also includes the *Operator Lifecycle Manager (OLM)*, which facilitates the discovery, installation, and update of applications and infrastructure components packaged as operators.

Red Hat, in collaboration with AWS, Google Cloud, and Microsoft, launched the *OperatorHub*, accessible at `https://operatorhub.io`. The platform is a public repository and marketplace for operators compatible with OpenShift and other distributions of Kubernetes that include the OLM.

References

Further information is available in the Red Hat OpenShift Container Platform 4.2 product documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/

Quiz

Describing OpenShift Container Platform Features

Choose the correct answers to the following questions:

1. Which of the following definitions best describes container orchestration platforms?

a. They extend your application's operational knowledge and provide a way to package and distribute them.
b. They allow you to manage a cluster of servers that run containerized applications. They add features such as self-service, high availability, monitoring, and automation.
c. They allow you to provision Infrastructure-as-a-Service clusters on a variety of cloud providers, including AWS, GCP, and Microsoft Azure.
d. They enable developers to write, package, and publish their applications as operators to the operator catalog.

2. Which three of the following key features enable high availability for your applications? (Choose three.)

a. An OpenShift etcd cluster keeps the cluster state available for all nodes.
b. OpenShift HAProxy load balancers allow external access to applications.
c. OpenShift services load balance access to applications from inside the cluster.
d. OpenShift deployment configurations ensure application containers are restarted in scenarios such as loss of a node.

3. Which two of the following statements about OpenShift are true? (Choose two.)

a. Developers can create and start cloud applications directly from a source code repository.
b. OpenShift patches Kubernetes to add features that would not be available to other distributions of Kubernetes.
c. OpenShift Dedicated gives you access to an exclusive set of operators that Red Hat curates and maintains. This helps to ensure that the operators are secure and safe to run in your environment.
d. OpenShift cluster administrators can discover and install new operators from the operator catalog.

4. Which two of the following services do OpenShift components use for load balancing their traffic? (Choose two.)

a. The OpenShift API, which is accessible over the external load balancer.
b. Services, which use Netfilter for load balancing.
c. Services, which use HAProxy for load balancing.
d. Routes, which use Netfilter for load balancing.
e. Routes, which use the HAProxy for load balancing.

5. Which two of the following statements about OpenShift high availability and scaling are true? (Choose two.)

a. OpenShift does not provide high availability by default. You need to use third-party high availability products.
b. OpenShift uses metrics from Prometheus to dynamically scale application pods.
c. High availability and scaling are restricted to applications that expose a REST API.
d. OpenShift can scale applications up and down based on demand.

Solution

Describing OpenShift Container Platform Features

Choose the correct answers to the following questions:

1. **Which of the following definitions best describes container orchestration platforms?**
   a. They extend your application's operational knowledge and provide a way to package and distribute them.
   b. They allow you to manage a cluster of servers that run containerized applications. They add features such as self-service, high availability, monitoring, and automation.
   c. They allow you to provision Infrastructure-as-a-Service clusters on a variety of cloud providers, including AWS, GCP, and Microsoft Azure.
   d. They enable developers to write, package, and publish their applications as operators to the operator catalog.

2. **Which three of the following key features enable high availability for your applications? (Choose three.)**
   a. An OpenShift etcd cluster keeps the cluster state available for all nodes.
   b. OpenShift HAProxy load balancers allow external access to applications.
   c. OpenShift services load balance access to applications from inside the cluster.
   d. OpenShift deployment configurations ensure application containers are restarted in scenarios such as loss of a node.

3. **Which two of the following statements about OpenShift are true? (Choose two.)**
   a. Developers can create and start cloud applications directly from a source code repository.
   b. OpenShift patches Kubernetes to add features that would not be available to other distributions of Kubernetes.
   c. OpenShift Dedicated gives you access to an exclusive set of operators that Red Hat curates and maintains. This helps to ensure that the operators are secure and safe to run in your environment.
   d. OpenShift cluster administrators can discover and install new operators from the operator catalog.

4. **Which two of the following services do OpenShift components use for load balancing their traffic? (Choose two.)**
   a. The OpenShift API, which is accessible over the external load balancer.
   b. Services, which use Netfilter for load balancing.
   c. Services, which use HAProxy for load balancing.
   d. Routes, which use Netfilter for load balancing.
   e. Routes, which use the HAProxy for load balancing.

5. **Which two of the following statements about OpenShift high availability and scaling are true? (Choose two.)**
   a. OpenShift does not provide high availability by default. You need to use third-party high availability products.
   b. OpenShift uses metrics from Prometheus to dynamically scale application pods.
   c. High availability and scaling are restricted to applications that expose a REST API.
   d. OpenShift can scale applications up and down based on demand.

Describing OpenShift Container Platform Architecture

Objectives

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

Introducing the Declarative Architecture of Kubernetes

The architecture of OpenShift is based on the declarative nature of Kubernetes. Most system administrators are used to imperative architectures, where you perform actions that indirectly change the state of the system, such as starting and stopping containers on a given server. In a declarative architecture, you change the state of the system and the system updates itself to comply with the new state. For example, with Kubernetes, you define a pod resource that specifies that a certain container should run under specific conditions. Then Kubernetes finds a server (a node) that can run that container under these specific conditions.

Declarative architectures allow for self-optimizing and self-healing systems that are easier to manage than imperative architectures.

Kubernetes defines the state of its cluster, including the set of deployed applications, as a set of *resources* stored in the etcd database. Kubernetes also runs controllers that monitor these resources and compares them to the current state of the cluster. These controllers take any action necessary to reconcile the state of the cluster with the state of the resources, for example by finding a node with sufficient CPU capacity to start a new container from a new pod resource.

Kubernetes provides a REST API to manage these resources. All actions that an OpenShift user takes, either using the command-line interface or the web console, are performed by invoking this REST API.

Introducing the OpenShift Control Plane

A Kubernetes cluster consists of a set of *nodes* that run the *kubelet* system service and a container engine. OpenShift runs exclusively the *CRI-O* container engine.

Some nodes are *master* nodes that run the REST API, the etcd database, and the platform controllers. OpenShift configures its master nodes so that they are not schedulable to run end-user application pods and are dedicated to running the control plane services. OpenShift calls its non-master nodes *worker* nodes.

Figure 1.4 provides an overview of an OpenShift master node, illustrating the main processes that run in a regular node and in a master node, as either system services or containers.

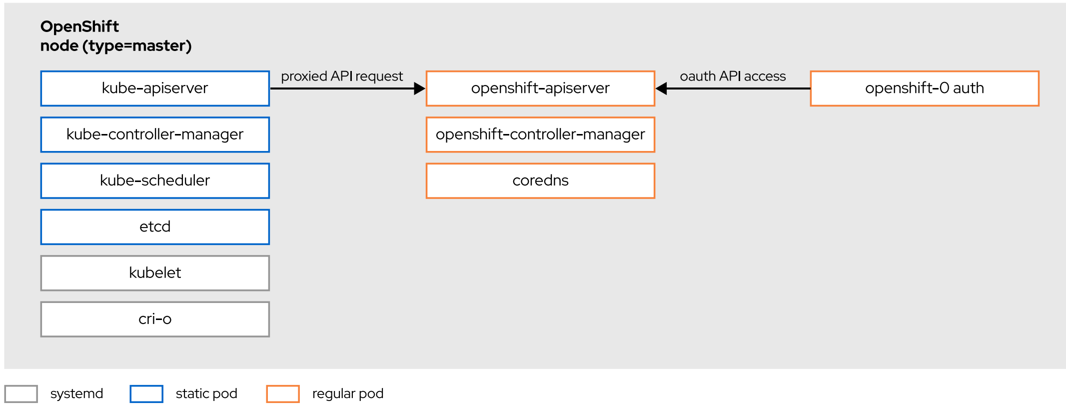


Figure 1.4: Architecture of an OpenShift master node

Depending on the node settings, the **kubelet** agent starts different sets of *static pods*. Static pods are pods that do not require connection to the API server to start. The **kubelet** agent manages the pod's life-cycle. Static pods can provide either control plane services, such as the scheduler, or node services, such as software-defined networking (SDN). OpenShift provides operators that create pod resources for these static pods so that they are monitored like regular pods.

Describing OpenShift Extensions

A lot of functionality from Kubernetes depends on external components, such as ingress controllers, storage plug-ins, network plug-ins, and authentication plug-ins. Similar to Linux distributions, there are many ways to build a Kubernetes distribution by picking and choosing different components.

A lot of functionality from Kubernetes also depends on extension APIs, such as access control and network isolation.

OpenShift is a Kubernetes distribution that provides many of these components already integrated and configured, and managed by operators. OpenShift also provides preinstalled applications, such as a container image registry and a web console, managed by operators.

OpenShift also adds to Kubernetes a series of extension APIs and custom resources. For example, build configurations for the Source-to-Image process, and route resources to manage external access to the cluster.

Red Hat develops all extensions as open source projects and works with the larger Kubernetes community not only to make these official components of Kubernetes but also to evolve the Kubernetes platform to allow easier maintainability and customization.

With OpenShift 3 these extensions were sometimes patches (or forks) of upstream Kubernetes. With OpenShift 4 and operators, these extensions are standard Kubernetes extensions that could be added to any distribution of Kubernetes.

Introducing the OpenShift Default Storage Class

Unlike many container platforms that focus on cloud-native, stateless applications, OpenShift also supports stateful applications that do not follow the standard *Twelve-Factor App* methodology.

OpenShift supports stateful applications by offering a comprehensive set of storage capabilities and supporting operators. OpenShift ships with integrated storage plug-ins and storage classes that rely on the underlying cloud or virtualization platform to provide *dynamically provisioned* storage.

For example, if you install OpenShift on Amazon Web Services (AWS), your OpenShift cluster comes pre-configured with a default *storage class* that uses AWS EBS service automatically to provision storage volumes on-demand. Users can deploy an application that requires persistent storage, such as a database, and OpenShift automatically creates an EBS volume to host the application data.

OpenShift cluster administrators can later define additional storage classes that use different EBS service tiers. For example, you could have one storage class for high-performance storage that sustains a high input-output operations per second (IOPS) rate, and another storage class for low-performance, low-cost storage. Cluster administrators can then allow only certain applications to use the high-performance storage class, and configure data archiving applications to use the low-performance storage class.

References

Further information is accessible in the Red Hat OpenShift Container Platform 4.2 product documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/

Quiz

Describing OpenShift Container Platform Architecture

Choose the correct answers to the following questions:

1. OpenShift is based on which of the following container orchestration technologies?

a. Docker Swarm
b. Rancher
c. Kubernetes
d. Mesosphere Marathon
e. CoreOS Fleet

2. Which two of the following statements are true of OpenShift Container Platform? (Choose two.)

a. OpenShift provides an OAuth server that authenticates calls to its REST API.
b. OpenShift requires the CRI-O container engine.
c. Kubernetes follows a declarative architecture, but OpenShift follows a more traditional imperative architecture.
d. OpenShift extension APIs run as system services.

3. Which of the following servers runs Kubernetes control components?

a. Workers
b. Nodes
c. Masters

4. Which of following components does OpenShift add to upstream Kubernetes?

a. The etcd database
b. A container engine
c. A registry server
d. A scheduler
e. The Kubelet

5. Which of the following sentences is true regarding support for storage with OpenShift?

a. Users can only store persistent data in the etcd database.

b. Users can only deploy on OpenShift cloud-native applications that conform to the Twelve-Factor App methodology.

c. Administrators must configure storage plug-ins appropriate for their cloud providers.

d. Administrators must define persistent volumes before any user can deploy applications that require persistent storage.

e. Users can deploy applications that require persistent storage by relying on the default storage class.

Solution

Describing OpenShift Container Platform Architecture

Choose the correct answers to the following questions:

1. **OpenShift is based on which of the following container orchestration technologies?**
   a. Docker Swarm
   b. Rancher
   c. Kubernetes
   d. Mesosphere Marathon
   e. CoreOS Fleet

2. **Which two of the following statements are true of OpenShift Container Platform? (Choose two.)**
   a. OpenShift provides an OAuth server that authenticates calls to its REST API.
   b. OpenShift requires the CRI-O container engine.
   c. Kubernetes follows a declarative architecture, but OpenShift follows a more traditional imperative architecture.
   d. OpenShift extension APIs run as system services.

3. **Which of the following servers runs Kubernetes control components?**
   a. Workers
   b. Nodes
   c. Masters

4. **Which of following components does OpenShift add to upstream Kubernetes?**
   a. The etcd database
   b. A container engine
   c. A registry server
   d. A scheduler
   e. The Kubelet

5. Which of the following sentences is true regarding support for storage with OpenShift?

a. Users can only store persistent data in the etcd database.
b. Users can only deploy on OpenShift cloud-native applications that conform to the Twelve-Factor App methodology.
c. Administrators must configure storage plug-ins appropriate for their cloud providers.
d. Administrators must define persistent volumes before any user can deploy applications that require persistent storage.
e. Users can deploy applications that require persistent storage by relying on the default storage class.

Describing Cluster Operators

Objectives

After completing this section, you should be able to describe what a cluster operator is, how it works, and name the major cluster operators.

Introducing Kubernetes Operators

Kubernetes *operators* are applications that invoke the Kubernetes API to manage Kubernetes resources. As for any Kubernetes application, you deploy an operator by defining Kubernetes resources such as services and deployments that reference the operator's container image. Because operators, unlike common applications, require direct access to the Kubernetes resources, they usually require custom security settings.

Operators usually define *custom resources (CR)* that store their settings and configurations. An OpenShift administrator manages an operator my editing its custom resources. The syntax of a custom resource is defined by a *custom resource definition (CRD)*.

Most operators manage another application; for example, an operator that manages a database server. In that case, the operator creates the resources that describe that other application using the information from its custom resource.

The purpose of an operator is usually to automate tasks that a human administrator (or human operator) would perform to deploy, update, and manage an application.

Introducing the Operator Framework

You can develop operators using your preferred programming language. Technically you do not need a special-purpose SDK to develop an operator. All you need is the ability to invoke REST APIs and consume secrets that contain access credentials to the Kubernetes APIs.

The *Operator Framework* is an open source toolkit for building, testing, and packaging operators. The Operator Framework makes these tasks easier than coding directly to low-level Kubernetes APIs by providing the following components:

Operator Software Development Kit (Operator SDK)
: Provides a set of Golang libraries and source code examples that implement common patterns in operator applications. It also provides a container image and playbook examples that allow you to develop operators using Ansible.

Operator Life Cycle Manager (OLM)
: Provides an application that manages the deployment, resource utilization, updates, and deletion of operators that have been deployed through an *operator catalog*. The OLM itself is an operator that comes preinstalled with OpenShift.

The Operator Framework also defines a set of recommended practices for implementing operators and CRDs and a standard way of packaging an operator manifest, as a container image, that allows an operator to be distributed using an *operator catalog*. The most common form of an operator catalog is an image registry server.

An operator container image that follows the Operator Framework standards contains all resource definitions required to deploy the operator application. This way the OLM can install an operator automatically. If an operator is not built and packaged by following the Operator Framework standards, the OLM will not be able to install nor manage that operator.

Introducing OperatorHub

OperatorHub provides a web interface to discover and publish operators that follow the Operator Framework standards. Both open source operators and commercial operators can be published to the Operator hub. Operator container images can be hosted at different image registries, for example Quay.io.

Introducing OpenShift Cluster Operators

Cluster operators are regular operators except that they are not managed by the OLM. They are managed by the OpenShift *Cluster Version Operator*, which is sometimes called a *first-level operator*. All cluster operators are also called *second-level operators*.

OpenShift cluster operators provide OpenShift extension APIs and infrastructure services such as:

- The OAuth server, which authenticates access to the master and extensions APIs.
- The core DNS server, which manages service discovery inside the cluster.
- The web console, which allows graphical management of the cluster.
- The internal image registry, which allow developers to host container images inside the cluster, using either S2I or another mechanism.
- The monitoring stack, which generates metrics and alerts about the cluster health.

Some cluster operators manage node or control plane settings. For example, with upstream Kubernetes you edit a node configuration file to add storage and network plug-ins, and these plug-ins may require additional configuration files. OpenShift supports operators that manage configuration files in all nodes and reload the node services that are affected by changes to these files.

Important

OpenShift 4 deprecates the usage of SSH sessions to manage nodes configuration and system services. This ensures that you do not customize the nodes, and that they can be safely added or removed from a cluster. You are expected to perform all administrative actions indirectly by editing custom resources and then wait for their respective operators to apply your changes.

Exploring OpenShift Cluster Operators

Usually an operator and its managed application share the same project. In the case of cluster operators, these are in the **`openshift-*`** projects.

Every cluster operator defines a custom resource of type **`ClusterOperator`**. Cluster operators manage the cluster itself, including the API server, the web console, or the network stack. Each cluster operator defines a set of custom resources, to further control its components. The **`ClusterOperator`** API resource exposes information such as the health of the update, or the version of the component.

Operators are apparent from their name, for example, the **`console`** cluster operator provides the web console, the **`ingress`** cluster operator enables ingresses and routes. The following lists some of the cluster operators.

- **`dns`**
- **`network`**
- **`ingress`**
- **`storage`**
- **`authentication`**
- **`console`**
- **`monitoring`**
- **`image-registry`**
- **`autoscale`**
- **`openshift-apiserver`**
- **`openshift-dns`**
- **`openshift-controller-manager`**
- **`dns`**
- **`cloud-credential`**

References

Further information is accessible in the Red Hat OpenShift Container Platform 4.2 product documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/

Introducing the Operator Framework
https://blog.openshift.com/introducing-the-operator-framework/

Quiz

Describing Cluster Operators

Match the terms below to their correct definition in the table:

Custom Resource Definition

Operator

Operator Catalog

Operator Image

Operator Lifecycle Manager (OLM)

Operator SDK

OperatorHub

Operator Terminology	Name
An open source toolkit for building, testing, and packaging operators.	
A repository for discovering and installing operators.	
An extension of the Kubernetes API that defines the syntax of a custom resource.	
The artifact defined by the Operator Framework that you can publish for consumption by an OLM instance.	
An application that manages Kubernetes resources.	
An application that manages Kubernetes operators.	
A public web service where you can publish operators that are compatible with the OLM.	

Solution

Describing Cluster Operators

Match the terms below to their correct definition in the table:

Operator Terminology	Name
An open source toolkit for building, testing, and packaging operators.	Operator SDK
A repository for discovering and installing operators.	Operator Catalog
An extension of the Kubernetes API that defines the syntax of a custom resource.	Custom Resource Definition
The artifact defined by the Operator Framework that you can publish for consumption by an OLM instance.	Operator Image
An application that manages Kubernetes resources.	Operator
An application that manages Kubernetes operators.	Operator Lifecycle Manager (OLM)
A public web service where you can publish operators that are compatible with the OLM.	OperatorHub

Summary

In this chapter, you learned:

- Red Hat OpenShift Container Platform is based on Red Hat Enterprise Linux CoreOS, the CRI-O container engine, and Kubernetes.
- RHOCP 4 provides a number of services on top of Kubernetes, such as an internal container image registry, storage, networking providers, and centralized logging and monitoring.
- Operators package applications that manage Kubernetes resources, and the Operator Lifecycle Manager (OLM) handles installation and management of operators.
- OperatorHub.io is an online catalog for discovering operators.

Chapter 2

Verifying a Cluster

Goal	Review installation methods and verify the functionality of a newly installed cluster.
Objectives	• Describe the various installation methods and platforms supported. • Configure the OpenShift Cluster used for classroom exercises. • Execute commands that assist in troubleshooting common problems.
Sections	• Describing Installation Methods (and Quiz) • Configuring a Lab Environment (and Guided Exercise) • Executing Troubleshooting Commands (and Guided Exercise)

Describing Installation Methods

Objectives

After completing this section, you should be able to describe the various installation methods and platforms supported.

Introducing OpenShift Installation Methods

Red Hat OpenShift Container Platform provides two main installation methods:

Full-Stack Automation
: With this method, the OpenShift installer provisions all compute, storage, and network resources from a cloud or virtualization provider. You provide the installer with minimum data, such as credentials to a cloud provider and the size of the initial cluster, and then the installer deploys a fully functional OpenShift cluster.

Pre-existing Infrastructure
: With this method, you configure a set of compute, storage, and network resources and the OpenShift installer configures an initial cluster using these resources. You can use this method to set up an OpenShift cluster using bare-metal servers and cloud or virtualization providers that are not supported by the Full-Stack Automation method.

  When using a pre-existing infrastructure, you must provide all of the cluster infrastructure and resources, including the bootstrap node. You must run the installation program to generate the required configuration files, and then run the installation program again to deploy an OpenShift cluster on your infrastructure.

At the time of the Red Hat OpenShift Container Platform 4.2 release, the set of cloud providers supported for the Full-Stack Automation method includes Amazon Web Services (AWS), Google Cloud Computing Engine (GCE), Microsoft Azure, and Red Hat OpenStack Platform using the standard Intel architecture (x86).

Future releases are expected to add more cloud and virtualization providers, such as VMware, Red Hat Virtualization, and IBM System Z. These releases are also expected to support customization of Full-Stack Automation installations, for example reusing existing AWS Virtual Private Clouds (VPC), and provisioning compute instances and load balancers in those VPCs.

Comparing OpenShift Installation Methods

Certain features of OpenShift require using the Full-Stack Automation method, for example, cluster automatic scaling. However, it is expected that future releases might relax such requirements.

Using the Full-Stack Automation method, all nodes of the new cluster run Red Hat Enterprise Linux CoreOS (RHEL CoreOS). Using the Pre-existing Infrastructure method, worker nodes can be setup using Red Hat Enterprise Linux (RHEL), but the control plane (master nodes) still requires RHEL CoreOS.

Describing the Deployment Process

The installation takes place in several stages, starting with the creation of a bootstrap machine that runs Red Hat Enterprise Linux CoreOS using the assets that the installer generates.

The bootstrapping process for the cluster is as follows:

1. The bootstrap machine boots, and then starts hosting the remote resources required for booting the control plane machines.
2. The control plane machines fetch the remote resources from the bootstrap machine and finish booting.
3. The control plane machines form an Etcd cluster.
4. The bootstrap machine starts a temporary Kubernetes control plane using the newly-created Etcd cluster.
5. The temporary control plane schedules the control plane to the control plane machines.
6. The temporary control plane shuts down and yields to the control plane.
7. The bootstrap node injects components specific to OpenShift into the control plane.
8. Finally, the installer tears down the bootstrap machine.

The result of this bootstrapping process is a fully running OpenShift control plane, which includes the API server, the controllers (such as the SDN), and the Etcd cluster. The cluster then downloads and configures the remaining components needed for day-to-day operation via the Cluster Version operator, including the automated creation of compute machines on supported platforms.

Customizing an OpenShift Installation

The OpenShift installer allows very little customization of the initial cluster that it provisions. Most customization is performed after installation, including:

- Changing the custom resources of cluster operators.
- Adding new operators to a cluster.
- Defining new machine sets.

References

For more information on the various installation methods, refer to the *installing* guide for *Red Hat OpenShift Container Platform 4.2* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html/installing/index

Quiz

Describing Installation Methods

Choose the correct answers to the following questions:

1. Which of the following installation methods requires using the OpenShift installer to configure master and worker nodes?

a. Full-Stack Automation.
b. Pre-existing Infrastructure.
c. Both Full-Stack Automation and Pre-existing Infrastructure.
d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

2. Which of the following installation methods allows setting up nodes using Red Hat Enterprise Linux?

a. Full-Stack Automation.
b. Pre-existing Infrastructure.
c. Both Full-Stack Automation and Pre-existing Infrastructure.
d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

3. Which of the following installation methods allows using an unsupported virtualization provider at the expense of some OpenShift features?

a. Full-Stack Automation.
b. Pre-existing Infrastructure.
c. Both Full-Stack Automation and Pre-existing Infrastructure.
d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

4. Which installation method allows using several supported cloud providers with minimum effort?

a. Full-Stack Automation.
b. Pre-existing Infrastructure.
c. Both Full-Stack Automation and Pre-existing Infrastructure.
d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

5. Which of the following installation methods allows extensive customization of the cluster settings by providing input to the OpenShift installer?

a. Full-Stack Automation.
b. Pre-existing Infrastructure.
c. Both Full-Stack Automation and Pre-existing Infrastructure.
d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

Solution

Describing Installation Methods

Choose the correct answers to the following questions:

1. **Which of the following installation methods requires using the OpenShift installer to configure master and worker nodes?**
   a. Full-Stack Automation.
   b. Pre-existing Infrastructure.
   c. Both Full-Stack Automation and Pre-existing Infrastructure.
   d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

2. **Which of the following installation methods allows setting up nodes using Red Hat Enterprise Linux?**
   a. Full-Stack Automation.
   b. Pre-existing Infrastructure.
   c. Both Full-Stack Automation and Pre-existing Infrastructure.
   d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

3. **Which of the following installation methods allows using an unsupported virtualization provider at the expense of some OpenShift features?**
   a. Full-Stack Automation.
   b. Pre-existing Infrastructure.
   c. Both Full-Stack Automation and Pre-existing Infrastructure.
   d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

4. **Which installation method allows using several supported cloud providers with minimum effort?**
   a. Full-Stack Automation.
   b. Pre-existing Infrastructure.
   c. Both Full-Stack Automation and Pre-existing Infrastructure.
   d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

5. **Which of the following installation methods allows extensive customization of the cluster settings by providing input to the OpenShift installer?**
   a. Full-Stack Automation.
   b. Pre-existing Infrastructure.
   c. Both Full-Stack Automation and Pre-existing Infrastructure.
   d. Neither Full-Stack Automation nor Pre-existing Infrastructure.

Configuring a Lab Environment

Objectives

After completing this section, you should be able to configure the OpenShift cluster used for classroom exercises.

Configuring Your workstation

The lab environment provides a **workstation** virtual machine (VM) on which you run commands required for all hands-on activities in this course. Because your OpenShift cluster and your **workstation** machine are independent entities, you must configure **workstation** with information to access your OpenShift cluster. Keeping the OpenShift cluster and **workstation** machine loosely connected allows customer support to troubleshoot your cluster, or recreate it if required. It also allows your instructor, classmates, and support staff to access your OpenShift cluster to help with troubleshooting, or when you need help completing a hands-on activity.

To configure your **workstation** machine, run the **lab-configure** command that comes preinstalled on the **workstation** machine. Answer all the prompts of the **lab-configure** command. The information that you need is displayed in the Red Hat Online Training Environment (ROL), after the provisioning of your dedicated OpenShift cluster. All prompts from the **lab-configure** command provide full line editing capabilities.

The **lab-configure** command tries to access your OpenShift cluster; if successful, then your configuration is saved in a local configuration file accessible at **/usr/local/etc/ocp4.config**. Most hands-on activities and lab scripts in this course use the shell variables in the configuration file.

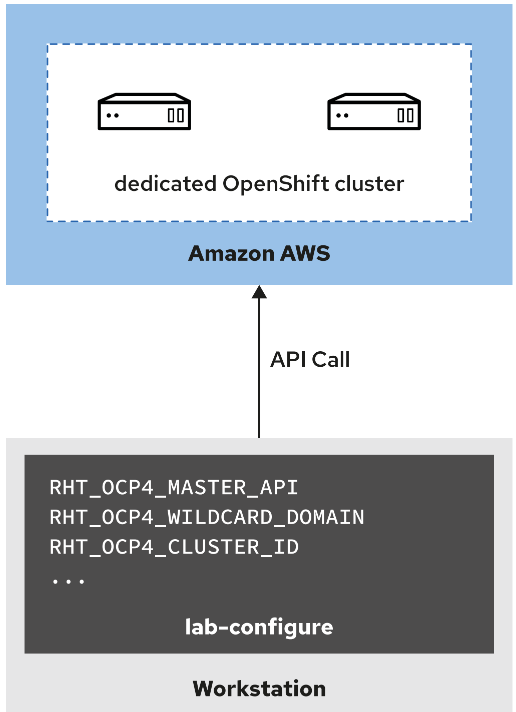


Figure 2.1: Accessing the cluster from your workstation

The following excerpt shows a completed **`/usr/local/etc/ocp4.config`**

```
RHT_OCP4_MASTER_API=https://api.cluster.domain.example.com:6443
RHT_OCP4_WILDCARD_DOMAIN=apps.cluster.domain.example.com
RHT_OCP4_CLUSTER_ID=6ffeab7d-51a8-47e7-9a18-dba9fd46e26c
RHT_OCP4_KUBEADM_PASSWD=ikkaa-6VVJJ-zzIQQ-T6969
RHT_OCP4_USER_PASSWD=9bc3b0f409eb727a68850fe0407849d0
```

If necessary, you can rerun the **`lab-configure`** command again at a later time to point your **`workstation`** machine to a different OpenShift cluster.

Guided Exercise

Configuring a Lab Environment

In this exercise, you will configure the lab environment to access the dedicated, cloud-based OpenShift cluster that ROL provides for each student.

Outcomes

Your lab environment accesses your dedicated OpenShift cluster.

Before You Begin

To perform this exercise, ensure you have access to:

- The Red Hat Online Learning Environment (ROL).
- The access credentials and ID of your dedicated cluster.

> **Important**
> The following activity requires the credentials that you obtained in the Guided Exercise: Creating a Lab Environment. Refer to this activity to retrieve the credentials to connect to your cluster.
>
> The **`lab-configure`** command saves the connection information to your OpenShift cluster in a configuration file. Rerun the command to modify the configuration.

> **Note**
> To avoid problems pasting text into the workstation machine, open the Firefox browser on workstation and navigate to https://rol.redhat.com. Copy the necessary values from Firefox and paste them into the GNOME Terminal window.

1. Run the **`lab-configure`** command to configure your workstation environment to connect to your OpenShift cluster on Amazon Web Services (AWS). The **`lab-configure`** command provides interactive prompts. Enter the connection information to your OpenShift cluster. This information is available on ROL.

    1.1. Run the **`lab-configure`** command and provide the Master API URL of your cluster. Make sure to include **`https://`** if it is not part of the Master API URL provided to you:

```
[student@workstation ~]$ lab-configure

This script configures the connection parameters to access the OpenShift cluster
 for your lab scripts

· Enter the Master API URL: https://api.cluster.domain.example.com:6443
```

1.2. Provide the cluster ID of your cluster:

```
· Enter the Cluster ID: long-and-random-hexadecimal-string
```

1.3. The script attempts to determine the correct wildcard domain for your cluster, as displayed in the following output. If the domain is incorrect, make the necessary changes.

```
· Enter the Wildcard Domain: apps.cluster.domain.example.com
```

1.4. Verify that the **`lab-configure`** identified the correct wildcard domain for your cluster.

```
· Enter the Kubeadmin User Password: long-and-random-password
```

2. The **`lab-configure`** command displays all values that you provided and does not wait for you to confirm them. Then, the **`lab-configure`** command verifies that it can access your cluster.

2.1. Verify that you provided the correct values for your OpenShift cluster to the **`lab-configure`** command.

```
You entered:
 · Master API URL:           https://api.cluster.domain.example.com:6443
 · Cluster ID:               long-and-random-hexadecimal-string
 · Wildcard Domain:          apps.cluster.domain.example.com
 · Kubeadmin User Password:  long-and-random-password
```

2.2. Wait until the **`lab-configure`** command verifies that it can connect to your OpenShift cluster and saves your configuration to the **`/usr/local/etc/ocp4.config`** file.

```
Verifying your Master API URL...

Verifying your kubeadmin user credentials...

Verifying your cluster configuration...

Saving your lab configuration file...

All fine, lab config saved to /usr/local/etc/ocp4.config. You can now proceed with
 your exercises.
```

2.3. If the **`lab-configure`** command finds any issues, then it displays an error message and exits. You must verify the information you got from ROL and run the **`lab-configure`** command again. The following listing shows an example of a verification error:

```
...output omitted...

Verifying your Master API URL...

ERROR:
Cannot connect to an OpenShift 4 API using your URL.
Please verify your network connectivity and that the URL does not point to an
 OpenShift 3.x nor to a non-OpenShift Kubernetes API.
No changes made to your lab configuration.
```

If your configuration saved without errors, then you are ready to start any of the exercises for this course. If there were any errors, then do not start any exercise until you can execute the **`lab-configure`** command successfully.

Finish

This exercise has no command to finish it. You should now be set up to perform any exercise in this course.

This concludes the guided exercise.

Executing Troubleshooting Commands

Objectives

After completing this section, you should be able to:

- Execute commands that assist in troubleshooting common problems.
- Verify that your OpenShift cluster nodes are healthy.
- Troubleshoot common issues with OpenShift and Kubernetes styles of deployment.

Troubleshooting Common Issues with an OpenShift Cluster

Most troubleshooting of the OpenShift cluster is very similar to troubleshooting application deployments, because most components of Red Hat OpenShift 4 are operators, and operators are Kubernetes applications. For each operator, you can identify the project where it resides, the deployment that manages the operator application, and its pods. If that operator has configuration settings that you need to change, then you can identify the custom resource (CR), or sometimes the configuration map or secret resource that stores these settings.

Most OpenShift operators manage applications that are also deployed from standard Kubernetes Workload API resources, such as daemon sets and deployments. The role of the operator is usually to create these resources and keep them in sync with the CR.

This section begins by focusing on cluster issues that are not directly related to operators or application deployments; later in this section, you learn how to troubleshoot application deployments.

Verifying the Health of OpenShift Nodes

The following commands display information about the status and health of nodes in an OpenShift cluster:

`oc get nodes`
: Displays a column with the status of each node. If a node is not **`Ready`**, then it cannot communicate with the OpenShift control plane, and is effectively dead to the cluster.

`oc adm top nodes`
: Displays the current CPU and memory usage of each node. These are actual usage numbers, not the resource requests that the OpenShift scheduler considers as the available and used capacity of the node.

`oc describe node` ***`my-node-name`***
: Displays the resources available and used from the scheduler point of view, and other information. Look for the headings "Capacity," "Allocatable," and "Allocated resources" in the output. The heading "Conditions" indicates whether the node is under memory pressure, disk pressure, or some other condition that would prevent the node from starting new containers.

Reviewing the Cluster Version Resource

To log in to the cluster, start by exporting the **KUBECONFIG** variable, whose value points to the **auth/kubeconfig** file in the installation directory. The **oc** command uses this environment variable to locate the API endpoint of the cluster.

```
[user@demo ~]$ export KUBECONFIG=INSTALL_DIR/auth/kubeconfig
```

Run **oc login** to connect to the cluster with a user name of **kubeadmin**. The password of the **kubeadmin** user is in the **kubeadmin-password** file in the same **auth** directory.

```
[user@dem ~]$ oc login -u kubeadmin -p MMTUc-TnXjo-NFyh3-aeWmC
Login successful.
...output omitted...
```

ClusterVersion is a custom resource that holds high-level information about the cluster, such as the update channels, the status of the cluster operators, and the cluster version (for example 4.2.2). Use this resource to declare the version of the cluster you wish to run. Defining a new version for the cluster instructs the **cluster-version** operator to upgrade the cluster to that version.

You can retrieve the cluster version to verify that it is running the desired version, but also to ensure that the cluster uses the right subscription channel.

- Run **oc get clusterversion** to retrieve the cluster version. The output lists the version, including minor releases, the cluster uptime for a given version, and the overall status of the cluster.

```
[user@demo ~]$ oc get clusterversion
NAME      VERSION  AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.2.2    True        False         14d     Cluster version is 4.2.2
```

- Run **oc describe clusterversion** to obtain more detailed information about the cluster status.

```
[user@demo ~]$ oc describe clusterversion
Name:          version
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   config.openshift.io/v1
Kind:          ClusterVersion
...output omitted...
Spec:
  Channel:     stable-4.2 (1)
  Cluster ID:  f33267f8-260b-40c1-9cf3-ecc406ce035e (2)
  Upstream:    https://api.openshift.com/api/upgrades_info/v1/graph (3)
Status:
  Available Updates: (4)
    Image:     quay.io/openshift-release-dev/ocp-release@sha256:...
    Version:   4.2.13
    Image:     quay.io/openshift-release-dev/ocp-release@sha256:...
    Version:   4.2.7
  Conditions:
```

```
      Last Transition Time:   2019-04-02T23:17:25Z
      Message:                Done applying 4.2.2 5
      Status:                 True
      Type:                   Available
...output omitted...
    Desired:
      Image:    quay.io/openshift-release-dev/ocp-release@sha256:...
      Version:  4.2.2
...output omitted...
    History:
      Completion Time:    2019-04-02T23:17:25Z 6
      Image:              quay.io/openshift-release-dev/ocp-release@sha256:...
      Started Time:       2019-04-02T23:06:33Z
      State:              Completed 7
      Version:            4.2.2
    Observed Generation:  1
    ...output omitted...
```

1. Displays the version of the cluster and its channel. Depending on your subscription, the channel might be different.
2. Displays the unique identifier for the cluster. This identifier is used by Red Hat to identify clusters and cluster entitlements.
3. This URL corresponds to the Red Hat update server. The endpoint allows the cluster to determine its upgrade path when updating to a new version.
4. This entry lists the available images for updating the cluster. In the following example, the cluster is two versions behind the most current OpenShift version; images for the two required updates are available on Red Hat container images registry.
5. This entry lists the history. The output indicates that an update completed.
6. This entry shows when the cluster deployed the version indicated in the **`Version`** entry
7. This entry indicates that the version successfully deployed. Use this entry to determine whether or not the cluster is healthy.

Reviewing Cluster Operators

OpenShift Container Platform *cluster operators* are top level operators that manage the cluster. They are responsible for the main components, such as the API server, the web console, storage, or the SDN. Their information is accessible through the **`ClusterOperator`** resource, which allows you to access the overview of all cluster operators, or detailed information on a given operator.

- Run **`oc get clusteroperators`** to retrieve the list of all cluster operators.

```
[user@demo ~]$ oc get clusteroperators
NAME                    VERSION     AVAILABLE   PROGRESSING    FAILING    SINCE
authentication 1                 4.2.2       True 2         False 3
 False 4       14d
cloud-credential                4.2.2     True        False          False      52d
cluster-autoscaler              4.2.2     True        False          False      52d
console                         4.2.2     True        False          False      33d
dns                             4.2.2     True        False          False      6d5h
image-registry                  4.2.2     True        False          False      6d4h
ingress                         4.2.2     True        False          False      60m
insights                        4.2.2     True        False          False      52d
kube-apiserver                  4.2.2     True        False          False      52d
kube-controller-manager         4.2.2     True        False          False      52d
kube-scheduler                  4.2.2     True        False          False      52d
```

```
machine-api                    4.2.2    True     False       False      52d
machine-config                 4.2.2    True     False       False      33d
marketplace                    4.2.2    True     False       False      60m
monitoring                     4.2.2    True     False       False      60m
network                        4.2.2    True     False       False      52d
node-tuning                    4.2.2    True     False       False      61m
openshift-apiserver            4.2.2    True     False       False      19d
openshift-controller-manager   4.2.2    True     False       False      52d
openshift-samples              4.2.2    True     False       False      52d
operator-lifecycle-manager     4.2.2    True     False       False      52d
...output omitted...
```

1. Indicates the name of the operator, in this case, the operator is responsible for managing authentication.
2. This entry indicates that the operator deployed successfully and is available for use in the cluster. Notice that a cluster operator might return a status of available even if its degraded. An operator reports *degraded* when its current state does not match its desired state over a period of time. For example, if the operator requires three running pods, but one pod is crashing, the operator is available but in a degraded state.
3. This entry indicates whether an operator is being updated to a newer version by the top level operator. If new resources are being deployed by the **`cluster version`** operator, then the columns read **`True`** .
4. The entry returns the health of the operator. The entry reads **`True`** if the operator encounters an error that prevents it from working properly. The operator services might still be available, however, all the requirements might not be satisfied. This can indicate that the operator will fail and require user intervention.

Displaying the Logs of OpenShift Nodes

Most of the infrastructure components of OpenShift are containers inside pods; you can view their logs the same way you view logs for any end-user application. Some of these containers are created by the Kubelet, and thus invisible to most distributions of Kubernetes, but OpenShift cluster operators create pod resources for them.

An OpenShift node based on Red Hat Enterprise Linux CoreOS runs very few local services that would require direct access to a node to inspect their status. Most of the system services in Red Hat Enterprise Linux CoreOS run as containers. The main exceptions are the CRI-O container engine and the Kubelet, which are Systemd units. To view these logs, use the **`oc adm node-logs`** command as shown in the following examples:

```
[user@demo ~]$ oc adm node-logs -u crio my-node-name
```

```
[user@demo ~]$ oc adm node-logs -u kubelet my-node-name
```

You can also display all journal logs of a node:

```
[user@demo ~]$ oc adm node-logs my-node-name
```

Opening a Shell Prompt on an OpenShift Node

Administrators who manage Red Hat OpenShift Cluster Platform 3 and other distributions of Kubernetes frequently open SSH sessions to their nodes to inspect the state of the control plane and the container engine, or to make changes to configuration files. Although this can still be done, it is no longer recommended with Red Hat OpenShift Cluster Platform 4.

If you install your cluster using the full-stack automation method, then your cluster nodes are not directly accessible from the internet because they are on a virtual private network, which AWS calls Virtual Private Cloud (VPC). To open SSH sessions, a bastion server on the same VPC of your cluster that is also assigned a public IP address is required. Creating a bastion server depends on your cloud provider and is out of scope for this course.

The **oc debug node** command provides a way to open a shell prompt in any node of your cluster. That prompt comes from a special-purpose tools container that mounts the node root file system at the **/host** folder, and allows you to inspect any files from the node.

To run local commands directly from the node, while in a **oc debug node** session, you must start a chroot shell in the **/host** folder. Then you can inspect the local file systems of the node, the status of its systemd services, and perform other tasks that would otherwise require a SSH session. The following is an example **oc debug node** session:

```
[user@demo ~]$ oc debug node/my-node-name
...output omitted...
sh-4.2# chroot /host
sh-4.2# systemctl is-active kubelet
active
```

A shell session started from the **oc debug node** command depends on the OpenShift control plane to work. It uses the same tunneling technology that allows opening a shell prompt inside a running pod (see the **oc rsh** command later in this section). The **oc debug node** command is not based on the SSH or RSH protocols.

If your control plane is not working, your node is not ready, or for some reason your node is not able to communicate with the control plane, then you cannot rely on the **oc debug node** command and will require a bastion host.

> **Warning**
> Exercise care when using the **oc debug node** command. Some actions can render your node unusable, such as stopping the Kubelet, and you cannot recover using only **oc** commands.

Troubleshooting The Container Engine

From an **oc debug node** session, use the **crictl** command to get low-level information about all local containers running on the node. You cannot use the **podman** command for this task because it does not have visibility on containers created by CRI-O. The following example lists all containers running on a node. The **oc describe node** command provides the same information but organized by pod instead of by container.

```
[user@demo ~]$ oc debug node/my-node-name
...output omitted...
sh-4.2# chroot /host
sh-4.2# crictl ps
...output omitted...
```

Troubleshooting Application Deployments

You can usually ignore the differences between Kubernetes deployments and OpenShift deployment configurations when troubleshooting applications. The common failure scenarios and the ways to troubleshoot them are essentially the same.

There are many scenarios that will be described in later chapters of this course, such as pods that cannot be scheduled. This section focuses on common scenarios that apply to generic applications, and the same scenarios usually apply to operators also.

Troubleshooting Pods That Fail to Start

A common scenario is that OpenShift creates a pod and that pod never establishes a **Running** state. This means that OpenShift could not start the containers inside that pod. Start troubleshooting using the **oc get pod** and **oc status** commands to verify whether your pods and containers are running. At some point, the pods are in an error state, such as **ErrImagePull** or **ImagePullBackOff**.

When this happens, the first step is listing events from the current project using the **oc get events** command. If your project contains many pods, then you can get a list of events filtered by pod using the **oc describe pod** command. You can also run similar **oc describe** commands to filter events by deployments and deployment configurations.

Troubleshooting Running and Terminated Pods

Another common scenario is that OpenShift creates a pod, and for a short time no problem is encountered. The pod enters the **Running** state, which means at least one of its containers started running. Later, an application running inside one of the pod containers stops working. It might either terminate or return error messages to user requests.

If the application is managed by a properly designed deployment, then it should include health probes that will eventually terminate the application and stop its container. If that happens, then OpenShift tries to restart the container several times. If the application continues terminating, due to health probes or other reasons, then the pod will be left in the **CrashLoopBackOff** state.

A container that is running, even for a very short time, generates logs. These logs are not discarded when the container terminates. The **oc logs** command displays the logs from any container inside a pod. If the pod contains a single container, then the **oc logs** command only requires the name of the pod.

```
[user@demo ~]$ oc logs my-pod-name
```

If the pod contains multiple containers, then the **oc logs** command requires the **-c** option.

```
[user@demo ~]$ oc logs my-pod-name -c my-container-name
```

Interpreting application logs requires specific knowledge of that particular application. If all goes well, the application provides clear error messages that can help you find the problem.

Introducing OpenShift Aggregated Logging

Red Hat OpenShift Container Platform 4 provides the Cluster Logging subsystem, based on Elasticsearch, Fluentd or Rsyslog, and Kibana, which aggregates logs from the cluster and its containers.

Deploying and configuring the OpenShift Cluster Logging subsystem through its operator is beyond the scope of this course. Refer to the references section at the end of this section for more information.

Creating Troubleshooting Pods

If you are not sure whether your issues relate to the application container image, or to the settings it gets from its OpenShift resources, then the **`oc debug`** command is very useful. This command creates a pod based on an existing pod, a deployment configuration, a deployment, or any other resource from the Workloads API.

The new pod runs an interactive shell instead of the default entry point of the container image. It also runs with health probes disabled. This way, you can easily verify environment variables, network access to other services, and permissions inside the pod.

The command-line options of the **`oc debug`** command allow you to specify settings that you do not want to clone. For example, you could change the container image, or specify a fixed user id. Some settings might require cluster administrator privileges.

A common scenario is creating a pod from a deployment, but running as the root user and thus proving that the deployment references a container image that was not designed to run under the default security policies of OpenShift:

```
[user@demo ~]$ oc debug deployment/my-deployment-name --as-root
```

Changing a Running Container

Because container images are immutable, and containers are supposed to be ephemeral, it is not recommended that you make changes to running containers. However, sometimes making these changes can help with troubleshooting application issues. After you try changing a running container, do not forget to apply the same changes back to the container image and its application resources, and then verify that the now permanent fixes work as expected.

The following commands help with making changes to running containers. They all assume that pods contain a single container. If not, you must add the **`-c`** ***`my-container-name`*** option.

`oc rsh` ***`my-pod-name`***
: Opens a shell inside a pod to run shell commands interactively and non-interactively.

`oc cp` ***`/local/path my-pod-name`*****`:`*****`/container/path`***
: Copies local files to a location inside a pod. You can also reverse arguments and copy files from inside a pod to your local file system. See also the **`oc rsync`** command for copying multiple files at once.

`oc port-forward` ***`my-pod-name local-port`*****`:`*****`remote-port`***
: Creates a TCP tunnel from ***`local-port`*** on your workstation to ***`local-port`*** on the pod. The tunnel is alive as long as you keep the **`oc port-forward`** running. This allows you to get network access to the pod without exposing it through a route. Because the tunnel starts at your localhost, it cannot be accessed by other machines.

Troubleshooting OpenShift CLI Commands

Sometimes, you cannot understand why an **`oc`** command fails and you need to troubleshoot its low-level actions to find the cause. Maybe you need to know what a particular invocation of the **`oc`** command does behind the scenes, so you can replicate the behavior with an automation tool that makes OpenShift and Kubernetes API requests, such as Ansible Playbooks using the **`k8s`** module.

The **`--log-level`** ***`level`*** option displays OpenShift API requests, starting with level 6. As you increase the level, up to 10, more information about those requests is added, such as their HTTP request headers and response bodies. Level 10 also includes a **`curl`** command to replicate each request.

You can try these two commands, from any project, and compare their outputs.

```
[user@demo ~]$ oc get pod --log-level 6
```

```
[user@demo ~]$ oc get pod --log-level 10
```

Sometimes, you only need the authentication token that the **`oc`** command uses to authenticate OpenShift API requests. With this token, an automation tool can make OpenShift API requests as if it was logged in as your user. To get your token, use the **`-t`** option of the **`oc whoami`** command:

```
[user@demo ~]$ oc whoami -t
```

References

For more information about OpenShift events, *Working with clusters* chapter in the *Red Hat OpenShift Container Platform 4.2 Nodes Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/nodes/index#nodes-containers-events

For more information about how to copy files to running containers, refer to the *Working with containers* chapter in the *Red Hat OpenShift Container Platform 4.2 Nodes Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/nodes/index#nodes-containers-copying-files

For more information about how to execute commands on running containers, refer to the *Working with containers* chapter in the *Red Hat OpenShift Container Platform 4.2 Nodes Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/nodes/index#nodes-containers-remote-commands

For more information about how to forward local ports to running containers, refer to the *Working with containers* chapter in the *Red Hat OpenShift Container Platform 4.2 Nodes Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/nodes/index#nodes-containers-port-forwarding

For more information about aggregated logging, refer to the *Red Hat OpenShift Container Platform 4.2 Logging Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/logging/index

ClusterOperator Custom Resource
https://github.com/openshift/cluster-version-operator/blob/master/docs/dev/clusteroperator.md

Guided Exercise

Executing Troubleshooting Commands

In this exercise, you will execute commands that assist in troubleshooting common problems with the OpenShift control plane and with application deployments.

Outcomes

You should be able to:

- Inspect the general state of an OpenShift cluster.
- Inspect local services and pods running in an OpenShift worker node.
- Diagnose and fix issues with the deployment of an application.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**`/usr/local/bin/oc`**).
- The PostgreSQL container image from Red Hat (**`registry.access.redhat.com/rhscl/postgresql-96-rhel7:1`**).

As the **`student`** user on the **`workstation`** machine, use the **`lab`** command to prepare your system for the exercise.

This command ensures that the cluster API is reachable, and creates the resource files that you will be using in the activity. It also creates the **`execute-troubleshoot`** project with an application that you will diagnose and fix during this exercise.

```
[student@workstation ~]$ lab execute-troubleshoot start
```

1. Log in to the OpenShift cluster and inspect the status of your cluster nodes.

   1.1. Source the classroom configuration file that is accessible at **`/usr/local/etc/ocp4.config`**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to the cluster as the **`kubeadmin`** user. When prompted, accept the insecure certificate.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
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.
...output omitted...
```

1.3. Verify that all nodes on your cluster are ready. The names of the nodes from your cluster will differ.

```
[student@workstation ~]$ oc get nodes
NAME                                         STATUS   ROLES    ...
ip-10-0-134-122.us-west-1.compute.internal   Ready    master   ...
ip-10-0-140-84.us-west-1.compute.internal    Ready    worker   ...
ip-10-0-152-96.us-west-2.compute.internal    Ready    worker   ...
...output omitted...
```

1.4. Verify whether any of your worker nodes are close to using all of the CPU and memory available to them. Use the **`node-role.kubernetes.io/worker`** label to filter the **`oc adm top`** command output to show only worker nodes.

Repeat the following command a few times to prove that you see actual usage of CPU and memory from your nodes. The numbers you see should change slightly each time you repeat the command.

```
[student@workstation ~]$ oc adm top node -l node-role.kubernetes.io/worker
NAME                                        CPU(cores)  CPU%   MEMORY(bytes)  ...
ip-10-0-140-84.us-west-1.compute.internal   489m        13%    6447Mi         ...
ip-10-0-152-96.us-west-2.compute.internal   159m        4%      921Mi         ...
...output omitted...
```

1.5. Pick up one of your worker nodes and pass its name to the **`oc describe`** command to verify that all of the conditions that might indicate problems are false.

```
[student@workstation ~]$ oc describe node \
>    ip-10-0-140-84.us-west-1.compute.internal
...output omitted...
Conditions:
  Type             Status   ...   Message
  ----             ------   ...   -------
  MemoryPressure   False    ...   kubelet has sufficient memory available
  DiskPressure     False    ...   kubelet has no disk pressure
  PIDPressure      False    ...   kubelet has sufficient PID available
  Ready            True     ...   kubelet is posting ready status
Addresses:
...output omitted...
```

2. Review the logs of the internal registry operator, the internal registry server, and the Kubelet of a node. You are not looking for anything in these logs, but you must be able to find them.

2.1. List all pods inside the **openshift-image-registry** project, and then identify the pod that runs the operator and the pod that runs the internal registry server.

```
[student@workstation ~]$ oc get pod -n openshift-image-registry
NAME                                               READY   STATUS    ...
cluster-image-registry-operator-564bd5dd8f-s46bz   2/2     Running   ...
image-registry-794dfc7978-w7w69                    1/1     Running   ...
...output omitted...
```

2.2. Follow the logs of the operator pod (**cluster-image-registry-operator-*xxx***). The following command fails because that pod has two containers.

```
[student@workstation ~]$ oc logs --tail 3 -n openshift-image-registry \
>    cluster-image-registry-operator-564bd5dd8f-s46bz
Error from server (BadRequest): a container name must be specified for pod
 cluster-image-registry-operator-564bd5dd8f-s46bz, choose one of: [cluster-image-
registry-operator cluster-image-registry-operator-watch]
```

2.3. Follow the logs of the first container of the operator pod. Your output might be different than the following example.

```
[student@workstation ~]$ oc logs --tail 3 -n openshift-image-registry \
>    -c cluster-image-registry-operator \
>    cluster-image-registry-operator-564bd5dd8f-s46bz
I0925 13:15:48.252294      13 controller.go:260] event from workqueue successfully
 processed
I0925 13:15:51.261479      13 controller.go:260] event from workqueue successfully
 processed
I0925 13:15:54.273422      13 controller.go:260] event from workqueue successfully
 processed
```

2.4. Follow the logs of the image registry server pod (**image-registry-*xxx*** from the output of the **oc get pod** command run previously). Your output might be different than the following example.

```
[student@workstation ~]$ oc logs --tail 3 -n openshift-image-registry \
>     image-registry-794dfc7978-w7w69
time="2019-09-25T21:18:06.824891032Z" level=info msg="authorized
 request" go.version=go1.11.6 http.request.host="10.129.2.44:5000"
 http.request.id=e160dea2-d336-48aa-aed3-cf154343f32b http.request.method=GET
 http.request.remoteaddr="10.129.2.50:59500" http.request.uri=/
extensions/v2/metrics http.request.useragent=Prometheus/2.11.0
 openshift.auth.user="system:serviceaccount:openshift-monitoring:prometheus-k8s"
time="2019-09-25T21:18:06.827613809Z" level=info msg="response
 completed" go.version=go1.11.6 http.request.host="10.129.2.44:5000"
 http.request.id=e160dea2-d336-48aa-aed3-cf154343f32b http.request.method=GET
 http.request.remoteaddr="10.129.2.50:59500" http.request.uri=/extensions/v2/
metrics http.request.useragent=Prometheus/2.11.0 http.response.contenttype="text/
plain; version=0.0.4" http.response.duration=12.008217ms http.response.status=200
 http.response.written=2326
time="2019-09-25T21:18:06.827703727Z" level=info msg=response
 go.version=go1.11.6 http.request.host="10.129.2.44:5000"
 http.request.id=f4d83df5-8ed7-4651-81d4-4ed9f758c67d http.request.method=GET
 http.request.remoteaddr="10.129.2.50:59500" http.request.uri=/extensions/v2/
metrics http.request.useragent=Prometheus/2.11.0 http.response.contenttype="text/
plain; version=0.0.4" http.response.duration=12.141585ms http.response.status=200
 http.response.written=2326
```

2.5. Follow the logs of the Kubelet from the same node that you inspected for CPU and memory usage in the previous step. Your output might be different than the following example.

```
[student@workstation ~]$ oc adm node-logs --tail 3 -u kubelet \
>     ip-10-0-140-84.us-west-1.compute.internal
-- Logs begin at Fri 2019-09-20 09:39:27 UTC, end at Wed 2019-09-25 13:19:54 UTC.
 --
Sep 25 13:19:54 ip-10-0-140-84 hyperkube[2015]: I0925 13:19:54.325329
    2015 prober.go:125] Liveness probe for "kube-apiserver-
ip-10-0-140-84.us-west-2.compute.internal_openshift-kube-
apiserver(bfe09eb62916b8951088b20e1aa485f0):kube-apiserver-12" succeeded
Sep 25 13:19:54 ip-10-0-140-84 hyperkube[2015]: I0925 13:19:54.375124
    2015 prober.go:125] Readiness probe for "kube-apiserver-
ip-10-0-140-84.us-west-2.compute.internal_openshift-kube-
apiserver(bfe09eb62916b8951088b20e1aa485f0):kube-apiserver-12" succeeded
Sep 25 13:19:54 ip-10-0-140-84 hyperkube[2015]: I0925 13:19:54.701851
    2015 prober.go:125] Liveness probe for "kube-controller-manager-
ip-10-0-140-84.us-west-2.compute.internal_openshift-kube-controller-
manager(07e7f7c664aa27443aec3bb3245638c7):kube-controller-manager-7" succeeded
```

3. Start a shell session to the same worker node that you previously used to inspect its OpenShift services and pods. Do not make any change to the node, such as stopping services or editing configuration files.

3.1. Start a shell session on the working node, and then use the **chroot** command to enter the local file system of the host.

```
[student@workstation ~]$ oc debug node/ip-10-0-140-84.us-west-1.compute.internal
Starting pod/ip-10-0-140-20us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.0.140.84
If you don't see a command prompt, try pressing enter.
sh-4.2# chroot /host
sh-4.2#
```

3.2. Still using the same shell session, verify that the Kubelet and the CRI-O container engine are running. Type **q** to exit the command.

```
sh-4.2# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset:
 enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
           └─10-default-env.conf
   Active: active (running) since Mon 2019-09-23 16:41:32 UTC; 3h 59min ago
...output omitted...
q
```

Rerun the same command against the **cri-o** service. Type **q** to exit from the command.

```
sh-4.2# systemctl status cri-o
● crio.service - Open Container Initiative Daemon
   Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; vendor preset:
 disabled)
  Drop-In: /etc/systemd/system/crio.service.d
           └─10-default-env.conf
   Active: active (running) since Mon 2019-09-23 16:41:30 UTC; 4h 1min ago
...output omitted...
q
```

3.3. Still using the same shell session, verify that the **openvswitch** pod is running.

```
sh-4.2# crictl ps --name openvswitch
CONTAINER ID    ...   STATE     NAME          ATTEMPT   POD ID
13f0b0ed3497a   ...   Running   openvswitch   1         4bc278dddf007
```

3.4. Terminate the **chroot** session and shell session to the node. This also terminates the **oc debug node** command.

```
sh-4.4# exit
exit
sh-4.2# exit
exit

Removing debug pod ...
[student@workstation ~]$
```

4. Enter the **execute-troubleshoot** project to diagnose a pod that is in an error state.

4.1. Use the **execute-troubleshoot** project.

```
[student@workstation ~]$ oc project execute-troubleshoot
Now using project "execute-troubleshoot" on server
"https://api.cluster.domain.example.com:6443".
```

4.2. Verify that the project has a single pod in either the **ErrImagePull** or **ImagePullBackOff** status.

```
[student@workstation ~]$ oc get pod
NAME                     READY   STATUS             ...
psql-7d4cc9d6d-m5r59     0/1     ImagePullBackOff   ...
```

4.3. Verify that the project includes a Kubernetes deployment that manages the pod.

```
[student@workstation ~]$ oc status
...output omitted...
deployment/psql deploys registry.access.redhat.com/rhscl/postgresq-96-rhel7:1
  deployment #1 running for 8 minutes - 0/1 pods
...output omitted...
```

4.4. List all events from the current project and look for error messages related to the pod.

```
[student@workstation ~]$ oc get events
LAST SEEN   TYPE      REASON              OBJECT                       MESSAGE
112s        Normal    Scheduled           pod/psql-7d4cc9d6d-m5r59     Successfully
 assigned execute-troubleshoot/psql-7d4cc9d6d-m5r59 to ip-10-0-143-197.us-
west-1.compute.internal
21s         Normal    Pulling             pod/psql-7d4cc9d6d-m5r59     Pulling
 image "registry.access.redhat.com/rhscl/postgresq-96-rhel7:1"
21s         Warning   Failed              pod/psql-7d4cc9d6d-m5r59
  Failed to pull image "registry.access.redhat.com/rhscl/postgresq-96-
rhel7:1": rpc error: code = Unknown desc = Error reading manifest 1-40 in
 registry.rht-16-1.dev.nextcle.com/rhscl/postgresq-96-rhel7: manifest unknown:
 manifest unknown
21s         Warning   Failed              pod/psql-7d4cc9d6d-m5r59     Error:
 ErrImagePull
8s          Normal    BackOff             pod/psql-7d4cc9d6d-m5r59     Back-off
 pulling image "registry.access.redhat.com/rhscl/postgresq-96-rhel7:1"
8s          Warning   Failed              pod/psql-7d4cc9d6d-m5r59     Error:
 ImagePullBackOff
112s        Normal    SuccessfulCreate    replicaset/psql-7d4cc9d6d    Created pod:
 psql-7d4cc9d6d-m5r59
112s        Normal    ScalingReplicaSet   deployment/psql              Scaled up
 replica set psql-7d4cc9d6d to 1
```

This output also indicates a problem getting the image for deploying the pod.

4.5. Use Skopeo to find information about the container image from the events.

```
[student@workstation ~]$ skopeo inspect \
>    docker://registry.access.redhat.com/rhscl/postgresq-96-rhel7:1
FATA[0003] Error reading manifest 1 in registry.access.redhat.com/rhscl/
postgresq-96-rhel7: name unknown: Repo not found
```

4.6. It looks like the container image is misspelled. Verify that it works if you replace **`postgresq-96-rhel7`** with **`postgresql-96-rhel7`**.

```
[student@workstation ~]$ skopeo inspect \
>    docker://registry.access.redhat.com/rhscl/postgresql-96-rhel7:1
{
    "Name": "registry.access.redhat.com/rhscl/postgresql-96-rhel7",
...output omitted...
```

4.7. To verify that the image name is the root cause of the error, edit the **`psql`** deployment to correct the name of the container image. The **`oc edit`** commands uses **`vi`** as the default editor.

> **Warning**
> In a real-world scenario, you would ask whoever deployed the PostgreSQL database to fix their YAML and redeploy their application.

```
[student@workstation ~]$ oc edit deployment/psql
...output omitted...
    spec:
      containers:
      - env:
        - name: POSTGRESQL_DATABASE
          value: db
        - name: POSTGRESQL_PASSWORD
          value: pass
        - name: POSTGRESQL_USER
          value: user
        image: registry.access.redhat.com/rhscl/postgresql-96-rhel7:1
...output omitted...
```

4.8. Verify that a new deployment is active.

```
[student@workstation ~]$ oc status
...output omitted...
  deployment #2 running for 10 seconds - 0/1 pods
  deployment #1 deployed 5 minutes ago
```

4.9. List all pods in the current project. You might see both the old failing pod and the new pod for a few moments. Repeat the following command until you see that the new pod is ready and running, and you no longer see the old pod.

```
[student@workstation ~]$ oc get pods
NAME                                 READY    STATUS     RESTARTS    AGE
psql-544c9c666f-btlw8  1/1     Running    0          55s
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab execute-troubleshoot finish
```

This concludes the guided exercise.

Summary

In this chapter, you learned:

- Red Hat OpenShift Container Platform provides two main installation methods: full-stack automation, and pre-existing infrastructures.
- Future releases are expected to add more cloud and virtualization providers, such as VMware, Red Hat Virtualization, and IBM System Z.
- An OpenShift node based on Red Hat Enterprise Linux CoreOS runs very few local services that would require direct access to a node to inspect their status. Most of the system services run as containers, the main exceptions are the CRI-O container engine and the Kubelet.
- The **`oc get node`**, **`oc adm top`**, **`oc adm node-logs`**, and **`oc adm debug`** commands provide troubleshooting information about OpenShift nodes.

Chapter 3

Configuring Authentication

Goal	Configure authentication with an identity provider
Objectives	• Describe the resources associated with OpenShift authentication and authorization. • Authenticate as a cluster administrator using the kubeconfig file or kubeadmin virtual user. • Configure the HTPasswd identity provider for OpenShift authentication. • Assign a user to a cluster role.
Sections	• Configuring Identity Providers (and Guided Exercise)

Configuring Identity Providers

Objectives

After completing this section, you should be able to Configure the HTPasswd identity provider for OpenShift authentication.

Describing OpenShift Users and Groups

There are a number of OpenShift resources related to authentication and authorization. The following is a list of the main resource types and their definitions:

User
: In the OpenShift Container Platform architecture, users are entities that interact with the API server. The user resource is a representation of an actor within the system. Assign permissions by adding roles to the user directly, or to the groups of which the user is a member.

Identity
: Identity is a resource that keeps a record of successful authentication attempts from a specific user and identity provider. Any data concerning the source of the authentication is stored on the identity. Only a single user resource is associated with an identity resource.

Service Account
: In OpenShift, applications can communicate with the API independently when user credentials cannot be acquired. To preserve the integrity of a regular user's credentials, credentials are not shared and service accounts are used instead. Service accounts enable you to control API access without the need borrow a regular user's credentials.

Group
: Groups represent a specific set of users. Users are assigned to one or to multiple groups. Groups are leveraged when implementing authorization policies to assign permissions to multiple users at the same time. For example, if you want to allow twenty users access to objects within a project, it is advantageous to use a group instead of granting access to each of the users individually. OpenShift Container Platform also provides system groups or *virtual groups* that are provisioned automatically by the cluster.

Role
: A role is a set of permissions that enables a user to perform API operations over one or more resource types. You grant permissions to users, groups, and service accounts by assigning roles to them.

User and identity resources are usually not created in advance. They are usually created automatically by OpenShift after a successful interactive log in using OAuth.

Authenticating API Requests

Authorization and Authentication are the two security layers responsible for enabling a user to interact with the cluster. When a user makes a request to the API, that user is associated with the request. The authentication layer is responsible for identifying the user. Information concerning the requesting user from the authentication layer is then used by the authorization layer to determine if the request is honored. After a user is authenticated, the RBAC policy determines what the user is authorized to do. If an API request contains invalid authentication, it is authenticated as a request by the anonymous system user.

The OpenShift API has two methods for authenticating requests:

- OAuth Access Tokens
- X.509 Client Certificates

If the request does not present an access token or certificate, then the authentication layer assigns it the **`system:anonymous`** virtual user, and the **`system:unauthenticated`** virtual group.

Introducing the Authentication Operator

The OpenShift Container Platform provides the Authentication operator, which runs an OAuth server. The OAuth server provides OAuth access tokens to users when they attempt to authenticate to the API. An identity provider must be configured and available to the OAuth server. The OAuth server uses an identity provider to validate the identity of the requester. The server reconciles the user with the identity and creates the OAuth access token that is then granted to the user. Identity and user resources are created automatically upon logging in.

Introducing Identity Providers

OpenShift OAuth server can be configured to use a number of identity providers. The following lists includes the more common ones:

HTPasswd
: Validate user names and passwords against a secret that stores credentials generated using the **`htpasswd`**.

Keystone
: Enables shared authentication with an OpenStack Keystone v3 server.

LDAP
: Configure the LDAP identity provider to validate user names and passwords against an LDAPv3 server, using simple bind authentication.

GitHub or GitHub Enterprise
: Configure a GitHub identity provider to validate user names and passwords against GitHub or the GitHub Enterprises OAuth authentication server.

OpenID Connect
: Integrates with an OpenID Connect identity provider using an Authorization Code Flow.

The OAuth custom resource must be updated with your desired identity provider. You can define multiple identity providers, of the same or different kinds, on the same OAuth custom resource.

Authenticating as a Cluster Administrator

Before you can configure an identity provider and manage users, you must access your OpenShift cluster as a cluster administrator. A just-installed OpenShift cluster provides two ways to authenticate API requests with cluster administrator privileges:

- The **`kubeadmin`** virtual user and password that grants an OAuth access token.
- The **`kubeconfig`** file that embeds an X.509 client certificate that never expires.

To create additional users and grant them different access levels, you must configure an identity provider and assign roles to your users.

Authenticating Using the X.590 Certificate

The **kubeconfig** file is created during the installation process and is specific to the cluster. It is stored in the **auth** directory that the OpenShift installer created during the installation. The file contains specific details and parameters used by the CLI to connect a client to the correct API server, including an X.509 certificate.

The installation logs provide the location where the **kubeconfig** file is saved:

```
INFO Run 'export KUBECONFIG=root/auth/kubeconfig' to manage the cluster with 'oc'.
```

To use the **kubeconfig** file to authenticate **oc** commands, you must copy the file to your workstation and set the absolute or relative path to the **KUBECONFIG** environment variable. Then, you can run any **oc** that requires cluster administrator privileges without logging in to OpenShift.

```
[user@demo ~]$ export KUBECONFIG=/home/user/auth/kubeconfig
[user@demo ~]$ oc get nodes
```

As an alternative, you can use the **--config** option of the **oc** command.

```
[user@demo ~]$ oc --config /home/user/auth/kubeconfig get nodes
```

Authenticating Using the Virtual User

After an installation has completed, OpenShift creates the **kubeadmin** virtual user. There are no user or identity resources for that virtual user. The **kubeadmin** user is hard-coded as a cluster administrator and you cannot change its privileges. It is also hard-coded to authenticate using the password stored in **kubeadmin**, from the **kube-system** project.

The **kubeadmin** password is dynamically generated by the installer and completely unique to your environment. The installation logs provide **kubeadmin** credentials used to log in to the cluster. The cluster installation logs also provide log in, password, and the URL for console access.

```
..output omitted..
INFO The cluster is ready when'oc login -u kubeadmin -p shdU_trbi_6ucX_edbu_aqop'
...output omitted...
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.demo.openshift4.2-gls.com
INFO Login to the console with user: kubeadmin, password: shdU_trbi_6ucX_edbu_aqop
```

Deleting the Virtual User

After you define an identity provider, create a new user, and assign that user the **cluster-admin** role, you can remove the **kubeadmin** user credentials to improve cluster security.

```
[user@demo ~]$ oc delete secret kubeadmin -n kube-system
```

> **Warning**
> If you delete the **kubeadmin** secret before you configure another user with cluster admin privileges, the only way you can administer your cluster would be using the **kubeconfig** file. If you do not have a copy of this file in a safe location, then there is no way to recover administrative access to your cluster. The only alternative is destroying and reinstalling your cluster.

Configuring the HTPasswd Identity Provider

The HTPasswd identity provider validates users against a secret that that contains user names and passwords generated with the **htpasswd** command from the Apache HTTP Server project. Only a cluster administrator is allowed to change the data inside the secret. Regular users cannot change their own passwords.

Managing users using the HTPasswd Identity Provider might suffice for a proof-of-concept environment with a small set of users. However, most production environments require a more powerful identity provider that integrates with the organization's identity management system.

Configuring the OAuth Custom Resource

To use the HTPasswd identity provider, the OAuth custom resource must be edited to add an entry to the **.spec.identityProviders** array:

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_htpasswd_provider (1)
    mappingMethod: claim (2)
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret (3)
```

1. This provider name is prefixed to provider user names to form an identity name.
2. Controls how mappings are established between provider identities and user objects.
3. An existing secret containing a data generated using the **htpasswd** command.

Updating the OAuth Custom Resource

To edit update the OAuth custom resource, use the **oc get** command to export the existing OAuth cluster resource to a file in YAML format.

```
[user@demo ~]$ oc get -o yaml oauth cluster > oauth.yaml
```

Then, open the resulting file in a text editor and make the needed changes to the embedded identity provider settings.

After completing modifications and saving the file, you must apply the new custom resource using the **oc replace** command.

```
[user@demo ~]$ oc create -f oauth.yaml
```

Managing Users with the HTPasswd Identity Provider

Managing user credentials with the HTPasswd Identity Provider requires creating a temporary **htpasswd** file, making changes to the file, and applying these changes to the secret.

Creating an HTPasswd File

The **htpasswd** utility is part of the **httpd-utils** package and it must be installed and available on your system.

Create or update the **htpasswd** file.

```
[user@demo ~]$ htpasswd -c -B -b htpasswd student redhat123
```

Add or update credentials.

```
[user@demo ~]$ htpasswd -b htpasswd student redhat1234
```

Delete credentials.

```
[user@demo ~]$ htpasswd -D htpasswd student
```

Creating the HTPasswd Secret

To use the HTPasswd provider, you must create a secret that contains the **htpasswd** file data. In the following example, the secret is named **htp-secret**.

```
[user@demo ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/home/user/htpasswd -n openshift-config
```

Updating the HTPasswd Secret

The secret must be updated after adding, changing, or deleting users. All data inside a secret must be encoded in base64. One way to encode the data is using the **oc create secret**, as if you are creating a new secret and sending the output YAML to the standard output. Then, you can pipe that output to the **oc replace** command to update the existing secret. To update the secret named **htp-secret**, run the following command:

```
[user@demo ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/path/to/your/file  --dry-run -o yaml \
>    | oc replace -n openshift-config -f -
```

When you attempt to log in after updating the secret you might receive an error. If this occurs, then wait a few moments and attempt to log in again. When the previous **oc create secret** command completes successfully, the OAuth operator reloads and it takes some time to update data from the secret.

Extracting Secret Data

When adding or removing users, an admin cannot assume the validity of a local htpasswd file. Moreover, the admin might not even be on a system that has the htpasswd file. In a real world scenario, it would behoove the administrator to use the **oc extract** command. The extract command downloads the contents of a configuration map or secret into a directory. The data can then be redirected to a file or displayed as standard output. To extract data from the secret and redirect it to a file name **temp**, use the following command:

```
[user@demo ~]$ oc extract secret/htp-secret -n openshift-config \
>    --to - > temp
```

Deleting Users and Identities

When a scenario occurs that requires you to delete a user, it is not sufficient to delete the user from the identity provider. The user and identity resources must also be deleted.

You must remove the password from the htpasswd secret. The user must be removed from the local htpasswd file, and then the secret must be updated.

To delete the user from htpasswd, run the following command:

```
[user@demo ~]$ htpasswd -D temp manager
```

Update the secret to remove all remnants of the user's password.

```
[user@demo ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/home/user/temp  --dry-run -o yaml \
>    | oc replace -n openshift-config -f -
```

Remove the user resource with the following command:

```
[user@demo ~]$ oc delete user manager
```

To delete all identity resources, run the following command:

```
[user@demo ~]$ oc delete identity manager
```

Assigning Administrative Privileges

The cluster-wide **cluster-admin** role grants cluster administration privileges to users and groups. It enables the user to perform any action on any resources within the cluster. In the following example, the **cluster-admin** role is assigned to the **student** user.

```
[user@demo ~]$ oc adm policy add-cluster-role-to-user cluster-admin student
```

References

For more information on identity providers, refer to the *Understanding identity provider configuration* section of the *Authentication* chapter in the *Red Hat OpenShift Container Platform 4.2 Authentication* guide at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/authentication/index

Guided Exercise

Configuring Identity Providers

In this exercise, you will configure the HTPasswd identity provider and create users for cluster administrators.

Outcomes

You should be able to:

- Create users and passwords for HTPasswd authentication.
- Configure the Identity Provider for HTPasswd authentication.
- Assign cluster administration rights to users.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**/usr/local/bin/oc**)

On **workstation**, run the following **lab** command to prepare your system for the exercise.

```
[student@workstation ~]$ lab auth-provider start
```

The command ensures that the cluster API is reachable, the **httpd-utils** package is installed, and that the authentication settings are configured to the installation defaults.

1. Add an entry for two htpasswd users, **admin** and **developer**. Assign the password from the variable **RHT_OCP4_USER_PASSWD** to both users.

    1.1. Source the classroom configuration file that is accessible at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

    1.2. Create an htpasswd authentication file named **temp** in the **~/DO280/labs/auth-provider** directory, and add the **admin** user with the password **${RHT_OCP4_USER_PASSWD}**. The name of the file is arbitrary and you can choose any name you prefer. In the following and subsequent examples throughout the exercise, the file is named **~/DO280/labs/auth-provider/temp**.

    The **htpasswd** command populates the htpasswd authentication file, **~/DO280/labs/auth-provider/temp**, with the user names and base64 encoded passwords.

```
[student@workstation ~]$ htpasswd -c -B -b ~/DO280/labs/auth-provider/temp admin
 ${RHT_OCP4_USER_PASSWD}
Adding password for user admin
```

1.3. Update your **~/DO280/labs/auth-provider/temp** file, and then add the **developer** user with the password **${RHT_OCP4_USER_PASSWORD}**. The **~/DO280/labs/auth-provider/temp** file is updated to reflect the additional user and password.

```
[student@workstation ~]$ htpasswd -b ~/DO280/labs/auth-provider/temp \
>    developer ${RHT_OCP4_USER_PASSWD}
Adding password for user developer
```

1.4. Review the contents of your **~/DO280/labs/auth-provider/temp** and verify that it includes two entries with hashed passwords: one for the **admin** user and another for the **developer** user.

```
[student@workstation ~]$ cat ~/DO280/labs/auth-provider/temp
admin:$2y$05$QPuzHdl06IDkJssT.tdkZuSmgjUHV1XeYU4FjxhQrFqKL7hs2ZUl6
developer:$apr1$0Nzmc1rh$yGtne1k.JX6L5s5wNa2ye.
```

2. Log in to OpenShift and create a secret that contains the HTPasswd users file.

2.1. Log in to the cluster as the **kubeadmin** user.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

2.2. Create a secret from the **/home/student/DO280/labs/auth-provider/temp** file. To use the HTPasswd identity provider, you must define a secret with a key named **htpasswd** that contains the HTPasswd user file **/home/student/DO280/labs/auth-provider/temp**.

```
[student@workstation ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/home/student/DO280/labs/auth-provider/temp \
>    -n openshift-config
secret/htp-secret created
```

2.3. Assign the **admin** user the **cluster-admin** role.

> **Note**
> The output that indicates that the **admin** user is not found can be safely ignored.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
>    cluster-admin admin
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "admin"
```

3. Update the HTPasswd identity provider for the cluster so that your users can authenticate. Configure the custom resource file and update the cluster.

3.1. Export the existing OAuth resource to a file named **oauth.yaml** in the **~/DO280/labs/auth-provider** directory.

```
[student@workstation ~]$ oc get -o yaml oauth cluster > ~/DO280/labs/auth-provider/oauth.yaml
```

> **Note**
> A **oauth.yaml** file was downloaded for your convenience to **~/DO280/solutions/auth-provider**, which contains the completed custom resource file.

3.2. Edit the **~/DO280/labs/auth-provider/oauth.yaml** file with your preferred text editor. You can choose the names of the **identityProviders** and **fileData** structures. For this exercise, use the **myusers** and **htp-secret** values respectively.

The completed custom resource should match the following. Note that **htpasswd**, **mappingMethod**, **name** and **type** are at the same indentation level.

```
apiVersion: config.openshift.io/v1
kind: OAuth
...output omitted...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: htp-secret
    mappingMethod: claim
    name: myusers
    type: HTPasswd
```

3.3. Apply the custom resource defined in the previous step.

```
[student@workstation ~]$ oc replace -f ~/DO280/labs/auth-provider/oauth.yaml
oauth.config.openshift.io/cluster replaced
```

4. Log in as **admin** and as **developer** to verify the HTPasswd user configuration.

4.1. Log in to the cluster as the **admin** user to verify the htpasswd authentication is configured correctly. If log in fails, wait a few seconds and try again. The authentication operator takes some time to load the configuration changes from the previous step.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

4.2. Use the **oc get nodes** command to verify that the **admin** user has the **cluster-admin** role. The names of the nodes from your cluster might be different.

```
[student@workstation ~]$ oc get nodes
NAME                                        STATUS   ROLES    ...
ip-10-0-134-122.us-west-1.compute.internal  Ready    master   ...
ip-10-0-140-84.us-west-1.compute.internal   Ready    worker   ...
ip-10-0-152-96.us-west-2.compute.internal   Ready    worker   ...
...output omitted...
```

4.3. Log in to the cluster as the **developer** user to verify the htpasswd authentication is configured correctly.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

4.4. Use **oc get nodes** command to verify that the **developer** and **admin** do not share the same level of access.

```
[student@workstation ~]$ oc get nodes
Error from server (Forbidden): nodes is forbidden: User "developer" cannot list
resource "nodes" in API group "" at the cluster scope
```

4.5. Log in as the **admin** user and list the current users.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc get users
NAME        UID                                    FULL NAME   IDENTITIES
admin       0de1d76a-0198-11ea-86f1-0a580a810062               myusers:admin
developer   18b2517a-0198-11ea-b781-0a580a80005b               myusers:developer
```

4.6. Display the list of current identities.

```
[student@workstation ~]$ oc get identity
NAME                IDP NAME   IDP USER NAME   USER NAME
myusers:admin       myusers    admin           admin       ...
myusers:developer   myusers    developer       developer   ...
```

5. As the **admin** user, create a new HTPasswd user named **manager** with the **${RHT_OCP4_USER_PASSWD}** password.

5.1. Extract the file data from the secret to the **~/DO280/labs/auth-provider/temp** file.

```
[student@workstation ~]$ oc extract secret/htp-secret -n openshift-config \
>     --to - > ~/DO280/labs/auth-provider/temp
# htpasswd
```

5.2. Add an entry your **~/DO280/labs/auth-provider/temp** file for the additional user **manager** with the password **${RHT_OCP4_USER_PASSWD}**.

```
[student@workstation ~]$ htpasswd -b ~/DO280/labs/auth-provider/temp manager
 ${RHT_OCP4_USER_PASSWD}
Adding password for user manager
```

5.3. Review the contents of your **~/DO280/labs/auth-provider/temp** file and verify that it includes three entries with hashed passwords: one each for the **admin**, **developer** and **manager** users.

```
[student@workstation ~]$ cat ~/DO280/labs/auth-provider/temp
admin:$2y$05$QPuzHdl06IDkJssT.tdkZuSmgjUHV1XeYU4FjxhQrFqKL7hs2ZUl6
developer:$apr1$0Nzmc1rh$yGtne1k.JX6L5s5wNa2ye.
manager:$apr1$CJ/tpa6a$sLhjPkIIAy755ZArTT5EH/
```

5.4. The secret must be updated after adding additional users. The **oc create secret** command takes the **/home/student/DO280/labs/auth-provider/temp** file as input, and pipes the output as stdin (standard input) to the **oc replace** command to update the secret. If you receive a failure, then re-run the command again after a few moments as the oauth operator might still be reloading.

```
[student@workstation ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/home/student/DO280/labs/auth-provider/temp \
>    --dry-run -o yaml | oc replace -n openshift-config -f -
secret/htp-secret replaced
```

5.5. Wait a few moments for the authentication operator to reload, and then log in to the cluster as the **manager** user.

> **Note**
> If the authentication fails, wait a few moments and try again.

```
[student@workstation ~]$ oc login -u manager -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

6. Create a new project named **auth-provider**, and then verify that the **developer** user cannot access the project.

6.1. As the **manager** user, create a new **auth-provider** project.

```
[student@workstation ~]$ oc new-project auth-provider
Now using project "auth-provider" on server https://
api.cluster.domain.example.com:6443".
...output omitted...
```

6.2. Log in as the **developer** user, and then attempt to delete the **auth-provider** project.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc delete project auth-provider
Error from server (Forbidden): projects.project.openshift.io "auth-provider"
is forbidden: User "developer" cannot delete resource "projects"
in API group "project.openshift.io" in the namespace "auth-provider"
```

7. Change the password for the **manager** user.

7.1. Log in as **admin** and extract the file data from the secret to the **~/DO280/labs/auth-provider/temp** file.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc extract secret/htp-secret -n openshift-config \
>    --to - > ~/DO280/labs/auth-provider/temp
# htpasswd
```

7.2. Generate a random user password. Your password will differ from what is displayed in the following example.

```
[student@workstation ~]$ openssl rand -hex 15
ce135e2c7a89014113ed3f62a86240
```

7.3. Store the password created in the previous step in the **${MANAGER_PASSWD}** variable. Copy only the random hex string from the output of the previous step. If you include the dash and the spaces, then the log in will fail.

```
[student@workstation ~]$ MANAGER_PASSWD='ce135e2c7a89014113ed3f62a86240'
```

7.4. Update the **manager** user with the password **${MANAGER_PASSWD}**.

```
[student@workstation ~]$ htpasswd -b ~/DO280/labs/auth-provider/temp manager
 ${MANAGER_PASSWD}
Updating password for user manager
```

7.5. Update the secret.

```
[student@workstation ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/home/student/DO280/labs/auth-provider/temp \
>    --dry-run -o yaml | oc replace -n openshift-config -f -
secret/htp-secret replaced
```

7.6. Log in as the **manager** user to verify the updated password.

```
[student@workstation ~]$ oc login -u manager -p ${MANAGER_PASSWD}
Login successful.
...output omitted...
```

8. Remove the **manager** user.

8.1. Log in as **admin** and extract the file data from the secret to the **~/DO280/labs/auth-provider/temp** file.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc extract secret/htp-secret -n openshift-config \
>    --to - > ~/DO280/labs/auth-provider/temp
# htpasswd
```

8.2. Delete the **manager** user from the **~/DO280/labs/auth-provider/temp** file.

```
[student@workstation ~]$ htpasswd -D ~/DO280/labs/auth-provider/temp manager
Deleting password for user manager
```

8.3. Update the secret.

```
[student@workstation ~]$ oc create secret generic htp-secret \
>    --from-file htpasswd=/home/student/DO280/labs/auth-provider/temp \
>    --dry-run -o yaml | oc replace -n openshift-config -f -
secret/htp-secret replaced
```

8.4. Delete the identity resource for the **manager** user.

```
[student@workstation ~]$ oc delete identity "myusers:manager"
identity.user.openshift.io "myusers:manager" deleted
```

8.5. Delete the user resource for the **manager** user.

```
[student@workstation ~]$ oc delete user manager
user.user.openshift.io manager deleted
```

8.6. Now, if you attempt to log in as the **manager** user, it will fail.

```
[student@workstation ~]$ oc login -u manager -p ${MANAGER_PASSWD}
Login failed (401 Unauthorized)
...output omitted...
```

8.7. List the current users to verify that the **manager** user is deleted.

```
[student@workstation ~]$ oc get users
NAME        UID                                    FULL NAME   IDENTITIES
admin       39e6f2ae-01a0-11ea-86f1-0a580a810062               myusers:admin
developer   404da1f8-01a0-11ea-b781-0a580a80005b               myusers:developer
```

8.8. Display the list of current identities to verify that the **manager** identity is deleted.

```
[student@workstation ~]$ oc get identity
NAME                 IDP NAME   IDP USER NAME   USER NAME
myusers:admin        myusers    admin           admin       ...
myusers:developer    myusers    developer       developer   ...
```

8.9. Extract the secret and verify that only the users **admin** and **developer** are displayed.

```
[student@workstation ~]$ oc extract secret/htp-secret -n openshift-config --to -
# htpasswd
admin:$apr1$n6.LnQwS$Llq47A305ssf6tN.ySXzP.
developer:$apr1$EZibs7bc$WkEZoTAN9G4Fv1Cp3SVYM0
```

9. Remove the identity provider and clean up all users.

9.1. Log in as the **kubeadmin** user.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD}
Login successful.
...output omitted...
```

9.2. Delete the **auth-provider** project.

```
[student@workstation ~]$ oc delete project auth-provider
project.project.openshift.io "auth-provider" deleted
```

9.3. Edit the resource in place to remove the identity provider from OAauth:

```
[student@workstation ~]$ oc edit oauth
```

Delete all the lines under **spec:**, and then append **{}** after **spec:**. Leave all the other information in the file. Your **spec:** line should match the following:

```
...output omitted...
spec: {}
```

Save your changes, and then verify that the **oc edit** command applied you changes:

```
oauth.config.openshift.io/cluster edited
```

9.4. Delete the secret **htp-secret**.

```
[student@workstation ~]$ oc delete secret htp-secret -n openshift-config
secret "htp-secret" deleted
```

9.5. Delete all user and identity resources.

```
[student@workstation ~]$ oc delete user --all
user.user.openshift.io "admin" deleted
user.user.openshift.io "developer" deleted
[student@workstation ~]$ oc delete identity --all
identity.user.openshift.io "myusers:admin" deleted
identity.user.openshift.io "myusers:developer" deleted
```

Finish

On the **`workstation`** machine, use the **`lab`** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab auth-provider finish
```

This concludes the guided exercise.

Summary

In this chapter, you learned:

- Creating OpenShift users requires valid credentials, managed by an identity provider, plus user and identity resources.
- Deleting OpenShift users requires deleting their credentials from the identity provider, and also deleting their user and identity resources.
- A just-installed OpenShift cluster provides two authentication methods that grant administrative access: the **`kubeconfig`** file and the **`kubeadmin`** virtual user.
- The HTPasswd identity provider authenticates users against credentials stored in a secret. The name of the secret, and other settings for the identity provider, are stored inside the OAuth custom resource.
- To manage user credentials using the HTPasswd identity provider, you must extract data from the secret, change that data using the **`htpasswd`** command, and then apply the data back to the secret.
- To grant a user cluster administrator privileges, assign the **`cluster-admin`** role to that user.

Chapter 4

Controlling Access to OpenShift Resources

Goal	Define and apply role-based access controls and protect sensitive information with secrets.
Objectives	• Define role-based access controls and apply permissions to users. • Create and apply secrets to manage sensitive information. • Create service accounts and apply permissions to pods.
Sections	• Defining and Applying Permissions using RBAC (and Guided Exercise) • Managing Sensitive Information with Secrets (and Guided Exercise) • Controlling Application Permissions with Security Context Constraints (and Guided Exercise)
Lab	Chapter Review

Defining and Applying Permissions Using RBAC

Objectives

After completing this section, you should be able to define role-based access controls and apply permissions to users.

Role-based Access Control (RBAC)

Role-based access control (RBAC) is a technique for managing access to resources in a computer system. In Red Hat OpenShift, RBAC determines if a user can perform certain actions within the cluster or project. There are two types of roles that can be used depending on the level of responsibility of the user: cluster and local.

> **Note**
> Authorization is a separate step from authentication.

Authorization Process

The authorization process is managed by rules, roles, and bindings.

RBAC Object	Description
Rule	Allowed actions for objects or groups of objects
Role	Sets of rules. Users and groups can be associated with multiple roles.
Binding	Assignation of users or groups to a role.

RBAC Scope

Red Hat OpenShift Container Platform (RHOCP) defines two groups of roles and bindings depending on the scope and responsibility of users: cluster roles and local roles.

Role Level	Description
`Cluster Role`	Users or groups with this role level can manage the OpenShift cluster.
`Local Role`	Users or groups with this role level can only manage elements at a project level.

> **Note**
> Cluster role bindings take precedence over local role bindings.

Managing RBAC Using the CLI

To add a cluster role to a user, use the **`add-cluster-role-to-user`** subcommand:

```
[user@demo ~]$ 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:

```
[user@demo ~]$ oc adm policy add-cluster-role-to-user cluster-admin username
```

To remove a cluster role from a user, use the **remove-cluster-role-from-user** subcommand:

```
[user@demo ~]$ 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:

```
[user@demo ~]$ oc adm policy remove-cluster-role-from-user cluster-admin username
```

Rules are defined by an action and a resource. For example, the **create user** rule is part of the **cluster-admin** role.

You can use the **oc adm policy who-can** command to determine if a user can execute an action on a resource. For example:

```
[user@demo ~]$ oc adm policy who-can delete user
```

Default Roles

OpenShift ships with a set of default cluster roles that can be assigned locally or to the entire cluster. You can modify these roles for fine-grained access control to OpenShift resources, but additional steps are required that are outside the scope of this course.

Default roles	Description
admin	Users with this role can manage all project resources, including granting access to other users to the project.
basic-user	Users with this role have read access to the project.
cluster-admin	Users with this role have superuser access to the cluster resources. These users can perform any action on the cluster, and have full control of all projects.
cluster-status	Users with this role can get cluster status information.
edit	Users with this role can create, change, and delete common application resources from the project, such as services and deployment configurations. These users cannot act on management resources such as limit ranges and quotas, and cannot manage access permissions to the project.
self-provisioner	Users with this role can create new projects. This is a cluster role, not a project role.

Default roles	Description
`view`	Users with this role can view project resources, but cannot modify project resources.

The **admin** role gives a user access to project resources such as quotas and limit ranges, and also the ability to 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.

Add a specified role to a user with the **add-role-to-user** subcommand. For example:

```
[user@demo ~]$ 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:

```
[user@demo ~]$ 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 via **rolebindings**.

Regular users
: This is the way most interactive OpenShift Container Platform users are represented. Regular users are represented with the **User** object. This type of user represents a person that has been allowed access to the platform. 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 often used to give extra privileges to pods or deployment configurations. 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.

References

For more information about Kubernetes namespaces refer to the
Kubernetes Documentation
https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

References

For more information about RBAC, refer to the *Using RBAC to define and apply permissions* chapter in the *OpenShift Container Platform 4.2 Official Documentation* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/authentication/index#using-rbac

For more information about groups, refer to the *Understanding authentication* chapter in the *OpenShift Container Platform 4.2 Official Documentation* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/authentication/index#understanding-authentication

Guided Exercise

Defining and Applying Permissions using RBAC

In this exercise, you will define role-based access controls and apply permissions to users.

Outcomes

You should be able to:

- Remove from all Red Hat OpenShift users the privilege to create new projects.
- Create OpenShift groups and add members to these groups.
- Create a project and assign project administration privileges over that project to a non-cluster administrator user.
- As a project administrator, assign read and write privileges to different groups of users.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**`/usr/local/bin/oc`**)
- The Apache HTTP Server container image from OpenShift.

As the **`student`** user on the **`workstation`** machine, use the **`lab`** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates some HTPasswd users for the exercise.

```
[student@workstation ~]$ lab authorization-rbac start
```

1. Log in to the OpenShift cluster and determine which cluster role bindings assign the **`self-provisioner`** cluster role.

    1.1. Source the classroom configuration file at **`/usr/local/etc/ocp4.config`**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

    1.2. Log in to the cluster as the **`admin`** user you created in Guided Exercise: Configuring Identity Providers

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.3. List all cluster role bindings that reference the **self-provisioner** cluster role.

```
[student@workstation ~]$ oc get clusterrolebinding -o wide \
>    | grep -E 'NAME|self-provisioner'
NAME                          ROLE                            ...
self-provisioners    ...      ClusterRole/self-provisioner    ...
```

2. Remove the privilege to create new projects from all users by deleting the **self-provisioner** cluster role from the **system:authenticated:oauth** virtual group.

2.1. Confirm that the **self-provisioners** cluster role bindings that you found in the previous step assign the **self-provisioner** cluster role to the **system:authenticated:oauth** group.

```
[student@workstation ~]$ oc describe clusterrolebindings self-provisioners
Name:         self-provisioners
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  self-provisioner
Subjects:
  Kind   Name                         Namespace
  ----   ----                         ---------
  Group  system:authenticated:oauth
```

2.2. Remove the **self-provisioner** cluster role from to the **system:authenticated:oauth** virtual group, which deletes the **self-provisioners** role binding. You can safely ignore the warning about your changes being lost.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group  \
>    self-provisioner system:authenticated:oauth
Warning: Your changes may get lost whenever a master is restarted,
 unless you prevent reconciliation of this rolebinding using the
 following command: oc annotate clusterrolebinding.rbac self-provisioner
 'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite
clusterrole.rbac.authorization.k8s.io/self-provisioner removed:
 "system:authenticated:oauth"
```

2.3. Verify that the role has been removed from the group. The cluster role binding **self-provisioners** should not exist.

```
[student@workstation ~]$ oc describe clusterrolebindings self-provisioners
Error from server (NotFound): clusterrolebindings.rbac.authorization.k8s.io "self-
provisioners" not found
```

2.4. Determine if any other cluster role bindings reference the **self-provisioner** cluster role.

```
[student@workstation ~]$ oc get clusterrolebinding -o wide \
>     | grep -E 'NAME|self-provisioner'
NAME                          ROLE    ...
```

2.5. Log in as the **leader** user, using the password from the **RHT_OCP4_USER_PASSWD** variable, and try to create a project. It should fail.

```
[student@workstation ~]$ oc login -u leader -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project test
Error from server (Forbidden): You may not request a new project via this API.
```

3. Create a project and add project administration privileges to the **leader** user.

3.1. Log in as the **admin** user and create the **authorization-rbac** project.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project authorization-rbac
Now using project "authorization-rbac" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

3.2. Grant project administration privileges to the **leader** user on the **authorization-rbac** project.

```
[student@workstation ~]$ oc policy add-role-to-user admin leader
clusterrole.rbac.authorization.k8s.io/admin added: "leader"
```

4. Create the **dev-group** and **qa-group** groups and add their respective members.

4.1. Create a group called **dev-group**.

```
[student@workstation ~]$ oc adm groups new dev-group
group.user.openshift.io/dev-group created
```

4.2. Add the **developer** user to **dev-group**.

```
[student@workstation ~]$ oc adm groups add-users dev-group developer
group.user.openshift.io/dev-group added: "developer"
```

4.3. Create a second group called **qa-group**.

```
[student@workstation ~]$ oc adm groups new qa-group
group.user.openshift.io/qa-group created
```

4.4. Add the **qa-engineer** user to **qa-group**.

```
[student@workstation ~]$ oc adm groups add-users qa-group qa-engineer
group.user.openshift.io/qa-group added: "qa-engineer"
```

4.5. Review all existing OpenShift groups to verify that they have the correct members.

```
[student@workstation ~]$ oc get groups
NAME        USERS
dev-group   developer
qa-group    qa-engineer
```

5. As the **leader** user, assign write privileges for **dev-group** and read privileges for **qa-group** to the **authorization-rbac** project.

5.1. Log in as the **leader** user.

```
[student@workstation ~]$ oc login -u leader -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
Using project "authorization-rbac".
```

5.2. Add write privileges to **dev-group** on the **authorization-rbac** project.

```
[student@workstation ~]$ oc policy add-role-to-group edit dev-group
clusterrole.rbac.authorization.k8s.io/edit added: "dev-group"
```

5.3. Add read privileges to **qa-group** on the **authorization-rbac** project.

```
[student@workstation ~]$ oc policy add-role-to-group view qa-group
clusterrole.rbac.authorization.k8s.io/view added: "qa-group"
```

5.4. Review all role bindings on the **authorization-rbac** project to verify that they assign roles to the correct groups and users. The following output omits default role bindings assigned by OpenShift to service accounts.

```
[student@workstation ~]$ oc get rolebindings -o wide
NAME      AGE   ROLE                USERS    GROUPS      ...
admin     58s   ClusterRole/admin   admin
admin-0   51s   ClusterRole/admin   leader
edit      12s   ClusterRole/edit             dev-group
...output omitted...
view      8s    ClusterRole/view             qa-group
```

6. As the **developer** user, deploy an Apache HTTP Server to prove that it has write privileges in the project. Also try to grant write privileges to the **qa-engineer** user to prove that the **developer** user has no project administration privileges.

6.1. Log in as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
Using project "authorization-rbac".
```

6.2. Deploy an Apache HTTP Server using the standard image stream from OpenShift.

```
[student@workstation ~]$ oc new-app --name httpd httpd:2.4
...output omitted...
--> Creating resources ...
imagestreamtag.image.openshift.io "httpd:2.4" created
deploymentconfig.apps.openshift.io "httpd" created
service "httpd" created
--> Success
...output omitted...
```

6.3. Try to grant write privileges to the **qa-engineer** user. It should fail.

```
[student@workstation ~]$ oc policy add-role-to-user edit qa-engineer
Error from server (Forbidden): rolebindings.rbac.authorization.k8s.io is
 forbidden: User "developer" cannot list resource "rolebindings" in API group
 "rbac.authorization.k8s.io" in the namespace "authorization-rbac"
```

7. Verify that the **qa-engineer** user only has read privileges on the **httpd** application.

7.1. Log in as the **qa-engineer** user.

```
[student@workstation ~]$ oc login -u qa-engineer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
Using project "authorization-rbac".
```

7.2. Attempt to scale the **httpd** application. It should fail.

```
[student@workstation ~]$ oc scale dc httpd --replicas 3
Error from server (Forbidden): deploymentconfigs.apps.openshift.io "httpd" is
 forbidden: User "qa-engineer" cannot update resource "deploymentconfigs/scale" in
 API group "apps.openshift.io" in the namespace "authorization-rbac"
```

8. Restore project creation privileges to all users.

8.1. Log in as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

8.2. Restore project creation privileges for all users by recreating the **self-provisioners** cluster role binding created by the OpenShift installer. You can safely ignore the warning that the group was not found.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
>     --rolebinding-name self-provisioners \
>     self-provisioner system:authenticated:oauth
Warning: Group 'system:authenticated:oauth' not found
clusterrole.rbac.authorization.k8s.io/self-provisioner added:
 "system:authenticated:oauth"
```

Finish

On the **workstation** machine, run the **lab** command to complete this exercise.

```
[student@workstation ~]$ lab authorization-rbac finish
```

This concludes the section.

Managing Sensitive Information with Secrets

Objectives

After completing this section, you should be able to:

- Create and apply secrets to manage sensitive information.
- Share secrets between applications.

Secrets Overview

Modern applications are designed to loosely couple code, configuration, and data. Configuration files and data are not hard-coded as part of the software. Instead, the software loads configuration and data from an external source. This enables application deployment to different environments without requiring a change to the application source code.

Often applications require access to sensitive information. As an example, a back-end web application requires access to database credentials to perform a database query.

Kubernetes and OpenShift uses secret resources to hold sensitive information, such as:

- Passwords.
- Sensitive configuration files.
- Credentials to an external resource, such as an SSH key or OAuth token.

A secret can store any type of data. Data in a secret is Base64-encoded, so it is not stored in plain text. Secret data is not encrypted; you can decode the secret from Base64 format to access the original data.

Although secrets can store any type of data, Kubernetes and OpenShift support different types of secrets. Different types of secret resources exist, including service account tokens, SSH keys, and TLS certificates. When you store information in a specific secret resource type, Kubernetes validates that the data conforms to the type of secret.

Features of Secrets

The main features of secrets include:

- Secret data can be shared within a project namespace.
- Secret data is referenced independently of secret definition. Administrators can create and manage a secret resource, and other team members reference the secret in their deployment configurations.
- Secret data is injected into pods when OpenShift creates a pod. You can expose a secret as an environment variable or as a mounted file in the pod.
- If the value of a secret changes during pod execution, the secret data does not update in the pod. After a secret value changes, you must create new pods to inject the new secret data.
- Any secret data that OpenShift injects into a pod is ephemeral. If OpenShift exposes sensitive data to a pod as environment variables, those variables are destroyed when the pod is destroyed.

If a secret is mounted as a file in the pod, the file is also destroyed when the pod is destroyed. Secret data volumes are backed by temporary file storage.

Use Cases for Secrets

Two primary use cases for secrets are storing credentials and securing communication between services. These are discussed below.

Credentials

Store sensitive information, such as passwords and user names, in a secret.

If an application expects to read sensitive information from a file, then you mount the secret as a data volume to the pod. The application can read the secret as an ordinary file to access the sensitive information. Some databases, for example, read credentials from a file to authenticate users.

Some applications use environment variables to read configuration and sensitive data. You can link secret variables to pod environment variables in a deployment configuration.

Transport Layer Security (TLS) and Key Pairs

You can secure communication to a service by having the cluster generate a signed certificate and key pair into a secret within the project 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.

Creating a Secret

If a pod requires access to sensitive information, then create a secret for the information before you deploy the pod.

To create a secret:

- 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 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.

Exposing Secrets to Pods

To expose a secret to a pod, first create the secret. Assign each piece of sensitive data to a key. After creation, the secret contains key-value pairs.

The following command creates a generic secret named **demo-secret** with two keys: **username** with the **demo-user** value and **root_password** with the **=zT1KTgk** value. The secret contains a single key-value pair.

```
[user@demo ~]$ oc create secret generic demo-secret \
>    --from-literal username=demo-user
>    --from-literal root_password=zT1KTgk
```

Secrets as Pod Environment Variables

Consider a database application that reads the database administrator password from the **MySQL_ROOT_PASSWORD** environment variable. Modify the environment variables section of the deployment configuration to use values from the secret:

```
env:
  - name: MYSQL_ROOT_PASSWORD❶
    valueFrom:
      secretKeyRef:
        name: demo-secret❷
        key: root_password❸
```

❶ The environment variable name in the pod, which contains data from a secret.
❷ The name of the secret that contains the desired sensitive information.
❸ The key contains the sensitive information in the secret.

You can also use the **oc set env** command to set application environment variables from secret values:

```
[user@demo ~]$ oc set env dc/demo --from=secret/demo-secret
```

The previous command creates an environment variable for each key in the secret. If a secret key is lowercase, the corresponding environment variable is converted to uppercase.

Secrets as Files in a Pod

The following command creates a volume mount in the pod that contains the contents of a secret:

```
[user@demo ~]$ oc set volume dc/demo❶ \
>    --add❷ \
>    --type=secret❸ \
>    --secret-name=demo-secret❹ \
>    --mount-path=/app-secrets❺
```

❶ Modify the volume configuration in the **demo** deployment configuration.
❷❸ Add a new volume from a secret.
❹ Use the **demo-secret** secret.
❺ Make the secret data available in the **/app-secrets** directory in the pod.

References

For more information on secrets, refer to the OpenShift Container Platform Documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html/nodes/working-with-pods#nodes-pods-secrets-about_nodes-pods-secrets

Guided Exercise

Managing Sensitive Information With Secrets

In this exercise, you will understand how to manage information using secrets

Outcomes

You should be able to:

- Manage secrets and use them to initialize environment variables in applications.
- Use secrets for a MySQL database application.
- Assign secrets to an application that connects to a MySQL database.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**/usr/local/bin/oc**).
- The **rhscl/mysql-57-rhel7:5.7-47** container image from Red Hat.
- The **redhattraining/famous-quotes:1.0** image from Quay.io.

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and downloads the resource files necessary for this exercise.

```
[student@workstation ~]$ lab authorization-secrets start
```

1. Log in to the OpenShift cluster and create the **authorization-secrets** project.

   1.1. Source the classroom configuration file at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>   ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.3. Create the **authorization-secrets** project.

```
[student@workstation ~]$ oc new-project authorization-secrets
Now using project "authorization-secrets" on server "https:///
api.cluster.domain.example.com:6443".
...output omitted...
```

2. Create a secret with the credentials and connection information to access a MySQL database.

```
[student@workstation ~]$ oc create secret generic mysql \
>   --from-literal user=myuser --from-literal password=redhat123 \
>   --from-literal database=test_secrets --from-literal hostname=mysql
secret/mysql created
```

3. Deploy a database and add the secret for user and database configuration.

3.1. Try to deploy an ephemeral database server. This should fail because the MySQL image needs environment variables for its initial configuration. The values for these variables can not be assigned using secrets because the **oc new-app** command does not support the use of this type of secrets.

```
[student@workstation ~]$ oc new-app --name mysql \
>   --docker-image registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47
--> Creating resources ...
    imagestream.image.openshift.io "mysql" created
    deploymentconfig.apps.openshift.io "mysql" created
    service "mysql" created
--> Success
...output omitted...
```

3.2. Run the **oc get pods** command with the **-w** option to retrieve the status of the deployment in real time. Notice how the database pod is in a failed state. Type **Ctrl+C** to exit the command.

```
[student@workstation ~]$ oc get pods -w
NAME             READY   STATUS              RESTARTS   AGE
mysql-1-deploy   1/1     Running             0          15s
mysql-1-8mjmm    0/1     ContainerCreating   0          7s
NAME             READY   STATUS              RESTARTS   AGE
mysql-1-8mjmm    0/1     ContainerCreating   0          8s
mysql-1-8mjmm    0/1     Error               0          28s
mysql-1-8mjmm    0/1     Error               1          29s
mysql-1-8mjmm    0/1     CrashLoopBackOff    1          30s
...output omitted...
```

Note
It might take a while for the pod to reach the error state.

3.3. Use the **mysql** secret to initialize environment variables on the **mysql** deployment configuration. The deployment needs the **MYSQL_USER**, **MYSQL_PASSWORD**, and

MYSQL_DATABASE environment variables for a successful initialization. The secret has the **user**, **password**, and **database** keys that can be assigned to the deployment configuration as environment variables, adding the prefix **MYSQL_**.

Setting the **mysql** secret as an environment variable to the deployment configuration triggers a new build of the application.

```
[student@workstation ~]$ oc set env dc/mysql --prefix MYSQL_ \
>   --from secret/mysql
deploymentconfig.apps.openshift.io/mysql updated
```

3.4. Verify that the **mysql** application was deployed successfully after adding the secret.

```
[student@workstation ~]$ oc get pods
NAME              READY   STATUS      RESTARTS   AGE
mysql-2-deploy    0/1     Completed   0          33s
mysql-2-rqp77     1/1     Running     0          24s
```

Take note of the pod name in the **Running** state; you will need it in further steps.

4. Verify that the database now authenticates using the environment variables initialized from the **mysql** secret.

4.1. Open a remote shell session to the **mysql** pod in the **Running** state.

```
[student@workstation ~]$ oc rsh mysql-2-rqp77
sh-4.2$
```

4.2. Start a MySQL session to verify that the environment variables initialized by the **mysql** secret were used to configure the **mysql** application.

```
sh-4.2$ mysql -u myuser --password=redhat123 test_secrets -e 'show databases;'
mysql: [Warning] Using a password on the command line interface can be insecure.
+--------------------+
| Database           |
+--------------------+
| information_schema |
| test_secrets       |
+--------------------+
```

4.3. Close the remote shell session to continue from your **workstation**.

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

5. Create a new application that uses the MySQL database.

5.1. Create a new application using the **redhattraining/famous-quotes** image from Quay.io.

```
[student@workstation ~]$ oc new-app --name quotes \
>   --docker-image quay.io/redhattraining/famous-quotes:1.0
--> Found container image c511b06 (19 hours old) from quay.io for "quay.io/
redhattraining/famous-quotes:1.0"
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "quotes" created
    deploymentconfig.apps.openshift.io "quotes" created
    service "quotes" created
--> Success
...output omitted...
```

5.2. Verify the **quotes** application pod status. This should display an error because you have not added the environment variables it needs to connect to the database. This might take a while to display in the output.

```
[student@workstation ~]$ oc get pods -l app=quotes -w
NAME             READY   STATUS              RESTARTS   AGE
quotes-1-8cnjb   0/1     ContainerCreating   0          1s
NAME             READY   STATUS              RESTARTS   AGE
quotes-1-8cnjb   0/1     ContainerCreating   0          8s
quotes-1-8cnjb   0/1     Error               0          35s
...output omitted...
```

6. Use the **mysql** secret to initialize the environment variables that the **quotes** application needs to connect to the database.

The **quotes** application requires the following environment variables: **QUOTES_USER**, **QUOTES_PASSWORD**, **QUOTES_DATABASE**, and **QUOTES_HOSTNAME**, which correspond to the **user**, **password**, **database**, and **hostname** keys of the **mysql** secret. As in the MySQL application case, the secret can be added to the **quotes** application to initialize its environment variables with the prefix **QUOTES_**.

6.1. Use the **mysql** secret to initialize environment variables in the application so it can connect to the **mysql** database.

```
[student@workstation ~]$ oc set env dc/quotes --prefix QUOTES_ \
>   --from secret/mysql
deploymentconfig.apps.openshift.io/quotes updated
```

6.2. Wait until the **quotes** application is successfully redeployed. The older containers should be automatically terminated.

```
[student@workstation ~]$ oc get pods -l app=quotes
NAME             READY   STATUS    RESTARTS   AGE
quotes-2-6zn7f   1/1     Running   0          9m35s
```

> **Note**
> It might take a while for the pod to finish the deployment.

7. Verify that the **quotes** application environment variables were successfully set and that the application is working properly.

   7.1. Verify that the application started correctly.

```
[student@workstation ~]$ oc logs pod/quotes-2-6zn7f
... Connecting to the database: myuser:redhat123@tcp(...:3306)/test_secrets
... Database connection OK
...output omitted...
```

   7.2. Expose the application service so that it can be accessed from outside the cluster.

```
[student@workstation ~]$ oc expose service quotes
route.route.openshift.io/quotes exposed
```

   7.3. Verify the application host name.

```
[student@workstation ~]$ oc get route quotes
NAME      HOST/PORT                                                         ...
quotes    quotes-authorization-secrets.apps.cluster.domain.example.com      ...
```

   7.4. Verify that the variables are being properly set in the application by accessing the **env** REST entry point.

```
[student@workstation ~]$ curl \
>   http://quotes-authorization-secrets.apps.cluster.domain.example.com/env
...output omitted...
                <li>QUOTES_USER: test_user </li>
                <li>QUOTES_PASSWORD: redhat123 </li>
                <li>QUOTES_DATABASE: test_secrets</li>
                <li>QUOTES_HOST: mysql</li>
...output omitted...
```

   7.5. Access the application **status** REST API entry point to test the database connection.

```
[student@workstation ~]$ curl \
>   http://quotes-authorization-secrets.apps.cluster.domain.example.com/status
Database connection OK
```

   7.6. Test application functionality by accessing the **random** REST API entry point.

```
[student@workstation ~]$ curl \
>   http://quotes-authorization-secrets.apps.cluster.domain.example.com/random
8: Those who can imagine anything, can create the impossible.
- Alan Turing
```

8. Remove the **authorization-secrets** project.

```
[student@workstation ~]$ oc delete project authorization-secrets
project.project.openshift.io "authorization-secrets" deleted
```

Finish

On **workstation**, run the **lab** command to complete this exercise.

```
[student@workstation ~]$ lab authorization-secrets finish
```

This concludes the section.

Controlling Application Permissions with Security Context Constraints (SCCs)

Objectives

After completing this section, you should be able to:

- Manage security context constraints.
- Create service accounts and apply permissions to pods.

Security Context Constraints (SCCs)

Red Hat OpenShift provides *security context constraints (SCCs)*, a security mechanism that restricts access to resources, but not to operations in OpenShift.

SCCs limits the access from a running pod in OpenShift to the host environment. SCCs control:

- 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 might require relaxed security context constraints to access resources that are forbidden by default, such as file systems, sockets or to access a SELinux context.

You can run the following command as a cluster administrator to list the SCCs defined by OpenShift:

```
[user@demo ~]$ oc get scc
```

OpenShift provides eight SCCs:

- **anyuid**
- **hostaccess**
- **hostmount-anyuid**
- **hostnetwork**
- **node-exporter**
- **nonroot**
- **privileged**
- **restricted**

To get additional information about an SCC, use the **oc describe** command.

```
[user@demo ~]$ 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`** SCC, 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:

```
[user@demo ~]$ oc create serviceaccount service-account-name
```

To associate the service account with an SCC, use the following command:

```
[user@demo ~]$ 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. This returns all the security context constraints that can be used to overcome the limitations of a container. For example:

```
[user@demo ~]$ oc get pod podname  -o yaml | \
>    oc adm policy scc-subject-review -f -
```

Privileged Containers

Some containers might need to access the runtime environment of the host. 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.

References

For more information, refer to the *Managing Security Context Constraints* chapter in the *OpenShift Container Platform 4.2 Official Documentation* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/authentication/index#managing-pod-security-policies

Guided Exercise

Controlling Application Permissions with Security Context Constraints (SCC)

In this exercise, you will deploy applications that require pods with extended permissions.

Outcomes

You should be able to:

- Create service accounts and assign security context constraints (SCCs) to them.
- Assign a service account to a deployment configuration.
- Run applications that need **root** privileges.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**/usr/local/bin/oc**)
- The **gitlab/gitlab-ce:8.4.3-ce.0** container image from Docker Hub.

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates some HTPasswd users for the exercise.

```
[student@workstation ~]$ lab authorization-scc start
```

1. Log in to the OpenShift cluster and create the **authorization-scc** project.

   1.1. Source the classroom configuration file at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. Create the **authorization-scc** project.

```
[student@workstation ~]$ oc new-project authorization-scc
Now using project "authorization-scc" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

2. Deploy a **gitlab/gitlab-ce:8.4.3-ce.0** application and verify that it fails because the container image needs **root** privileges.

 2.1. Deploy a **gitlab/gitlab-ce:8.4.3-ce.0** application.

```
[student@workstation ~]$ oc new-app --name gitlab gitlab/gitlab-ce:8.4.3-ce.0
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "gitlab" created
    deploymentconfig.apps.openshift.io "gitlab" created
    service "gitlab" created
--> Success
...output omitted...
```

 2.2. Determine if the application was successfully deployed. It should give an error because this image needs **root** privileges to properly deploy.

```
[student@workstation ~]$ oc get pods
NAME              READY    STATUS       RESTARTS    AGE
gitlab-1-deploy   0/1      Completed    0           89s
gitlab-1-ltjhs    0/1      Error        1           81s
```

Note

It might take some time for the image to reach the **Error** state. You might also see the **CrashLoopBackOff** status while checking the pod's health.

 2.3. Review the application logs to confirm that the failure is caused by insufficient privileges.

```
[student@workstation ~]$ oc logs pod/gitlab-1-ltjhs
...output omitted...
================================================================================
Recipe Compile Error in /opt/gitlab/embedded/cookbooks/\
cache/\cookbooks/gitlab/recipes/default.rb
================================================================================

Chef::Exceptions::InsufficientPermissions
-----------------------------------------
directory[/etc/gitlab] (gitlab::default line 26) had an error: \
Chef::Exceptions::InsufficientPermissions: Cannot create directory[/etc/gitlab] at
 \
/etc/gitlab due to insufficient permissions
...output omitted...
```

3. Create a new service account and add the **anyuid** SCC to it.

3.1. Create a service account named **gitlab-sa**.

```
[student@workstation ~]$ oc create sa gitlab-sa
serviceaccount/gitlab-sa created
```

3.2. Log in as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

3.3. Assign the **anyuid** SCC to the **gitlab-sa** service account.

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z gitlab-sa
securitycontextconstraints.security.openshift.io/anyuid added to:
 ["system:serviceaccount:authorization-scc:gitlab-sa"]
```

4. Modify the **gitlab** application so that it uses the newly created service account. Verify that the new deployment is successful.

4.1. Log in as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

4.2. Assign the **gitlab-sa** service account to the **gitlab** deployment.

```
[student@workstation ~]$ oc set serviceaccount deploymentconfig gitlab gitlab-sa
deploymentconfig.apps.openshift.io/gitlab serviceaccount updated
```

4.3. Verify that the **gitlab** redeployment was successful. You may need to run the **oc get pods** command multiple times until you see a running application pod.

```
[student@workstation ~]$ oc get pods
NAME              READY   STATUS      RESTARTS   AGE
gitlab-1-deploy   0/1     Completed   0          47m
gitlab-2-cjkvd    1/1     Running     0          119s
gitlab-2-deploy   0/1     Completed   0          2m8s
```

5. Verify that the **gitlab** application is properly working.

5.1. Expose the **gitlab** application.

```
[student@workstation ~]$ oc expose service gitlab --port 80
route.route.openshift.io/gitlab exposed
```

5.2. Get the exposed route.

```
[student@workstation ~]$ oc get route gitlab
NAME      HOST/PORT                                             PATH     ...
gitlab    gitlab-authorization-scc.apps.cluster.domain.example.com  gitlab   ...
```

5.3. Verify that the **gitlab** application is answering HTTP queries.

```
[student@workstation ~]$ curl http://gitlab-authorization-scc.apps.
>cluster.domain.example.com/users/sign_in | grep title
...output omitted...
<title>Sign in · GitLab</title>
...output omitted...
```

6. Delete the **authorization-scc** project.

```
[student@workstation ~]$ oc delete project authorization-scc
project.project.openshift.io "authorization-scc" deleted
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab authorization-scc finish
```

This concludes the section.

Lab

Controlling access to OpenShift resources

In this lab, you will configure users with different access levels on the cluster and on a project, and protect access credentials inside a project with a secret.

Outcomes

You should be able to:

- Remove the ability to create projects at the cluster level.
- Grant groups the ability to create projects.
- Manage user privileges in projects by granting privileges to groups.
- Use security context constraints to give extra privileges to service accounts.
- Use secrets to provide sensitive information to deployments.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and downloads the resource files necessary for this exercise.

```
[student@workstation ~]$ lab authorization-review start
```

1. Log in to your OpenShift cluster as the **admin** user and remove the ability to create projects cluster wide.
2. Create a group named **wp-mgrs** for the WordPress managers and grant project creation privileges to it. Add the **leader** user to the group and create the **authorization-review** project.
3. Create a group named **wp-devs** and grant edit privileges on the **authorization-review** project. Add the **developer** user to the group.
4. Create a group named **wp-qa** and grant view privileges on the **authorization-review** project. Add the **qa** user to the group.
5. Allow the **wordpress** application to run as **root**: create a service account named **wordpress-sa** and grant the **anyuid** SCC to it.
6. As the **developer** user, create a secret named **review-secret**, which you will use with the MySQL database and WordPress application.

   The secret should include three key-value pairs: **user=wpuser**, **password=redhat123**, and **database=wordpress**.
7. Deploy a MySQL database application named **mysql** using the image located at **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47**. After it was deployed, modify the deployment configuration to use the **review-secret** secret, as environment variables with the **MYSQL_** prefix.

8. Deploy a WordPress application named **wordpress** using the container image located at **docker.io/library/wordpress:5.3.0**. Add the **WORDPRESS_DB_HOST=mysql** and **WORDPRESS_DB_NAME=wordpress** environmental variables when creating the application. Once deployed, modify the **wordpress** deployment configuration to use the **review-secret** secret as environment variables with the **WORDPRESS_DB_** prefix. The application needs these additional variables to connect to the database. Because the **wordpress** application needs extra privileges, assign the **wordpress-sa** service account to it.
9. As the **qa** user, verify the **mysql** database and **wordpress** application status and try to make a change to the **wordpress** deployment.

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab authorization-review finish
```

This concludes the section.

Solution

Controlling access to OpenShift resources

In this lab, you will configure users with different access levels on the cluster and on a project, and protect access credentials inside a project with a secret.

Outcomes

You should be able to:

- Remove the ability to create projects at the cluster level.
- Grant groups the ability to create projects.
- Manage user privileges in projects by granting privileges to groups.
- Use security context constraints to give extra privileges to service accounts.
- Use secrets to provide sensitive information to deployments.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and downloads the resource files necessary for this exercise.

```
[student@workstation ~]$ lab authorization-review start
```

1. Log in to your OpenShift cluster as the **admin** user and remove the ability to create projects cluster wide.

   1.1. Source the classroom configuration file at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to the cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. Remove the **self-provisioner** cluster role from the **system:authenticated:oauth** virtual group.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group  \
>    self-provisioner system:authenticated:oauth
Warning: Your changes may get lost whenever a master is restarted,
 unless you prevent reconciliation of this rolebinding using the
 following command: oc annotate clusterrolebinding.rbac self-provisioner
 'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite
clusterrole.rbac.authorization.k8s.io/self-provisioner removed:
 "system:authenticated:oauth"
```

Note

You can safely ignore the warning about your changes being lost.

2. Create a group named **wp-mgrs** for the WordPress managers and grant project creation privileges to it. Add the **leader** user to the group and create the **authorization-review** project.

2.1. Create a group named **wp-mgrs**.

```
[student@workstation ~]$ oc adm groups new wp-mgrs
group.user.openshift.io/wp-mgrs created
```

2.2. Grant cluster creation privileges to the **wp-mgrs** group.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group  \
>    self-provisioner wp-mgrs
clusterrole.rbac.authorization.k8s.io/self-provisioner added: "wp-mgrs"
```

2.3. Add the **leader** user to the **wp-mgrs** group.

```
[student@workstation ~]$ oc adm groups add-users wp-mgrs leader
group.user.openshift.io/wp-mgrs added: "leader"
```

2.4. As the **leader** user, create the **authorization-review** project.

```
[student@workstation ~]$ oc login -u leader -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

```
[student@workstation ~]$ oc new-project authorization-review
Now using project "authorization-review" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

3. Create a group named **wp-devs** and grant edit privileges on the **authorization-review** project. Add the **developer** user to the group.

3.1. Log in to the cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

3.2. Create a group named **wp-devs**.

```
[student@workstation ~]$ oc adm groups new wp-devs
group.user.openshift.io/wp-devs created
```

3.3. Add the **developer** user to **wp-devs**.

```
[student@workstation ~]$ oc adm groups add-users wp-devs developer
group.user.openshift.io/wp-devs added: "developer"
```

3.4. Grant edit privileges to the **wp-devs** group on the **authorization-review** project.

```
[student@workstation ~]$ oc policy add-role-to-group edit wp-devs
clusterrole.rbac.authorization.k8s.io/edit added: "wp-devs"
```

4. Create a group named **wp-qa** and grant view privileges on the **authorization-review** project. Add the **qa** user to the group.

4.1. Create a group named **wp-qa**.

```
[student@workstation ~]$ oc adm groups new wp-qa
group.user.openshift.io/wp-qa created
```

4.2. Add the **qa** user to **wp-qa**.

```
[student@workstation ~]$ oc adm groups add-users wp-qa qa
group.user.openshift.io/wp-qa added: "qa"
```

4.3. Grant view privileges to the **wp-qa** group on the **authorization-review** project.

```
[student@workstation ~]$ oc policy add-role-to-group view wp-qa
clusterrole.rbac.authorization.k8s.io/view added: "wp-qa"
```

5. Allow the **wordpress** application to run as **root**: create a service account named **wordpress-sa** and grant the **anyuid** SCC to it.

5.1. Create a service account named **wordpress-sa**.

```
[student@workstation ~]$ oc create sa wordpress-sa
serviceaccount/wordpress-sa created
```

5.2. Grant **anyuid** SCC to the **wordpress-sa** service account.

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z wordpress-sa
securitycontextconstraints.security.openshift.io/anyuid added to:
 ["system:serviceaccount:authorization-review:wordpress-sa"]
```

6. As the **developer** user, create a secret named **review-secret**, which you will use with the MySQL database and WordPress application.

The secret should include three key-value pairs: **user=wpuser**, **password=redhat123**, and **database=wordpress**.

6.1. Log in as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

6.2. Create a secret named **review-secret**.

```
[student@workstation ~]$ oc create secret generic review-secret \
>   --from-literal user=wpuser --from-literal password=redhat123 \
>   --from-literal database=wordpress
secret/review-secret created
```

7. Deploy a MySQL database application named **mysql** using the image located at **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47**. After it was deployed, modify the deployment configuration to use the **review-secret** secret, as environment variables with the **MYSQL_** prefix.

7.1. Create a new application to deploy a **mysql** database server.

```
[student@workstation ~]$ oc new-app --name mysql \
>   --docker-image registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "mysql" created
    deploymentconfig.apps.openshift.io "mysql" created
    service "mysql" created
--> Success
...output omitted...
```

7.2. Use the **review-secret** secret to initialize the environment variables on the **mysql** deployment.

```
[student@workstation ~]$ oc set env dc/mysql --prefix MYSQL_ \
>   --from secret/review-secret
deploymentconfig.apps.openshift.io/mysql updated
```

> **Note**
> It may take a while for the deployment to roll out successfully after setting the secret.

8. Deploy a WordPress application named **wordpress** using the container image located at **docker.io/library/wordpress:5.3.0**. Add the **WORDPRESS_DB_HOST=mysql** and **WORDPRESS_DB_NAME=wordpress** environmental variables when creating the application. Once deployed, modify the **wordpress** deployment configuration to use the **review-secret** secret as environment variables with the **WORDPRESS_DB_** prefix. The application needs these additional variables to connect to the database. Because the **wordpress** application needs extra privileges, assign the **wordpress-sa** service account to it.

    8.1. Deploy a **wordpress** application.

```
[student@workstation ~]$ oc new-app --name wordpress \
>    --docker-image docker.io/library/wordpress:5.3.0 \
>    -e WORDPRESS_DB_HOST=mysql \
>    -e WORDPRESS_DB_NAME=wordpress
...output omitted...
-> Creating resources ...
    imagestream.image.openshift.io "wordpress" created
    deploymentconfig.apps.openshift.io "wordpress" created
    service "wordpress" created
--> Success
...output omitted...
```

    8.2. Set the **wordpress-sa** service account to the **wordpress** deployment configuration.

```
[student@workstation ~]$ oc set serviceaccount deploymentconfig wordpress \
>    wordpress-sa
deploymentconfig.apps.openshift.io/wordpress serviceaccount updated
```

    8.3. Use the **review-secret** secret to initialize the environment variables on the **wordpress** deployment.

```
[student@workstation ~]$ oc set env dc/wordpress --prefix WORDPRESS_DB_ \
>   --from secret/review-secret
deploymentconfig.apps.openshift.io/wordpress updated
```

> **Note**
> It may take a while for the deployment to roll out successfully after setting the secret.

9. As the **qa** user, verify the **mysql** database and **wordpress** application status and try to make a change to the **wordpress** deployment.

    9.1. Log in as the **qa** user.

```
[student@workstation ~]$ oc login -u qa -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

    9.2. Verify the **wordpress** application status.

```
[student@workstation ~]$ oc status
...output omitted...
svc/mysql - 172.30.158.22:3306
  dc/mysql deploys istag/mysql:5.7-47
    deployment #2 deployed 14 minutes ago - 1 pod
    deployment #1 failed 14 minutes ago: newer deployment was found running

svc/wordpress - 172.30.6.39:80
  dc/wordpress deploys istag/wordpress:5.3.0
    deployment #3 deployed 26 seconds ago - 1 pod
    deployment #2 deployed 12 minutes ago
    deployment #1 deployed 12 minutes ago
...output omitted...
```

9.3. Try to delete the **wordpress** application to verify that the **qa** user does not have edit privileges in the project.

```
[student@workstation ~]$ oc delete all -l app=wordpress
Error from server ... User "qa" cannot delete resource "pods" in API group "" in
 the namespace "authorization-review"
Error from server ... User "qa" cannot delete resource "replicationcontrollers" in
 API group "" in the namespace "authorization-review"
Error from server ... User "qa" cannot delete resource "services" in API group ""
 in the namespace "authorization-review"
Error from server ... User "qa" cannot delete resource "deploymentconfigs" in API
 group "apps.openshift.io" in the namespace "authorization-review"
Error from server ... User "qa" cannot delete resource "imagestreams" in API group
 "image.openshift.io" in the namespace "authorization-review"
```

This concludes the lab.

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab authorization-review finish
```

Summary

In this chapter, you learned:

- OpenShift uses role-based access control (RBAC) to control user actions. A role is a collection of rules that govern interaction with OpenShift resources. Default roles exist for cluster administrators, developers, and auditors.
- To control user interaction, you assign a user to one or more roles. A role binding contains all of the associations of a role to users and groups.
- Secret resources allow you to separate sensitive information from application pods. You expose secrets to an application pod either as environment variables or as ordinary files.
- OpenShift uses security context constraints (SCCs) to define allowed pod interactions with system resources. By default, pods operate under the **`restricted`** context which limits access to node resources.

Chapter 5

Configuring OpenShift Networking Components

Goal	Identify the components of Red Hat OpenShift Container Platform software-defined networking and configure some of the components.
Objectives	• Troubleshoot OpenShift software-defined networking using the command-line interface. • Describe the ingress components of OpenShift and create a route.
Sections	• Troubleshooting OpenShift Software-defined Networking (and Guided Exercise) • Controlling Cluster Network Ingress (and Guided Exercise)
Lab	Configuring OpenShift Networking Components

Troubleshooting OpenShift Software-defined Networking

Objectives

After completing this section, you should be able to troubleshoot OpenShift software-defined networking using the command-line interface.

Introducing OpenShift Software-defined Networking

OpenShift implements a *software-defined network (SDN)* to manage the network infrastructure of the cluster and user applications. Software-defined networking is a networking model that allows you to manage network services through the abstraction of several networking layers. It decouples the software that handles the traffic, called the *control plane*, and the underlying mechanisms that route the traffic, called the *data plane*. Among the many features of SDN, open standards enable vendors to propose their solutions, centralized management, dynamic routing, and tenant isolation.

In OpenShift Container Platform, the SDN solves the following five requirements:

- Managing the network traffic and network resources programmatically, so that the organization teams can decide how to expose their applications.
- Managing communication between containers that run in the same project.
- Managing communication between pods, whether they belong to a same project or run in separate projects.
- Managing network communication from a pod to a service.
- Managing network communication from an external network to a service, or from containers to external networks.

The SDN implementation creates a backwards-compatible model, in which pods are akin to virtual machines in terms of port allocation, IP address leasing, and reservation.

Discussing OpenShift Networking Model

The OpenShift SDN uses Linux namespaces to partition the usage of resources and processes on physical and virtual hosts. This implementation allows containers inside pods to share network resources, such as devices, IP stacks, firewall rules, and also routing tables. The OpenShift SDN allocates a unique routable IP to each pod so that you can access the pod from any other service in the same network.

Migrating Legacy Applications

The SDN design makes it easy to containerize your legacy applications because you do not need to change the way the application components communicate with each other. If your application is comprised of many services that communicate over the TCP/UDP stack, this approach still works as containers in a pod share the same network stack. Although using OpenShift services is the recommended approach, you can seamlessly migrate those services before considering migrating all your services in OpenShift.

The following diagram shows how all pods are connected to a shared network.

> **Note**
> The OpenShift *Cluster Network Operator* manages the network, as discussed later.

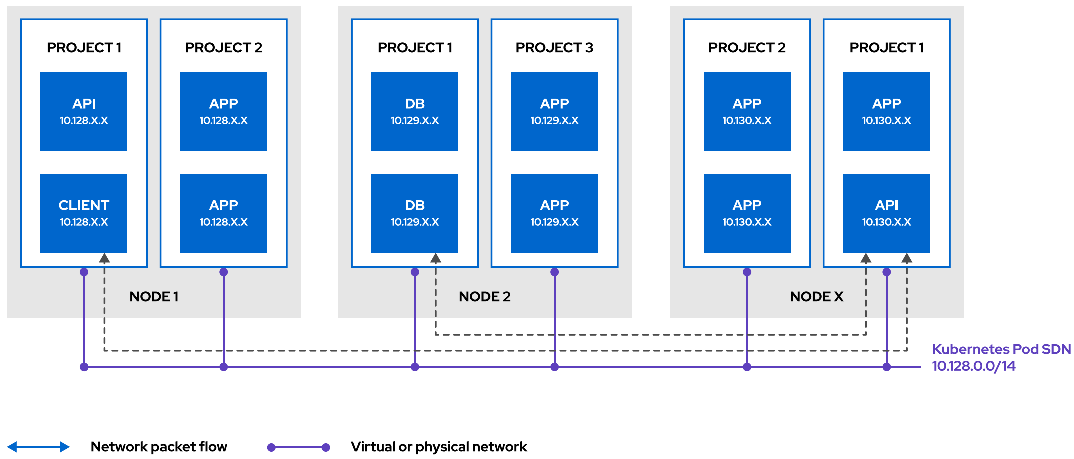


Figure 5.1: Kubernetes basic networking

Using Services for Accessing Pods

Kubernetes provides the concept of a *service*, which is an essential resource in any OpenShift application. Services allow for the logical grouping of pods under a common access route. A service acts as a load balancer in front of one or more pods, thus decoupling the application specifications (such as the number running of replicas) from the access to said application. It load-balances client requests across member pods, and provides a stable interface that enables communication with pods without tracking individual pod IP addresses.

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, such as during the deployment of a new application version or when draining a node for maintenance. Pods are assigned a different IP address each time they are created; thus, pods are not easily addressable. Instead of having a pod discover the IP address of another pod, you can use services, which provide a single, unique IP address for other pods to use, independent of where the pods are running.

Services rely on selectors (labels) that indicate which pods receive the traffic through the service. Each pod matching these selectors is added to the service resource as an endpoint. As pods are created and killed, the service automatically updates the endpoints.

Using selectors brings flexibility to the way you design the architecture and routing of your applications. For example, you can divide the application into tiers and decide to create a service for each tier. Selectors allow a design that is flexible and highly resilient.

OpenShift uses two subnets: one subnet for pods, and one subnet for services. The traffic is forwarded in a transparent way to the pods; an agent (depending on the network mode that you use) manages routing rules to route traffic to the pods that match the selectors.

The following diagram shows how three API pods are running on separate nodes. The **`service1`** service is able to balance the load between these three pods.

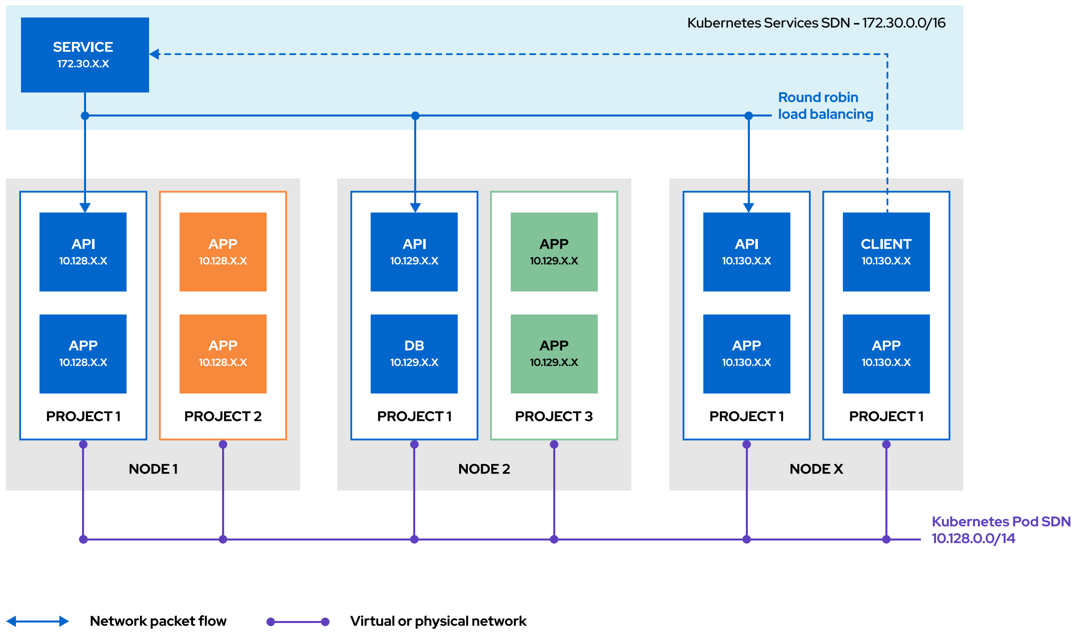


Figure 5.2: Using services for accessing applications

The following YAML definition shows how you create a service. This defines the **`application-frontend`** service, which creates a virtual IP that exposes the TCP port 443. The front-end application listens on the unprivileged port 8843.

```
kind: Service
apiVersion: v1
metadata:
  name: application-frontend ❶
  labels:
    app: frontend-svc ❷
spec:
  ports: ❸
    - name: HTTP
      protocol: TCP
      port: 443 ❹
      targetPort: 8443 ❺
  selector: ❻
    app: shopping-cart
    name: frontend
```

❶ The name of the service. This identifier allows you to manage the service after its creation.

❷ A label that you can use as a selector. This allow you to logically group your services.

❸ An array of objects that describes network ports to expose.

Each entry defines the name for the port mapping. This value is generic and is used for identification purposes only.

❹ This is the port that the service exposes. You use this port to connect to the application that the service exposes.

❺ This is the port on which the application listens. The service creates a forwarding rule from the service port to the service target port.

❻ The selector defines which pods are in the service pool. Services use this selector to determine where to route the traffic.

In this example, the service targets all pods whose labels match **app: shopping-cart** and **name: frontend**.

Defining Service Types

OpenShift supports many service types to accommodate a variety of use cases. The following lists presents the available service types.

- Cluster IP (**ClusterIP**): This service type exposes the service using an IP internal to the cluster. As such, this IP is not accessible from an external network. This is the default value when you define a service; as an administrator, you can configure the CIDR for these IPs at installation time.
- Node port (**NodePort**): This service type instructs the OpenShift control plane to map to an IP address that a node in the cluster uses. This is a legacy type, and Red Hat recommends using *routes* instead, which expose the service as a host name. Routes are discussed in the next section.

  When using this service type, each node proxies the same port number on every node to your service.

  Node ports are in the 30000-32767 range by default, which means that a node port is unlikely to match an intended port for a service.
- Load balancer (**LoadBalancer**): This service type exposes the service through a cloud provider load balancer. You access the virtual IP of the provider's load balancer, and the OpenShift control plane automatically creates a node port or cluster IP to route the incoming packets.
- External name (**ExternalName**): This service type creates a CNAME in the DNS zone that matches an external host name. You typically use this service type to create different access points for applications that are external to the cluster. The service returns the CNAME record whose value matches the external name.

The various service types allow you to control how to access your services. Some service types may be better suited for your environment, such as the **LoadBalancer** type when deploying in Amazon Web Services (AWS), Microsoft Azure, or Google Compute Engine (GCE). With this service type, OpenShift redirects the traffic from the external load balancer to the back-end pods, however, the provider determines the load balancing mechanism and strategy. This can increase the administration overhead associated with the cluster, as you must also manage the permissions for developers to create or delete their services.

Note

The configuration of load balancers is outside the scope of this course.

The **NodePort** type is an older Kubernetes-based approach, whereby the control plan exposes the service to external clients by binding to available ports on the node host. The node then proxies connections to the service IP address. Then you access your application through the node host and the port value. When using this service type, you must manually maintain the list of ports that you use to avoid port collisions. Moreover, you must use a valid port number, that is, a port that is inside the range and that is configured for the **NodePort** service type.

The **ExternalName** service type is a recent addition. Services that use this service type do not map to selectors. Instead, they use DNS names to determine how to access the application that matches the host name. The control plane creates a CNAME record whose value corresponds to the external name. This is a convenient solution for migrating your legacy applications to the cluster, as you can run parts of the application outside of the cluster until their migration is complete.

Discussing the DNS Operator

The DNS operator deploys and runs a DNS server managed by CoreDNS, a lightweight DNS server written in GoLang. The DNS operator provides DNS name resolution between pods, which enables services to discover their endpoints.

Every time you create a new application, OpenShift configures the pods so that they contact the CoreDNS service IP for DNS resolution.

Run the following command to review the configuration of the DNS operator.

```
[user@demo ~]$ oc describe dns.operator/default
Name:         default
...output omitted...
API Version:  operator.openshift.io/v1
Kind:         DNS
...output omitted...
Spec:
Status:
  Cluster Domain:  cluster.local
  Cluster IP:      172.30.0.10
...output omitted...
```

The DNS operator is responsible for the following:

- Creating a default cluster DNS name (**`cluster.local`**).
- Assigning a DNS name to a namespace (for example, **`backend.cluster.local`**).
- Assigning DNS names to services that you define (for example, **`db.backend.cluster.local`**).
- Assigning DNS names to pods in a namespace (such as **`db001.backend.cluster.local`**).

Managing DNS Records for Services

This DNS implementation allows pods to seamlessly resolve DNS names for resources in a project or the cluster. Pods can use a predictable naming scheme for accessing a service. For example, querying the **`db.backend.cluster.local`** from a container returns the IP address of the service. In this case, **db** is the name of the service, **backend** is the project name, and **`cluster.local`** is the cluster DNS name.

CoreDNS creates two kind of records for services: **A** records that resolve to services, and **SRV** records that match the following format:

```
_port-name._port-protocol.svc.namespace.svc.cluster.local
```

For example, if you use a service that exposes the TCP port 443 through the **HTTPS** service, the **SRV** record is created as follows:

```
_443._tcp.https.frontend.svc.cluster.local
```

> **Note**
> When services do not have a cluster IP, the DNS operator assigns them a DNS **A** record that resolves to the set of IPs of the pods behind the service.
>
> Similarly, the **SRV** record that is created resolves to all the pods that are behind the service.

Introducing the Cluster Network Operator

OpenShift Container Platform uses the Cluster Network Operator for managing the SDN. This includes the network CIDR to use, the network mode, the network provider, and the IP address pools.

Run the following **oc get** command as an administrative user to consult the SDN configuration, which is managed by the **Network.config.openshift.io** custom resource definition.

```
[user@demo ~]$ oc get \
>    Network.config.openshift.io cluster -oyaml
apiVersion: config.openshift.io/v1
kind: Network
...output omitted...
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14 ❶
    hostPrefix: 23 ❷
  externalIP:
    policy: {}
  networkType: OpenshiftSDN ❸
  serviceNetwork:
  - 172.30.0.0/16
...output omitted...
```

❶ Defines the CIDR for all pods in the cluster. In this example, the SDN has a netmask of **255.252.0.0** and can allocate 262144 IP addresses.

❷ Defines the host prefix. A value of **23** indicates a netmask of **255.255.254.0**, which translates to 512 allocatable IPs.

❸ Shows the current SDN provider. You can choose between **OpenShiftSDN**, **OVNKubernetes**, and **Kuryr**. Red Hat only supports the **OpenShiftSDN** network provider.

> **Note**
> Configuring additional networks is outside the scope of the course. For more information on the Kubernetes network custom resource definition, consult the *Kubernetes Network Custom Resource Definition De-facto Standard Version 1* document listed in the references section.

Introducing OpenShift Network Modes

The OpenShift software-defined networking (SDN) provides the network layer that the pods in the cluster use for communication. To establish this network layer, the OpenShift SDN uses a plug-in model that allows you to select the appropriate technology for your use case.

In OpenShift Container Platform 4, the **OpenShifSDN** API resource defines and manages the SDN mode. Configure the mode to use in the **defaultNetwork** section of the **install-config.yaml** installation file.

> **Important**
> The SDN configuration is done at installation time.

OpenShift supports three modes: **Multitenant**, **Subnet**, and **NetworkPolicy**; the default mode is **NetworkPolicy**.

The following excerpt shows the SDN mode configuration:

```
defaultNetwork:
  type: OpenShiftSDN
  openshiftSDNConfig:
    mode: NetworkPolicy
    mtu: 1450
    vxlanPort: 4789
```

The configuration instructs the installer to use the **NetworkPolicy** mode, and to define the values of the MTU overlay and the VXLAN UDP port for creating the tunnel. Adjust the MTU value to prevent packets fragmentation, which typically happens when you encapsulate an Ethernet frame into another frame, such as for overlay networks.

Comparing and Contrasting Network Modes

The **Subnet** mode allows you to create a flat network, in which all pods can communicate with each other across projects and tenants.

The **Mutitenant** mode implements segregation at the project level, which provides an extra layer of isolation for pods and services. When using this mode, each project receives a unique VLAN ID that identifies traffic from the pods that belong to the project. Pods are restricted to accessing those pods whose network packet tags use the same VNID. Pods cannot communicate with pods and services in a different project.

> **Note**
> Projects with a VNID of 0 can communicate with all other pods, and vice versa. This is true for all pods that you create in the default project, which assigns a VNID of 0 to the pods.

The **NetworkPolicy** mode provides an extra level of flexibility by allowing you to define network policies for your pods. By default, without any network policy resources defined, pods in a project can access any other pod.

To isolate one or more pods in a project, define a **NetworkPolicy** resource in that project to indicate the allowed ingress and egress connections.

> **Important**
> After declaring any network policy in a project, OpenShift defaults to preventing all ingress and egress traffic, unless you explicitly define a rule to allow network connections.

The following excerpt shows how to allow external users to access an application with labels matching **`app: product-catalog`** over a TCP connection on port 8080.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: external-access
spec:
  podSelector:
    matchLabels:
      app: product-catalog
  ingress:
  - ports:
    - protocol: TCP
      port: 8080
```

One benefit of using network policies is the management of security between projects (tenants), which you cannot do with layer 2 technologies such as VLANs. Network policies allow you to create tailored policies between projects to make sure applications and users can only access what they should.

Introducing Multus Container Network Interface (CNI)

As container adoption increases, so does the need to manage the traffic flow between applications. This means having ways of segregating traffic based on policy, performance, and security.

One way to segregate and manage this traffic flow is to use *network function virtualization software (NFVS)*. NFVS allows you to control and manage the traffic flow on both the data plane and the control plane. Using NFVS allows you to work work with a variety of protocols for performance and security reasons.

Multus is an open source project to support multiple network cards in Kubernetes. One of the challenges that this solves is the migration of NFVS and network function virtualization to containers. Multus is a container network interface (CNI) that acts as a broker and arbiter of other CNI plug-ins for managing the implementation and life-cycle of supplementary network devices in containers. Multus supports plug-ins such as SR-IOV, vHost CNI, Flannel, and Calico.

The following diagram shows how you can design two separate workloads: a kernel-based workload (SR-IOV), and a DPDK-based workload. With these workloads, the control plane network (OpenShift) manages the pods, and each pod connects to an extra data plane through a second network device.

This separation of functions improves the performance for the DPDK-based workload since it no longer relies on the SR-IOV performance.

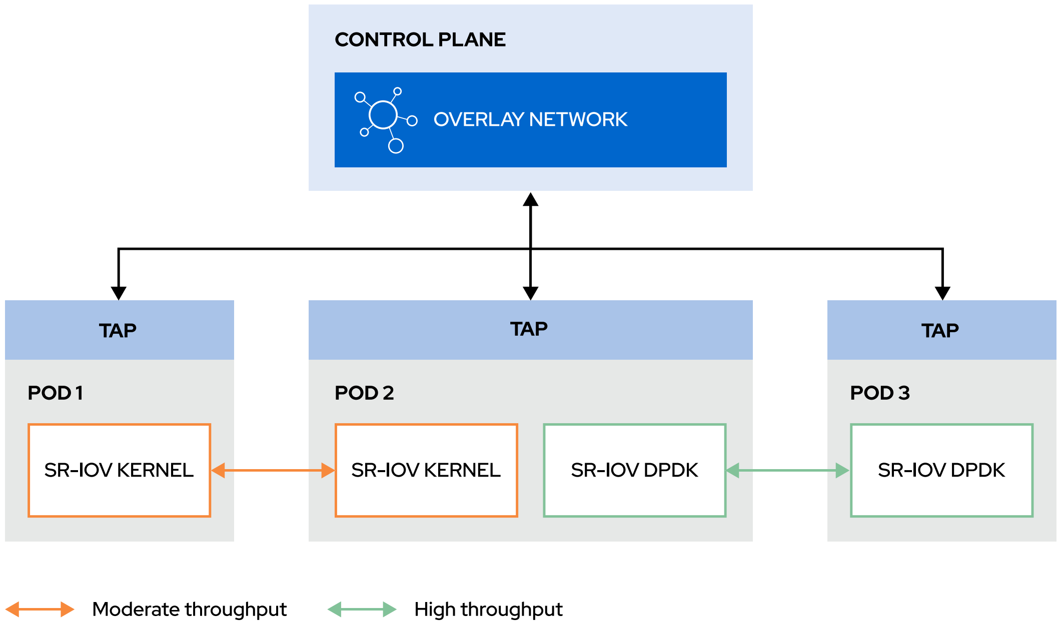


Figure 5.3: Two workloads in OpenShift with Multus CNI

References

For more information, refer to the *Configuring network policy with OpenShift SDN* chapter in the *Red Hat OpenShift Container Platform 4.2 Networking Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/networking/index#configuring-networkpolicy

Network Policy Objects in Action
https://blog.openshift.com/network-policy-objects-action/

Kubernetes Networking Design Document
https://kubernetes.io/docs/concepts/cluster-administration/networking/

Kubernetes Network Custom Resource Definition De-facto Standard Version 1
https://github.com/k8snetworkplumbingwg/multi-net-spec/blob/master/v1.0/%5Bv1%5D%20Kubernetes%20Network%20Custom%20Resource%20Definition%20De-facto%20Standard.md

CoreDNS: DNS and Service Discovery
https://coredns.io/

Multus-CNI
https://github.com/intel/multus-cni

Guided Exercise

Troubleshooting OpenShift Software-defined Networking

In this exercise, you will diagnose and fix connectivity issues with a Kubernetes-style application deployment.

Outcomes

You should be able to:

- Deploy the **To Do** Node.js application.
- Create a route to expose an application service.
- Use **oc debug** to troubleshoot communication between pods in your application.
- Update an OpenShift service.

Before You Begin

To perform this exercise, ensure that you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**/usr/local/bin/oc**).

On the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab network-sdn start
```

As an OpenShift developer, you just completed the migration of a **To Do** Node.js application to OpenShift. The application is comprised of two deployments, one for the database, and one for the front end. It also contains two services for communication between pods.

Although the application seems to initialize, you cannot access it via a web browser. In this activity, you will troubleshoot your application and correct the issue.

1. Log in to the OpenShift cluster and create the **network-sdn** project.

   1.1. From **workstation**, open a terminal and source the classroom configuration file that is accessible at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.3. Create the **network-sdn** project.

```
[student@workstation ~]$ oc new-project network-sdn
Now using project "network-sdn" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
```

2. Deploy the database and restore its data.

The **/home/student/DO280/labs/network-sdn/todo-db.yaml** file defines the following resources:

- A deployment that creates a container based on a MySQL image.
- A service that points to the **mysql** application.

2.1. Go to the **network-sdn** directory and list the files. In a later step, you will use **db-data.sql** to initialize the database for the application.

```
[student@workstation ~]$ cd ~/DO280/labs/network-sdn
[student@workstation network-sdn]$ ls
db-data.sql  todo-db.yaml  todo-frontend.yaml
```

2.2. Use **oc create** with the **-f** against **todo-db.yml** to deploy the database server pod.

```
[student@workstation network-sdn]$ oc create -f todo-db.yaml
deployment.apps/mysql created
service/mysql created
```

2.3. Run **oc status** to review the resources that are present in the project. The **mysql** service points to the database pod.

```
[student@workstation network-sdn]$ oc status
In project network-sdn on server https://api.cluster.domain.example.com:6443

svc/mysql - 172.30.223.41:3306
  deployment/mysql deploys mysql:8.0
    deployment #1 running for 4 seconds - 0/1 pods
...output omitted...
```

2.4. Wait a few moments to ensure that the database pod is running. Retrieve the name of the database pod to restore the tables of the **items** database.

```
[student@workstation network-sdn]$ oc get pods
NAME                    READY   STATUS    RESTARTS   AGE
mysql-94dc6645b-hjjqb   1/1     Running   0          33m
```

2.5. Use **oc cp** to transfer the database dump to the pod. Make sure to replace the pod name with the one you obtained in the previous step.

```
[student@workstation network-sdn]$ oc cp db-data.sql mysql-94dc6645b-hjjqb:/tmp/
```

Note

The command does not produce any output.

2.6. Use **oc rsh** to connect to the pod and restore the database.

```
[student@workstation network-sdn]$ oc rsh mysql-94dc6645b-hjjqb bash
1000570000@mysql-94dc6645b-hjjqb:/$ mysql -u root -p$MYSQL_ROOT_PASSWORD \
>    items < /tmp/db-data.sql

mysql: [Warning] Using a password on the command line interface can be insecure.
```

2.7. Ensure that the **Item** table is present in the database.

```
1000570000@mysql-94dc6645b-hjjqb:/$ mysql -u root -p$MYSQL_ROOT_PASSWORD \
>     items -e "show tables;"

mysql: [Warning] Using a password on the command line interface can be insecure.
+-----------------+
| Tables_in_items |
+-----------------+
| Item            |
+-----------------+
```

2.8. Exit the container.

```
1000570000@mysql-94dc6645b-hjjqb:/$ exit
```

3. Deploy the front end application. The **/home/student/DO280/labs/network-sdn/todo-frontend.yaml** file defines the following resources:

- A deployment that creates the **Todo** Node.js application.
- A service that points to the **frontend** application.

3.1. Use **oc create** to create the front-end application.

```
[student@workstation network-sdn]$ oc create -f todo-frontend.yaml
deployment.apps/frontend created
service/frontend created
```

3.2. Wait a few moments for the front end container to start, and then run **oc get pods**.

```
[student@workstation network-sdn]$ oc get pods
NAME                       READY   STATUS    RESTARTS   AGE
frontend-57b8b445df-f56qh  1/1     Running   0          34s
...output omitted...
```

4. Create a route to access the **frontend** service and access the application.

4.1. You must create a route to access the application from an external network. To do so, use the **oc expose** command against the **frontend** service. Use the **--hostname** option to override the default FQDN that OpenShift creates.

```
[student@workstation network-sdn]$ oc expose service frontend \
>     --hostname todo.${RHT_OCP4_WILDCARD_DOMAIN}
route.route.openshift.io/frontend exposed
```

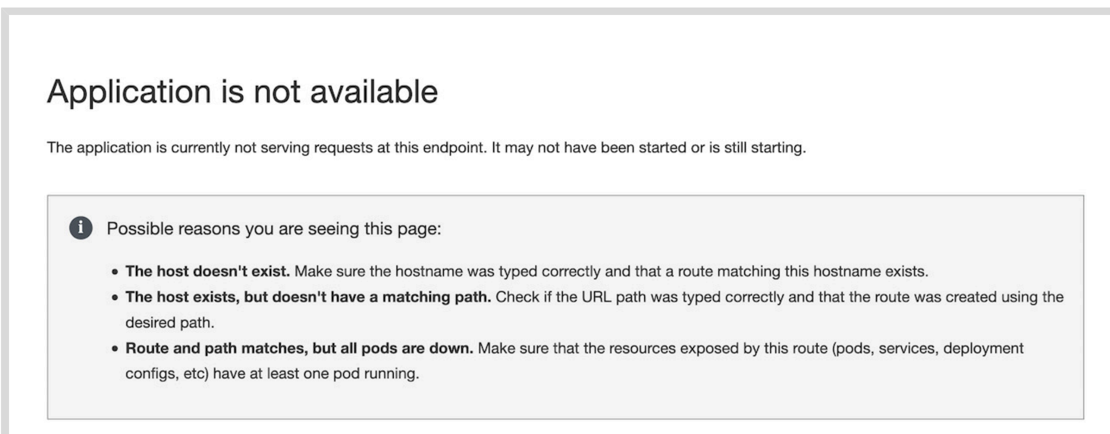
4.2. List the routes in the project.

```
[student@workstation network-sdn]$ oc get routes
NAME       HOST/PORT                            ... PORT  ...
frontend   todo.apps.cluster.domain.example.com ... 8080 ...
```

As you can see, OpenShift detects the port on which the application listens and creates a forwarding rule from port 80 to port 8080, which is the *target* port.

4.3. From **workstation**, open Firefox and access `http://todo.apps.`*`cluster.domain.example.com`*`/todo/`. Make sure to replace the content of the URL with your cluster values and add the trailing slash.

The application is not reachable, as shown in the following screen capture.



4.4. Inspect the pod logs for errors. The output does not indicate any errors.

```
[student@workstation network-sdn]$ oc logs frontend-57b8b445df-f56qh
App is ready at : 8080
```

5. Run **oc debug** to create a carbon copy of an existing pod in the **frontend** deployment. You use this pod to check connectivity to the database.

5.1. Before creating a debug pod, retrieve the database service IP. In a later step, you use **curl** to access the database endpoint.

The JSONPath expression allows you to retrieve the service IP.

```
[student@workstation network-sdn]$ oc get service/mysql \
>    -o jsonpath="{.spec.clusterIP}{'\n'}"
172.30.103.29
```

5.2. Run the **oc debug** command against the **frontend** deployment, which runs the web application pod.

```
[student@workstation network-sdn]$ oc debug -t deployment/frontend
Starting pod/frontend-debug ...
Pod IP: 10.131.0.144
If you don't see a command prompt, try pressing enter.
sh-4.2$
```

5.3. One way to test the connectivity between the **frontend** and the database is the usage of **curl**, which supports a variety of protocols.

Use **curl** to connect to the database over the port 3306, which is MySQL's default port; make sure to replace the IP address with the one that you obtained in Step 5.1. When done, type **Ctrl+C** to exit the session, and type **exit** to exit the debug pod.

```
sh-4.2$ curl -v telnet://172.30.103.29:3306
* About to connect() to 172.30.103.29 port 3306 (#0)
*   Trying 172.30.103.29
* TCP_NODELAY set
* Connected to 172.30.103.29... (172.30.103.29) port 3306 (#0)
* RCVD IAC 193
Ctrl+C
sh-4.2$ exit
exit

Removing debug pod ...
```

The output indicates that the database is up and running, and that it is accessible from the **frontend** deployment.

6. In the following steps, you ensure that the network connectivity inside the cluster is operational by connecting to the front end container from the database container.

Obtain some information about the **frontend** pod and use the **oc debug** command to diagnose the issue from the **mysql** deployment.

6.1. Before creating a debug pod, retrieve IP address of the **frontend** service.

```
[student@workstation network-sdn]$ oc get service/frontend \
>    -o jsonpath="{.spec.clusterIP}{'\n'}"
172.30.23.147
```

6.2. Run the **oc debug** command to create a container for troubleshooting based on the **mysql** deployment. You must override the container image because the MySQL Server image does not provide the **curl** command.

```
[student@workstation network-sdn]$ oc debug -t deployment/mysql \
>    --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/mysql-debug ...
Pod IP: 10.131.0.146
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

6.3. Use **curl** to connect to the **frontend** application over the port 8080. Make sure to replace the IP address with the one that you obtained in Step 6.1

The following output indicates that **curl** times out. This could either indicate that the application is not running or that the service is not able to access the application.

```
sh-4.4$ curl -m 10 -v http://172.30.23.147:8080
* Rebuilt URL to: http://172.30.23.147:8080/
*   Trying 172.30.23.147...
* TCP_NODELAY set
* Connection timed out after 10000 milliseconds
* Closing connection 0
curl: (28) Connection timed out after 10000 milliseconds
```

6.4. Exit the debug pod.

```
sh-4.4$ exit

Removing debug pod ...
```

7. In the following steps, you connect to the **frontend** pod through its private IP. This allows testing whether or not the issue is related to the service.

7.1. Retrieve the IP of the **frontend** pod.

```
[student@workstation network-sdn]$ oc get pods -o wide -l name=frontend
NAME                        READY   STATUS    RESTARTS   AGE   IP            ...
frontend-57b8b445df-f56qh   1/1     Running   0          39m   10.128.2.61   ...
```

7.2. Create a debug pod from the **mysql** deployment.

```
[student@workstation network-sdn]$ oc debug -t deployment/mysql \
>    --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/mysql-debug ...
Pod IP: 10.131.1.27
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

7.3. Run **curl** in verbose mode against the **frontend** pod on port 8080. Replace the IP with the one that you obtained in Step 7.1 and append the name of the application.

```
sh-4.2$ curl -v http://10.128.2.61:8080/todo/
*   Trying 10.128.2.61...
* TCP_NODELAY set
```

```
* Connected to 10.128.2.61 (10.128.2.61) port 8080 (#0)
> GET /todo/ HTTP/1.1
> Host: 10.128.2.61:8080
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: restify
< Cache-Control: public, max-age=3600
< Content-Length: 4508
< Content-Type: text/html
< Last-Modified: Thu, 05 Dec 2019 22:50:51 GMT
< Connection: Keep-Alive
< Date: Fri, 03 Jan 2020 19:53:04 GMT
< Request-Id: aaba2e54-ee4a-47db-8492-8a2f67fc97f1
< Response-Time: 1
...output omitted...
```

`Curl` can access the application through the pod's private IP.

7.4. Exit the debug pod.

```
sh-4.2$ exit
exit

Removing debug pod ...
```

8. Review the configuration of the **`frontend`** service.

8.1. List the services in the project and ensure that the **`frontend`** service exists.

```
[student@workstation network-sdn]$ oc get svc
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
frontend   ClusterIP   172.30.23.147   <none>        8080/TCP   93m
mysql      ClusterIP   172.30.103.29   <none>        3306/TCP   93m
```

8.2. Review the configuration and status of the **`frontend`** service. Notice the value of the service selector that indicates to which pod the service should forward packets.

```
[student@workstation network-sdn]$ oc describe svc/frontend
Name:              frontend
Namespace:         network-sdn
Labels:            app=todonodejs
                   name=frontend
Annotations:      <none>
Selector:          name=api
Type:              ClusterIP
IP:                172.30.23.147
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
Endpoints:         <none>
Session Affinity:  None
Events:            <none>
```

This *...output omitted...* also indicates that the service has no endpoint, so it is not able to forward incoming traffic to the application.

8.3. Retrieve the labels of the **frontend** deployment. The output shows that pods are created with a **name** label that has a value of **frontend**, whereas the service uses **api** as the value.

```
[student@workstation network-sdn]$ oc describe deployment/frontend | \
>    grep Labels -A3
Labels:           app=todonodejs
                  name=frontend
Annotations:      deployment.kubernetes.io/revision: 2
Selector:         app=todonodejs,name=frontend
--
  Labels:  app=todonodejs
           name=frontend
  Containers:
   todonodejs:
```

9. Update the **frontend** service and access the application.

9.1. Run **oc edit** to edit the **frontend** service. Update the selector to match the correct label.

```
[student@workstation network-sdn]$ oc edit svc/frontend
```

Locate the section that defines the selector, and then update the **name: frontend** label inside the selector. After making the changes, exit the editor.

```
...output omitted...
  selector:
    name: frontend
...output omitted...
```

Save your changes and verify that the **oc edit** command applied them.

```
service/frontend edited
```

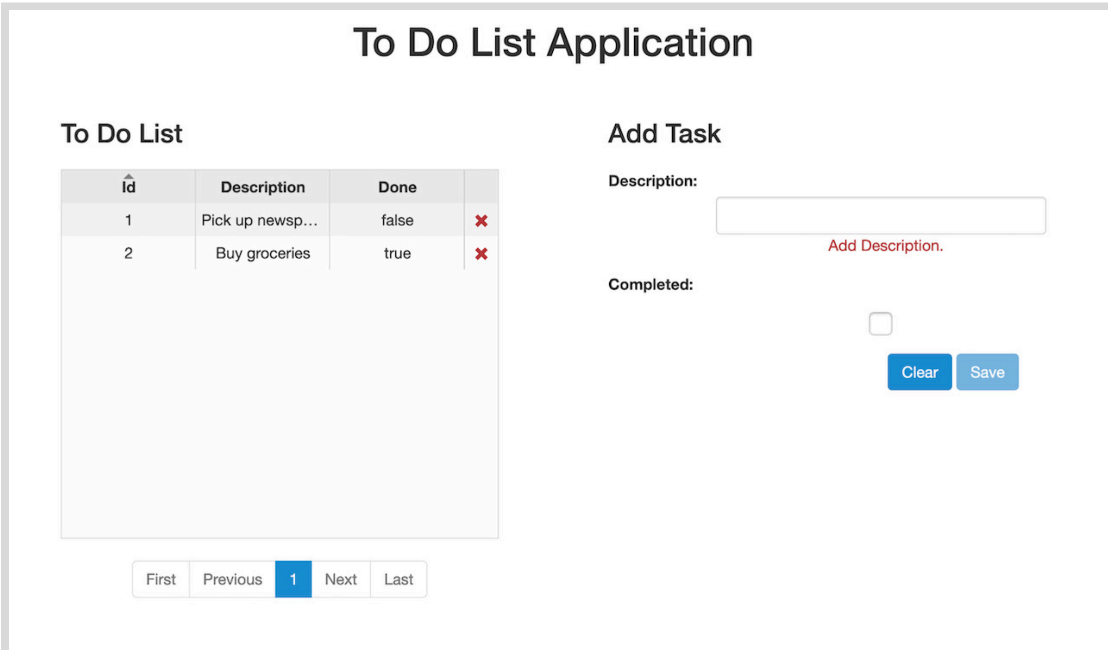
9.2. Review the service configuration to ensure that the service has an endpoint.

```
[student@workstation network-sdn]$ oc describe svc/frontend
Name:              frontend
Namespace:         network-sdn
Labels:            app=todonodejs
                   name=frontend
Annotations:       <none>
Selector:          name=frontend
Type:              ClusterIP
IP:                172.30.169.113
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
```

```
Endpoints:          10.128.2.61:8080
Session Affinity:   None
Events:             <none>
```

9.3. From **workstation**, open Firefox and access the To Do application at `http://todo.apps.cluster.domain.example.com/todo/`. Replace the content of the URL with your cluster values.

You should see the To Do application.



10. Go to the user home directory and delete the **network-sdn** project.

```
[student@workstation network-sdn]$ cd
[student@workstation ~]$ oc delete project network-sdn
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab network-sdn finish
```

This concludes the guided exercise.

Controlling Cluster Network Ingress

Objectives

After completing this section, you should be able to describe the ingress components of OpenShift and create a route.

Accessing Application from External Networks

OpenShift Container Platform offers many ways to expose your applications to external networks. You can expose HTTP and HTTPS traffic, TCP applications, and also non-TCP traffic. Some of these methods are service *types*, such as **`NodePort`** or load balancer, while others use their own API resource, such as **`Ingress`** and **`Route`**. The various types are discussed in the following section.

OpenShift *routes* allow you to expose your applications to external networks. With routes, you can access your application with a unique host name that is publicly accessible. Routes rely on a router *plug-in* to redirect the traffic from the public IP to pods.

The following diagram shows how a route exposes an application running as pods in your cluster.

> **Note**
>
> For performance reasons, routers send requests directly to pods based on service configuration.
>
> The dotted line indicates this implementation, that is, the router accesses the pods through the services network.

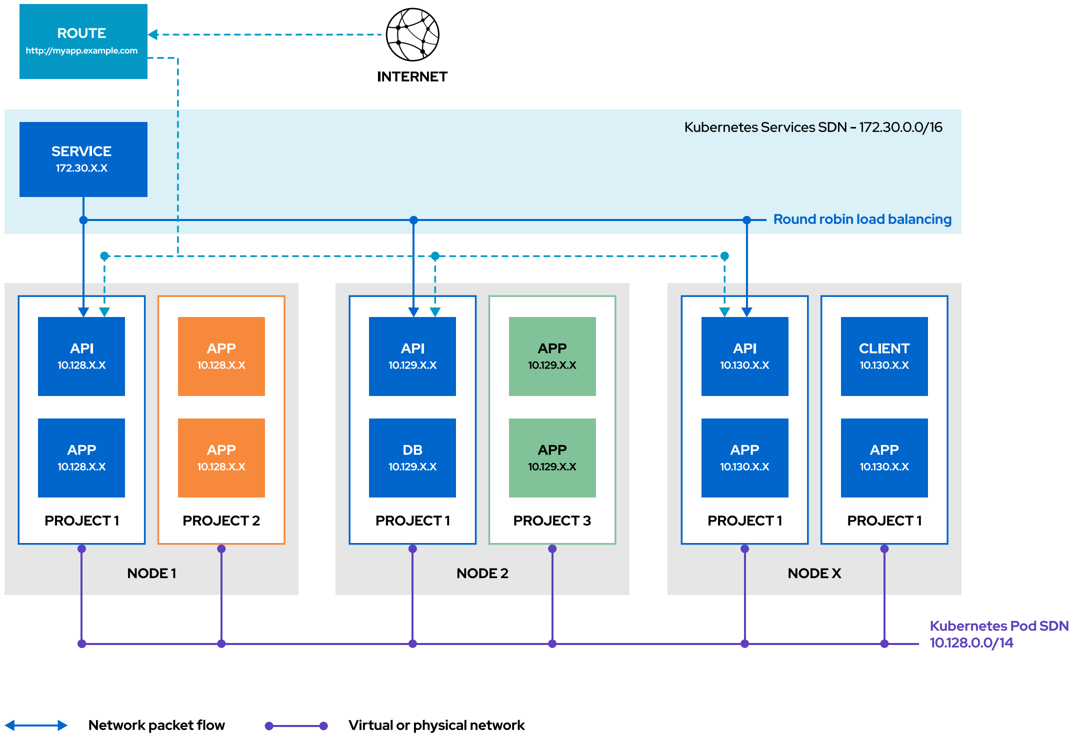


Figure 5.6: Using routes to expose applications

Describing Methods for Managing Ingress Traffic

OpenShift provides different methods for managing ingress traffic. The parenthesis indicates whether this is a service type or a resource.

- **Ingress** (resource). The *Ingress Operator* manages this resource. Ingresses accept external requests and proxy them based on the route. You can only route HTTP, HTTPS and *server name identification (SNI)*, and TLS with SNI.

> **Note**
>
> An ingress is a Kubernetes resource that provides some of the same features as routes (which are an OpenShift resource), but also includes specific features, such as path-based routing, TLS re-encryption, or support of wildcard domains. For more information about the differences between a route and an ingress, consult the *Kubernetes Ingress vs OpenShift Route* link listed in the references section.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example
  namespace: network-ingress
spec:
  rules:
    - host: frontend.apps.acme.com
      http:
        paths:
```

```
          - path: /testpath
            backend:
              serviceName: test
              servicePort: 80
```

- **External load balancer** (service type). This resources instructs OpenShift to spin up a load balancer in a cloud environment. A load balancer instructs OpenShift to interact with the cloud provider in which the cluster is running to provision a load balancer.

```
apiVersion: v1
kind: Service
metadata:
  name: api-lb
spec:
  ports:
  - name: db
    port: 3306
  loadBalancerIP:
  type: LoadBalancer
  selector:
    name: api-frontend
```

- **Service external IP** (service type). This method instructs OpenShift to set NAT rules to redirect traffic from one of the cluster IPs to the container.

- **NodePort** (service type). With this method, OpenShift exposes a service on a static port on the node IP address. You must ensure that the external IP addresses are properly routed to the nodes.

OpenShift ingresses and routes are implemented by a shared router service that runs as a pod inside the cluster. You can scale and replicate this pod 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 must point to the public-facing IP addresses of the nodes running the router. Router pods, unlike regular application pods, bind to public IP addresses of the nodes, instead of to the internal pod network. This is typically configured using a DNS wildcard.

You must provide the following value when creating a route:

- The name of a service. The route uses the service to determine the pods to which to route the traffic.

- A host name for the route. A route is always a subdomain of your cluster wildcard domain. For example, if you are using a wildcard domain of **`apps.dev-cluster.acme.com`**, and need to expose a front-end service through a route, then it will be named:

```
frontend.apps.dev-cluster.acme.com
```

> **Note**
> You can also let OpenShift automatically generate a host name for the route.

- An optional path, for path-based routes.

- A target port, which is the port on which the application listens. The target port usually corresponds to the port that you define in the **`targetPort`** key of a service.
- An encryption strategy, depending on whether you need a secure or insecure route.

The following listing shows a minimal definition for a route:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: a-simple-route ❶
  labels: ❷
    app: API
    name: api-frontend
spec:
  host: api.apps.acme.com ❸
  to:
    kind: Service
    name: api-frontend ❹
    weight: 100
  port: ❺
    targetPort: 8443
```

❶ The name of the route. This name must be unique.
❷ A set of labels that you can use as selectors.
❸ The host name of the route. This host name must be a subdomain of you wildcard domain, since OpenShift routes the wildcard domain to the routers.
❹ The service to which to redirect the traffic. Although you use a service name, the route only uses this information to determine the list of pods that receive the traffic.
❺ The application port. Because routes bypass services, this must match the application port and not the service port.

Describing Route Options and Route 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 presented in the following list.

OpenShift Secure Routes

Edge
: With edge termination, TLS termination occurs at the router, before the traffic is routed to the pods. The router serves the TLS certificates, so you must configure them into the route; otherwise, OpenShift assigns its own certificate to the router 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`
: With pass-through termination, encrypted traffic is sent straight to the destination pod without the router providing TLS termination. In this mode, the application is responsible for serving certificates for the traffic. This is currently the only method that supports mutual authentication between the application and a client that accesses it.

`Re-encryption`

Re-encryption is a variation on edge termination, whereby the router terminates TLS with a certificate, and 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 Insecure Routes

Insecure routes correspond to traffic that is not encrypted.

The easiest and preferred way to create a route (secure or insecure) is to use the **`oc expose service`** ***`service`*** command, where **`service`** corresponds to a service. Use the **`--hostname`** option to provide a custom host name for the route.

```
[user@demo ~]$ oc expose service api-frontend \
>    --hostname api.apps.acme.com
```

When omitting the host name, OpenShift generates a host name for you with the following structure: *<route-name>-<project-name>.<default-domain>*

For example, if you create a **`frontend`** route in a **`api`** project, in a cluster that uses **`apps.example.com`** as the wildcard domain, then the route host name will be:

frontend.api.apps.example.com.

> **Important**
>
> 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.

Securing Applications

Securing Applications with Edge Routes

Before creating a secure route, you need to generate a TLS certificate. The following command shows how to create a secure edge route with a TLS certificate.

The **`--key`** option requires the certificate private key, and the **`--cert`** option requires the certificate that has been signed with that key.

```
[user@demo ~]$ oc create route edge \
  >    --service api-frontend --api.apps.acme.com \
  >    --key api.key --cert api.crt
```

When using a route in edge mode, the traffic between the client and the router is encrypted, but traffic between the router and the application is not.

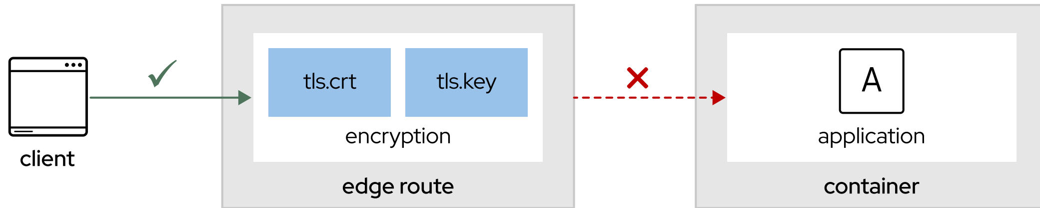


Figure 5.7: Securing applications with edge routes

> **Note**
> Network policies can help you protect the internal traffic between your applications or between projects. For more information on how to accomplish this, consult the *Network Policy Objects in Action* document in the references section.

Securing Applications with Pass-through Routes

The previous example demonstrates how to create an edge route, that is, an OpenShift route that presents a certificate at the edge. Pass-through routes offer a secure alternative because the application exposes its TLS certificate. As such, the traffic is encrypted between the client and the application.

To create a pass-through route, you need a certificate and a way for your application to access it. The best way to accomplish this is by using OpenShift TLS *secrets*. Secrets are exposed via a mount point into the container.

The following diagram shows how you can mount a **`secret`** resource in your container. The application is then able to access your certificate. With this mode, there is no encryption between the client and the router.

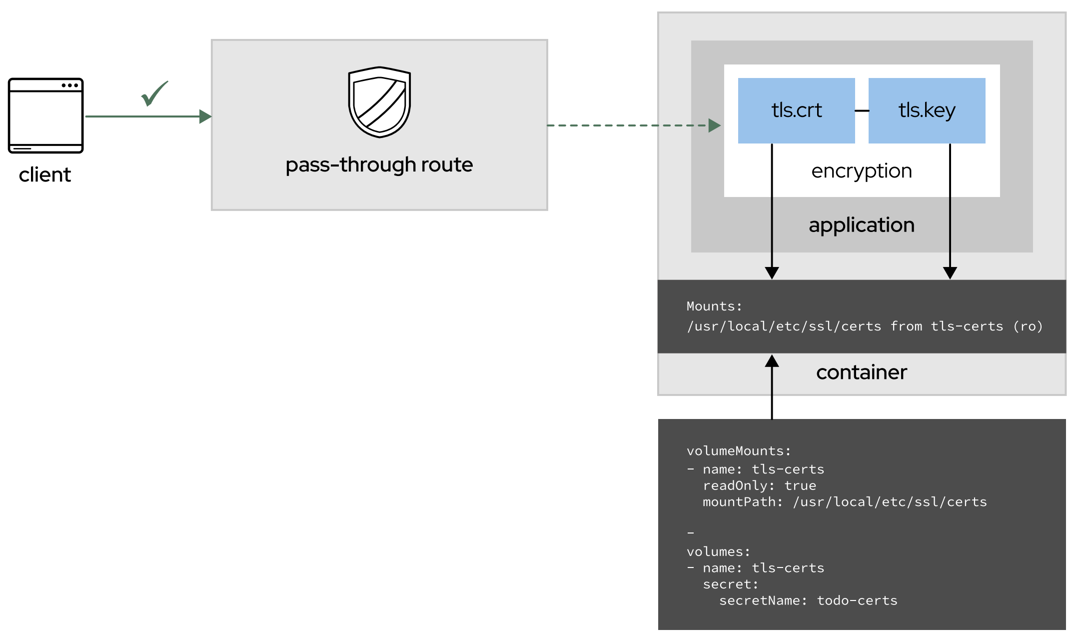


Figure 5.8: Securing applications with pass-through routes

References

For more information on how to manage routes, refer to *Configuring Routes* chapter in the *Red Hat OpenShift Container Platform 4.2 Networking Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/networking/index#configuring-routes

For more information on how to configure ingress cluster traffic, refer to *Configuring Routes* chapter in the *Red Hat OpenShift Container Platform 4.2 Networking Guide* at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/networking/index#configuring-ingress-cluster-traffic

For more information, refer to *Routes* section in the *Red Hat OpenShift Online Developer Guide* at
https://docs.openshift.com/online/pro/dev_guide/routes.html

Kubernetes Ingress vs OpenShift Route – OpenShift Blog
https://blog.openshift.com/kubernetes-ingress-vs-openshift-route/

Network Policy Objects in Action – OpenShift Blog
https://blog.openshift.com/network-policy-objects-action/

Guided Exercise

Controlling Cluster Network Ingress

In this exercise, you will expose an application secured by TLS certificates.

Outcomes

You should be able to:

- Deploy an application and create an unencrypted route for it.
- Create an OpenShift edge route with encryption.
- Update an OpenShift deployment to support a new version of the application.
- Create an OpenShift TLS secret and mount it to your application.
- Verify that the communication to the application is encrypted.

Before You Begin

To perform this exercise, ensure that you have access to:

- A running OpenShift cluster.

On the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable, and creates the **network-ingress** OpenShift project. It also gives the **developer** user edit access on the project.

```
[student@workstation ~]$ lab network-ingress start
```

As an application developer, you are ready to deploy your application in OpenShift. In this activity, you will deploy two versions of the application, one that is exposed over unencrypted traffic (HTTP), and one that is exposed over secure traffic.

The container image, accessible at **https://quay.io/redhattraining/todo-angular**, has two tags: **v1.1**, which is the insecure version of the application, and **v1.2**, which is the secure version. Your organization uses its own certificate authority (CA) that can sign certificates for the ***.apps.cluster.domain.example.com** and ***.cluster.domain.example.com** domains.

The CA certificate is accessible at **~/DO280/labs/network-ingress/certs/training-CA.pem**. The **passphrase.txt** contains a unique password that protects the CA key. The **certs** folder also contains the CA key.

1. Log in to the OpenShift cluster and create the **network-ingress** project.

   1.1. From **workstation**, open a terminal and source the classroom configuration file that is accessible at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

1.2. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>   ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.3. Create the **network-ingress** project.

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

2. The OpenShift deployment file for the application is accessible at **~/DO280/labs/network-ingress/todo-app-v1.yaml**. The deployment points to **quay.io/redhattraining/todo-angular:v1.1**, which is the initial and unencrypted version of the application. The file defines the **todo-https** service that points to the application pod.

Create the application and expose the service.

2.1. Use **oc create** to deploy the application in the **network-ingress** OpenShift project.

```
[student@workstation ~]$ oc create -f \
>   ~/DO280/labs/network-ingress/todo-app-v1.yaml
deployment.apps/todo-http created
service/todo-http created
```

2.2. Wait a couple of minutes, so that the application can start, and then review the resources in the project.

```
[student@workstation ~]$ oc status
In project network-sdn on server https://api.cluster.domain.example.com:6443

svc/todo-http - 172.30.247.75:80 -> 8080
  deployment/todo-http deploys quay.io/redhattraining/todo-angular:v1.1
    deployment #1 running for 16 seconds - 1 pod
...output omitted...
```

2.3. Run **oc expose** to create a route for accessing the application. Give the route a host name of **todo-http.apps.cluster.domain.example.com**.

```
[student@workstation ~]$ oc expose svc todo-http \
>   --hostname todo-http.${RHT_OCP4_WILDCARD_DOMAIN}
route.route.openshift.io/todo-http exposed
```

2.4. Retrieve the name of the route and copy it to the clipboard.

```
[student@workstation ~]$ oc get routes
NAME        HOST/PORT                                     PATH        ...
todo-http   todo-http.apps.cluster.domain.example.com     todo-http   ...
```

2.5. From **workstation**, open Firefox and access `http://todo-http.apps.`*`cluster.domain.example.com`*. Replace the domain with the content of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

Confirm that you can see the To Do application.

2.6. From **workstation** open a new terminal tab and run **tcpdump** with the following options to intercept the traffic on port 80:

- **-i ens*X*** intercepts traffic on the main interface.
- **-A** strips the headers and prints the packets in ASCII format.
- **-n** disables DNS resolution.
- **port 80** is the port of the application.

Optionally, the **grep** command allows you to filter on JavaScript resources.

Start by retrieving the name of the main interface whose IP is **172.25.250.9**.

```
[student@workstation ~]$ ip a | grep 172.25.250.9
inet 172.25.250.9/24 brd 172.25.250.255 scope global noprefixroute ens3
[student@workstation ~]$ sudo tcpdump -i ens3 -A \
>    -n port 80 | grep js
```

Note

The full command is available at **~/DO280/labs/network-ingress/tcpdump-command.txt**.

2.7. On Firefox, refresh the page and notice the activity in the terminal. Type **Ctrl**+**C** to stop the capture.

```
...output omitted...
             toBe('Pretty text with some links: http://angularjs.org/,
 us@somewhere.org, ' +
             toBe('Pretty text with some links: http://angularjs.org/,
 mailto:us@somewhere.org, ' +
            toBe('http://angularjs.org/');
...output omitted...
    /*jshint validthis: true */
    /*jshint validthis: true */
...output omitted...
```

3. Create a secure edge route. Edge certificates encrypt the traffic between the client and the router, but leave the traffic between the router and the service unencrypted. OpenShift generates its own certificate that it signs with its CA.

In later steps, you extract the CA to ensure the signing of the route certificate.

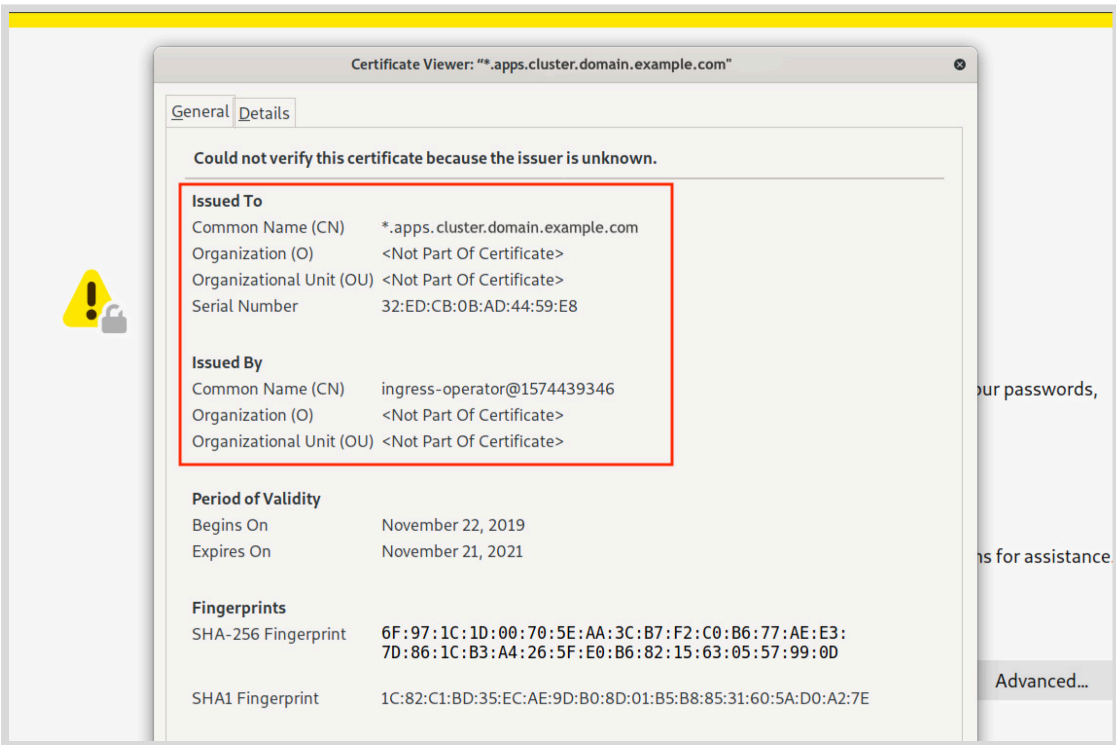
3.1. Go to **~/DO280/labs/network-ingress** and run the **oc create route** command to define the new route. Use the **--cert** and **key** options to embed the TLS certificates.

Give the route a host name of **todo-https.apps.*cluster.domain.example.com***.

```
[student@workstation ~]$ cd ~/DO280/labs/network-ingress
[student@workstation network-ingress]$ oc create route edge todo-https \
>   --service todo-http \
>   --hostname todo-https.${RHT_OCP4_WILDCARD_DOMAIN}
route.route.openshift.io/todo-https created
```

3.2. To test the route and read the certificate, open Firefox and access `https://todo-https.apps.cluster.domain.example.com`. Upon first access, Firefox warns you about the certificate. Click **Advanced**, then **Add Exception** and then **View Certificate** to read the certificate.

Locate the CN entry that indicates that the OpenShift ingress operator created the certificate with its own CA.



3.3. Use **`curl`** to further confirm rejection of the certificate.

```
[student@workstation network-ingress]$ curl \
>   https://todo-https.${RHT_OCP4_WILDCARD_DOMAIN}
curl: (60) SSL certificate problem: self signed certificate in certificate chain
...output omitted...
```

3.4. One way to verify how the certificate is signed by OpenShift is to retrieve the CA that the ingress operator uses. This allows you to validate the edge certificate against the CA.

From **`workstation`**, log in to the cluster as the **`admin`** user. The **`developer`** user is not allowed to retrieve the CA.

> **Important**
> In a real word scenario, administrators with access to the namespace retrieve that CA and give it to the developers.

```
[student@workstation network-ingress]$ oc login -u admin \
>   -p ${RHT_OCP4_USER_PASSWD}
```

3.5. Run **oc extract** to retrieve the CA present in the **openshift-ingress-operator** namespace.

```
[student@workstation network-ingress]$ oc extract secrets/router-ca \
>   --keys tls.crt -n openshift-ingress-operator
tls.crt
```

3.6. From the terminal, use **curl** to retrieve the connection headers. Use the **--cacert** option to pass the CA to Curl.

The **Server certificate** section shows some information about the certificate, and the alternative name matches the name of the route.

```
[student@workstation network-ingress]$ curl -I -v \
>   --cacert tls.crt https://todo-https.${RHT_OCP4_WILDCARD_DOMAIN}
...output omitted...
* Server certificate:
*  subject: CN=*.apps.cluster.domain.example.com
*  start date: Nov 22 16:15:50 2019 GMT
*  expire date: Nov 21 16:15:51 2021 GMT
*  subjectAltName: host "todo-https.apps.cluster.domain.example.com \
   matched cert's "*.apps.cluster.domain.example.com"
*  issuer: CN=ingress-operator@1574439346
*  SSL certificate verify ok.
...output omitted...
```

The output indicates that Curl trusts the remote certificate, since it matches the CA.

3.7. Log back as the **developer** user.

```
[student@workstation network-ingress]$ oc login -u developer \
>   -p ${RHT_OCP4_USER_PASSWD}
```

3.8. Although the traffic is encrypted at the edge with a certificate, you can still access the insecure traffic at the service level. This is because the pod behind the service does not offer an encrypted route.

Retrieve the IP address of the **todo-http** service.

```
[student@workstation network-ingress]$ oc get svc todo-http \
>   -o jsonpath="{.spec.clusterIP}{'\n'}"
172.30.102.29
```

3.9. Create a debug pod in the **todo-http** deployment. Use the Red Hat Universal Base Image (UBI), which contains some basic tools to interact with containers.

```
[student@workstation network-ingress]$ oc debug -t deployment/todo-http \
>   --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/todo-http-debug ...
Pod IP: 10.131.0.255
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

3.10. From the debug pod, use **curl** to access the service over HTTP. Replace the IP address with the one that you obtained in a previous step.

The output indicates that the application is available over HTTP.

```
sh-4.4$ curl -v 172.30.102.29
* Rebuilt URL to: 172.30.102.29/
*   Trying 172.30.102.29...
* TCP_NODELAY set
* Connected to 172.30.102.29 (172.30.102.29) port 80 (#0)
> GET / HTTP/1.1
> Host: 172.30.102.29
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.14.1
< Date: Fri, 29 Nov 2019 22:21:09 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
...output omitted...
```

3.11. Exit the debug pod.

```
sh-4.4$ exit
Removing debug pod ...
```

3.12. Delete the edge route. In the next steps, you define the pass-through route.

```
[student@workstation network-ingress]$ oc delete route todo-https
route.route.openshift.io "todo-https" deleted
```

4. Generate TLS certificates for the application.

In the following steps, you generate a CA-signed certificate that you attach as a secret to the pod. You then configure a secure route in pass-through mode and let the application expose that certificate.

4.1. Go to the **~/DO280/labs/network-ingress/certs** directory and list the files.

```
[student@workstation network-ingress]$ cd certs
[student@workstation certs]$ ls -al
total 16
drwxr-xr-x. 2 student student   94 Nov 29 19:20 .
drwxr-xr-x. 3 student student  113 Nov 29 19:06 ..
-rw-rw-r--. 1 student student  604 Nov 29 17:35 openssl-commands.txt
-rw-r--r--. 1 student student   33 Nov 29 17:35 passphrase.txt
-rw-r--r--. 1 student student 1743 Nov 29 17:35 training-CA.key
-rw-r--r--. 1 student student 1363 Nov 29 17:35 training-CA.pem
-rw-r--r--. 1 student student  406 Nov 29 17:35 training.ext
```

4.2. Generate the private key for your CA-signed certificate.

> **Note**
> The following commands for generating a signed certificate are all available in the **openssl-commands.txt** file, available in the directory.

```
[student@workstation certs~]$ openssl genrsa \
>   -out training.key 2048
Generating RSA private key, 2048 bit long modulus
....................
.....+++
..........................................+++
e is 65537 (0x10001)
```

4.3. Generate the certificate signing request (CSR) for **todo-apps.*cluster.domain.example.com***. Make sure to type the subject of the request in one line. Alternatively, remove the **-subj** option and its content. This prompts you for the values; make sure to indicate a common name (**CN**) of **todo-https.${RHT_OCP4_WILDCARD_DOMAIN}**.

```
[student@workstation certs]$ openssl req -new \
>   -subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\
> CN=todo-https.${RHT_OCP4_WILDCARD_DOMAIN}" \
>   -key training.key -out training.csr
```

4.4. Finally, generate the signed certificate. Notice the usage of the **-CA** and **-CAkey** options for signing the certificate against the CA. The **-passin** option allows you to reuse the password of the CA. The **extfile** option allows you to define a *Subject Alternative Name (SAN)*.

```
[student@workstation certs]$ openssl x509 -req -in training.csr \
>   -passin file:passphrase.txt \
>   -CA training-CA.pem -CAkey training-CA.key -CAcreateserial \
>   -out training.crt -days 1825 -sha256 -extfile training.ext
Signature ok
subject=/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/
CN=todo-https.apps.cluster.domain.example.com
Getting CA Private Key
```

4.5. Ensure that the newly created certificate and key are present in the current directory.

```
[student@workstation certs]$ ls -alt
total 36
-rw-rw-r--. 1 student student 1484 Nov 29 19:46 training.crt
-rw-rw-r--. 1 student student   17 Nov 29 19:46 training-CA.srl
-rw-rw-r--. 1 student student 1050 Nov 29 19:45 training.csr
-rw-rw-r--. 1 student student 1050 Nov 29 19:31 .
-rw-rw-r--. 1 student student 1675 Nov 29 19:45 training.key
drwxr-xr-x. 3 student student  109 Nov 29 19:42 ..
-rw-r--r--. 1 student student 1363 Nov 29 19:38 training-CA.pem
-rw-r--r--. 1 student student 1751 Nov 29 19:38 training-CA.key
-rw-r--r--. 1 student student  406 Nov 29 19:38 training.ext
-rw-r--r--. 1 student student   33 Nov 29 19:38 passphrase.txt
-rw-r--r--. 1 student student  604 Nov 29 19:32 openssl-commands.txt
```

4.6. Go back to the **network-ingress** directory. This is important as the next step involves the creation of a route using the self-signed certificate.

```
[student@workstation certs]$ cd ~/DO280/labs/network-ingress
```

5. Deploy a new version of your application. The new version of the application expects a certificate and a key inside the container at **/usr/local/etc/ssl/certs**. The web server in that version is configured with SSL support. Create a secret to import the certificate from **workstation**. In a later step, the application deployment requests that secret and exposes its content to the container at **/usr/local/etc/ssl/certs**.

5.1. Create a **tls** OpenShift secret named **todo-certs**. Pass the signed **training.csr** as the certificate, and **training.key** as the key.

```
[student@workstation network-ingress]$ oc create secret tls todo-certs \
>   --cert=certs/training.crt \
>   --key=certs/training.key
secret/todo-certs created
```

5.2. The deployment file, accessible at **~/DO280/labs/network-ingress/todo-app-v2.yaml**, points to version 2 of the container image. The new version of the application is configured to support SSL certificates. Run **oc create** to create a new deployment using that image.

```
[student@workstation network-ingress]$ oc create -f todo-app-v2.yaml
deployment.apps/todo-https created
service/todo-https created
```

5.3. Wait a couple of minutes to ensure that the application pod is running. Copy the pod name to your clipboard.

```
[student@workstation network-ingress]$ oc get pods
NAME                          READY   STATUS    RESTARTS   AGE
...output omitted...
todo-https-59d8fc9d47-265ds   1/1     Running   0          62s
```

5.4. Review the volumes that are mounted inside the pod. The output indicates that the certificates are mounted to **/usr/local/etc/ssl/certs**.

```
[student@workstation network-ingress]$ oc describe pod \
>   todo-https-59d8fc9d47-265ds | grep Mounts -A2
    Mounts:
      /usr/local/etc/ssl/certs from tls-certs (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-prz4k (ro)
```

6. Create the secure pass-through route.

6.1. Run the **oc create route** command to define the new route.

Give the route a host name of **todo-https.apps.cluster.domain.example.com**.

```
[student@workstation network-ingress]$ oc create route passthrough todo-https \
>   --service todo-https --port 8443 \
>   --hostname todo-https.${RHT_OCP4_WILDCARD_DOMAIN}
route.route.openshift.io/todo-https created
```

6.2. Use **curl** in verbose mode to test the route and read the certificate. Use the **--cacert** option to pass the CA certificate to **curl**.

The output indicates a match between the certificate chain and the application certificate. This indicates that the OpenShift router only forwards packets that are encrypted by the application web server certificate.

```
[student@workstation network-ingress]$ curl -vvI \
>   --cacert certs/training-CA.pem \
>   https://todo-https.${RHT_OCP4_WILDCARD_DOMAIN}
...output omitted...
* Server certificate:
*  subject: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; \
   CN=todo-https.apps.cluster.domain.example.com
*  start date: Jan  3 21:31:19 2020 GMT
*  expire date: Jan  1 21:31:19 2025 GMT
*  subjectAltName: host "todo-https.apps.cluster.domain.example.com" \
   matched cert's "*.apps.cluster.domain.example.com"
*  issuer: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; \
   CN=cluster.domain.example.com
*  SSL certificate verify ok.
```

7. Create a new debug pod to further confirm proper encryption at the service level.

7.1. Retrieve the IP address of the **todo-https** service.

```
[student@workstation network-ingress]$ oc get svc todo-https \
>   -o jsonpath="{.spec.clusterIP}{'\n'}"
172.30.121.154
```

7.2. Create a debug pod in the **todo-https** deployment with the Red Hat UBI.

```
[student@workstation network-ingress]$ oc debug -t deployment/todo-https \
>   --image registry.access.redhat.com/ubi8/ubi:8.0
Starting pod/todo-https-debug ...
Pod IP: 10.128.2.129
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

7.3. From the debug pod, use **curl** to access the service over HTTP. Replace the IP address with the one that you obtained in a previous step.

The output indicates that the application is not available over HTTP, and the web server redirects you to the secure version.

```
sh-4.4$ curl -I http://172.30.102.29
HTTP/1.1 301 Moved Permanently
Server: nginx/1.14.1
Date: Sat, 30 Nov 2019 00:01:35 GMT
Content-Type: text/html
Connection: keep-alive
Location: https://172.30.102.29:8443/
```

7.4. Finally, access the application over HTTPS. Use the **-k** option because the container does not have access to the CA certificate.

```
sh-4.4$ curl -k https://172.30.102.29:8443
<!DOCTYPE html>
<html lang="en" ng-app="todoItemsApp" ng-controller="appCtl">
<head>
    <meta charset="utf-8">
    <title>ToDo app</title>
```

7.5. Exit the debug pod.

```
sh-4.4$ exit
Removing debug pod ...
```

8. Go to the home directory and delete the **network-ingress** project.

```
[student@workstation network-ingress]$ cd
[student@workstation ~]$ oc delete project network-ingress
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab network-ingress finish
```

This concludes the guided exercise.

Lab

Configuring OpenShift Networking Components

In this lab, you will configure a TLS pass-through route for your application.

Outcomes

You should be able to:

- Deploy an application and configure an insecure route.
- Generate a TLS certificate for an application.
- Configure a pass-through termination route for an application with a TLS certificate.

Before You Begin

To perform this exercise, ensure that you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**/usr/local/bin/oc**).

On the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates the self-signed certificate authority (CA) that you use in this lab.

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

In this review, you deploy the PHP application named **PHP-SSL**, which prints some information about the system. The application is available in two flavors: one that runs with an unencrypted network that listens on port 8080, and one that uses a TLS certificate to encrypt the network traffic, which listens on port 8443.

The container image for this review is accessible at `quay.io/redhattraining/php-ssl`. It has two tags: **v1.0** for the insecure version of the application, and **v1.1** for the secure version.

1. As the OpenShift **developer** user, create the **network-review** project. Credentials for accessing the cluster are available at **/usr/local/etc/ocp4.config**

2. As the **developer** user, deploy the first insecure version of the **PHP-SSL** application in the **network-review** OpenShift project. Use the resources file available at **~/DO280/labs/network-review/php-http.yaml**.

   Before deploying the application, make the necessary changes to the file, namely, the location of the container image and the port on which it listens.

   After creating the application, wait a few moments to ensure that one pod is running.

3. Create a route named **php-http**, with a host name of `php-http.`*`apps.cluster.domain.example.com`*, to access the application.

Replace ***.apps.cluster.domain.example.com*** with the value of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

From **workstation**, use Firefox to access `http://php-http`*`.apps.cluster.domain.example.com`*. Confirm the availability of the application before proceeding to the next step.

4. Generate and sign a TLS certificate for the encrypted version of the application.

   Create a certificate signing request (CSR) for the **php-https*.apps.cluster.domain.example.com*** domain. Save the CSR to **~/DO280/labs/network-review/certs/training.csr**.

   Use the CSR to generate a certificate and save it to **~/DO280/labs/network-review/certs/training.crt**. To generate the certificate, pass as arguments the CA accessible at **~/DO280/labs/network-review/certs/training-CA.pem** and the CSR.

   You can use the **~/DO280/labs/network-review/certs/openssl-commands.txt** text file for help. This file contains the commands for generating the certificate signing request and the certificate. Make sure to replace the values in the file before copying and running the OpenSSL commands.

5. As the **developer** user, create an OpenShift TLS secret named **php-certs** in the **network-review** project. Use **~/DO280/labs/network-review/certs/training.crt** for the certificate and **~/DO280/labs/network-review/certs/training.key** for the key.

6. Use the resources file, available at **~/DO280/labs/network-review/php-https.yaml**, to deploy the second, secure version of the **PHP-SSL** application. Deploy the application in the **network-review** OpenShift project.

   Before deploying the application, make the necessary changes to the resources file, namely:

   - The location of the container.
   - The port the application listens on.
   - The name of the secret to mount as a volume.

7. Create a secure pass-through route named **php-https**, with a host name of `php-https`*`.apps.cluster.domain.example.com`*, to access the secure version of the application. Replace ***.apps.cluster.domain.example.com*** with the value of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

   From **workstation** use Firefox to access `https://php-https`*`.apps.cluster.domain.example.com`*. Accept the signed certificate and confirm the availability of the application.

8. **Optional step**: from **workstation**, use **curl** to access the HTTPS version of the application.

   Pass the CA certificate to the command so that Curl accepts the connection.

9. Return to your home folder as the **lab network-review finish** command will delete the **network-review** folder.

```
[student@workstation network-review]$ cd
```

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

This concludes the lab.

Solution

Configuring OpenShift Networking Components

In this lab, you will configure a TLS pass-through route for your application.

Outcomes

You should be able to:

- Deploy an application and configure an insecure route.
- Generate a TLS certificate for an application.
- Configure a pass-through termination route for an application with a TLS certificate.

Before You Begin

To perform this exercise, ensure that you have access to:

- A running OpenShift cluster.
- The OpenShift CLI (**/usr/local/bin/oc**).

On the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates the self-signed certificate authority (CA) that you use in this lab.

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

In this review, you deploy the PHP application named **PHP-SSL**, which prints some information about the system. The application is available in two flavors: one that runs with an unencrypted network that listens on port 8080, and one that uses a TLS certificate to encrypt the network traffic, which listens on port 8443.

The container image for this review is accessible at `quay.io/redhattraining/php-ssl`. It has two tags: **v1.0** for the insecure version of the application, and **v1.1** for the secure version.

1. As the OpenShift **developer** user, create the **network-review** project. Credentials for accessing the cluster are available at **/usr/local/etc/ocp4.config**

    1.1. From **workstation**, open a terminal and source the classroom configuration file that is accessible at **/usr/local/etc/ocp4.config**.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

    1.2. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.3. Create the **network-review** project.

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

2. As the **developer** user, deploy the first insecure version of the **PHP-SSL** application in the **network-review** OpenShift project. Use the resources file available at **~/DO280/labs/network-review/php-http.yaml**.

Before deploying the application, make the necessary changes to the file, namely, the location of the container image and the port on which it listens.

After creating the application, wait a few moments to ensure that one pod is running.

2.1. To deploy the first version of the application, go to **~/DO280/labs/network-review/** and list the files. Ensure that **php-http.yaml** is present.

```
[student@workstation ~]$ cd ~/DO280/labs/network-review/
[student@workstation network-review]$ ls
certs  php-https.yaml  php-http.yaml
```

2.2. Use your favorite text editor to update the **php-http.yaml** file as follows:

- Locate the **image** entry. Set it to use the container image accessible at `quay.io/redhattraining/php-ssl:v1.0`.

```
...output omitted...
      cpu: '0.5'
  image: 'quay.io/redhattraining/php-ssl:v1.0'
  name: php-http
...output omitted...
```

- Locate the **containerPort** entry. Set it to **8080**, which corresponds to the insecure endpoint.

```
...output omitted...
  ports:
  - containerPort: 8080
    name: php-http
...output omitted...
```

After making your changes, save and exit the file.

2.3. Use **oc create** to deploy the application. This creates a deployment and a service.

```
[student@workstation network-review]$ oc create -f php-http.yaml
deployment.apps/php-http created
service/php-http created
```

2.4. Wait a few moments, and then run **oc get pods** to ensure that there is a pod that is ready and running.

```
[student@workstation network-review]$ oc get pods
NAME                         READY   STATUS    RESTARTS   AGE
php-http-6cb58c847b-7qsbd    1/1     Running   0          8m11s
```

3. Create a route named **php-http**, with a host name of `php-http`*`.apps.cluster.domain.example.com`*, to access the application. Replace ***.apps.cluster.domain.example.com*** with the value of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

From **workstation**, use Firefox to access `http://php-http`*`.apps.cluster.domain.example.com`*. Confirm the availability of the application before proceeding to the next step.

3.1. Run **oc expose** to create a route for accessing the application. Give the route a host name of **php-http*.apps.cluster.domain.example.com***.

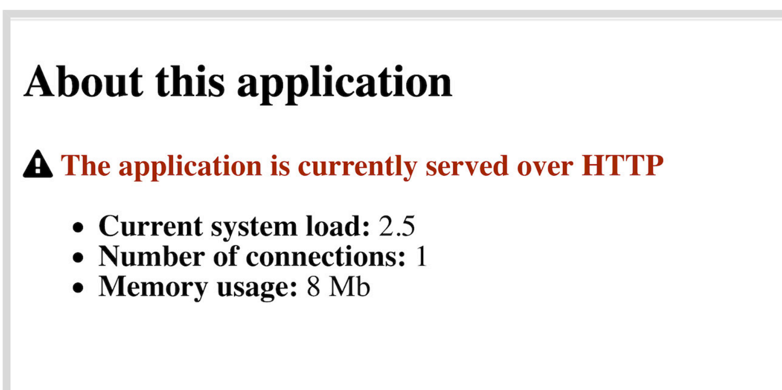
```
[student@workstation ~]$ oc expose svc php-http \
>    --hostname php-http.${RHT_OCP4_WILDCARD_DOMAIN}
route.route.openshift.io/php-http exposed
```

3.2. Retrieve the name of the route and copy it to the clipboard.

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT                               ... SERVICES   ...
php-http  php-http.apps.cluster.domain.example.com    php-http   ...
```

3.3. From **workstation**, open Firefox and access `http://php-http.apps`*`.cluster.domain.example.com`*. Replace the domain with the content of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

Confirm that you can see the application.



4. Generate and sign a TLS certificate for the encrypted version of the application.

Create a certificate signing request (CSR) for the **php-https*.apps.cluster.domain.example.com*** domain. Save the CSR to **~/DO280/labs/network-review/certs/training.csr**.

Use the CSR to generate a certificate and save it to **~/DO280/labs/network-review/certs/training.crt**. To generate the certificate, pass as arguments the CA accessible at **~/DO280/labs/network-review/certs/training-CA.pem** and the CSR.

You can use the **~/DO280/labs/network-review/certs/openssl-commands.txt** text file for help. This file contains the commands for generating the certificate signing request and the certificate. Make sure to replace the values in the file before copying and running the OpenSSL commands.

4.1. Go to the **~/DO280/labs/network-review/certs** directory and list the files.

```
[student@workstation network-review]$ cd certs
[student@workstation certs]$ ls -al
total 24
drwxr-xr-x. 2 student student  142 Dec 16 14:51 .
drwxr-xr-x. 3 student student   62 Dec 16 15:46 ..
-rw-r--r--. 1 student student  604 Dec 13 13:59 openssl-commands.txt
-rw-r--r--. 1 student student   33 Dec 16 14:51 passphrase.txt
-rw-------. 1 student student 1743 Dec 16 14:51 training-CA.key
-rw-r--r--. 1 student student 1383 Dec 16 14:51 training-CA.pem
-rw-r--r--. 1 student student  406 Dec 16 14:51 training.ext
-rw-------. 1 student student 1675 Dec 16 14:51 training.key
```

4.2. Generate the certificate signing request (CSR) for **php-https.*cluster.domain.example.com***. Make sure to type the subject of the request on one line. Alternatively, remove the **-subj** option and its content. The command prompts you for the values; make sure to indicate a common name (**CN**) of **php-https.${RHT_OCP4_WILDCARD_DOMAIN}**.

Note

Make sure there is no space after the trailing slash of the organization (**Red Hat**) and the common name (**CN**).

```
[student@workstation certs]$ openssl req -new \
>    -subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\
> CN=php-https.${RHT_OCP4_WILDCARD_DOMAIN}" \
>    -key training.key -out training.csr
```

Note

The command does not generate any output.

4.3. Alternatively, open the **openssl-commands.txt** text file. Copy and paste the first **openssl** command to your terminal. Replace the common name with **php-https.${RHT_OCP4_WILDCARD_DOMAIN}**, and the output file with **training.csr**.

4.4. Generate the signed certificate. Notice the usage of the **-CA** and **-CAkey** options for signing the certificate with the CA.

```
[student@workstation certs]$ openssl x509 -req -in training.csr \
>   -passin file:passphrase.txt \
>   -CA training-CA.pem -CAkey training-CA.key -CAcreateserial \
>   -out training.crt -days 3650 -sha256 -extfile training.ext
Signature ok
subject=/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/
CN=php-https.apps.cluster.domain.example.com
Getting CA Private Key
```

4.5. Alternatively, copy and paste the second **openssl** command in **openssl-commands.txt** file to your terminal. Replace the CSR file with **training.csr**, the CA with **training-CA.pem**, and the output certificate with **training.crt**.

```
...output omitted...
openssl x509 -req -in training.csr \
  -CA training-CA.pem -CAkey training-CA.key -CAcreateserial \
  -passin file:passphrase.txt \
  -out training.crt -days 3650 -sha256 -extfile training.ext
```

4.6. Ensure that the newly created certificate and the key are present in the current directory.

```
[student@workstation certs]$ ls -alt
total 36
-rw-rw-r--. 1 student student 1497 Dec 16 16:03 training.crt
drwxr-xr-x. 2 student student  205 Dec 16 16:03 .
-rw-rw-r--. 1 student student   41 Dec 16 16:03 training-CA.srl
-rw-rw-r--. 1 student student 1045 Dec 16 16:01 training.csr
-rw-r--r--. 1 student student  565 Dec 16 15:58 openssl-commands.txt
drwxr-xr-x. 3 student student   62 Dec 16 15:46 ..
-rw-------. 1 student student 1675 Dec 16 14:51 training.key
-rw-r--r--. 1 student student 1383 Dec 16 14:51 training-CA.pem
-rw-------. 1 student student 1743 Dec 16 14:51 training-CA.key
-rw-r--r--. 1 student student  406 Dec 16 14:51 training.ext
-rw-r--r--. 1 student student   33 Dec 16 14:51 passphrase.txt
```

4.7. Return to the **network-review** directory. This is important as the next step involves the creation of a route using the signed certificate.

```
[student@workstation certs]$ cd ~/DO280/labs/network-review
```

5. As the **developer** user, create an OpenShift TLS secret named **php-certs** in the **network-review** project. Use **~/DO280/labs/network-review/certs/training.crt** for the certificate and **~/DO280/labs/network-review/certs/training.key** for the key.

5.1. Use the **oc create secret** command to create the **php-certs** TLS secret. Pass the **training.csr** file as the certificate, and **training.key** as the key.

```
[student@workstation network-ingress]$ oc create secret tls php-certs \
>   --cert=certs/training.crt \
>   --key=certs/training.key
secret/php-certs created
```

5.2. Retrieve the list of secrets to make sure that it is present.

```
[student@workstation network-ingress]$ oc get secrets
NAME        TYPE                    DATA   AGE
...output omitted...
php-certs   kubernetes.io/tls       2      93s
```

6. Use the resources file, available at **~/DO280/labs/network-review/php-https.yaml**, to deploy the second, secure version of the **PHP-SSL** application. Deploy the application in the **network-review** OpenShift project.

Before deploying the application, make the necessary changes to the resources file, namely:

- The location of the container.
- The port the application listens on.
- The name of the secret to mount as a volume.

6.1. To deploy the second version of the application, ensure that the **~/DO280/labs/network-review/php-https.yaml/** resources file is present.

```
[student@workstation network-review]$ ls
certs  php-https.yaml php-http.yaml
```

6.2. Use your favorite text editor to update the **php-https.yaml** file as follows:

- Locate the **image** entry. Set it to use the container image accessible at quay.io/redhattraining/php-ssl:v1.1.

```
...output omitted...
      cpu: '0.5'
  image: 'quay.io/redhattraining/php-ssl:v1.1'
  name: php-https
...output omitted...
```

- Locate the **containerPort** entry. Set it to **8443**, which corresponds to the secure endpoint.

```
...output omitted...
  name: php-https
  ports:
  - containerPort: 8443
    name: php-https
...output omitted...
```

- Locate the **secretName** entry. Set it to **php-certs**, which corresponds to the name of the secret that you created in a previous step.

```
...output omitted...
volumes:
- name: tls-certs
  secret:
    secretName: php-certs
...output omitted...
```

After making your changes, save and exit the file.

6.3. Use **oc create** to deploy the secure application. This creates a deployment and a service.

```
[student@workstation network-review]$ oc create -f php-https.yaml
deployment.apps/php-https created
service/php-https created
```

6.4. Wait a few moments, and then run **oc get pods** to ensure that there is a pod named **php-https-*** that is ready and running.

```
[student@workstation network-review]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
php-http-6cb58c847b-7qsbd   1/1     Running   0          8m11s
php-https-84498cd794-hvf7w  1/1     Running   0          26s
```

7. Create a secure pass-through route named **php-https**, with a host name of php-https.*apps.cluster.domain.example.com*, to access the secure version of the application. Replace ***.apps.cluster.domain.example.com*** with the value of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

From **workstation** use Firefox to access https://php-https.*apps.cluster.domain.example.com*. Accept the signed certificate and confirm the availability of the application.

7.1. Run **oc expose** to create a pass-through route for accessing the application. Give the route a host name of **php-https.*apps.cluster.domain.example.com***. Use the **port** option to indicate the secure port 8443.

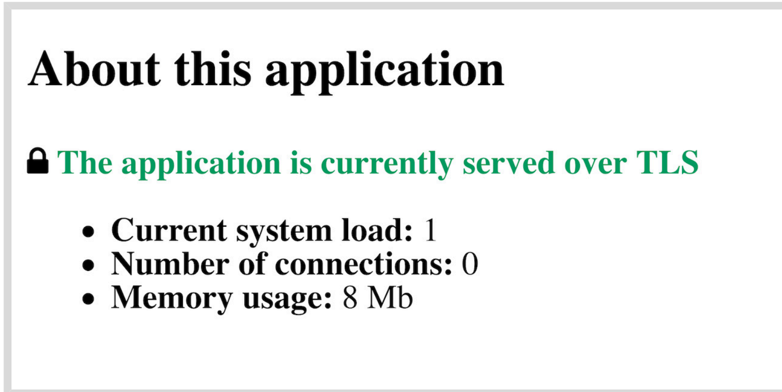
```
[student@workstation network-review]$ oc create route passthrough php-https \
>    --service php-https --port 8443 \
>    --hostname php-https.${RHT_OCP4_WILDCARD_DOMAIN}
route.route.openshift.io/php-https created
```

7.2. Retrieve the name of the route and copy it to the clipboard.

```
[student@workstation network-review]$ oc get routes
NAME        HOST/PORT                                 TERMINATION  ...
php-http    php-http.apps. cluster.domain.example.com              ...
php-https   php-https.apps.cluster.domain.example.com  passthrough  ...
```

7.3. From **workstation**, open Firefox and access https://php-https.apps.*cluster.domain.example.com*. Replace the domain with the content of the **RHT_OCP4_WILDCARD_DOMAIN** variable.

Accept the signed certificate and confirm that you can see secure version of the application.



8. **Optional step**: from **`workstation`**, use **`curl`** to access the HTTPS version of the application.

   Pass the CA certificate to the command so that Curl accepts the connection.

   Use the **`--cacert`** option to pass the CA certificate to **`curl`**.

```
[student@workstation network-ingress]$ curl -v \
>    --cacert certs/training-CA.pem \
>    https://php-https.${RHT_OCP4_WILDCARD_DOMAIN}
...output omitted...
* TLSv1.3 (OUT), TLS handshake, Finished (20):
...output omitted...
* Server certificate:
*  subject: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; \
CN=php-https.apps.cluster.domain.example.com
...output omitted...
  The application is currently served over TLS        </span></strong>
...output omitted...
```

9. Return to your home folder as the **`lab network-review finish`** command will delete the **`network-review`** folder.

```
[student@workstation network-review]$ cd
```

This concludes the lab.

Evaluation

As the **`student`** user on the **`workstation`** machine, use the **`lab`** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **`student`** user on the **`workstation`** machine, use the **`lab`** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

In this chapter, you learned:

- OpenShift implements a software-defined networking (SDN) to manage the network infrastructure of the cluster. SDN decouples the software that handles the traffic from the underlying mechanisms that route the traffic.
- Kubernetes provides *services* that allow the logical grouping of pods under a common access route. Services act as load balancers in front of one or more pods.
- Services use selectors (labels) that indicate which pods are running for the service.
- There are two kind of routes: secure, and insecure. Secure routes encrypt the traffic using TLS certificates, and insecure routes forward traffic over an unencrypted connection.

  Secure routes support three modes: edge, pass-through, and re-encryption.

Chapter 6

Controlling Pod Scheduling

Goal	Control the nodes on which a pod runs.
Objectives	• Describe pod scheduling algorithms, the methods used to influence scheduling, and apply these methods. • Limit the resources consumed by an application. • Control the number of replicas of a pod.
Sections	• Controlling Pod Scheduling Behavior (and Guided Exercise) • Limiting Resource Usage (and Guided Exercise) • Scaling an Application (and Guided Exercise)
Lab	Controlling Pod Scheduling

Controlling Pod Scheduling Behavior

Objectives

After completing this section, you should be able to:

- Describe pod scheduling algorithms.
- Describe the methods used to influence scheduling.

Introducing 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 4.2 supports the common data center concepts of zones and regions by using node labels, *affinity* rules, and *anti-affinity* rules.

The OpenShift pod scheduler algorithm follows a three step process:

1. Filtering nodes.

   The scheduler filters the list of running nodes by evaluating each node against a set of *predicates*, such as the availability of host ports, or whether a pod can be scheduled to a node experiencing either disk or memory pressure.

   Additionally, a pod can define a node selector that matches 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.

   Another filtering check evaluates if the list of nodes have any *taints*, and if so whether the pod has an associated *toleration* that can accept the taint. If a pod cannot accept the taint of a node, then the node is not eligible. By default, master nodes include the taint **`node-role.kubernetes.io/master:NoSchedule`**. A pod that does not have a matching toleration for this taint will not be scheduled to a master node.

   The end result of the filtering step is typically a shorter list of node candidates that are eligible to run the pod. In some cases, none of the nodes are filtered out, which means the pod could run on any of the nodes. In other cases, all of the nodes are filtered out, which means the pod cannot be scheduled until a node with the desired prerequisites becomes available.

   A full list of predicates and their descriptions can be found in the references section.

2. Prioritizing the filtered list of nodes.

   The list of candidate nodes is evaluated using multiple priority criteria that 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 synchronize with each other.

A common use for **anti-affinity** rules is to schedule related pods that are not too close to each other, for high availability. An 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 these 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 in a round-robin fashion.

The scheduler is flexible and can be customized for advanced scheduling situations. Additionally, although this course will focus on pod placement using node labels and node selectors, pods can also be placed using pod affinity and anti-affinity rules, as well as node affinity and anti-affinity rules. Customizing the scheduler and covering these alternative pod placement scenarios is outside the scope of this course.

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 switch, a storage array, or a uninterruptible power supply (UPS).

As an example of regions and zones, Amazon Web Services (AWS) has a region in northern Virginia (**us-east-1**) with 6 availability zones, and another region in Ohio (**us-east-2**) with 3 availability zones. Each of the AWS availability zones can contain multiple data centers potentially consisting of hundreds of thousands of servers.

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 deployment (or 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.

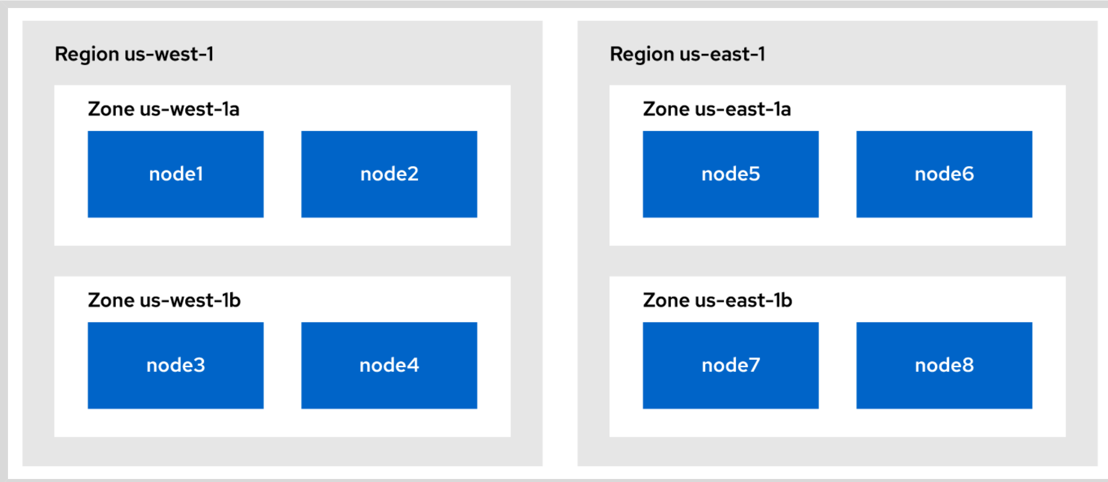


Figure 6.1: Sample cluster topology using regions and zones

Labeling Nodes

As an OpenShift cluster administrator, you can add additional labels to your nodes. For example, you might label nodes with the **env** label using the values of **dev**, **qa**, or **prod** with the intent that development, quality assurance, and production workloads will be deployed to a specific subset of nodes. The labels you choose are arbitrary, but you must publish the labels and their associated values to your developers so that they can configure their applications appropriately.

Use the **oc label** command as a cluster administrator to immediately add, update, or remove a node label. For example, use the following command to label a node with **env=dev**:

```
[user@demo ~]$ oc label node node1.us-west-1.compute.internal env=dev
```

Use the **--overwrite** option to change an existing label.

```
[user@demo ~]$ oc label node node1.us-west-1.compute.internal env=prod --overwrite
```

Remove a label by specifying the label name followed by a hyphen, such as **env-**.

```
[user@demo ~]$ oc label node node1.us-west-1.compute.internal env-
```

Important

Both labels and their values are case-sensitive. An application node selector must match the case of the actual label and the value applied to the node.

Use the **--show-labels** option with the **oc get nodes** command to see the case-sensitive labels assigned to a node.

```
[user@demo ~]$ oc get node node2.us-west-1.compute.internal --show-labels
NAME                               ...  ROLES    ...  LABELS
node2.us-west-1.compute.internal   ...  worker   ...  beta.kubernetes.io/
arch=amd64,beta.kubernetes.io/instance-type=m4.xlarge,beta.kubernetes.io/
os=linux,tier=gold,failure-domain.beta.kubernetes.io/region=us-
west-1,failure-domain.beta.kubernetes.io/zone=us-west-1c,kubernetes.io/
arch=amd64,kubernetes.io/hostname=node2,kubernetes.io/os=linux,node-
role.kubernetes.io/worker=,node.openshift.io/os_id=rhcos
```

Notice that a node might have several 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:

```
[user@demo ~]$ oc get node -L failure-domain.beta.kubernetes.io/region
NAME                                        ...   ROLES    ...   REGION
ip-10-0-131-214.us-west-1.compute.internal  ...   master   ...   us-west-1
ip-10-0-139-250.us-west-1.compute.internal  ...   worker   ...   us-west-1
ip-10-0-141-144.us-west-1.compute.internal  ...   master   ...   us-west-1
ip-10-0-152-57.us-west-1.compute.internal   ...   master   ...   us-west-1
ip-10-0-154-226.us-west-1.compute.internal  ...   worker   ...   us-west-1
```

Multiple **-L** options in the same **oc get** command are supported. For example:

```
[user@demo ~]$ oc get node -L failure-domain.beta.kubernetes.io/region \
>     -L failure-domain.beta.kubernetes.io/zone -L env
NAME                                        ...   REGION      ZONE         ENV
ip-10-0-131-214.us-west-1.compute.internal  ...   us-west-1   us-west-1b
ip-10-0-139-250.us-west-1.compute.internal  ...   us-west-1   us-west-1b   dev
ip-10-0-141-144.us-west-1.compute.internal  ...   us-west-1   us-west-1b
ip-10-0-152-57.us-west-1.compute.internal   ...   us-west-1   us-west-1c
ip-10-0-154-226.us-west-1.compute.internal  ...   us-west-1   us-west-1c
```

Labeling Machine Sets

Although node labels are persistent, if your OpenShift cluster contains machine sets (created if your cluster was installed using the full stack automation method), then you should also add labels to the machine set configuration. This ensures that new machines (and the nodes generated from them) will also contain the desired labels.

You can identify the relationship between machines and nodes by listing machines in the **openshift-machine-api** namespace and including the **-o wide** option.

```
[user@demo ~]$ oc get machines -n openshift-machine-api -o wide
NAME                                  ... NODE
...output omitted...
ocp-qz7hf-worker-us-west-1b-rvx6w ... ip-10-0-139-250.us-west-1.compute.internal
ocp-qz7hf-worker-us-west-1c-v4n4n ... ip-10-0-154-226.us-west-1.compute.internal
```

Machines used for **worker** nodes should come from a machine set. The name of a machine contains the name of the machine set from which it was generated.

```
[user@demo ~]$ oc get machineset -n openshift-machine-api
NAME                          DESIRED   CURRENT   READY   AVAILABLE   ...
ocp-qz7hf-worker-us-west-1b   1         1         1       1           ...
ocp-qz7hf-worker-us-west-1c   1         1         1       1           ...
```

Edit a machine set so that new machines generated from it will have the desired label or labels. Modifying a machine set will not apply changes to existing machines or nodes.

```
[user@demo ~]$ oc edit machineset ocp-qz7hf-worker-us-west-1b \
>    -n openshift-machine-api
```

The highlighted lines below show where to add a label within a machine set.

```
...output omitted...
    spec:
      metadata:
        creationTimestamp: null
        labels:
          env: dev
      providerSpec:
...output omitted...
```

Controlling Pod Placement

Many infrastructure-related pods in an OpenShift cluster are configured to run on master nodes. Examples include pods for the DNS operator, the OAuth operator, and the OpenShift API server. In some cases, this is accomplished by using the node selector **node-role.kubernetes.io/master: ''** in the configuration of a daemon set or a replica set.

Similarly, some user 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. Use node labels and node selectors to implement these kinds of scenarios.

A *node selector* is part of an individual pod definition. Define a node selector in either a deployment or a deployment configuration resource, so that any new pod generated from that resource will have the desired node selector. If your deployment or deployment configuration resource is under version control, then modify the resource file and apply the changes using the **oc apply -f** command.

Alternatively, a node selector can be added or modified using either the **oc edit** command or the **oc patch** command. For example, to configure the **myapp** deployment so that its pods only run on nodes that have the **env=qa** label, use the **oc edit** command:

```
[user@demo ~]$ oc edit deployment/myapp
```

```
...output omitted...
spec:
...output omitted...
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: myapp
    spec:
      nodeSelector:
        env: dev
      containers:
      - image: quay.io/redhattraining/scaling:v1.0
...output omitted...
```

The following **oc patch** command accomplishes the same thing:

```
[user@demo ~]$ oc patch deployment/myapp --patch \
>    '{"spec":{"template":{"spec":{"nodeSelector":{"env":"dev"}}}}}'
```

Whether using the **oc edit** command or the **oc patch** command, the change triggers a new deployment and the new pods are scheduled according to the node selector.

Configuring a Node Selector for a Project

If the cluster administrator does not want developers controlling the node selector for their pods, then a default node selector should be configured in the project resource. A cluster administrator can either define a node selector when a project is created, or can add or update a node selector after a project is created. Use the **oc adm new-project** command to add the node selector at project creation. For example, the following command creates a new project named **demo**, where all pods will be deployed to nodes that have the label of **tier=1**.

```
[user@demo ~]$ oc adm new-project demo --node-selector "tier=1"
```

To configure a default node selector for an existing project, add an *annotation* to the namespace resource with the **openshift.io/node-selector** key. The **oc annotate** command can add, modify, or remove a node selector annotation:

```
[user@demo ~]$ oc annotate namespace demo \
>    openshift.io/node-selector="tier=2" --overwrite
```

Scaling the Number of Pod Replicas

Although most deployment and deployment configuration resources start with creating a single pod, the number of replicas (or copies) of a pod is frequently increased. This is accomplished by scaling the deployment or deployment configuration. Multiple methods for scaling will be covered later, but one method uses the **oc scale** command. For example, the number of pods in the **myapp** deployment can be scaled to three using the following command:

```
[user@demo ~]$ oc scale --replicas 3 deployment/myapp
```

References

For more information, refer to the *Controlling pod placement onto nodes (scheduling)* chapter in the *Red Hat OpenShift Container Platform 4.2 Nodes* guide at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling

Amazon Web Services Regions and Availability Zones
https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

Guided Exercise

Controlling Pod Scheduling Behavior

In this exercise, you will configure an application to run on a subset of the cluster worker nodes.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Add a new label to a node.
- Deploy pods to nodes that match a specified label.
- Remove a label from a node.
- Troubleshoot when pods fail to deploy to a node.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab schedule-pods start
```

1. As the **developer** user, create a new project named **schedule-pods**.

   1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. Create a new project named **schedule-pods**.

```
[student@workstation ~]$ oc new-project schedule-pods
Now using project "schedule-pods" on server
 "https://api.cluster.domain.example.com:6443".
...output omitted...
```

2. Deploy and scale a test application.

2.1. Create a new application named **hello** using the container located at **quay.io/redhattraining/hello-world-nginx:v1.0**.

```
[student@workstation ~]$ oc create deployment hello \
>    --image quay.io/redhattraining/hello-world-nginx:v1.0
deployment.apps/hello created
```

2.2. Create a service for the application.

```
[student@workstation ~]$ oc expose deployment/hello --port 80 --target-port 8080
service/hello exposed
```

2.3. Make the service accessible by creating a route to it.

```
[student@workstation ~]$ oc expose svc/hello
route.route.openshift.io/hello exposed
```

2.4. Manually scale the application so there are four running pods.

```
[student@workstation ~]$ oc scale --replicas 4 deployment/hello
deployment.extensions/hello scaled
```

2.5. Verify that the four running pods are distributed between the two worker nodes.

```
[student@workstation ~]$ oc get pods -o wide
NAME                    ...  NODE                                        ...
hello-cb7797cdd-dqk8g   ...  ip-10-0-140-20.us-east-2.compute.internal   ...
hello-cb7797cdd-hq4xj   ...  ip-10-0-140-20.us-east-2.compute.internal   ...
hello-cb7797cdd-mbvdw   ...  ip-10-0-152-96.us-east-2.compute.internal   ...
hello-cb7797cdd-zbnnn   ...  ip-10-0-152-96.us-east-2.compute.internal   ...
```

Note

Depending on the existing load on each worker node, your output may be different. Although the scheduler will attempt to distribute the pods, the distribution may not be even.

3. Prepare the worker nodes so that application workloads can be distributed to either development or production nodes by assigning the **env** label. Assign the **env=dev** label to the first worker node and the **env=prod** label to the second worker node.

3.1. Log in to your OpenShift cluster as the **admin** user. A regular user does not have permission to view information about nodes, machines, or machine sets, nor can a regular user apply labels.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

3.2. The following two commands in this step are for information only. To keep this exercise simple, you will only label nodes. A more realistic scenario would label machine sets so that new worker machines (and the nodes associated with them) would have the desired label.

Display all of the machines in your environment. The output includes the AWS region and zone, as well as the specific node being used by each machine.

```
[student@workstation ~]$ oc get machines -n openshift-machine-api -o wide
NAME                                          ...  REGION     ZONE        AGE
 NODE
do280-33cc33cc-ntpkj-master-0                 ...  us-east-2  us-east-2a  63d
 ip-10-0-134-251.us-east-2.compute.internal
do280-33cc33cc-ntpkj-master-1                 ...  us-east-2  us-east-2b  63d
 ip-10-0-155-221.us-east-2.compute.internal
do280-33cc33cc-ntpkj-master-2                 ...  us-east-2  us-east-2c  63d
 ip-10-0-171-178.us-east-2.compute.internal
do280-33cc33cc-ntpkj-worker-us-east-2a-p2slq  ...  us-east-2  us-east-2a  63d
 ip-10-0-140-20.us-east-2.compute.internal
do280-33cc33cc-ntpkj-worker-us-east-2b-pntdb  ...  us-east-2  us-east-2b  41d
 ip-10-0-152-96.us-east-2.compute.internal
```

Display the machine sets used to generate your workers.

```
[student@workstation ~]$ oc get machinesets -n openshift-machine-api
NAME                                    DESIRED  CURRENT  READY  AVAILABLE  ...
do280-33cc33cc-ntpkj-worker-us-east-2a  1        1        1      1          ...
do280-33cc33cc-ntpkj-worker-us-east-2b  1        1        1      1          ...
```

3.3. Now that you have seen the correlation between machines and nodes (and the machine sets used to create the machines), verify that none of the worker nodes use the **env** label.

```
[student@workstation ~]$ oc get nodes -L env -l node-role.kubernetes.io/worker
NAME                                       STATUS   ROLES    ...   ENV
ip-10-0-140-20.us-east-2.compute.internal  Ready    worker   ...
ip-10-0-152-96.us-east-2.compute.internal  Ready    worker   ...
```

3.4. Add the **env=dev** label to the first worker node to indicate that it is a development node.

```
[student@workstation ~]$ oc label node \
>    ip-10-0-140-20.us-east-2.compute.internal \
>    env=dev
node/ip-10-0-140-20.us-east-2.compute.internal labeled
```

3.5. Add the **env=prod** label to the second worker node to indicate that it is a production node.

```
[student@workstation ~]$ oc label node \
>    ip-10-0-152-96.us-east-2.compute.internal \
>    env=prod
node/ip-10-0-152-96.us-east-2.compute.internal labeled
```

3.6. Verify that the worker nodes have the correct **env** label set. Make note of the node that has the **env=dev** label, as you will check later to see if the application pods have been deployed to that node.

```
[student@workstation ~]$ oc get nodes -L env -l node-role.kubernetes.io/worker
NAME                                         STATUS   ROLES    ...   ENV
ip-10-0-140-20.us-east-2.compute.internal    Ready    worker   ...   dev
ip-10-0-152-96.us-east-2.compute.internal    Ready    worker   ...   prod
```

4. Switch back to the **developer** user and modify the **hello** application so that it is deployed to the development node. After verifying this change, delete the **schedule-pods** project.

4.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
Using project "schedule-pods".
```

4.2. Modify the **deployment** resource for the **hello** application to select a development node. Make sure to add the node selector the **spec** group in the **template** section.

```
[student@workstation ~]$ oc edit deployment/hello
```

Add the highlighted lines below to the deployment resource, indenting as shown.

```
...output omitted...
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      nodeSelector:
        env: dev
      restartPolicy: Always
...output omitted...
```

The following output from **oc edit** is displayed after you save your changes.

```
deployment.extensions/hello edited
```

4.3. Verify that the application pods are deployed to the node with the **env=dev** label. Although it make take a little time to redeploy, the application pods must be deployed to the same worker node that you previously identified.

```
[student@workstation ~]$ oc get pods -o wide
NAME                     ...  NODE                                        ...
hello-5564c6bd8f-5b22n   ...  ip-10-0-140-20.us-east-2.compute.internal   ...
hello-5564c6bd8f-b9b2w   ...  ip-10-0-140-20.us-east-2.compute.internal   ...
hello-5564c6bd8f-g9sct   ...  ip-10-0-140-20.us-east-2.compute.internal   ...
hello-5564c6bd8f-gmjls   ...  ip-10-0-140-20.us-east-2.compute.internal   ...
```

4.4. Remove the **schedule-pods** project.

```
[student@workstation ~]$ oc delete project schedule-pods
project.project.openshift.io "schedule-pods" deleted
```

5. Finish cleaning up this portion of the exercise by removing the **env** label from the worker nodes.

5.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

5.2. Remove the **env** label from all worker nodes.

```
[student@workstation ~]$ oc label node -l node-role.kubernetes.io/worker env-
node/ip-10-0-140-20.us-east-2.compute.internal labeled
node/ip-10-0-152-96.us-east-2.compute.internal labeled
```

6. The **schedule-pods-ts** project contains an application that runs only on nodes that are labeled as **client=ACME**. In the following example, the application pod is pending and you must diagnose the problem using the following steps:

6.1. Log in to your OpenShift cluster as the **developer** user and ensure that you are using the **schedule-pods-ts** project.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
Using project "schedule-pods-ts".
```

If the output above does not show that you are using the **schedule-pods-ts** project, switch to it.

```
[student@workstation ~]$ oc project schedule-pods-ts
Now using project "schedule-pods-ts" on server ...
```

6.2. Verify that the application pod has a status of **Pending**.

```
[student@workstation ~]$ oc get pods
NAME                       READY   STATUS    RESTARTS   AGE
hello-ts-5dbff9f44-w6csj   0/1     Pending   0          7s
```

6.3. Because a pod with a status of pending does not have any logs, check details of the pod using the **oc describe pod** command to see if describing the pod provides any useful information.

```
[student@workstation ~]$ oc describe pod hello-ts-5dbff9f44-8h7c7
...output omitted...
QoS Class:       BestEffort
```

```
Node-Selectors:  client=acme
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason            ...              Message
  ----     ------            ...              -------
  Warning  FailedScheduling  ...  0/5 nodes are available: 5 node(s) didn't match
 node selector.
```

Based on this information, the pod should be scheduled to a node with the label **client=acme**, but none of the five nodes has this label. The information provided indicates that at least one worker node has the label **client=ACME**. You have found the problem. The application must be modified so that it uses the correct node selector.

6.4. Edit the deployment resource for the application to use the correct node selector.

```
[student@workstation ~]$ oc edit deployment/hello-ts
```

Change **acme** to **ACME** as shown below.

```
...output omitted...
      dnsPolicy: ClusterFirst
      nodeSelector:
        client: ACME
      restartPolicy: Always
...output omitted...
```

The following output from **oc edit** is displayed after you save your changes.

```
deployment.extensions/hello-ts edited
```

6.5. Verify that the a new application pod is deployed and has a status of **Running**.

```
[student@workstation ~]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
hello-ts-69769f64b4-wwhpc   1/1     Running   0          11s
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab schedule-pods finish
```

This concludes the lab.

Limiting Resource Usage

Objectives

After completing this section, you should be able to use limit ranges and resource quotas to limit the resources consumed by an application.

Defining Resource Requests and Limits for Pods

A pod definition can include both resource requests and resource limits:

Resource requests
: Used for scheduling, and to 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.

Resource request and resource limits should be defined for each container in either a deployment or a deployment configuration resource. If requests and limits have not been defined, then you will find a **`resources: {}`** line for each container.

Modify the **`resources: {}`** line to specify the desired requests and or limits. For example:

```
...output omitted...
    spec:
      containers:
      - image: quay.io/redhattraining/hello-world-nginx:v1.0
        name: hello-world-nginx
        resources:
          requests:
            cpu: "10m"
            memory: 20Mi
          limits:
            cpu: "80m"
            memory: 100Mi
status: {}
```

If you use the **`oc edit`** command to modify a deployment or a deployment configuration, ensure you use the correct indentation. Indentation mistakes can result in the editor refusing to save changes. To avoid indentation issues, you can use the **`oc set resources`** command to specify resource requests and limits. The following command sets the same requests and limits as the preceding example.

```
[user@demo ~]$ oc set resources deployment hello-world-nginx \
>    --requests cpu=10m,memory=20Mi --limits cpu=80m,memory=100Mi
```

If a resource quota applies to a resource request, then the pod should define a resource request. If a resource quota applies to a resource limit, then the pod should also define a resource limit. Red Hat recommends defining resource requests and limits even if quotas are not being used.

Viewing Requests, Limits, and Actual Usage

Using the OpenShift command-line interface, cluster administrators can view compute usage information on individual nodes. The **oc describe node** command displays detailed information about a node, including information about the pods running on the node. For each pod, it shows CPU requests and limits, as well as memory requests and limits. If a request or limit has not been specified, then the pod will show a 0 for that column. A summary of all resource requests and limits is also displayed.

```
[user@demo ~]$ oc describe node node1.us-west-1.compute.internal
Name:                   node1.us-west-1.compute.internal
Roles:                  worker
Labels:                 beta.kubernetes.io/arch=amd64
                        beta.kubernetes.io/instance-type=m4.xlarge
                        beta.kubernetes.io/os=linux
...output omitted...
Non-terminated Pods:                          (20 in total)
...   Name                  CPU Requests  ...   Memory Requests   Memory Limits  AGE
...   ----                  ------------  ...   ---------------   -------------  ---
...   tuned-vdwt4           10m (0%)      ...   50Mi (0%)         0 (0%)         8d
...   dns-default-2rpwf     110m (3%)     ...   70Mi (0%)         512Mi (3%)     8d
...   node-ca-6xwmn         10m (0%)      ...   10Mi (0%)         0 (0%)         8d
...output omitted...
  Resource                    Requests     Limits
  --------                    --------     ------
  cpu                         600m (17%)   0 (0%)
  memory                      1506Mi (9%)  512Mi (3%)
...output omitted...
```

Note

The summary columns for **Requests** and **Limits** display the sum totals of defined requests and limits. In the case of the output above, only 1 of the 20 pods running on the node defined a memory limit, and that limit was 512Mi.

The **oc describe node** command displays requests and limits, and the **oc adm top** command shows actual usage. For example, if a pod requests 10m of CPU, then the scheduler will ensure that it places the pod on a node with available capacity. Although the pod requested 10m of CPU, it might use more or less than this value, unless it is also constrained by a CPU limit. Similarly, a pod that does not specify resource requests will still use some amount of resources. The **oc adm top nodes** command shows actual usage for one or more nodes in the cluster, and the **oc adm top pods** command shows actual usage for each pod in a project.

```
[user@demo ~]$ oc adm top nodes -l node-role.kubernetes.io/worker
NAME                               CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
node1.us-west-1.compute.internal   519m         14%    3126Mi          20%
node2.us-west-1.compute.internal   167m         4%     1178Mi          7%
```

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 improves the stability of the OpenShift control plane, by avoiding unbounded growth of the **`Etcd`** database. 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 starve other applications in a shared cluster by using all the cluster capacity.

OpenShift manages quotas for the number of resources and the use of compute resources in a cluster by using a **`ResourceQuota`** resource, or a **`quota`**. A quota 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**
> Although a single quota resource can define all of the quotas for a project, a project can contain multiple quotas. For example, one quota resource might limit compute resources, such as total CPU allowed or total memory allowed. Another quota resource might limit object counts, such as the number of pods allowed or the number of services allowed. The effect of multiple quotas is cumulative, but it is expected that two different **`ResourceQuota`** resources for the same project do not limit the use of the same type of Kubernetes or compute resource. For example, two different quotas in a project should not both attempt to limit the maximum number of pods allowed.

The following table describes some resources that a quota can restrict by their count or number.

Resource Name	Quota 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 compute resources that can be restricted by a quota.

Compute Resource Name	Quota Description
cpu (**requests.cpu**)	Total CPU use across all containers
memory (**requests.memory**)	Total memory use across all containers
storage (**requests.storage**)	Total storage requests by containers across all persistent volume claims

Quota attributes can track either resource requests or resource limits for all pods in the project. By default, quota attributes track 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. This example specifies quotas for both the number of resources and the use of 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. One millicore is the equivalent to 1/1000 of a single CPU core.

Resource quotas can be created in the same way as any other OpenShift Container Platform resource; that is, by passing a YAML or JSON resource definition file to the **oc create** command:

```
[user@demo ~]$ oc create --save-config -f dev-quota.yml
```

Another way to create a resource quota is by using the **oc create quota** command, for example:

```
[user@demo ~]$ oc create quota dev-quota --hard services=10,cpu=1300,memory=1.5Gi
```

Use the **oc get resourcequota** command to list available quotas, and use the **oc describe resourcequota** command to view usage statistics related to any hard limits defined in the quota, for example:

```
[user@demo ~]$ oc get resourcequota
NAME            CREATED AT
compute-quota   2019-11-26T21:51:43Z
count-quota     2019-11-26T21:51:09Z
```

Without arguments, the **oc describe quota** command displays the cumulative limits set for all **ResourceQuota** resources in the project.

```
[user@demo ~]$ oc describe quota
Name:       compute-quota
Namespace:  schedule-demo
Resource    Used    Hard
--------    ----    ----
cpu         500m    10
memory      300Mi   1Gi


Name:                   count-quota
Namespace:              schedule-demo
Resource                Used  Hard
--------                ----  ----
pods                    1     3
replicationcontrollers  1     5
services                1     2
```

An active quota can be deleted by name using the **oc delete** command:

```
[user@demo ~]$ oc delete resourcequota 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.

Quotas are applied to new resources, but they do not affect existing resources. For example, if you create a quota to limit a project to 15 pods, but there are already 20 pods running, then the quota will not remove the additional 5 pods that exceed the quota.

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 a resource count, then the action is denied by the server and an appropriate error message is returned to the user. However, if the modification exceeds the quota for a compute resource, then 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, then OpenShift refuses to create pods that do not specify resource requests or resource limits for that compute resource. To use most templates and builders with a project restricted by quotas, the project must also contain a limit range resource 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, consider that a limit range defines valid ranges and default values for a single pod, and 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 limit range 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, then it takes the default value provided by the limit ranges for the project. If a new resource provides compute resource requests or limits that are smaller than the minimum specified by the project limit ranges, then 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 limit ranges, then the resource is not created.

The following table describes some of the compute resources that can be specified by a limit range resource.

Type	Resource Name	Description
Container	cpu	Minimum and maximum CPU allowed and default CPU set per container
	memory	Minimum and maximum memory allowed and default memory set per container
Pod	cpu	Minimum and maximum CPU allowed across all containers in a 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: "500m"
        memory: "750Mi"
      min:
        cpu: "10m"
        memory: "5Mi"
    - type: "Container"
      default:
        cpu: "100m"
        memory: "100Mi"
      max:
        cpu: "500m"
        memory: "750Mi"
      min:
        cpu: "10m"
        memory: "5Mi"
```

Users can create a limit range resource the same way as any other OpenShift resource; that is, by passing a YAML or JSON resource definition file to the **oc create** command:

```
[user@demo ~]$ oc create --save-config -f dev-limits.yml
```

Red Hat OpenShift Container Platform 4.2 does not provide an **oc create** command specifically for limit ranges like it does for resource quotas. The only alternative is to use YAML or JSON files.

Use the **oc describe limitrange** command to view the limit constraints enforced in a project.

```
[user@demo ~]$ oc describe limitrange dev-limits
Name:       dev-limits
Namespace:  schedule-demo
Type        Resource  Min  Max    Default Request  Default Limit  ...
----        --------  ---  ---    ---------------  -------------  ...
Pod         cpu       10m  500m   -                -              ...
Pod         memory    5Mi  750Mi  -                -              ...
Container   cpu       10m  500m   100m             100m           ...
Container   memory    5Mi  750Mi  100Mi            100Mi          ...
```

An active limit range can be deleted by name with the **oc delete** command:

```
[user@demo ~]$ oc delete limitrange dev-limits
```

After a limit range is created in a project, all requests to create new resources are evaluated against each limit range resource in the project. If the new resource violates the minimum or maximum constraint enumerated by any limit range, 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 limit range 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. A violation of **`LimitRange`** constraints prevents pod creation, and resulting error messages are displayed. A 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.

Cluster administrators can specify which projects are subject to cluster resource quotas in 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:

```
[user@demo ~]$ oc create clusterquota user-qa \
>     --project-annotation-selector openshift.io/requester=qa \
>     --hard pods=12,secrets=20
```

The following is an example of creating a cluster resource quota for all projects that have been assigned the **`environment=qa`** label:

```
[user@demo ~]$ oc create clusterquota env-qa \
>     --project-label-selector environment=qa \
>     --hard pods=10,services=5
```

Project users can use the **`oc describe QUOTA`** command to view cluster resource quotas that apply to the current project, if any.

Use the **`oc delete`** command to delete a cluster resource quota:

```
[user@demo ~]$ oc delete clusterquota QUOTA
```

> **Note**
> 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

For more information, refer to the *Quotas* chapter in the *Red Hat OpenShift Container Platform 4.2 Applications* guide at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/applications/index#quotas

Guided Exercise

Limiting Resource Usage

In this exercise, you will configure an application so that it does not consume all computing resources from the cluster and its worker nodes.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Configure an application to specify resource requests for CPU and memory usage.
- Modify an application to work within existing cluster restrictions.
- Create a quota to limit the total amount of CPU, memory, and configuration maps available to a project.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates the resource files that you will be using in the activity.

```
[student@workstation ~]$ lab schedule-limit start
```

1. As the **developer** user, create a new project named **schedule-limit**.

    1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

    1.2. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

    1.3. Create a new project for this guided exercise named **schedule-limit**.

```
[student@workstation ~]$ oc new-project schedule-limit
Now using project "schedule-limit" on server
   "https://api.cluster.domain.example.com:6443".
...output omitted...
```

2. Deploy a test application for this exercise that explicitly requests container resources for CPU and memory.

2.1. Create a deployment resource file and save it to **~/DO280/labs/schedule-limit/hello-limit.yaml**. Name the application **hello-limit** and use the container image located at **quay.io/redhattraining/hello-world-nginx:v1.0**.

```
[student@workstation ~]$ oc create deployment hello-limit \
>    --image quay.io/redhattraining/hello-world-nginx:v1.0 \
>    --dry-run -o yaml > ~/DO280/labs/schedule-limit/hello-limit.yaml
```

2.2. Edit **~/DO280/labs/schedule-limit/hello-limit.yaml** to replace the **resources: {}** line with the highlighted lines below. Ensure that you have proper indentation before saving the file.

```
[student@workstation ~]$ vim ~/DO280/labs/schedule-limit/hello-limit.yaml

...output omitted...
    spec:
      containers:
      - image: quay.io/redhattraining/hello-world-nginx:v1.0
        name: hello-world-nginx
        resources:
          requests:
            cpu: "3"
            memory: 20Mi
status: {}
```

2.3. Create the new application using your resource file.

```
[student@workstation ~]$ oc create --save-config \
>    -f ~/DO280/labs/schedule-limit/hello-limit.yaml
deployment.apps/hello-limit created
```

2.4. Although a new deployment was created for the application, the application pod should have a status of **Pending**.

```
[student@workstation ~]$ oc get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-limit-d86874d86b-fpmrt   0/1     Pending   0          10s
```

2.5. The pod cannot be scheduled because none of the worker nodes have sufficient CPU resources. This can be verified by viewing warning events.

```
[student@workstation ~]$ oc get events --field-selector type=Warning
LAST SEEN   TYPE      REASON             OBJECT                             MESSAGE
88s         Warning   FailedScheduling   pod/hello-limit-d86874d86b-fpmrt   0/5
 nodes are available: 5 Insufficient cpu.
```

3. Redeploy your application so that it requests fewer CPU resources.

3.1. Edit **~/DO280/labs/schedule-limit/hello-limit.yaml** to request 2 CPUs for the container. Change the **cpu: "3"** line to match the highlighted line below.

```
[student@workstation ~]$ vim ~/DO280/labs/schedule-limit/hello-limit.yaml

...output omitted...
        resources:
          requests:
            cpu: "2"
            memory: 20Mi
```

3.2. Apply the changes to your application.

```
[student@workstation ~]$ oc apply -f ~/DO280/labs/schedule-limit/hello-limit.yaml
deployment.apps/hello-limit configured
```

3.3. Verify that your application deploys successfully. You might need to run **oc get pods** multiple times until you see a running pod. The previous pod with a pending status will terminate and eventually disappear.

```
[student@workstation ~]$ oc get pods
NAME                          READY   STATUS        RESTARTS   AGE
hello-limit-d86874d86b-fpmrt  0/1     Terminating   0          2m19s
hello-limit-7c7998ff6b-ctsjp  1/1     Running       0          6s
```

4. Attempt to scale your application from 1 pod to 3 pods. After verifying that this change would exceed the capacity of your cluster, delete the resources associated with the **hello-limit** application.

4.1. Manually scale the **hello-limit** application up to 3 pods.

```
[student@workstation ~]$ oc scale --replicas 3 deployment/hello-limit
deployment.extensions/hello-limit scaled
```

4.2. Check to see if all three pods are running. You might need to run **oc get pods** multiple times until you see that two pods are running and one pod is pending. Depending on the current cluster load, both additional pods might also be in a pending state.

```
[student@workstation ~]$ oc get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-limit-7c7998ff6b-ctsjp  1/1     Running   0          76s
hello-limit-7c7998ff6b-j7mqx  1/1     Running   0          23s
hello-limit-7c7998ff6b-sz6kl  0/1     Pending   0          23s
```

4.3. Warning events indicate that one or more pods cannot be scheduled because none of the nodes has sufficient CPU resources. Your warning messages might be slightly different.

```
[student@workstation ~]$ oc get events --field-selector type=Warning
LAST SEEN   TYPE      REASON             OBJECT                            MESSAGE
38s         Warning   FailedScheduling   pod/hello-limit-7c7998ff6b-sz6kl  0/5
 nodes are available: 2 node(s) had taints that the pod didn't tolerate, 3
 Insufficient cpu.
...output omitted...
```

4.4. Delete all of the resources associated with the **hello-limit** application.

```
[student@workstation ~]$ oc delete all -l app=hello-limit
pod "hello-limit-7c7998ff6b-ctsjp" deleted
pod "hello-limit-7c7998ff6b-j7mqx" deleted
pod "hello-limit-7c7998ff6b-sz6kl" deleted
deployment.apps "hello-limit" deleted
```

5. Deploy a second application to test memory usage. This second application sets a memory limit of 200MB per container.

5.1. Use the resource file located at **/home/student/DO280/labs/schedule-limit/loadtest.yml** to create the **loadtest** application. In addition to creating a deployment, this resource file also creates a service and a route.

```
[student@workstation ~]$ oc create --save-config \
>    -f ~/DO280/labs/schedule-limit/loadtest.yaml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created
```

5.2. The **loadtest** container image is designed to increase either CPU or memory load on the container by making a request to the application API. Identify the fully-qualified domain name used in the route.

```
[student@workstation ~]$ oc get routes
NAME       HOST/PORT                                              ...
loadtest   loadtest-schedule-limit.apps.cluster.domain.example.com  ...
```

5.3. Open two additional terminal windows.

In the first terminal window, access the application API to simulate additional memory pressure on the container.

In the second terminal window, continuously monitor the status of each pod with **watch oc get pods**.

In the third terminal window, continuously monitor the CPU and memory usage of each pod with **watch oc adm top pod**.

6. Generate additional memory load that can be handled by the container.

6.1. In the first terminal window, use the application API to increase the memory load by 150MB for 60 seconds. Use the fully-qualified domain name previously identified in the route. While you wait for the **curl** command to complete, observe the output in the other two terminal windows.

```
[student@workstation ~]$ curl -X GET \
>    http://loadtest-schedule-limit.apps.cluster.domain.example.com\
> /api/loadtest/v1/mem/150/60
curl: (52) Empty reply from server
```

6.2. In the second terminal window, observe the output of **`watch oc get pods`**. Because the container can handle the additional load, you should see that the single application pod has a status of **`Running`** for the entire **`curl`** request.

```
Every 2.0s: oc get pods              Tue Nov  5 ...

NAME                       READY   STATUS    RESTARTS   AGE
loadtest-f7495948-tlxgm    1/1     Running   0          7m34s
```

6.3. In the third terminal window, observe the output of **`watch oc adm top pod`**. The starting memory usage for the pod is about 20-22Mi.

```
Every 2.0s: oc adm top pod              Tue Nov  5 ...

NAME                       CPU(cores)   MEMORY(bytes)
loadtest-f7495948-tlxgm    0m           20Mi
```

As the API request is made, you should see memory usage for the pod increase to about 170-172Mi.

```
Every 2.0s: oc adm top pod              Tue Nov  5 ...

NAME                       CPU(cores)   MEMORY(bytes)
loadtest-f7495948-tlxgm    0m           172Mi
```

A short while after the **`curl`** request completes, you should see memory usage drop back down to about 20-22Mi.

```
Every 2.0s: oc adm top pod              Tue Nov  5 ...

NAME                       CPU(cores)   MEMORY(bytes)
loadtest-f7495948-tlxgm    0m           20Mi
```

7. Generate additional memory load that cannot be handled by the container.

7.1. Use the application API to increase the memory load by 200MB for 60 seconds. Observe the output in the other two terminal windows.

```
[student@workstation ~]$ curl -X GET \
>    http://loadtest-schedule-limit.apps.cluster.domain.example.com\
> /api/loadtest/v1/mem/200/60
<html><body><h1>502 Bad Gateway</h1>
The server returned an invalid or incomplete response.
</body></html>
```

7.2. In the second terminal window, observe the output of **watch oc get pods**. Almost immediately after running the **curl** command, the status of the pod will transition to **OOMKilled**. You might even see a status of **Error**. The pod is out of memory and needs to be killed and restarted. The status might change to **CrashLoopBackOff** before returning to a **Running** status. The restart count will also increment.

```
Every 2.0s: oc get pods                Tue Nov  5 ...

NAME                      READY    STATUS      RESTARTS    AGE
loadtest-f7495948-tlxgm   0/1      OOMKilled   0           9m13s
```

In some cases the pod might have restarted and changed to a status of **Running** before you have time to switch to the second terminal window. The restart count will have incremented from 0 to 1.

```
Every 2.0s: oc get pods                Tue Nov  5 ...

NAME                      READY    STATUS    RESTARTS    AGE
loadtest-f7495948-tlxgm   1/1      Running   1           9m33s
```

7.3. In the third terminal window, observe the output of **watch oc adm top pod**. After the pod is killed, metrics will not be available for a brief period of time.

```
Every 2.0s: oc adm top pod             Tue Nov  5 ...

W1104 15:19:39.314609  114486 top_pod.go:259] Metrics not available for pod
 schedule-limit/loadtest-f7495948-tlxgm, age: 15m33.314602192s
error: Metrics not available for pod schedule-limit/loadtest-f7495948-tlxgm, age:
 15m33.314602192s
```

7.4. In the first terminal window, delete all of the resources associated with the second application.

```
[student@workstation ~]$ oc delete all -l app=loadtest
pod "loadtest-f7495948-tlxgm" deleted
service "loadtest" deleted
deployment.apps "loadtest" deleted
route.route.openshift.io "loadtest" deleted
```

In the second and third terminal windows, press **Ctrl**+**c** to end the **watch** command. Optionally, close the second and third terminal windows.

8. As a cluster administrator, create quotas for the **schedule-limit** project.

8.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

8.2. Create a quota named **project-quota** that limits the **schedule-limit** project to 3 CPUs, 1GB of memory, and 3 configuration maps.

```
[student@workstation ~]$ oc create quota project-quota \
>    --hard cpu="3",memory="1G",configmaps="3" \
>    -n schedule-limit
resourcequota/project-quota created
```

> **Note**
> This exercise places a quota on configuration maps to demonstrate what happens when a user tries to exceed the quota.

9. As a developer, attempt to exceed the configuration map quota for the project.

   9.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

   9.2. Use a loop to attempt to create four configuration maps. The first three should succeed and the fourth should fail because it would exceed the quota.

```
[student@workstation ~]$ for X in {1..4}
>    do
>    oc create configmap my-config${X} --from-literal key${X}=value${X}
>    done
configmap/my-config1 created
configmap/my-config2 created
configmap/my-config3 created
Error from server (Forbidden): configmaps "my-config4" is forbidden: exceeded
 quota: project-quota, requested: configmaps=1, used: configmaps=3, limited:
 configmaps=3
```

10. Clean up the lab environment by deleting the quota and the project.

   10.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

   10.2. Delete the **project-quota** quota.

```
[student@workstation ~]$ oc delete resourcequota project-quota
resourcequota "project-quota" deleted
```

   10.3. Delete the **schedule-limit** project.

```
[student@workstation ~]$ oc delete project schedule-limit
project.project.openshift.io "schedule-limit" deleted
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab schedule-limit finish
```

This concludes the lab.

Scaling an Application

Objectives

After completing this section, students should be able to:

- Control the number of replicas of a pod.
- Specify the number of replicas in a deployment or deployment configuration resource.
- Manually scale the number of replicas.
- Create a *horizontal pod autoscaler* (HPA) resource.

Specifying Pod Replicas in Configuration Workloads

The number of pod replicas for a specific deployment or deployment configuration can be increased or decreased to meet your needs. Despite the **`ReplicaSet`** and **`ReplicationController`** resources, the number of replicas needed for an application is typically defined in a deployment or deployment configuration resource. A *replica set* or *replication controller* (managed by a deployment or a deployment configuration) guarantees that the specified number of replicas of a pod are running at all times. The replica set or replication controller can add or remove pods as necessary to conform to the desired replica count.

Both deployments and deployment configurations contain:

- The desired number of replicas
- A selector for identifying managed pods
- A pod definition, or template, for creating a replicated pod (including labels to apply to the pod)

The following deployment resource (created using the **`oc create deployment`** command) displays the following items:

```
apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
  replicas: 1 ❶
  selector:
    matchLabels:
      app: scale ❷
  strategy: {}
  template: ❸
    metadata:
      labels:
        app: scale ❹
    spec:
      containers:
...output omitted...
```

1. Specifies the desired number of copies (or replicas) of the pod to run.
2. A replica set (for a deployment) or a replication controller (for a deployment configuration) uses a selector to count the number of running pods, in the same way that a service uses a selector to find the pods to load balance.
3. A template for pods that the replica set or replication controller creates.
4. Labels on pods created by the template must match those used for the selector.

A deployment configuration, typically created using either the **oc new-app** command or the web console, defines the same information, but in a slightly different way. Notice that an additional selector for **deploymentconfig** exists, and the matching **deploymentconfig** label is also added to the pod metadata.

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
...output omitted...
spec:
  replicas: 1
  selector:
    app: hello
    deploymentconfig: hello
  strategy:
    resources: {}
  template:
    metadata:
      annotations:
        openshift.io/generated-by: OpenShiftNewApp
      labels:
        app: hello
        deploymentconfig: hello
...output omitted...
```

If the deployment or deployment configuration resource is under version control, then modify the **replicas** line in the resource file and apply the changes using the **oc apply** command.

Whether in a deployment or a deployment configuration resource, a selector is a set of labels that all of the pods managed by the replica set or replication controller must match. The same set of labels must be included in the pod definition that the deployment or deployment configuration instantiates. This selector is used 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.

Manually Scaling the Number of Pod Replicas

Developers and administrators can choose to manually scale the number of pod replicas in a project. More pods may be needed for an anticipated surge in traffic, or the pod count may be reduced to reclaim resources that can be used elsewhere by the cluster. Whether increasing or decreasing the pod replica count, the first step is to identify the appropriate deployment or deployment configuration (dc) to scale using the **oc get** command.

```
[user@demo ~]$ oc get deployment
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
scale   1/1     1            1           8h
```

The number of replicas in a deployment or deployment configuration resource can be changed manually using the **oc scale** command.

```
[user@demo ~]$ oc scale --replicas 5 deployment/scale
```

The deployment resource propagates the change to the replica set; it reacts to the change by creating new pods (replicas) or deleting existing ones, depending on whether the new desired replica count is less than or greater than the existing count.

Although it is possible to manipulate a replica set or replication controller resource directly, the recommended practice is to manipulate the deployment or deployment configuration resource instead. A new deployment creates either a new replica set or a new replication controller and changes made directly to a previous replica set or replication controller are ignored.

Autoscaling Pods

OpenShift can autoscale a deployment or a deployment configuration based on current load on the application pods, by means of a **HorizontalPodAutoscaler** resource type.

A horizontal pod autoscaler resource uses performance metrics collected by the OpenShift Metrics subsystem. The Metrics subsystem comes pre-installed in OpenShift 4.2, rather than requiring a separate install, as in OpenShift 3.x. To autoscale a deployment or deployment configuration, you must specify resource requests for pods so that the horizontal pod autoscaler can calculate the percentage of usage.

The recommended way to create a horizontal pod autoscaler resource is using the **oc autoscale** command, for example:

```
[user@demo ~]$ oc autoscale dc/hello --min 1 --max 10 --cpu-percent 80
```

The previous command creates a horizontal pod autoscaler resource that changes the number of replicas on the **hello** deployment configuration to keep its pods under 80% of their total requested CPU usage.

The **oc autoscale** command creates a horizontal pod autoscaler resource using the name of the deployment or deployment configuration as an argument (**hello** in the previous example).

> **Note**
> Autoscaling for Memory Utilization continues to be a Technology Preview feature for Red Hat OpenShift Container Platform 4.2.

The maximum and minimum values for the horizontal pod autoscaler resource serve to accommodate bursts of load and avoid overloading the OpenShift cluster. If the load on the application changes too quickly, then it might be advisable to keep a number of spare pods to cope with sudden bursts of user requests. Conversely, too many pods can use up all cluster capacity and impact other applications sharing the same OpenShift cluster.

To get information about horizontal pod autoscaler resources in the current project, use the **`oc get`** command. For example:

```
[user@demo ~]$ oc get hpa
NAME   REFERENCE               TARGETS        MINPODS  MAXPODS  REPLICAS  ...
hello  DeploymentConfig/hello  <unknown>/80%  1        10       1         ...
scale  Deployment/scale        60%/80%        2        10       2         ...
```

Important

A value of <unknown> in the TARGETS column indicates that the deployment or deployment configuration does not define resource requests for the metric. The horizontal pod autoscaler will not scale these pods. Most of the pods created using either the **`oc create deployment`** or the **`oc new-app`** command do not define resource requests. Using the OpenShift autoscaler may therefore require editing the deployment or deployment configuration resources, creating custom YAML or JSON resource files for your application, or adding limit range resources to your project that define default resource requests.

References

For more information, refer to the *Automatically scaling pods* section in the *Red Hat OpenShift Container Platform 4.2 Nodes* guide at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/nodes/index#nodes-pods-autoscaling

Guided Exercise

Scaling an Application

In this exercise, you will scale an application manually and automatically.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Manually scale an application.
- Configure an application to automatically scale based on usage.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates the resource files that you will be using in the activity.

```
[student@workstation ~]$ lab schedule-scale start
```

1. As the **developer** user, create a new project named **schedule-scale**.

   1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. Create a new project for this guided exercise named **schedule-scale**.

```
[student@workstation ~]$ oc new-project schedule-scale
Now using project "schedule-scale" on server
   "https://api.cluster.domain.example.com:6443".
...output omitted...
```

2. Deploy a test application for this exercise which explicitly requests container resources for CPU and memory.

   2.1. Modify the resource file located at **~/DO280/labs/schedule-scale/loadtest.yaml** to set both requests and limits for CPU and memory usage.

```
[student@workstation ~]$ vim ~/DO280/labs/schedule-scale/loadtest.yaml
```

2.2. Replace the **resources: {}** line with the highlighted lines listed below. Ensure that you have proper indentation before saving the file.

```
...output omitted...
    spec:
      containers:
      - image: quay.io/redhattraining/loadtest:v1.0
        name: loadtest
        resources:
          requests:
            cpu: "25m"
            memory: 25Mi
          limits:
            cpu: "100m"
            memory: 100Mi
status: {}
```

2.3. Create the new application using your resource file.

```
[student@workstation ~]$ oc create --save-config \
>    -f ~/DO280/labs/schedule-scale/loadtest.yaml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created
```

2.4. Verify that your application pod has a status of **Running**. You might need to run the **oc get pods** command multiple times.

```
[student@workstation ~]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
loadtest-5f9565dbfb-jm9md   1/1     Running   0          23s
```

3. Manually scale the **loadtest** deployment by first increasing and then decreasing the number of running pods.

3.1. Scale the **loadtest** deployment up to five pods.

```
[student@workstation ~]$ oc scale --replicas 5 deployment/loadtest
deployment.extensions/loadtest scaled
```

3.2. Verify that all five application pods are running. You might need to run the **oc get pods** command multiple times.

```
[student@workstation ~]$ oc get pods
NAME                       READY   STATUS    RESTARTS   AGE
loadtest-5f9565dbfb-22f9s  1/1     Running   0          54s
loadtest-5f9565dbfb-8l2rn  1/1     Running   0          54s
loadtest-5f9565dbfb-jm9md  1/1     Running   0          3m17s
loadtest-5f9565dbfb-lfhns  1/1     Running   0          54s
loadtest-5f9565dbfb-prjkl  1/1     Running   0          54s
```

3.3. Scale the **loadtest** deployment back down to one pod.

```
[student@workstation ~]$ oc scale --replicas 1 deployment/loadtest
deployment.extensions/loadtest scaled
```

3.4. Verify that only one application pod is running. You might need to run the **oc get pods** command multiple times while waiting for the other pods to terminate.

```
[student@workstation ~]$ oc get pods
NAME                       READY   STATUS    RESTARTS   AGE
loadtest-5f9565dbfb-prjkl  1/1     Running   0          72s
```

4. Configure the **loadtest** application to automatically scale based on load, and then test the application by applying load.

4.1. Create a horizontal pod autoscaler that ensures the **loadtest** application always has 2 application pods running; that number can be increased to a maximum of 10 pods when CPU load exceeds 50%.

```
[student@workstation ~]$ oc autoscale deployment/loadtest \
>    --min 2 --max 10 --cpu-percent 50
horizontalpodautoscaler.autoscaling/loadtest autoscaled
```

4.2. The **loadtest** container image is designed to either increase CPU or memory load on the container by making a request to the application API. Identify the fully-qualified domain name used in the route.

```
[student@workstation ~]$ oc get route/loadtest
NAME       HOST/PORT                                         ...
loadtest   loadtest-schedule-scale.apps.cluster.domain.example.com   ...
```

4.3. Access the application API to simulate additional CPU pressure on the container. Move on to the next step while you wait for the **curl** command to complete.

```
[student@workstation ~]$ curl -X GET \
>    http://loadtest-schedule-scale.apps.cluster.domain.example.com\
> /api/loadtest/v1/cpu/1
curl: (52) Empty reply from server
```

4.4. Open a second terminal window and continuously monitor the status of the horizontal pod autoscaler.

> **Note**
> The increased activity of the application does not immediately trigger the autoscaler. Wait a few moments if you do not see any changes to the number of replicas.

```
[student@workstation ~]$ watch oc get hpa/loadtest
```

As the load increases (visible in the **TARGETS** column), you should see the count under **REPLICAS** increase. Observe the output for a minute or two before moving on to the next step. Your output will likely be different from what is displayed below.

```
Every 2.0s: oc get hpa/loadtest              Tue Nov  5 ...

NAME      REFERENCE             TARGETS    MINPODS  MAXPODS  REPLICAS  ...
loadtest  Deployment/loadtest  172%/50%   2        10       9         ...
```

> **Note**
> Although the horizontal pod **autoscalerr** resource can be quick to scale out, it is slower to scale in. Rather than waiting for the **loadtest** application to scale back down to two pods, continue with the rest of the exercise.

5. Back in the first terminal window, create a second application named **scaling** that uses a deployment configuration resource. Scale the application, and then verify the responses coming from the application pods.

   5.1. Create a new application with the **oc new-app** command using the container image located at **quay.io/redhattraining/scaling:v1.0**.

```
[student@workstation ~]$ oc new-app \
>    --docker-image quay.io/redhattraining/scaling:v1.0
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "scaling" created
    deploymentconfig.apps.openshift.io "scaling" created
    service "scaling" 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/scaling'
    Run 'oc status' to view your app.
```

   5.2. Create a route to the application by exposing the service for the application.

```
[student@workstation ~]$ oc expose svc/scaling
route.route.openshift.io/scaling exposed
```

5.3. The **oc create deployment** command creates a **deployment** resource, and the **oc new-app** command creates a **deploymentconfig** (dc) resource. Scale the application up to three pods using the deployment configuration for the application.

```
[student@workstation ~]$ oc scale --replicas 3 dc/scaling
deploymentconfig.apps.openshift.io/scaling scaled
```

5.4. Verify that all three pods for the **scaling** application are running, and identify their associated IP addresses.

```
[student@workstation ~]$ oc get pods -o wide -l app=scaling
NAME              READY   STATUS    ...  IP           ...
scaling-1-5n9nj   1/1     Running   ...  10.128.2.83  ...
scaling-1-8tnnq   1/1     Running   ...  10.128.2.84  ...
scaling-1-l9nxb   1/1     Running   ...  10.131.0.73  ...
```

5.5. Display the host name used to route requests to the **scaling** application.

```
[student@workstation ~]$ oc get route/scaling
NAME      HOST/PORT                                            ...
scaling   scaling-schedule-scale.apps.cluster.domain.example.com  ...
```

5.6. When you access the host name for your application, the PHP page will output the IP address of the pod that replied to the request. Send several requests to your application, and then sort the responses to count the number of requests sent to each pod.

```
[student@workstation ~]$ (for x in {1..100}
>     do curl -s \
>     http://scaling-schedule-scale.apps.cluster.domain.example.com
>     done) | sort | uniq -c
     34 Server IP: 10.128.2.83
     33 Server IP: 10.128.2.84
     33 Server IP: 10.131.0.73
```

6. Optional: Check the status of the horizontal pod autoscaler running for the **loadtest** application. If the **watch oc get hpa/loadtest** command is still running in the second terminal window, switch to it an observe the output. Provided enough time has passed, the replica count should be back down to two. When finished, press **Ctrl+c** to exit the **watch** command, and then close the second terminal window.

```
Every 2.0s: oc get hpa/loadtest             Tue Nov  5 ...

NAME       REFERENCE             TARGETS  MINPODS  MAXPODS  REPLICAS   ...
loadtest   Deployment/loadtest   0%/50%   2        10       2          ...
```

7. Clean up the lab environment by deleting the **schedule-scale** project.

```
[student@workstation ~]$ oc delete project schedule-scale
project.project.openshift.io "schedule-scale" deleted
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab schedule-scale finish
```

This concludes the lab.

Lab

Controlling Pod Scheduling

Performance Checklist

In this lab, you will configure an application to run on a subset of the cluster worker nodes, and to scale with load.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Add a new label to worker nodes.
- Deploy pods to nodes that match a specified label.
- Request CPU and memory resources for pods.
- Configure an application to scale automatically.
- Create a quota to limit the amount of resources a project can consume.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates a directory for exercise files.

```
[student@workstation ~]$ lab schedule-review start
```

1. Source the **/usr/local/etc/ocp4.config** file, and then log in to your OpenShift cluster as the **admin** user. Label the two worker nodes with the **tier** label. Give one worker node the label of **tier=gold** and the other worker node the label of **tier=silver**.
2. Switch to the **developer** user and create a new project named **schedule-review**.
3. Create a new application named **loadtest** using the container image located at **quay.io/redhattraining/loadtest:v1.0**. The **loadtest** application should be deployed to nodes labeled with **tier=silver**. Ensure that each container requests **100m** of CPU and **20Mi** of memory.
4. Create a route to your application named **loadtest** using the default (automatically generated) host name. Depending on how you created your application, you might need to create a service before creating the route. Your application works as expected if running **curl http://loadtest-schedule-review.*apps.cluster.domain.example.com*/api/loadtest/v1/healthz** returns **{"health":"ok"}**.
5. Create a horizontal pod autoscaler named **loadtest** for the **loadtest** application that will scale from **2** pods to a maximum of **40** pods if CPU load exceeds **70%**. You can test the horizontal pod autoscaler with the following command: **curl -X GET http://loadtest-**

schedule-review.*apps.cluster.domain.example.com*/api/loadtest/v1/cpu/3

> **Note**
> Although the horizontal pod autoscaler will scale the **loadtest** application, your OpenShift cluster will run out of resources before it reaches a maximum of 40 pods.

6. As the **admin** user, implement a quota named **review-quota** on the **schedule-review** project. Limit the **schedule-review** project to a maximum of **1** full CPU, **2G** of memory, and **20** pods.

Grading

Run the following **lab** command to verify your work. If the **lab** command reports any errors, review your changes, make corrections, and run the **lab** command again until successful.

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

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab schedule-review finish
```

This concludes the lab.

Solution

Controlling Pod Scheduling

Performance Checklist

In this lab, you will configure an application to run on a subset of the cluster worker nodes, and to scale with load.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Add a new label to worker nodes.
- Deploy pods to nodes that match a specified label.
- Request CPU and memory resources for pods.
- Configure an application to scale automatically.
- Create a quota to limit the amount of resources a project can consume.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates a directory for exercise files.

```
[student@workstation ~]$ lab schedule-review start
```

1. Source the **/usr/local/etc/ocp4.config** file, and then log in to your OpenShift cluster as the **admin** user. Label the two worker nodes with the **tier** label. Give one worker node the label of **tier=gold** and the other worker node the label of **tier=silver**.

    1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

    1.2. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

    1.3. Identify the two worker nodes.

```
[student@workstation ~]$ oc get nodes -l node-role.kubernetes.io/worker -o name
node/ip-10-0-139-250.us-west-1.compute.internal
node/ip-10-0-154-226.us-west-1.compute.internal
```

1.4. Label the first worker node with the label **tier=gold**.

```
[student@workstation ~]$ oc label node \
>     ip-10-0-139-250.us-west-1.compute.internal tier=gold
node/ip-10-0-139-250.us-west-1.compute.internal labeled
```

1.5. Label the second worker node with the label **tier=silver**.

```
[student@workstation ~]$ oc label node \
>     ip-10-0-154-226.us-west-1.compute.internal tier=silver
node/ip-10-0-154-226.us-west-1.compute.internal labeled
```

1.6. Confirm that the labels have been added correctly.

```
[student@workstation ~]$ oc get nodes -l node-role.kubernetes.io/worker -L tier
NAME                                         STATUS   ROLES    ...   TIER
ip-10-0-139-250.us-west-1.compute.internal   Ready    worker   ...   gold
ip-10-0-154-226.us-west-1.compute.internal   Ready    worker   ...   silver
```

2. Switch to the **developer** user and create a new project named **schedule-review**.

2.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

2.2. Create the **schedule-review** project.

```
[student@workstation ~]$ oc new-project schedule-review
Now using project "schedule-review" on server
   "https://api.cluster.domain.example.com:6443".
...output omitted...
```

3. Create a new application named **loadtest** using the container image located at **quay.io/redhattraining/loadtest:v1.0**. The **loadtest** application should be deployed to nodes labeled with **tier=silver**. Ensure that each container requests **100m** of CPU and **20Mi** of memory.

3.1. In order to make the upcoming adjustments easier, create a deployment resource file without actually creating the application.

```
[student@workstation ~]$ oc create deployment loadtest --dry-run \
>     --image quay.io/redhattraining/loadtest:v1.0 \
>     -o yaml > ~/DO280/labs/schedule-review/loadtest.yaml
```

3.2. Edit **~/DO280/labs/schedule-review/loadtest.yaml** to specify a node selector. Add the highlighted lines listed below and ensure that you have proper indentation.

```
[student@workstation ~]$ vim ~/DO280/labs/schedule-review/loadtest.yaml

...output omitted...
    spec:
      containers:
      - image: quay.io/redhattraining/loadtest:v1.0
        name: loadtest
        resources: {}
      nodeSelector:
        tier: silver
status: {}
```

3.3. Continue editing **~/DO280/labs/schedule-review/loadtest.yaml**. Replace the **resources: {}** line with the highlighted lines listed below. Ensure that you have proper indentation before saving the file.

```
...output omitted...
    spec:
      containers:
      - image: quay.io/redhattraining/loadtest:v1.0
        name: loadtest
        resources:
          requests:
            cpu: 100m
            memory: 20Mi
      nodeSelector:
        tier: silver
status: {}
```

3.4. Create the **loadtest** application.

```
[student@workstation ~]$ oc create --save-config \
>    -f ~/DO280/labs/schedule-review/loadtest.yaml
deployment.apps/loadtest created
```

4. Create a route to your application named **loadtest** using the default (automatically generated) host name. Depending on how you created your application, you might need to create a service before creating the route. Your application works as expected if running **curl http://loadtest-schedule-review.*apps.cluster.domain.example.com*/api/loadtest/v1/healthz** returns **{"health":"ok"}**.

4.1. Create a service by exposing the deployment for the **loadtest** application.

```
[student@workstation ~]$ oc expose deployment/loadtest \
>    --port 80 --target-port 8080
service/loadtest exposed
```

4.2. Create a route named **loadtest** by exposing the **loadtest** service.

```
[student@workstation ~]$ oc expose service/loadtest --name loadtest
route.route.openshift.io/loadtest exposed
```

4.3. Identify the host name created by the **loadtest** route.

```
[student@workstation ~]$ oc get route/loadtest
NAME       HOST/PORT                                                ...
loadtest   loadtest-schedule-review.apps.cluster.domain.example.com  ...
```

4.4. Verify access to the **loadtest** application using the host name identified in the previous step.

```
[student@workstation ~]$ curl http://loadtest-schedule-review.\
apps.cluster.domain.example.com/api/loadtest/v1/healthz
{"health":"ok"}
```

5. Create a horizontal pod autoscaler named **loadtest** for the **loadtest** application that will scale from **2** pods to a maximum of **40** pods if CPU load exceeds **70%**. You can test the horizontal pod autoscaler with the following command: **curl -X GET http://loadtest-schedule-review.*apps.cluster.domain.example.com*/api/loadtest/v1/cpu/3**

> **Note**
> Although the horizontal pod autoscaler will scale the **loadtest** application, your OpenShift cluster will run out of resources before it reaches a maximum of 40 pods.

5.1. Create the horizontal pod autoscaler for the **loadtest** application.

```
[student@workstation ~]$ oc autoscale deployment/loadtest --name loadtest \
>    --min 2 --max 40 --cpu-percent 70
horizontalpodautoscaler.autoscaling/loadtest autoscaled
```

5.2. Test the horizontal pod autoscaler by applying CPU load. Use the previously identified host name for the **loadtest** route. Wait for the **curl** command to complete.

```
[student@workstation ~]$ curl -X GET http://loadtest-schedule-review.\
apps.cluster.domain.example.com/api/loadtest/v1/cpu/3
```

5.3. Verify that additional pods have been added. You might need to run the **oc get hpa/loadtest** command multiple times. Your output will likely differ, but check that the replica count is greater than 2.

```
[student@workstation ~]$ oc get hpa/loadtest
NAME       REFERENCE           TARGETS    MINPODS   MAXPODS   REPLICAS   ...
loadtest   Deployment/loadtest 1043%/70%  2         40        21         ...
```

> **Note**
>
> An **<unknown>** entry in the **TARGETS** column indicates that the containers for the **loadtest** application are not requesting CPU resources.

6. As the **admin** user, implement a quota named **review-quota** on the **schedule-review** project. Limit the **schedule-review** project to a maximum of **1** full CPU, **2G** of memory, and **20** pods.

    6.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

    6.2. Create the resource quota.

```
[student@workstation ~]$ oc create quota review-quota \
>    --hard cpu="1",memory="2G",pods="20"
resourcequota/review-quota created
```

> **Note**
>
> The quota will not impact existing pods, but the scheduler will evaluate the quota if new resources, such as pods, are requested.

This concludes the lab.

Grading

Run the following **lab** command to verify your work. If the **lab** command reports any errors, review your changes, make corrections, and run the **lab** command again until successful.

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

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab schedule-review finish
```

Summary

In this chapter, you learned:

- The default pod scheduler uses regions and zones to achieve both performance and redundancy.
- Labeling nodes and using node selectors influences pod placement.
- Resource requests define the minimum amount of resources a pod needs in order to be scheduled.
- Quotas restrict the amount of resources a project is allowed to consume.
- The **`oc scale`** command manually scales the number of replicas of a pod.
- Horizontal pod autoscalers dynamically scale pod replicas based on load.

Chapter 7

Scaling an OpenShift Cluster

Goal	Control the size of an OpenShift cluster.
Objectives	• Issue commands to control the number of workers in a cluster. • Define parameters to automatically control the size of a cluster.
Sections	• Manually Scaling an OpenShift Cluster (and Guided Exercise) • Automatically Scaling an OpenShift Cluster (and Guided Exercise)

Manually Scaling an OpenShift Cluster

Objectives

After completing this section, you should be able to issue commands to control the number of workers in a cluster.

Introducing the Machine API

OpenShift Container Platform can scale the cluster in and out by adding or removing workers through the Machine API. The scaling capabilities allow the cluster to provide enough computing power for your applications. The process can be either manual or automatic, depending on your needs.

The following diagram shows the Machine API (running as an operator) and the API resources that it manages. The operator provides a variety of controllers that interact with the cluster resources, such as the **`MachineSet`** controller, which describes a group of worker nodes.

> **Note**
> The API resources for automatic scaling (**`ClusterAutoscaler`** and **`MachineAutoscaler`**) are not managed by the Machine API operator. The Cluster Autoscaler operator manages these two resources.

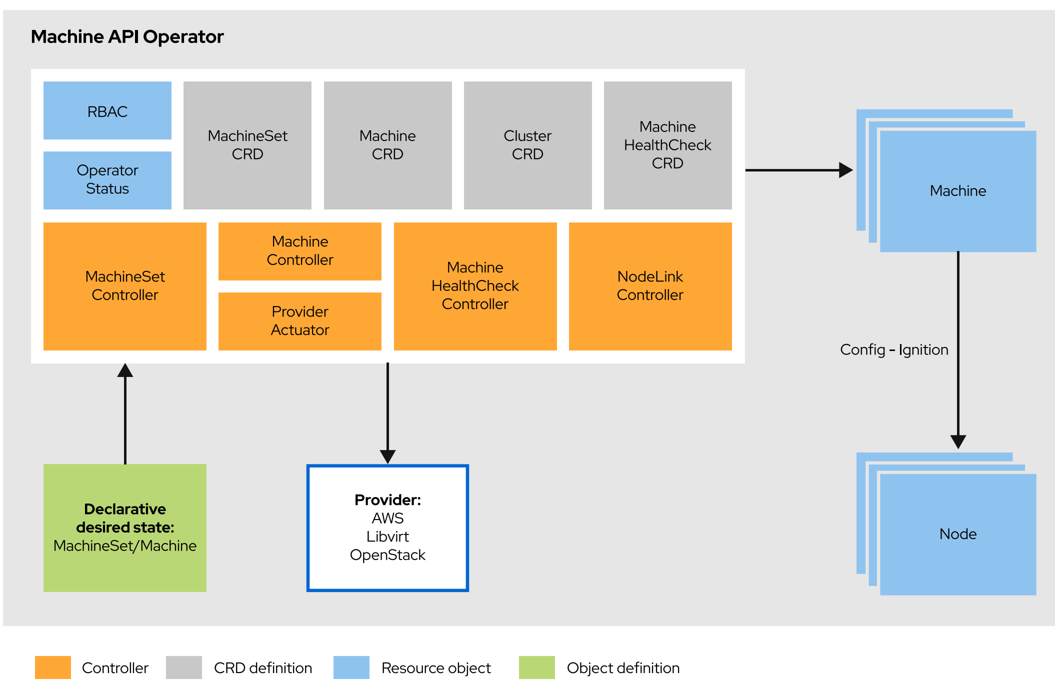


Figure 7.1: The Machine API operator

The Machine API provides the following custom resources:

- **`Machines`** are the fundamental compute unit in a cluster. Each **`Machine`** resource correlates to a physical or virtual node.

  The following example shows the various values for a worker node. Notice the placement, the AMI, and the block storage configuration.

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  generateName: ocp-demo-xxyy-worker-us-west-2b-
  name: ocp-demo-xxyy-worker-us-west-2b-rcm72
  uid: 9e4e1744-0d42-11ea-9a46-025ef26e9de6
  ...
  labels:
    machine.openshift.io/cluster-api-machine-type: worker
    ...
    machine.openshift.io/instance-type: m4.xlarge
    machine.openshift.io/region: us-west-2
    machine.openshift.io/zone: us-west-2b
spec:
  providerID: 'aws:///us-west-2b/i-0c537ed8c82ea894d'
  providerSpec:
    value:
    ...
      placement:
        availabilityZone: us-west-2b
        region: us-west-2
      credentialsSecret:
        name: aws-cloud-credentials
      instanceType: m4.xlarge
      blockDevices:
        - ebs:
            iops: 0
            volumeSize: 120
            volumeType: gp2
      securityGroups:
        - filters:
            - name: 'tag:Name'
              values:
                - ocp-demo-xxyy-worker-sg
      deviceIndex: 0
      ami:
        id: ami-08e10b201e19fd5e7
```

- **`MachineSets`** describe a group of hosts. **`MachineSets`** are to machines what **`ReplicaSets`** are to pods. You can scale in and out the number of replicas (**`machines`**) specified by this resource.

  In a default installation, each worker has its own machine set, excluding the master nodes. By default, OpenShift creates one machine set per availability zone when the cluster is running in AWS. Use a machine set to customize your cluster topology, for example, if you need to define a set per region.

> **Note**
>
> Like **`ReplicaSets`**, **`MachineSets`** use labels to determine their members.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: ocp-demo-xxyy-worker-us-west-2b
...
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: ocp-demo-rnh2q
      machine.openshift.io/cluster-api-machineset: ocp-demo-xxyy-worker-us-west-2b
  template:
    metadata:
      creationTimestamp: null
      labels:
        machine.openshift.io/cluster-api-cluster: ocp-demo-xxyy
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: ocp-demo-xxyy-worker-us-
west-2b
    spec:
      metadata:
        creationTimestamp: null
      providerSpec:
      ...
          placement:
            availabilityZone: us-west-2b
            region: us-west-2
          credentialsSecret:
            name: aws-cloud-credentials
          instanceType: m4.xlarge
          metadata:
            creationTimestamp: null
          publicIp: null
 ...
```

- **`MachineAutoscaler`** and **`ClusterAutoscaler`** are used to automatically scale resources in a cloud deployment. Automatic scaling also ensures that if a worker node becomes unresponsive, then pods are quickly evacuated to another worker node for improved availability. Scaling is useful when the cluster is under stress, or for workloads whose computing requirements change.

  This resource allows you to scale a variety of components, such as nodes, cores, and memory.

> **Note**
>
> See *About the Cluster Autoscaler* link in the references section for more information on how to use these resources.

> **Note**
> The next section discusses the Cluster Autoscaler resource.

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  resourceLimits:
    maxNodesTotal: 20
  scaleDown:
    enabled: true
    delayAfterAdd: 10s
    delayAfterDelete: 10s
    delayAfterFailure: 10s
```

- **`MachineHealthCheck`** verifies the health of a machine (such as a worker node) and takes action if the resource is unhealthy.

> **Note**
> In OpenShift Container Platform version 4.2, **`MachineHealthChecks`** is a technology preview feature.

Manually Scaling Worker Nodes

As mentioned, scaling a cluster can be done in two ways: manually and automatically. The availability of automatic scaling depends on the provider and is supported by the Full-Stack Automation mode (IPI). Reasons to scale a cluster include minimizing resources congestion and better management of resources, such as dedicating a set of nodes to a particular workload.

Like deployments and deployment configurations, you change the number of instances by adjusting the replicas specified by the **`MachineSet`** resource.

```
[user@demo ~]$ oc scale --replicas=2 \
>    machineset MACHINE-SET -n openshift-machine-api
```

If the number of workers in the availability zone does not match the replica count, OpenShift spawns news workers until the number of worker matches the replica count.

> **Note**
> Aside from scaling, changes to a machine set do not affect existing machines. New machines inherit changes made to a machine set, such as new or updated labels.

Scaling down is straightforward. Scaling down the machine set instructs the scheduler to evacuate the pods running on machines in the machine set.

Scaling down only works if the remaining nodes have enough capacity to run the pods. If there are not enough resources, the controller refuses to scale in.

References

About the Cluster Autoscaler
https://docs.openshift.com/container-platform/4.2/machine_management/applying-autoscaling.html#cluster-autoscaler-about_applying-autoscaling

Scaling an OpenShift 4 Cluster
https://github.com/openshift/training/blob/master/docs/04-scaling-cluster.md

Guided Exercise

Manually Scaling an OpenShift Cluster

In this exercise, you manually scale a machine set to increase the computing capacity of the cluster and to ensure that new nodes are created with a specific label.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Label an existing worker node.
- Configure a machine set to add a node label to a new worker node.
- Manually scale the replica count for a machine set.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab scale-manual start
```

1. As the **admin** user, modify a machine set to use the **env=prod** node label.

   1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>     ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. Identify the available machine sets in your cluster.

```
[student@workstation ~]$ oc get machinesets -n openshift-machine-api
NAME                          DESIRED   CURRENT   READY   AVAILABLE   ...
ocp-qz7hf-worker-us-west-1b   1         1         1       1           ...
ocp-qz7hf-worker-us-west-1c   1         1         1       1           ...
...output omitted...
```

   1.4. Edit the first machine set listed so that new machines created from it (and the new worker node associated with the new machine) will have the **env=prod** label.

```
[student@workstation ~]$ oc edit machineset ocp-qz7hf-worker-us-west-1b \
>     -n openshift-machine-api
```

Add the highlighted lines below to the machine set configuration. Ensure that the file has the correct indentation, using spaces instead of tabs, before saving your changes.

```
...output omitted...
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: ocp-qz7hf
      machine.openshift.io/cluster-api-machineset: ocp-qz7hf-worker-us-west-1b
  template:
    metadata:
      creationTimestamp: null
      labels:
        machine.openshift.io/cluster-api-cluster: ocp-qz7hf
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: ocp-qz7hf-worker-us-west-1b
    spec:
      metadata:
        creationTimestamp: null
        labels:
          env: prod
...output omitted...
```

Important

In a real-world scenario, you would add the **env=prod** node label to more than one machine set. Adding the node label to only one machine set effectively limits pod deployments to one availability zone.

2. Scale the modified machine set so that it has two replica machines. Ensure existing nodes associated with the modified machine set have the **env=prod** label.

2.1. Scale the machine set modified in the previous step so that it contains two machines.

```
[student@workstation ~]$ oc scale machineset ocp-qz7hf-worker-us-west-1b \
>     --replicas 2 -n openshift-machine-api
machineset.machine.openshift.io/ocp-qz7hf-worker-us-west-1b scaled
```

Note

It can take 5 minutes or more before the new node created from the new machine has a status of running. You can safely continue with the lab.

2.2. Identify the machines and nodes associated with the machine set that you modified in the previous step. The name of each machine includes the name of the machine set from which it was generated. The new machine does not initially have a node associated with it.

```
[student@workstation ~]$ oc get machines -n openshift-machine-api -o wide \
>    -l machine.openshift.io/cluster-api-machine-role=worker
NAME                              ...  NODE                                ...
ocp-qz7hf-worker-us-west-1b-5x98x  ...                                      ...
ocp-qz7hf-worker-us-west-1b-rvx6w  ...  node1.us-west-1.compute.internal  ...
ocp-qz7hf-worker-us-west-1c-v4n4n  ...  node2.us-west-1.compute.internal  ...
```

2.3. Add the **env=prod** label to the node associated with the older machine. Remember that machine names include the name of the machine set from which they were generated. The node associated with the new machine already has the **env=prod** label.

```
[student@workstation ~]$ oc label node node1.us-west-1.compute.internal env=prod
node/node.us-west-1.compute.internal labeled
```

2.4. Verify that the **env=prod** label is set correctly. The new node might not yet exist.

```
[student@workstation ~]$ oc get nodes -l node-role.kubernetes.io/worker -L env
NAME                              STATUS   ROLES    ...   ENV
node1.us-west-1.compute.internal  Ready    worker   ...   prod
node2.us-west-1.compute.internal  Ready    worker   ...
```

3. Create a project named **scale-manual** that deploys pods to nodes labeled with **env=prod**. Allow the **developer** user to create new applications within this new project.

3.1. Create the **scale-manual** project using **env=prod** as a node selector.

```
[student@workstation ~]$ oc adm new-project scale-manual --node-selector env=prod
Created project scale-manual
```

3.2. Assign the **edit** role to the **developer** user on the **scale-manual** project.

```
[student@workstation ~]$ oc adm policy add-role-to-user edit developer \
>    -n scale-manual
clusterrole.rbac.authorization.k8s.io/edit added: "developer"
```

4. As the **developer** user, deploy a new application in the **scale-manual** project and then scale the deployment to six pods.

4.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

4.2. Ensure that you are using the **scale-manual** project.

```
[student@workstation ~]$ oc project scale-manual
Now using project "scale-manual" on server
   "https://api.cluster.domain.example.com:6443".
```

4.3. Create a new deployment named **manual** using the container image located at **quay.io/redhattraining/hello-world-nginx:v1.0**.

```
[student@workstation ~]$ oc create deployment manual \
>    --image quay.io/redhattraining/hello-world-nginx:v1.0
deployment.apps/manual created
```

4.4. Scale the deployment so that it has 6 application pods.

```
[student@workstation ~]$ oc scale --replicas 6 deployment/manual
deployment.extensions/manual scaled
```

4.5. Verify that the application pods are placed on two nodes. You might need to run the **oc get** command multiple times before all of the application pods are running.

```
[student@workstation ~]$ oc get pods -o wide
NAME                     ...  STATUS   ...  NODE                              ...
manual-9d86c4687-895mm   ...  Running  ...  node3.us-west-1.compute.internal  ...
manual-9d86c4687-pd5sc   ...  Running  ...  node3.us-west-1.compute.internal  ...
manual-9d86c4687-rhwmj   ...  Running  ...  node3.us-west-1.compute.internal  ...
manual-9d86c4687-wqpn7   ...  Running  ...  node3.us-west-1.compute.internal  ...
manual-9d86c4687-x5h5f   ...  Running  ...  node1.us-west-1.compute.internal  ...
manual-9d86c4687-zrlr9   ...  Running  ...  node3.us-west-1.compute.internal  ...
```

Note

The default scheduler might not distribute the pods evenly. Because the new node likely has more free resources available, it might have more application pods scheduled.

5. As the **admin** user, verify that the new node automatically received the **env=prod** label and that it has application pods running.

5.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

5.2. Verify that two worker nodes have the **env=prod** label.

```
[student@workstation ~]$ oc get nodes -l node-role.kubernetes.io/worker -L env
NAME                               STATUS   ROLES    ...   ENV
node1.us-west-1.compute.internal   Ready    worker   ...   prod
node3.us-west-1.compute.internal   Ready    worker   ...   prod
node2.us-west-1.compute.internal   Ready    worker   ...
```

5.3. Use the **oc describe** command to view details about the new worker node. Based on the output of the previous command, use the node with the smallest value for age. The age of the new node displays a unit of minutes (m), as compared to the other nodes that display age with a unit of days (d) or hours (h).

```
[student@workstation ~]$ oc describe node node3.us-west-1.compute.internal
...output omitted...
Capacity:
 attachable-volumes-aws-ebs:  39
 cpu:                         4
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      16419376Ki
 pods:                        250
...output omitted...
Non-terminated Pods:                      (14 in total)
  Namespace                               Name
  ---------                               ----
  openshift-cluster-node-tuning-operator  tuned-2cmnx
  openshift-dns                           dns-default-8bxsm
  openshift-image-registry                node-ca-s86rf
  openshift-machine-config-operator       machine-config-daemon-dj2vg
  openshift-marketplace                   certified-operators-7494d79f45-xsc9h
  openshift-monitoring                    node-exporter-cmd76
  openshift-multus                        multus-rlh22
  openshift-sdn                           ovs-gc5jm
  openshift-sdn                           sdn-w9pv8
  scale-manual                            manual-9d86c4687-895mm
  scale-manual                            manual-9d86c4687-pd5sc
  scale-manual                            manual-9d86c4687-rhwmj
  scale-manual                            manual-9d86c4687-wqpn7
  scale-manual                            manual-9d86c4687-zrlr9
...output omitted...
```

The new worker node increased cluster capacity by 4 CPUs and about 16GB of memory. In addition to application pods from the **manual** deployment, various daemon sets have also placed pods on the new worker node.

6. Clean up the resources used in this exercise.

6.1. Delete the **scale-manual** project.

```
[student@workstation ~]$ oc delete project scale-manual
project.project.openshift.io "scale-manual" deleted
```

6.2. Use the **oc get** command to identify the machine set with two worker nodes, and then use the **oc scale** command to scale it back down to one worker node.

```
[student@workstation ~]$ oc get machinesets -n openshift-machine-api
NAME                          DESIRED    CURRENT    READY    AVAILABLE   ...
ocp-qz7hf-worker-us-west-1b   2          2          2        2           ...
ocp-qz7hf-worker-us-west-1c   1          1          1        1           ...
```

```
[student@workstation ~]$ oc scale machineset ocp-qz7hf-worker-us-west-1b \
>    --replicas 1 -n openshift-machine-api
machineset.machine.openshift.io/ocp-qz7hf-worker-us-west-1b scaled
```

6.3. Remove the **env=prod** node label from the modified machine set.

```
[student@workstation ~]$ oc edit machineset ocp-qz7hf-worker-us-west-1b \
>    -n openshift-machine-api
```

Remove the lines in bold, and then save your changes.

```
...output omitted...
    spec:
      metadata:
        creationTimestamp: null
        labels:
          env: prod
...output omitted...
```

6.4. Remove the **env** label from any worker node where it is set.

```
[student@workstation ~]$ oc label nodes -l env env-
node/node1.us-west-1.compute.internal labeled
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab scale-manual finish
```

This concludes the guided exercise.

Automatically Scaling an OpenShift Cluster

Objectives

After completing this section, you should be able to define parameters to control the size of a cluster automatically.

Automatically Scaling a Cluster

As discussed in the section Manually Scaling an OpenShift Cluster, the Machine API provides several resources for managing the workloads of your cluster. You can scale your cluster resources in two ways: manually and automatically.

Manual scaling requires updating the number of replicas in a machine set. Automatic scaling of a cluster involves using two custom resources: **`MachineAutoscaler`** and **`ClusterAutoscaler`**.

A **`MachineAutoscaler`** resource automatically scales the number of replicas in a machine set, depending on the load. This API resource interacts with the machine sets and instructs them to add more worker nodes to the cluster. The resource supports the definition of lower and upper boundaries. The **`ClusterAutoscaler`** enforces limits for the whole cluster, such as the total number of nodes. For example, **`MaxNodesTotal`** sets the maximum number of cores in the whole cluster, and **`MaxMemoryTotal`** sets the maximum memory in the whole cluster.

Each OpenShift cluster can only have one **`ClusterAutoscaler`** resource. The **`ClusterAutoscaler`** resource operates at a higher level and defines the maximum number of nodes and other resources, such as cores, memory, and Graphical Processing Units (GPUs). This prevents the **`MachineAutoscaler`** resources from scaling out in an uncontrolled manner.

> **Note**
> You must define a **`ClusterAutoscaler`** resource for your cluster, otherwise, automatic scaling does not work.

The following excerpt describes a **`ClusterAutoscaler`** resource:

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10
  resourceLimits:
    maxNodesTotal: 6
scaleDown:
    enabled: true
    delayAfterAdd: 3m
    unneededTime: 3m
```

Use a **MachineAutoscaler** resource to scale the number of machines defined by a **MachineSet** resource. Scaling only works if you defined a **ClusterAutoscaler**, and adding a new machine does not exceed any of the values defined by the **ClusterAutoscaler** resource. For example, adding one **m4.xlarge** machine from AWS adds one node, 4 CPU cores, and 16 GB of memory. On its own, a **MachineAutoscaler** does not scale the cluster in or out unless cluster autoscaling is allowed.

The following diagram shows how the **MachineAutoscaler** resource interacts with machine sets, which scale worker nodes in and out.

- If the **MachineAutoscaler** resource must scale, then it checks to see if a **ClusterAutoscaler** resource exists. If it does not, then no scaling occurs.

  If a **ClusterAutoscaler** resource does exist, then the **MachineAutoscaler** resource evaluates whether adding the new machine violates any of the limits defined in the **ClusterAutoscaler**.

- Provided the request does not exceeds the limits, a new machine is created.

- When the new machine is ready, OpenShift schedules pods to it as a new node.

Properties such as **minReplicas** and **maxReplicas** define the lower and upper boundaries of automatic scaling.

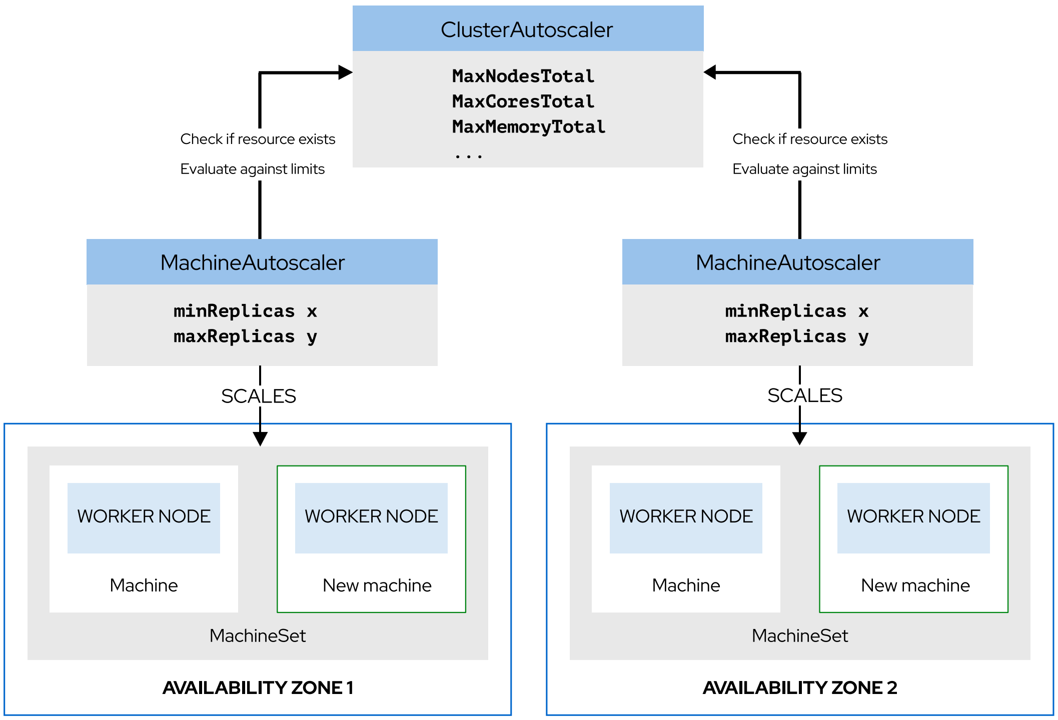


Figure 7.2: Dynamic scaling with full-stack automation

> **Note**
> There is a one to one mapping between a **`MachineAutoscaler`** and a **`MachineSet`** resource. This mapping allows each **`MachineAutoscaler`** resource to manage its own machine set.

> **Note**
> In a default full-stack automation deployment, each **`MachineSet`** resource maps to an availability zone, but this design is arbitrary. For example, you can use a **`MachineSet`** resource to separate workers based on their disk speeds.

Implementing Automatic Scaling

For successful automatic scaling, the following are required:

- A cluster deployed in full-stack automation, because automatic scaling must interact with a cloud service when adding or removing workers.
- A **`ClusterAutoscaler`** resource, if the infrastructure supports it. Additionally, the **`ClusterAutoscaler`** resource might limit the maximum number of nodes, and define a minimum and maximum values for cores, memory, and GPUs. The **`enabled: true`** entry in the **`scaleDown`** section of the **`ClusterAutoscaler`** resource authorizes the cluster to automatically scale in the number of machines when they are not used.
- At least one **`MachineAutoscaler`** resource. Each **`MachineAutoscaler`** resource defines a minimum and a maximum number of replicas for a specific machine set.

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
  name: "scale-automatic"
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1
  maxReplicas: 2
  scaleTargetRef:
    apiVersion: machine.openshift.io/v1beta1 kind: MachineSet
    name: MACHINE-SET-NAME
```

After creating these resources, if the cluster cannot manage a load, then the automatic addition of worker nodes is triggered.

The **`ClusterAutoscaler`** resource scales out the number of workers when pods fail to schedule on any of the current nodes due to insufficient resources, or when another node is necessary to meet deployment needs.

> **Important**
> OpenShift does not increase the cluster resources beyond the limits that you specify in the **`ClusterAutoscaler`** resource.

> **References**
> For more information on the cluster automatic scaler, refer to the *Applying Autoscaling to an OpenShift Container Platform Cluster* chapter in the *Red Hat OpenShift Container Platform 4.2 Official Machine Management Guide* at
> https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/machine_management/index#applying-autoscaling
>
> **Scaling an OpenShift 4 Cluster**
> https://github.com/openshift/training/blob/master/docs/04-scaling-cluster.md

Guided Exercise

Automatically Scaling an OpenShift Cluster

In this exercise, you will automatically scale a machine set to increase its computing capacity based on current loads.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Create a **`ClusterAutoscaler`** resource.
- Create a **`MachineAutoscaler`** resource.

Before You Begin

On the **`workstation`** machine, use the **`lab`** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and creates the files needed for this exercise.

```
[student@workstation ~]$ lab scale-automatic start
```

1. As the **`admin`** user, create a **`ClusterAutoscaler`** resource that allows your cluster to scale out to six nodes. Your cluster should also be able to scale in.

   1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to your OpenShift cluster as the **`admin`** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. Modify the sample cluster autoscaler resource file at **`/home/student/DO280/labs/scale-automatic/clusterautoscaler.yaml`**.

```
[student@workstation ~]$ vim ~/DO280/labs/scale-automatic/clusterautoscaler.yaml
```

   Ensure the file matches the highlighted lines before saving the file.

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10
  resourceLimits:
    maxNodesTotal: 6
  scaleDown:
    enabled: true
    delayAfterAdd: 3m
    unneededTime: 3m
```

1.4. Use the modified resource file to create the cluster autoscaler.

```
[student@workstation ~]$ oc create --save-config \
>    -f ~/DO280/labs/scale-automatic/clusterautoscaler.yaml
clusterautoscaler.autoscaling.openshift.io/default created
```

2. Create a **MachineAutoscaler** resource for the first machine set in the **openshift-machine-api** namespace, so that the machine set can scale out to two nodes.

2.1. Identify the machine sets available in your environment. The machine autoscaler must use the name of an existing machine set.

```
[student@workstation ~]$ oc get machinesets -n openshift-machine-api
NAME                           DESIRED   CURRENT   READY   AVAILABLE   ...
ocp-qz7hf-worker-us-west-1b    1         1         1       1           ...
ocp-qz7hf-worker-us-west-1c    1         1         1       1           ...
```

2.2. Modify the sample machine autoscaler resource file at **/home/student/DO280/labs/scale-automatic/machineautoscaler.yaml**.

```
[student@workstation ~]$ vim ~/DO280/labs/scale-automatic/machineautoscaler.yaml
```

Increase **maxReplicas** to a value of **2**. Replace the highlighted instance of **MACHINE-SET-NAME** with the name of the first machine set identified in the preceding step.

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
  name: "scale-automatic"
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1
  maxReplicas: 2
  scaleTargetRef:
    apiVersion: machine.openshift.io/v1beta1
    kind: MachineSet
    name: MACHINE-SET-NAME
```

2.3. Use the modified resource file to create the machine autoscaler.

```
[student@workstation ~]$ oc create --save-config \
>     -f ~/DO280/labs/scale-automatic/machineautoscaler.yaml \
>     -n openshift-machine-api
machineautoscaler.autoscaling.openshift.io/scale-automatic created
```

3. As the **developer** user, create the **scale-automatic** project, and then create a new deployment in the project.

3.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

3.2. Create the **scale-automatic** project.

```
[student@workstation ~]$ oc new-project scale-automatic
Now using project "scale-automatic" on server
   "https://api.cluster.domain.example.com:6443".
```

3.3. Create the **loadtest** deployment, service, and route using the resource file located at **/home/student/DO280/labs/scale-automatic/loadtest.yaml**.

```
[student@workstation ~]$ oc create --save-config \
>     -f ~/DO280/labs/scale-automatic/loadtest.yaml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created
```

Note

The **loadtest** deployment requests two full CPUs per container (2000 millicores) so that only one pod can fit on each worker node.

4. Switch back to the **admin** user, and then trigger the machine autoscaler by manually scaling the **loadtest** deployment.

4.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

4.2. Scale the **loadtest** deployment to three replica pods in the **scale-automatic** project.

```
[student@workstation ~]$ oc scale --replicas 3 deployment/loadtest \
>     -n scale-automatic
deployment.extensions/loadtest scaled
```

4.3. Use the **watch** command to continuously monitor the output of **oc get pods -o wide** in the **scale-automatic** namespace. The second application pod should move to a status of running, but the third application pod is pending because the cluster does not have enough resources to schedule it. Leave the **watch** command running in the terminal window.

```
[student@workstation ~]$ watch oc get pods -o wide -n scale-automatic
Every 2.0s: oc get pods -o wide -n scale-automatic                        ...

NAME                          ... STATUS  ... NODE
loadtest-6b9c55876f-6xd9f ... Pending ... <node>                          ...
loadtest-6b9c55876f-spvd8 ... Running ... node1.us-west-1.compute.internal ...
loadtest-6b9c55876f-wt6jz ... Running ... node2.us-west-1.compute.internal ...
```

4.4. In a new terminal window, use the **watch** command to continuously monitor the output of **oc get nodes -l node-role.kubernetes.io/worker**. Press **Ctrl**+**c** to exit the **watch** command; after the three worker nodes have a status of Ready, close the terminal window.

```
[student@workstation ~]$ watch oc get nodes -l node-role.kubernetes.io/worker
Every 2.0s: oc get nodes -l node-role.kubernetes.io/worker                 ...

NAME                              STATUS   ROLES    ...
node1.us-west-1.compute.internal  Ready    worker   ...
node3.us-west-1.compute.internal  Ready    worker   ...
node2.us-west-1.compute.internal  Ready    worker   ...
```

Note

Although it can take five minutes or more to create the new node, the new machine is created almost immediately. Run the **oc get machines -n openshift-machine-api** command to verify that you see a total of six machines. If not, confirm that you correctly configured the ClusterAutoscaler and MachineAutoscaler resources.

4.5. When the new worker node is ready, the pending pod will move to a status of ContainerCreating and then to a status of Running. The initial terminal window should still be running the **watch** command to display the output of **oc get pods -o wide -n scale-automatic**. Verify that all three application pods have a status of Running, and then press **Ctrl**+**c** to exit the **watch** command.

```
Every 2.0s: oc get pods -o wide -n scale-automatic                        ...

NAME                          ... STATUS  ... NODE
loadtest-6b9c55876f-6xd9f ... Running ... node3.us-west-1.compute.internal ...
loadtest-6b9c55876f-spvd8 ... Running ... node1.us-west-1.compute.internal ...
loadtest-6b9c55876f-wt6jz ... Running ... node2.us-west-1.compute.internal ...
```

5. Clean up the lab environment by scaling the machine set back down to one machine. Delete the cluster autoscaler, the machine autoscaler, and the **scale-automatic** project.

5.1. Identify the machine set in the **openshift-machine-api** namespace that has two machines.

```
[student@workstation ~]$ oc get machinesets -n openshift-machine-api
NAME                         DESIRED   CURRENT   READY   AVAILABLE   ...
ocp-qz7hf-worker-us-west-1b  2         2         2       2           ...
ocp-qz7hf-worker-us-west-1c  1         1         1       1           ...
```

5.2. Manually scale the machine set back to one machine.

```
[student@workstation ~]$ oc scale machineset ocp-qz7hf-worker-us-west-1b \
>    --replicas 1 -n openshift-machine-api
machineset.machine.openshift.io/ocp-qz7hf-worker-us-west-1b scaled
```

5.3. Delete the **default** cluster autoscaler.

```
[student@workstation ~]$ oc delete clusterautoscaler default
clusterautoscaler.autoscaling.openshift.io "default" deleted
```

5.4. Delete the **scale-automatic** machine autoscaler in the **openshift-machine-api** namespace.

```
[student@workstation ~]$ oc delete machineautoscaler scale-automatic \
>    -n openshift-machine-api
machineautoscaler.autoscaling.openshift.io "scale-automatic" deleted
```

5.5. Delete the **scale-automatic** project.

```
[student@workstation ~]$ oc delete project scale-automatic
project.project.openshift.io "scale-automatic" deleted
```

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab scale-automatic finish
```

This concludes the guided exercise.

Summary

In this chapter, you learned:

- OpenShift Container Platform provides mechanisms for automatically scaling in and scaling out the environment. You can scale your cluster manually or automatically.
- When scaling a cluster manually, you interact with the **`MachineSets`** resource, which controls the **`Machines`** resource. **`Machines`** map to worker nodes.
- When scaling a cluster automatically, you interact with the **`MachineAutoscaler`** resource, which instructs a **`MachineSet`** resource to update its number of replicas.
- The **`ClusterAutoscaler`** resource controls the maximum number of resources that the **`MachineAutoscalers`** can create. This allows you to prevent the uncontrolled growth of your cluster.

Chapter 8

Performing Cluster Updates

Goal	Describe how to perform a cluster update.
Objectives	Describe the cluster update process.
Sections	• Describing the Cluster Update Process (and Quiz)

Describing the Cluster Update Process

Objectives

After completing this section, you should be able to describe the cluster update process.

Introducing Cluster Updates

Red Hat OpenShift Container Platform 4 adds many new features by using Red Hat Enterprise Linux CoreOS. Red Hat released a new software distribution system that provides the best upgrade path to update your cluster and the underlying operating system. This new distribution system is one of the significant benefits of OpenShift 4 architectural changes, enabling clusters to upgrade *Over-the-Air (OTA)*.

This software distribution system for OTA manages the controller manifests, cluster roles, and any other resources necessary to update a cluster to a particular version.

This feature ensures that a cluster runs the latest available version seamlessly. OTA also enables a cluster to use new features as they become available, including the latest bug fixes and security patches. OTA substantially decreases downtime due to upgrades.

> **Note**
>
> Red Hat hosts and manages this service at `https://cloud.redhat.com`, and hosts cluster images at `https://quay.io`.
>
> As of OpenShift 4.2, the OTA system requires a persistent connection to the Internet. It is not possible to deploy this feature on-premise.

OTA enables faster updates by allowing the skipping of intermediary versions. For example, you can update from 4.2.1 to 4.2.3, thus bypassing 4.2.2.

You use a single interface (`https://cloud.redhat.com`) to manage the life cycle of all your OpenShift clusters.

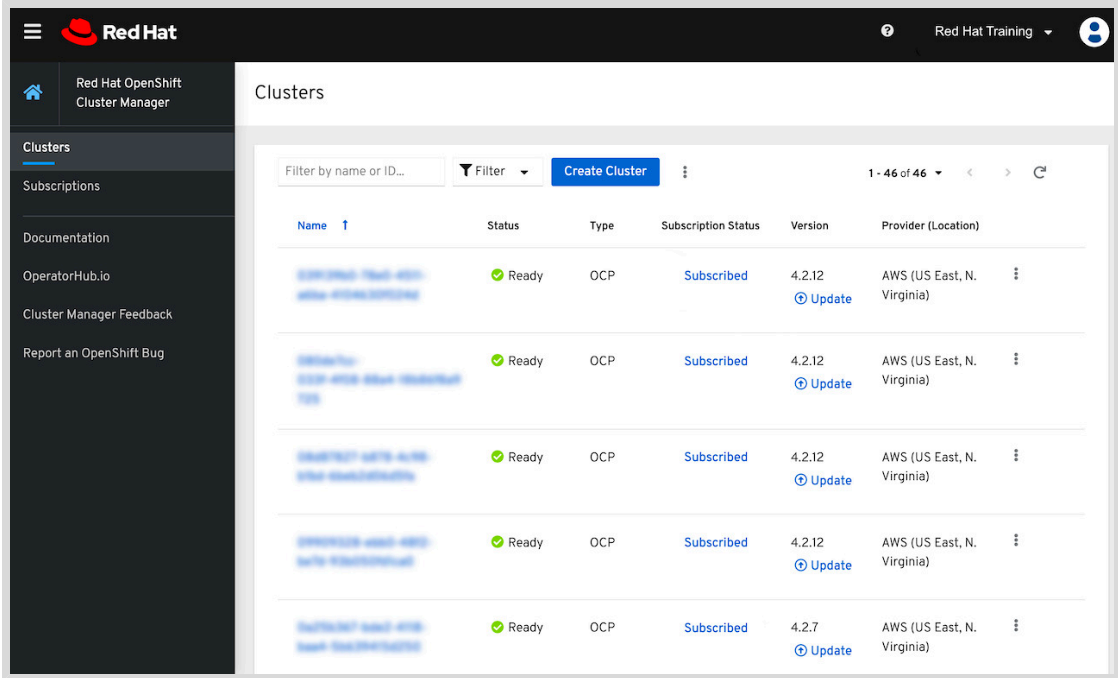


Figure 8.1: Managing clusters at cloud.redhat.com

The service defines *upgrade paths* that correspond to cluster eligibility for certain updates.

Update paths belong to upgrade *channels*. A channel can be visualized as a representation of the upgrade path. The channel controls the frequency and stability of updates. The OTA policy engine represents channels as a series of pointers to particular versions within the upgrade path.

A channel name consists of three parts: the tier (release candidate, fast, and stable), the major version (4), and the minor version (.2). Example channel names include: **`candidate-4.2`**, **`fast-4.2`**, and **`stable-4.2`**. Each channel delivers patches, also called *z-stream* updates, for a given cluster version.

Minor versions support the z-stream channel, such as 4.2.**2**, or 4.2.**3**. For instance, the **`fast-4.2`** channel can deliver 4.2.1 and 4.2.2 updates but not 4.3.1.

The following diagram shows two theoretical update graphs for alpha and beta channels.

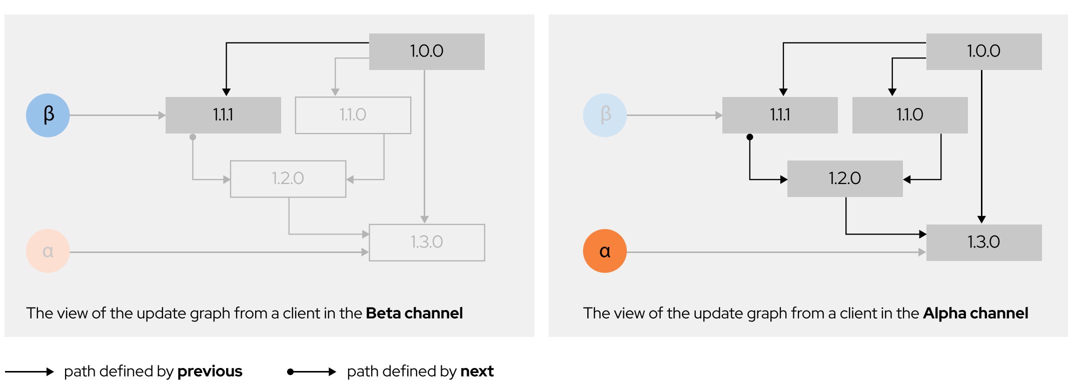


Figure 8.2: Update graphs for alpha and beta channels

The beta channel offers updates up to a version 1.1.1 of the software from the 1.0.0 version, however, minor updates that are more recent are not available in this channel (such as 1.2.0). You can think of this channel as a production channel, wherein cluster versions more recent than 1.1.1 have not been tested for production yet.

The alpha channel offers several update paths, from 1.0.0 to 1.3.0, but also from 1.0.0 to 1.1.1 or 1.1.0. This release candidate channel allows the cluster to install the latest available updates (1.3.0), but also any intermediary version, if needed.

The *stable* and *fast* channels are classified as General Availability (GA), whereas the *candidate* (Release Candidate (RC) channel) channel is not supported by Red Hat.

To ensure the stability of the cluster and proper level of support, you cannot switch from a stable channel or fast channel to the candidate channel. However, you can switch from a stable channel to a fast one, and vice versa.

Describing the Fast Channel

The fast channel delivers updates as soon as they are available. This channel is best suited for pre-production and QA environments.

> **Note**
> Customers can help to improve OpenShift by joining the Red Hat connected customers program. If you join this program, then your cluster is registered to the fast channel.

Discussing the Stable Channel

- The stable channel contains delayed updates.

  Red Hat support and site reliability engineering (SRE) teams monitor operational clusters with new fast updates. If operational clusters pass additional testing and validation, updates in the fast channel are enabled in the stable channel.

  If Red Hat observes operational issues from a fast channel update, then that update is skipped in the stable channel. The stable channel delay provides time to observe unforeseen problems in actual OpenShift clusters that testing did not reveal.

  The stable channel is better suited for production environments.

> **Note**
> The release of updates for patch and CVE fixes ranges from several hours to a day. This delay provides time to assess any operational impacts to OpenShift clusters.

- The stable channel delivers only minor updates for a given cluster version; for instance, the `stable-4.2` channel delivers 4.2.1 and 4.2.2 updates, but not 4.3.1.

To ensure the stability of your applications, configure a development cluster to use the fast channel, and a production cluster to use the stable channel. This can help you troubleshoot issues early.

Describing OTA

OTA follows a client-server approach. Red Hat hosts the cluster images and the update infrastructure. One feature of OTA is the generation of all possible update paths for your cluster.

OTA gathers information about the cluster and your entitlement to determine the available upgrade paths.

You are notified via the web console when a new update is available.

The following diagram describes the updates architecture: Red Hat hosts both the cluster images and a "watcher", which automatically detects new images that are pushed to Quay. The *Cluster Version Operator* (CVO) receives its update status from that watcher. The CVO starts by updating the cluster components via their operators, and then updates any extra components that the *Operator Lifecycle Manager* (OLM) manages.

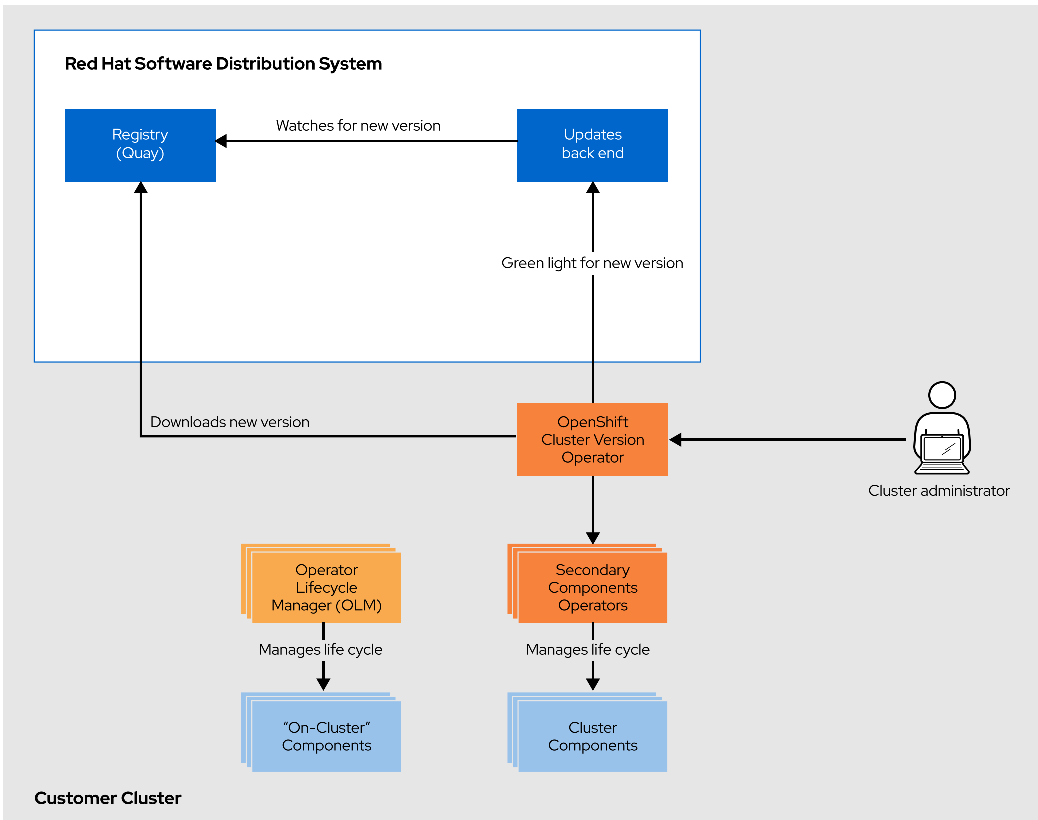


Figure 8.3: OpenShift Container Platform Updates Architecture

Telemetry allows Red Hat to determine the update path.

The cluster uses Prometheus-based telemetry to report on the state of each cluster operator. The data is anonymized and sent back to Red Hat servers that advise cluster administrators about potential new releases.

> **Note**
>
> Red Hat values customer privacy. For a complete list of the data that Telemeter gathers, consult the *Telemeter – Sample Metrics* document listed in the references section.

In the future, Red Hat intends to extend the list of updated operators that are included in the upgrade path to include *independent software vendors (ISV)* operators.

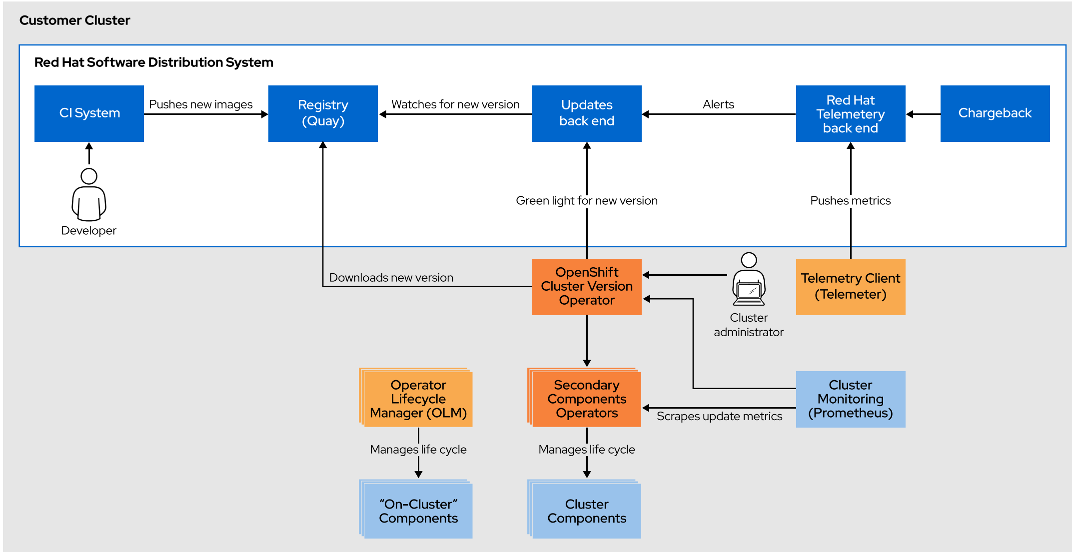


Figure 8.4: Managing cluster updates using telemetry

Discussing the Update Process

Machine Config Operator
: The Machine Config Operator applies the desired machine state to each of the nodes. This component also handles the rolling upgrade of nodes in the cluster, and uses CoreOS Ignition as the configuration format.

Operator Lifecycle Manager (OLM)
: The Operator Lifecycle Manager (OLM) orchestrates updates to any operators running in the cluster.

Updating the Cluster

You can update the cluster via the web console, or from the command-line. Updating via the web console is easier than using the command-line. The cluster administration page displays when a new update is available. From this page, click **Update now** to being the process.

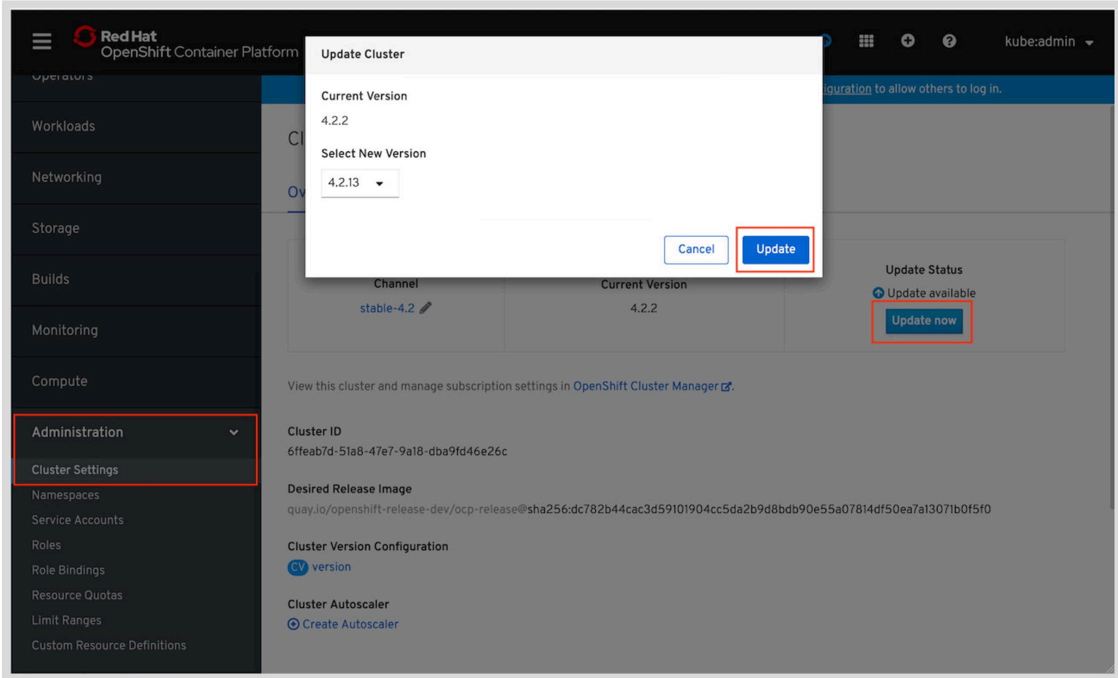


Figure 8.5: Updating the cluster using the web console

Important

Red Hat does not support reverting your cluster to a previous version, or rollback. Red Hat only supports upgrading to a newer version.

The update process also updates the underlying operating system when there are updates available. It uses the **rpm-ostree** technology for managing transactional upgrades. Updates are delivered via container images and are part of the OpenShift update process. When the update deploys, the nodes pull the new image, extract it, write the packages to the disk, and then modify the bootloader to boot into the new version. The machine reboots and implements a rolling update to ensure that the cluster capacity is minimally impacted.

The following steps describe the procedure for updating a cluster as a cluster administrator using the command-line interface.

1. Retrieve the cluster version.

```
[user@demo ~]$ oc get clusterversion
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE ...
version   4.2.2     True        False         47d   ...
```

2. Review the current update channel information and confirm the channel. If you are running the cluster in production, then ensure that the channel reads **stable**.

```
[user@demo ~]$ oc get clusterversion \
>    -o json|jq ".items[0].spec"
{
  "channel": "stable-4.2",
  "clusterID": "6ffeac7d-51a8-47e7-9a18-dba9fd46e26c",
  "upstream": "https://api.openshift.com/api/upgrades_info/v1/graph"
}
```

3. View the available updates and note the version number of the update that you want to apply.

```
[user@demo ~]$ oc adm upgrade
Cluster version is 4.2.2

Updates:

VERSION IMAGE
4.2.4   quay.io/openshift-release-dev/ocp-release@sha256:...
4.2.7   quay.io/openshift-release-dev/ocp-release@sha256:...
4.2.8   quay.io/openshift-release-dev/ocp-release@sha256:...
4.2.9   quay.io/openshift-release-dev/ocp-release@sha256:...
4.2.10  quay.io/openshift-release-dev/ocp-release@sha256:...
4.2.12  quay.io/openshift-release-dev/ocp-release@sha256:...
4.2.13  quay.io/openshift-release-dev/ocp-release@sha256:...
```

4. Run the following command to install the latest available update for your cluster.

```
[user@demo ~]$ oc adm upgrade --to-latest=true
```

5. Run the following command to install a specific version. *VERSION* corresponds to one of the available versions that the **oc adm upgrade** command returns.

```
[user@demo ~]$ oc adm upgrade --to=VERSION
```

6. The previous command initializes the update process. Run the following command to review the status of the Cluster Version Operator (CVO). The **version** key now matches the version that you have selected.

```
[user@demo ~]$ oc get clusterversion \
>    -o json|jq ".items[0].spec"
{
  "channel": "stable-4.2",
  "clusterID": "990f7ab8-109b-4c95-8480-2bd1deec55ff",
  "desiredUpdate": {
    "force": false,
    "image": "quay.io/openshift-release-dev/ocp-release@sha256:...
    "version": "4.2.13"
  },
  "upstream": "https://api.openshift.com/api/upgrades_info/v1/graph"
}
```

7. The following command allows you to review the cluster version status history to monitor the status of the update. It might take some time for all the objects to finish updating.

   The history contains a list of the most recent versions applied to the cluster. This value is updated when the CVO applies an update. The list is ordered by date, where the newest update is first in the list.

   Updates in the history have a state of **Completed** if the rollout completed. Otherwise, the update has a state of **Partial** if the update failed or did not complete.

```
[user@demo ~]$ oc get clusterversion \
>    -o json|jq ".items[0].status.history"
[
  {
    "completionTime": "2019-11-22T16:22:43Z",
    "image": "quay.io/openshift-release-dev/ocp-release@sha256:...
    "startedTime": "2019-11-22T16:08:10Z",
    "state": "Completed",
    "verified": false,
    "version": "4.2.13"
  }
]
```

Important

If an upgrade fails, the operator stops and reports the status of the failing component. Rolling your cluster back to a previous version is not supported. If your upgrade fails, contact Red Hat support.

8. After the update completes, you can confirm that the cluster version has updated to the new version.

```
[user@demo ~]$ oc get clusterversion
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   ...
version   4.2.13     True        False          1d     ...
```

References

For more information on Red Hat Enterprise Linux CoreOS (RHCOS), consult the *Red Hat Enterprise Linux CoreOS (RHCOS)* chapter at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/architecture/index#architecture-rhcos

OpenShift Container Platform (OCP) 4 upgrade paths
https://access.redhat.com/articles/4495171

OpenShift 4.2 Upgrades phased roll out
https://access.redhat.com/solutions/4583231

Telemeter – Sample Metrics
https://github.com/openshift/telemeter/blob/master/docs/sample-metrics.md

Cincinnati update algorithm
https://github.com/openshift/cincinnati/blob/master/docs/design/cincinnati.md#cincinnatin

Quiz

Describing the Cluster Update Process

Choose the correct answers to the following questions:

1. Which two of the following updates would be available in the `fast-4.2` update channel? (Choose two.)

a. 4.1.2
b. 4.2.1
c. 4.3.1
d. 4.2.5

2. Which of the following components retrieves the updated cluster images from Quay.io?

a. Cluster Monitoring (Prometheus)
b. Operator Lifecycle Manager (OLM)
c. Cluster Version Operator (CVO)
d. Telemetry client (Telemeter)

3. Which of the following components manages the updates of operators that are not considered to be cluster operators?

a. Operator Lifecycle Manager (OLM)
b. Telemetry client (Telemeter)
c. Cluster Version Operator (CVO)

4. Which two of the following commands allow you to retrieve the version of the cluster that is currently running? (Choose two.)

a. oc adm upgrade
b. oc get clusterchannel
c. oc get clusterversion

5. Which of the following statements is true regarding the OTA feature?

a. The *stable* channel is classified as General Availability (GA), whereas the *fast* channel is classified as Release Candidate (RC).
b. When using the stable channel, you cannot skip intermediary versions. For example, when updating from 4.1 to 4.3, OpenShift installs the 4.2 version first.
c. It is not possible to switch from a stable or fast channel to a candidate channel. However, you can switch from a fast to a stable channel and vice versa.
d. Rolling back a failed update is only supported by Red Hat when you are attempting to update from a z-stream version to one another (for example, from 4.2.2 to 4.2.3, but not from 4.2.3 to 4.3).

6. Which two of the following channels are classified as general availability? (Choose two.)

a. candidate-4.2.stable
b. stable-4.2
c. candidate-stable-4.2
d. fast-4.1
e. fast-4.1.1

Solution

Describing the Cluster Update Process

Choose the correct answers to the following questions:

1. **Which two of the following updates would be available in the `fast-4.2` update channel? (Choose two.)**
   a. 4.1.2
   b. 4.2.1
   c. 4.3.1
   d. 4.2.5

2. **Which of the following components retrieves the updated cluster images from Quay.io?**
   a. Cluster Monitoring (Prometheus)
   b. Operator Lifecycle Manager (OLM)
   c. Cluster Version Operator (CVO)
   d. Telemetry client (Telemeter)

3. **Which of the following components manages the updates of operators that are not considered to be cluster operators?**
   a. Operator Lifecycle Manager (OLM)
   b. Telemetry client (Telemeter)
   c. Cluster Version Operator (CVO)

4. **Which two of the following commands allow you to retrieve the version of the cluster that is currently running? (Choose two.)**
   a. oc adm upgrade
   b. oc get clusterchannel
   c. oc get clusterversion

5. **Which of the following statements is true regarding the OTA feature?**
   a. The *stable* channel is classified as General Availability (GA), whereas the *fast* channel is classified as Release Candidate (RC).
   b. When using the stable channel, you cannot skip intermediary versions. For example, when updating from 4.1 to 4.3, OpenShift installs the 4.2 version first.
   c. It is not possible to switch from a stable or fast channel to a candidate channel. However, you can switch from a fast to a stable channel and vice versa.
   d. Rolling back a failed update is only supported by Red Hat when you are attempting to update from a z-stream version to one another (for example, from 4.2.2 to 4.2.3, but not from 4.2.3 to 4.3).

6. Which two of the following channels are classified as general availability? (Choose two.)

a. candidate-4.2.stable

b. stable-4.2

c. candidate-stable-4.2

d. fast-4.1

e. fast-4.1.1

Summary

In this chapter, you learned:

- One of the major benefits of OpenShift 4 architectural changes is that you can update your clusters *Over-the-Air* (OTA).
- Red Hat provides a new software distribution system that ensures the best path for updating your cluster and the underlying operating system.
- There are two distribution channels: *fast*, and *stable*. The *fast* channel delivers updates are soon as they are available. The *stable* channel delivers delayed updates.
- Red Hat does not support reverting your cluster to a previous version. Red Hat only supports upgrading to a newer version.

Chapter 9

Managing a Cluster with the Web Console

Goal	Manage a Red Hat OpenShift cluster using the web console.
Objectives	• Perform cluster administration with the web console. • Manage cluster workloads with the web console. • Examine the metrics page and dashboard within the web console.
Sections	• Performing Cluster Administration (and Guided Exercise) • Managing Workloads (and Guided Exercise) • Examining Cluster Metrics (and Guided Exercise)
Lab	Managing a Cluster with the Web Console

Performing Cluster Administration

Objectives

After completing this section, you should be able to perform cluster administration with the web console.

Describing the Web Console

The Red Hat OpenShift web console provides a graphical user interface to perform administrative, management, and troubleshooting tasks. It supports both **`Administrator`** and **`Developer`** perspectives. This course explores the **`Administrator`** perspective.

The following list outlines some of the most important parts of the web console, grouped by the main navigation menu items.

Home
: The dashboard provides a quick overview of the cluster, including health metrics, resource counts, and a streaming list of events, such as machine updates or pod failures.

  You can navigate to **Home** → **Search** page to find or create resources of any type. This is also a useful starting point to navigate to resources that do not have dedicated navigation in the menu, such as Groups.

  The **Home** → **Events** page displays a filterable stream of events that occur in the cluster and is a good starting point for troubleshooting.

Operators
: Explore and install operators curated by Red Hat using **`OperatorHub`**, then navigate to the **`Installed Operators`** page to managed the operators.

Workloads, Networking, and Storage
: Manage common resources such as deployments, services, and persistent volumes. Of particular interest for troubleshooting is the ability to view pod logs and connect to a terminal.

Builds
: Manage build configurations, builds, and image streams.

Monitoring
: View alerts and perform ad hoc Prometheus queries.

Compute
: View and manage compute resources such as nodes, machines, and machine autoscalers.

Administration
: View and manage a wide variety of settings that are of particular interest to cluster administrators, such as cluster updates, cluster operators, CRDs, role bindings, and resource quotas.

Accessing the OpenShift Web Console

The OpenShift web console runs as pods in the **openshift-console** project and is managed by an operator running in the **openshift-console-operator** project. You can discover the URL by listing the route.

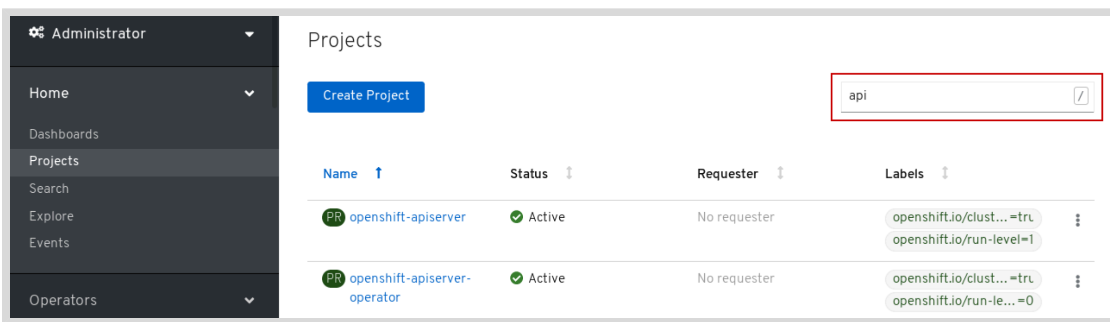
```
[student@workstation ~]$ oc get routes -n openshift-console
NAME       HOST/PORT                                      ...  PORT   ...
console    console-openshift-console.apps.cluster.example.com    ...  https  ...
downloads  downloads-openshift-console.apps.cluster.example.com  ...  http   ...
```

In non-production systems, self-signed certificates are commonly used for the HTTPS endpoint. Web browsers will warn you about the certificate, and you will need to add a security exception when navigating to the web console for the first time.

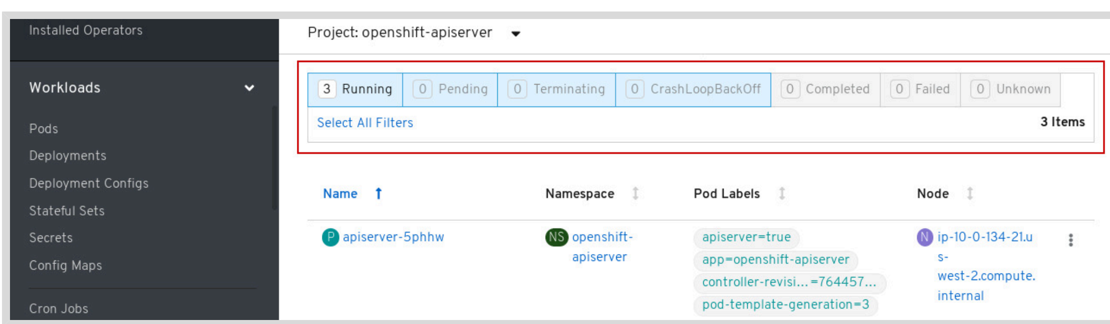
Finding Resources

The web UI provides multiple ways to locate resources. Many common resources, such as **Deployments** and **Services**, are available in the main menu on the left. You can use the **Home** → **Search** page to find other resource types. This page provides a complete menu of resource types and a label selector field.

Use the name filter to quickly locate resources on pages with long lists such as the **Projects** page.



It may be useful to filter pods by state to identify potential issues or problematic deployments.



The details page of a resource displays common useful information. The contents of this page vary for different types. For example, the **Pod Details** page displays metrics and status information and the **Secret Details** page allows you to reveal or copy data stored in the secret.

Detail pages provide a YAML editor to view and modify the resource specification from the web console. Some resource types, such as **secrets** and **role bindings**, provide more advanced UIs tailored to the resource type.

Creating Users and Groups

The **secrets** editor and similar tools can make it quicker and easier to manage users and groups than using the **oc** command.

As discussed in Chapter 3, *Configuring Authentication*, OpenShift supports a variety of identity providers (IdPs), including LDAP and OpenID Connect. This course discusses HTPasswd, which is a simple identity provider that uses a flat file stored as a secret.

After using a terminal to generate the HTPasswd entry, switch to the web console to append the entry to the secret. In the web UI, locate the secret in the **openshift-config** project and then click **Actions** → **Edit Secret**. The **Edit Key/Value Secret** tool handles the base64 encryption for you.

Groups do not have a specialized page in the web console, nor do they have a menu option to navigate directly to them. The easiest way to create a group is to select **Group** from the resource type list on the **Search** page and then click **Create Group** to use the YAML editor.

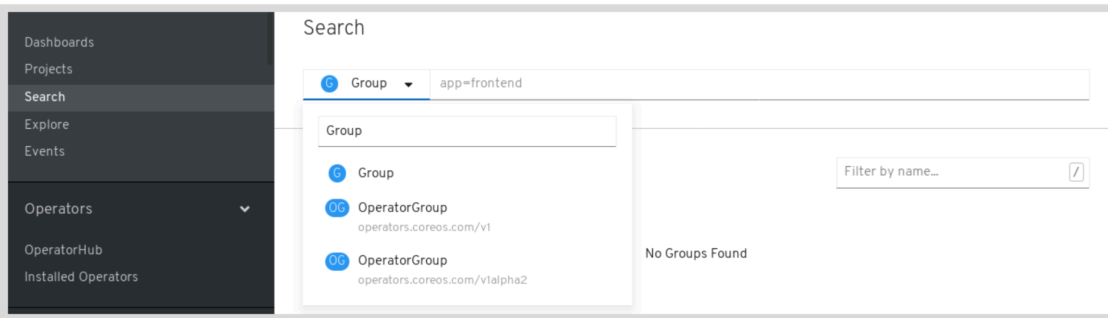


Figure 9.3: Creating a group from the Search page.

Creating a Project

The web UI features a variety of pages and forms for configuring projects.

1. Navigate to the **Home** → **Projects** page to display the full list of projects. Click **Create Project** and complete the form to create a new project.

2. After you have created your new project, you can navigate to the **Role Bindings** tab on the project details page.

3. Red Hat recommends that administrators responsible for multitenant clusters configure **Resource Quotas** and **Limit Ranges**, which enforce total project limits and container limits, respectively. Navigate to either **Administration** → **Resource Quotas** or **Administration** → **Limit Ranges** to access the appropriate YAML editor, where you can configure these limits.

Discussing Limitations

The OpenShift web console is a powerful tool for graphically administrating OpenShift clusters, however some administrative tasks are not currently available in the web console. For example, viewing node logs and executing node debug sessions requires the **oc** command-line tool.

References

For more information, refer to the *Red Hat OpenShift Container Platform 4.2 Web Console* guide at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/web_console/index

Guided Exercise

Performing Cluster Administration

In this exercise, you will perform cluster administration with the web console.

Outcomes

You should be able to use the OpenShift web console to:

- Find resources associated with an operator.
- Review a pod's status, YAML definition, and logs.
- View and edit cluster configuration resources.
- Create a new project and configure its resource quotas, limit ranges, and role-based access control (RBAC).

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates the resources required for this exercise.

```
[student@workstation ~]$ lab console-admin start
```

1. As the **admin** user, locate and navigate to the OpenShift web console.

   1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. List the routes in the **openshift-console** namespace. Identify the **console** host and port.

```
[student@workstation ~]$ oc get routes -n openshift-console
NAME        HOST/PORT                                         ...  PORT   ...
console     console-openshift-console.apps.cluster.example.com    ...  https  ...
downloads   downloads-openshift-console.apps.cluster.example.com  ...  http   ...
```

1.4. Echo the value of **`${RHT_OCP4_USER_PASSWD}`** to retrieve the **`admin`** password.

```
[student@workstation ~]$ echo ${RHT_OCP4_USER_PASSWD}
...output omitted...
```

1.5. Open a web browser and navigate to the **`console-openshift-console.apps.`*`cluster.example`*`.com`** console route URL discovered in a previous step.

1.6. Click **Advanced** to reveal the untrusted certificate message, and then click **Add Exception**. In the **Add Security Exception** dialog box, click **Confirm Security Exception**.

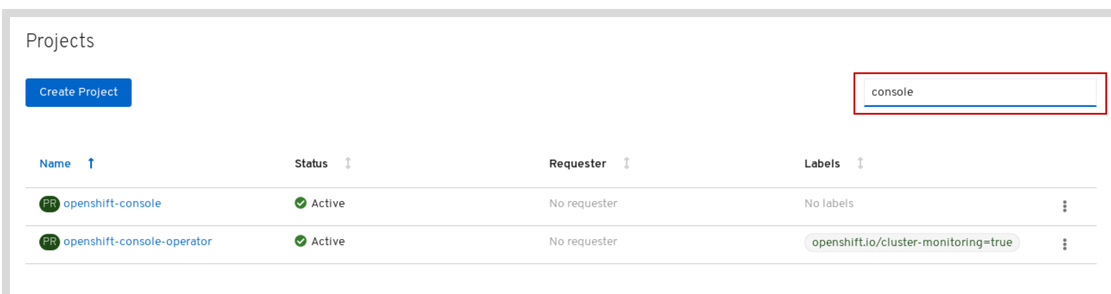
You need to do this twice to skip the warnings about the self-signed SSL certificate for both the **`console-openshift-console`** and **`oauth-openshift`** subdomains.

1.7. Click **localusers** and log in as the **`admin`** user with the password from the **`${RHT_OCP4_USER_PASSWD}`** variable.

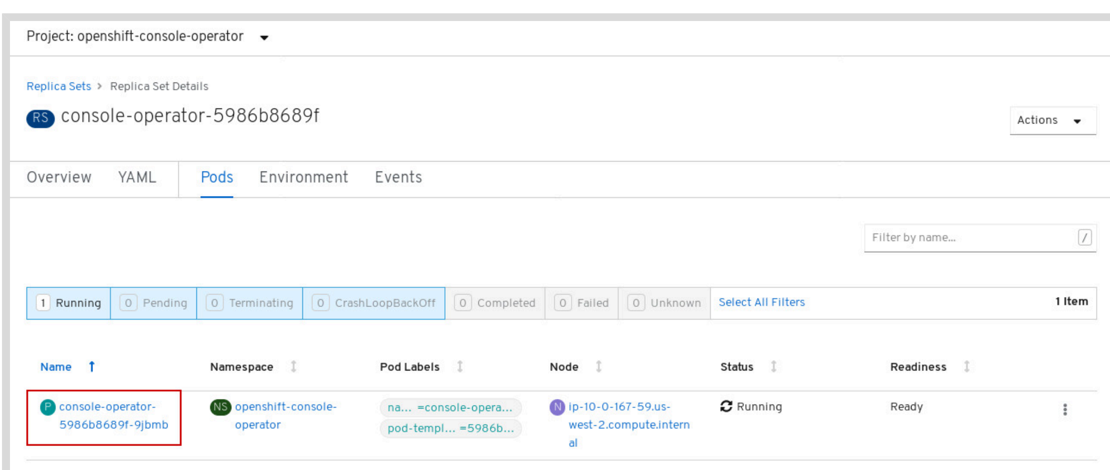
2. Review the **`openshift-console-operator`** and **`openshift-console`** pod logs.

2.1. In the Red Hat OpenShift Container Platform web UI, click **Home** → **Projects** to display the **Projects** page.

2.2. Type **`console`** in the **Filter by name** field and then click the **`openshift-console-operator`** link.



2.3. Click **Workloads** and then click **1 of 1 pods** to navigate to the **`console-operator`** replica set. Click the pod name marked by the P icon to navigate to the **`console-operator`** pod.

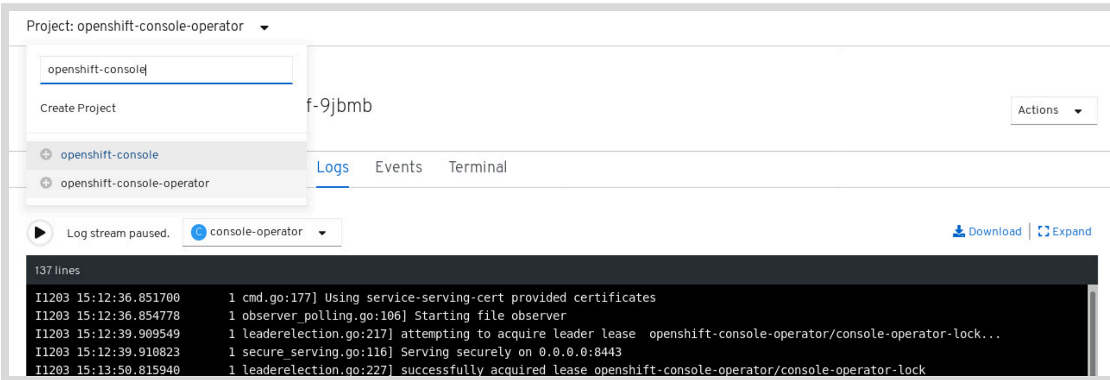


2.4. Review the pod details **Overview** page and notice the pod metrics, running status, and volumes. Click **YAML** to navigate to the pod resource editor.

2.5. Click **View Schema** to see resource documentation.

2.6. Click **Logs** to view the console operator logs.

2.7. Open the **Project** list and type **`openshift-console`** to switch to the **`openshift-console`** project.



2.8. Click the first pod in the table and then click **Logs** to view the console pod logs.

3. Review the Console, Image, and OAuth cluster settings.

3.1. Click **Administration** → **Cluster Settings** to view the **Cluster Settings** page. Notice that information about the cluster's update channel and current version are listed at the top and a section for the cluster's update history is listed further below.

3.2. Click **Global Configuration** to navigate to the list of cluster configuration resources.

3.3. Click **Console** and then click **YAML** to review the **`Console`** resource.

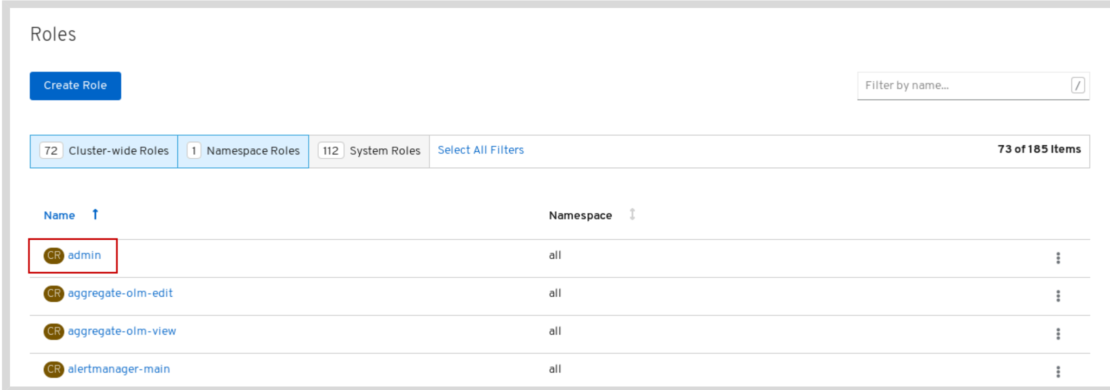
3.4. Return to the **Cluster Settings** Global Configuration page. Click **Image** and then click **YAML**. Notice the **`internalRegistryHostname`** is configured to use the internal image registry.

3.5. Return to the **Cluster Settings** Global Configuration page and click **OAuth**. The OAuth overview page has a special section for listing and adding identity providers. Navigate to the YAML page to view additional configuration details.

4. Review the **`admin`**, **`edit`**, and **`view`** cluster roles.

4.1. Click **Administration** → **Roles** to view the **Roles** page.

4.2. Click **admin** next to the CR icon. Review the **Rules** table which describes the allowed actions for various resources.



4.3. Return to the **Cluster Roles** page and click the cluster role named **edit** to view the **edit** cluster role details.

4.4. Return to the **Cluster Roles** page and type **view** in the **Filter by name** field. Click the cluster role named **view** to navigate to the **view** cluster role details. Notice that this role only allows **get**, **list**, and **watch** actions on the listed resources.

5. Add a **tester** user entry to the **localusers** secret.

5.1. In the OpenShift Container Platform web UI, click **Workloads** → **Secrets** and then select **openshift-config** from the **Project** filter list to display the secrets for the **openshift-config** project.

5.2. Use the filter or scroll to the bottom of the page to locate and then click the **localusers** link.

5.3. Click **Actions** → **Edit Secret** at the upper right of the page to navigate to the **Edit Key/Value Secret** tool.

5.4. Use the **workstation** terminal to generate an htpasswd entry using the **${RHT_OCP4_USER_PASSWD}** password.

```
[student@workstation ~]$ htpasswd -n -b tester ${RHT_OCP4_USER_PASSWD}
tester:$apr1$oQ3BtWOp.HtW97.$wVbJBofBNsNd4sd
```

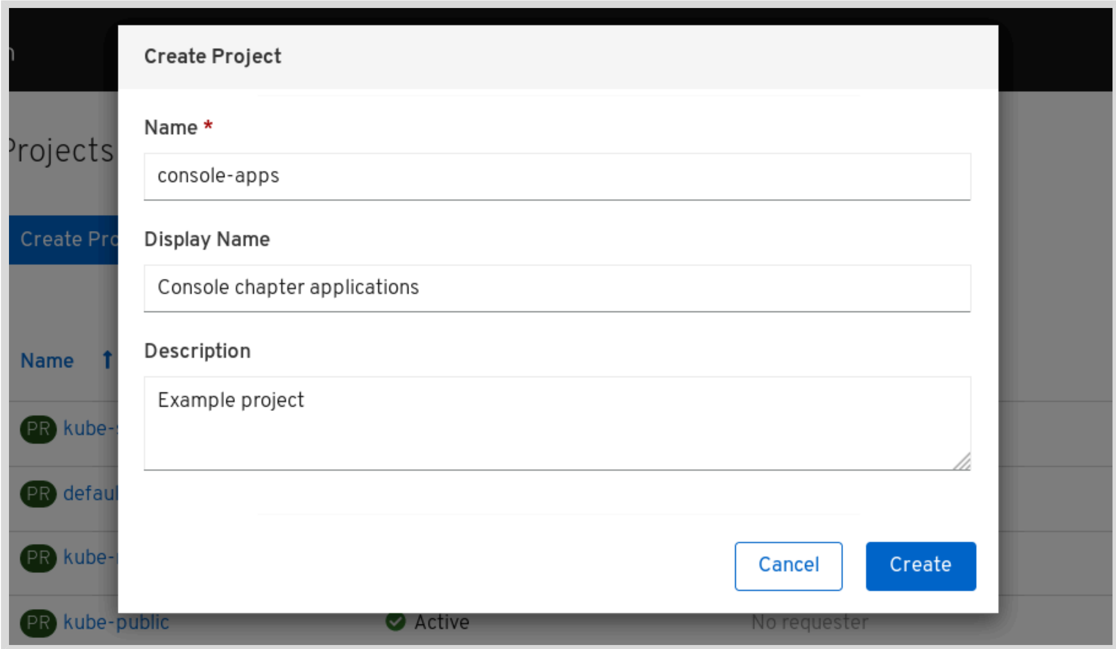
5.5. Append the terminal output from the **htpasswd** command to the **htpasswd** value in the OpenShift web console's secrets editor, and then click **Save**.

```
admin:$apr1$Au9.fFr$0k5wvUBd3eeBt0baa77.dae
leader:$apr1$/abo4Hybn7a.tG5ZoOBn.QWefXckiy1
developer:$apr1$RjqTY4cv$xql3.BQfg42moSxwnTNkh.
tester:$apr1$oQ3BtWOp.HtW97.$wVbJBofBNsNd4sd
```

6. Create and configure a new project named **console-apps**.

6.1. Click **Home** → **Projects** to view the **Projects** page, and then click **Create Project**.

6.2. Type **console-apps** in the **Name** field, and then provide an optional **Display Name** and **Description**. Click **Create**.



6.3. Click **Administration** → **Resource Quotas** and then click **Create Resource Quota**. Modify the YAML document as follows:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: console-apps
spec:
  hard:
    pods: '10'
    requests.cpu: '2'
    requests.memory: 8Gi
    limits.cpu: '4'
    limits.memory: 12Gi
```

Click **Create**.

6.4. Click **Administration** → **Limit Ranges** and then click **Create Limit Range**. Modify the YAML document to specify default memory and CPU container limits and requests:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-range
  namespace: console-apps
spec:
```

```
  limits:
    - default:
        cpu: 500m
        memory: 5Gi
      defaultRequest:
        cpu: 10m
        memory: 100Mi
      type: Container
```

6.5. Click **Home** → **Search** and then select **Group** from the **Select Resource** list.

Click **Create Group** and use the editor to define a Group resource as follows:

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  name: project-team
users:
  - developer
  - tester
```

Click **Create** to create the new **project-team** group.

6.6. Click **Home** → **Projects** and then click **console-apps** to navigate back to the project. Click **Role Bindings**.

6.7. Click **Create Binding** and fill out the form as follows to create a role binding for the **project-team** group.

Team Role Binding Form

Field	Value
Binding Type	`Namespace Role Binding (RoleBinding)`
Name	`team`
Namespace	`console-apps`
Role Name	`edit`
Subject	`Group`
Subject Name	`project-team`

Click **Create** to create the namespaced RoleBinding.

6.8. Return to the **Role Bindings** page and click **Create Binding** to create a role binding for the **leader** user. Fill out the form as follows:

Leader Role Binding Form

Field	Value
Binding Type	`Namespace Role Binding (RoleBinding)`
Name	`leader`
Namespace	`console-apps`
Role Name	`admin`
Subject	`User`
Subject Name	`leader`

Click **Create** to create the namespaced RoleBinding.

6.9. Click **admin** → **Log out** and then log back in as the **`developer`** user. The password is the same as the **`admin`** password that you retrieved in the initial steps.

Ensure that the **`developer`** account can only see the **`console-apps`** project.

> **Note**
> Previous projects from guided exercises that were not deleted upon completion may also display in the list.

6.10. You will continue to use the new **`console-apps`** project in the next section, so you do not need to delete it.

Finish

On the **`workstation`** machine, use the **`lab`** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab console-admin finish
```

> **Important**
> Do not delete the **`console-apps`** project. It will be used in the upcoming sections.

This concludes the section.

Managing Workloads

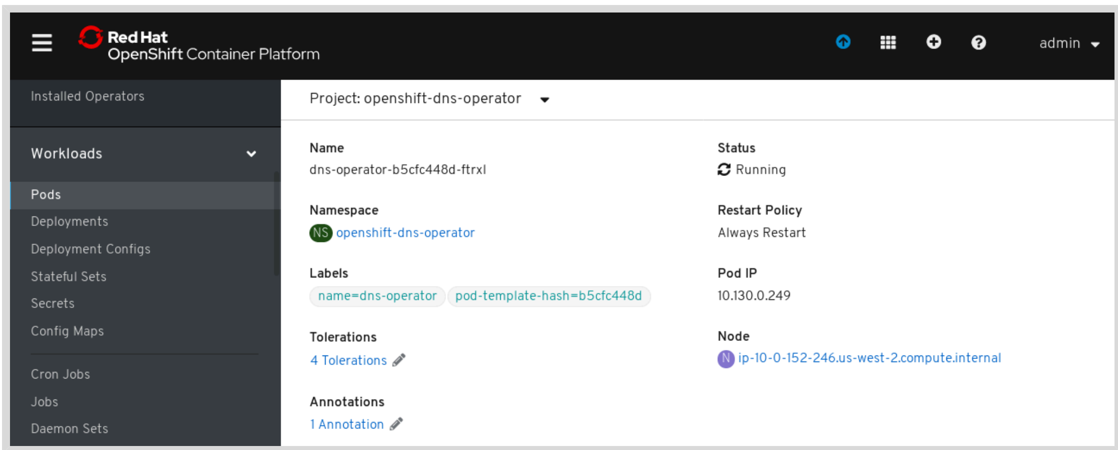
Objectives

After completing this section, you should be able to manage cluster workloads with the web console.

Exploring Workload Resources

Workload resources such as Pods, Deployments, Stateful Sets, and Config Maps are listed under the **Workloads** menu. Click a resource type to see a list of resources, and then click the name of the resource to navigate to the details page for that resource.

For example, to navigate to the OpenShift DNS operator pod, click **Workloads** → **Pods**, select **`opeshift-dns-operator`** from the Project list at the top of the page, and click the name of the pod listed in the table.



There are often multiple ways to navigate to common resources. Throughout the web UI, associated resources will often link to each other. The deployment details page displays a list of pods. Click any pod name in that list to display the pod details page for that pod.

Managing Workloads

The web console provides specialized editor pages for many workload resources. Use the **Actions** menu on the resource's details page to navigate to the specialized editor pages.

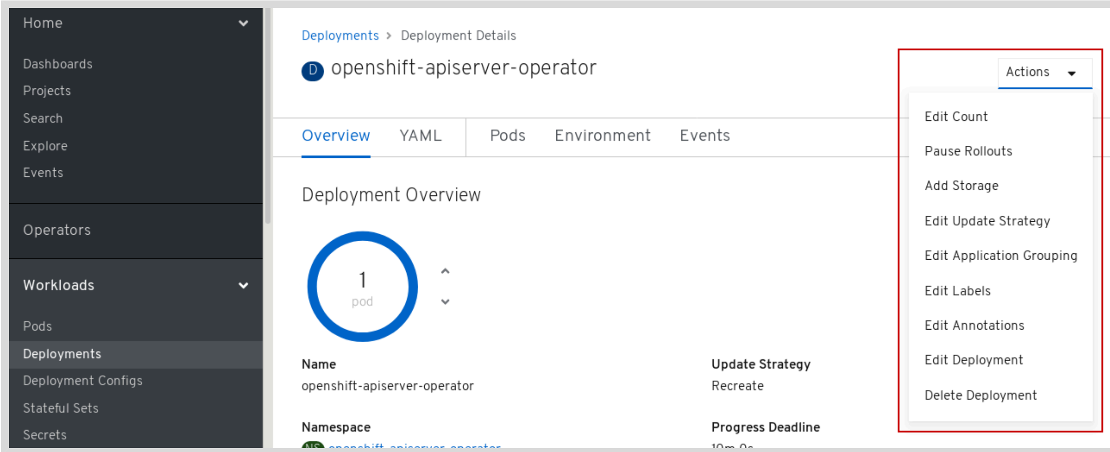


Figure 9.10: Using the Actions menu to modify a deployment.

Some useful action pages are described below:

- All resources feature **`Edit Labels`** and **`Edit Annotations`** editors.
- Click **Actions** → **Add Storage** to add a Persistent Volume Claim (PVC) to a deployment.
- To edit the replica count, navigate to the **Deployment Details** page and click **Actions** → **Edit Count**.
- To modify the rolling update parameters, navigate to the **Deployment Details** page and click **Actions** → **Edit Update Strategy**.
- Navigate to the **Secret Details** page and click **Actions** → **Edit Secrets** to display the **Edit Key/Value Secret** tool which automatically uses Base64 to encode and decode values.

You can also use the embedded YAML editor to create or modify workload resources. Drag and drop a JSON or YAML file into the browser-based editor to update the resource from a file without using the **oc** command.

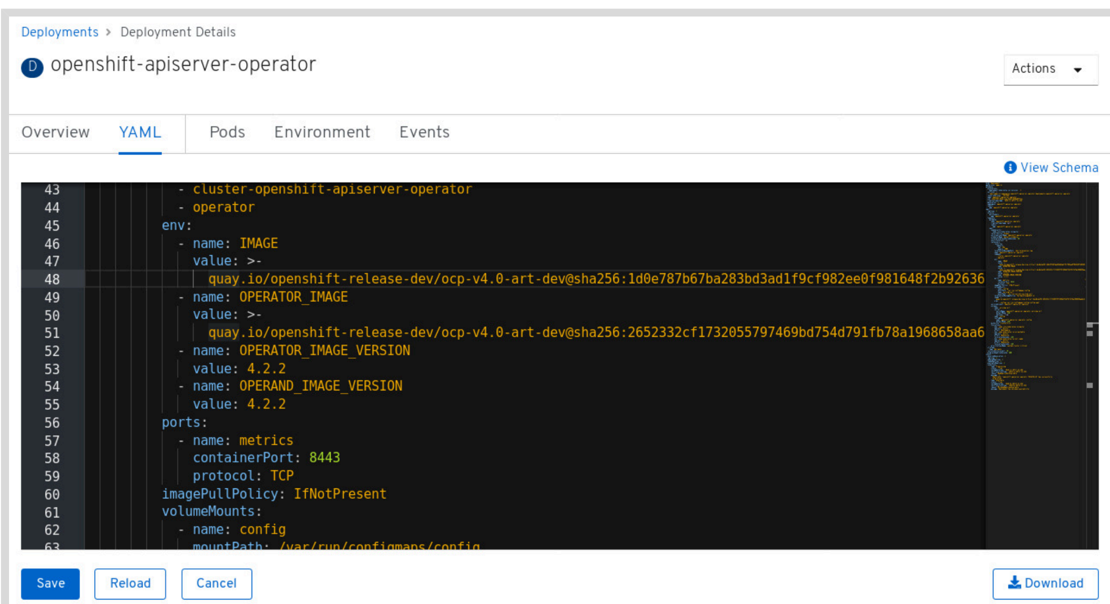


Figure 9.11: Editing a resource using the embedded YAML editor.

Along with the ability to edit resources in a dedicated page or the embedded YAML editor, you can perform many other common operations directly from the OpenShift web console.

For example, to delete a resource, navigate to the resource's details page and click **Actions** → **Delete** *Resource Type*.

There is often more than one way to perform a particular task. For example, to manually scale a deployment you can navigate to the **Deployment Details** page and then click **Actions** → **Edit Count**, or you can click the arrows next to the pod count without leaving the page.

Deploying Applications

You can create deployments and deployment configurations from the **Workloads** → **Deployments** and **Workloads** → **DeploymentConfigs** pages, respectively. Each of these sections provides a YAML editor with a prepopulated specification to define your YAML resource.

The **Builds** section contains tools for:

- Creating build configurations for Source-to-Image (S2I), Dockerfile, or custom builds.
- Listing and inspecting builds.
- Managing image streams.

After you initiate a deployment or build, use the resource's details and events pages to verify success, or start investigating the cause of a deployment failure.

Installing and Using Operators

Explore community and partner operators on the OpenShift web console's **Operators** → **OperatorHub** page. Over 150 operators are available for installation from the web UI. This includes community operators, which Red Hat does not support.

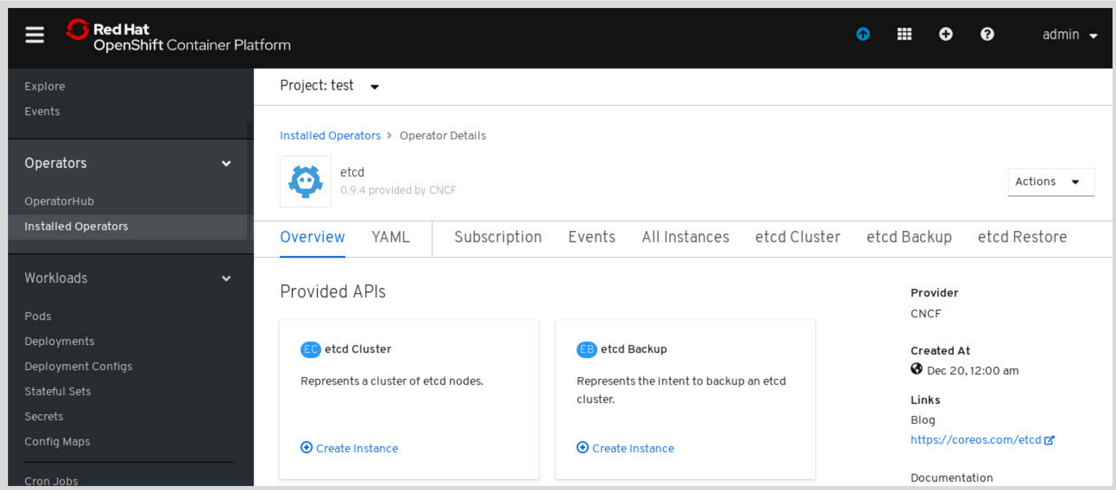
Operators add features and services to your cluster along with automation traditionally performed by human operators, such as deployment coordination or automatic backups. Operators cover a broad range of categories including:

- Traditional databases such as PostgreSQL and MySQL.
- Popular big data frameworks such as Apache Spark.
- Kafka-based streaming platforms such as Red Hat AMQ streams.
- The Knative serverless framework OpenShift Serverless Operator.

Click the operator listing to view details about the operator, such as its version and where to find documentation. After you have selected the operator to install, click **Install**.

Before installing an operator, complete the **Create Operator Subscription** form to select the target namespace and operator update strategy. You can install operators to target all or only specific namespaces. Be aware, however, that not all operators support all installation target options.

After you have installed an operator, it appears on the **Operators** → **Installed Operators** page. If it is installed for a specific namespace, make sure you select the correct project using the project filter at the top of the page.



The operator details page lists the APIs provided by the operator and allows you to create instances of those resources. For example, from the etcd operator page you can create instances of an etcd cluster, backup request, or restore request.

Guided Exercise

Managing Workloads

In this exercise, you will manage cluster workloads with the web console.

Outcomes

You should be able to use the OpenShift web console to:

- Install an operator from OperatorHub.
- Use a custom resource to create a database.
- Deploy and troubleshoot an application that uses the operator-managed resources.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates the resources required for this activity.

```
[student@workstation ~]$ lab console-workloads start
```

1. As the **admin** user, locate and navigate to the OpenShift web console.

    1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

    1.2. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>     ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

    1.3. List the routes in the **openshift-console** namespace. Identify the **console** host and port.

```
[student@workstation ~]$ oc get routes -n openshift-console
NAME       HOST/PORT                                           ...  PORT   ...
console    console-openshift-console.apps.cluster.example.com   ...  https  ...
downloads  downloads-openshift-console.apps.cluster.example.com ...  http   ...
```

    1.4. Echo the value of **${RHT_OCP4_USER_PASSWD}** to retrieve the **admin** password.

```
[student@workstation ~]$ echo ${RHT_OCP4_USER_PASSWD}
...output omitted...
```

1.5. Open a web browser and navigate to the console route URL discovered in a previous step.

1.6. Click **Advanced** to reveal the untrusted certificate message, and then click **Add Exception**. In the **Add Security Exception** dialog box, click **Confirm Security Exception**.

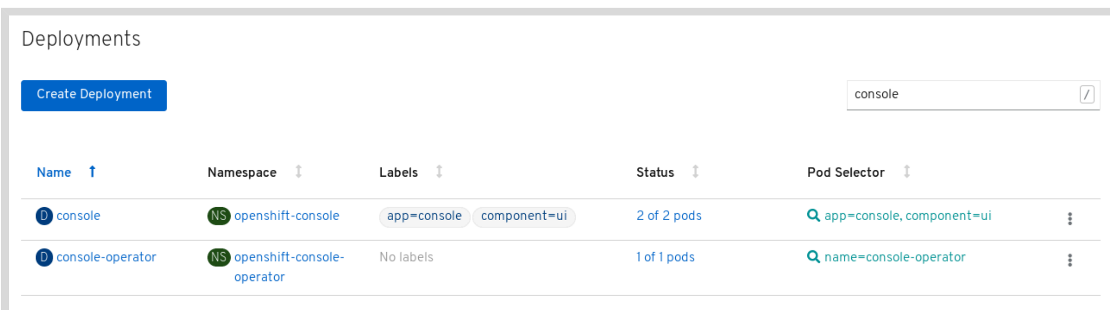
You will need to do this twice to skip the warnings about the self-signed SSL certificate for both the **`console-openshift-console`** and **`oauth-openshift`** subdomains.

1.7. Click **localusers** and log in as the **`admin`** user with the password from the **`${RHT_OCP4_USER_PASSWD}`** variable.

2. Inspect the **`openshift-console-operator`** and **`openshift-console`** deployments, replica sets, and pods.

2.1. Click **Workloads** → **Deployments** and select **`all projects`** from the project list at the top. Type **`console`** in the **Filter by name** field.

Notice that OpenShift has a deployment named **`console-operator`** with a single pod in the **`openshift-console-operator`** namespace, which operates a deployment named **`console`** in the **`openshift-console`** namespace.



2.2. Click **Workloads** → **Replica Sets** and type **`console`** in the **Filter by name** field.

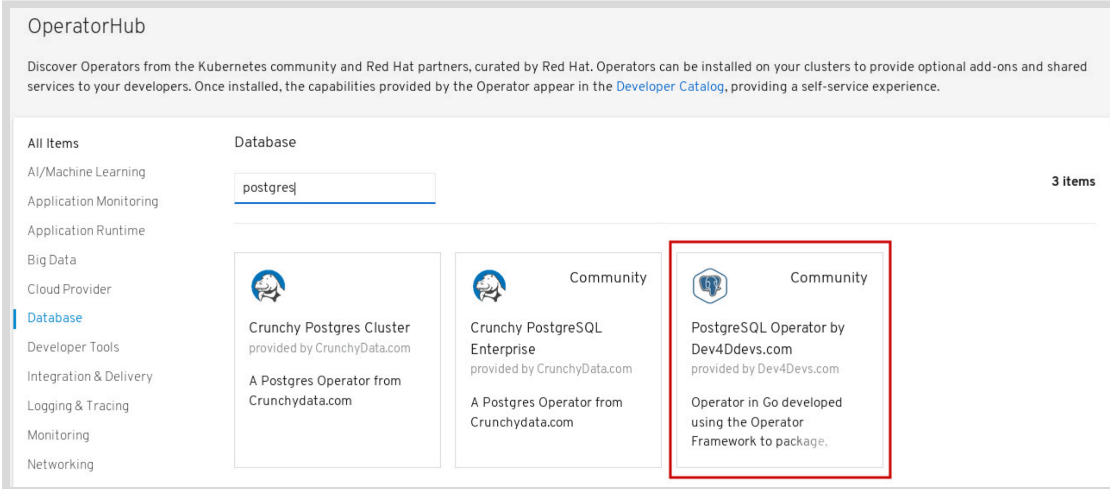
Deployments declare a **`ReplicaSet`** to ensure that a specified number of pods are always running.

2.3. In the status column, click **2 of 2 pods** to display the console **`ReplicaSet`** pod list.

3. Install the community PostgreSQL operator provided by Dev4Devs.com from the **OperatorHub** page.

3.1. Click **Operators** → **OperatorHub** and then click **Database** to display the list of database operators available from OperatorHub.

3.2. Type **`postgres`** in the **Filter by keyword** field, and then click **PostgreSQL Operator by Dev4Ddevs.com**. Click **Continue** to view the community operator page, and then click **Install**.



3.3. Select the **console-apps** namespace and then click **Subscribe** to install the operator for use in the **console-apps** project. Leave the other form fields unchanged.

4. Log out as the **admin** user and log in as the **developer** user.

4.1. Click **admin** → **Log out**.

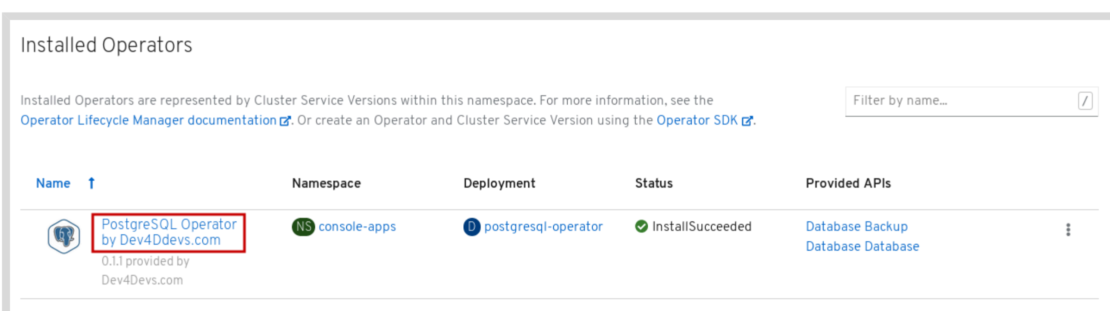
4.2. Log in as the **developer** user. The password is the same as the **admin** password that you retrieved in the initial steps.

5. Provision a PostgreSQL database using the installed operator and **Database** Custom Resource Definition (CRD).

5.1. On the **Projects** page, click the **console-apps** link to see the resources associated with the **console-apps** project.

5.2. Click **Operators** → **Installed Operators**, and then click the **PostgreSQL Operator by Dev4Devs.com** link to display the **Operator Details** page.

> **Note**
> If the **Installed Operators** list does not load, make sure that the **console-apps** project is selected at the top of the page.



5.3. Click the **Database Database** tab and then click **Create Database**.

5.4. Update the **Database** YAML to specify the PostgreSQL image provided by Red Hat. Do not change the other default values.

```
apiVersion: postgresql.dev4devs.com/v1alpha1
kind: Database
metadata:
  name: database
  namespace: console-apps
spec:
  databaseCpu: 30m
  databaseCpuLimit: 60m
  databaseMemoryLimit: 512Mi
  databaseMemoryRequest: 128Mi
  databaseName: example
  databaseNameKeyEnvVar: POSTGRESQL_DATABASE
  databasePassword: postgres
  databasePasswordKeyEnvVar: POSTGRESQL_PASSWORD
  databaseStorageRequest: 1Gi
  databaseUser: postgres
  databaseUserKeyEnvVar: POSTGRESQL_USER
  image: registry.redhat.io/rhscl/postgresql-96-rhel7:1-51
  size: 1
```

5.5. Click **Create** to add the **Database** resource. The PostgreSQL operator will read the specification and automatically create the workload, network, and storage for the new database.

6. Review the resources created by the operator.

6.1. Click **Workloads** → **Deployments** and inspect the list of deployments. You will notice a **database** deployment and a **postgresql-operator** deployment.

6.2. Click the **database** deployment, and then click the **Pods** tab to see the pod deployed by the **database** deployment. Click the pod name to display the **Pod Details** page.

6.3. Click **Networking** → **Services** and then click the **database** service name to see the details of the service created by the PostgreSQL operator.

6.4. Click **Storage** → **Persistent Volume Claims** and then click the **database** PVC to see the details of the Persistent Volume Claim created by the PostgreSQL operator.

7. Create a **deployment**, **service**, and **route** for a simple web application. The application will display a list of books stored in the database.

7.1. Click **Workloads** → **Deployments** and then click **Create Deployment** to display the web console YAML editor. Update the YAML as follows and then click **Create**.

> **Note**
> You can copy the YAML from the **~/DO280/labs/console-workloads/deployment.yaml** file on **workstation**.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: books
  namespace: console-apps
spec:
  replicas: 1
  selector:
    matchLabels:
      app: books
  template:
    metadata:
      labels:
        app: books
    spec:
      containers:
        - name: books
          image: 'quay.io/redhattraining/books:v0.9'
          ports:
            - containerPort: 8080
              protocol: TCP
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8080
          env:
            - name: DB_HOST
              value: database.console-apps.svc.cluster.local
            - name: DB_PORT
              value: '5432'
            - name: DB_USER
              value: postgres
            - name: DB_PASSWORD
              value: postgres
            - name: DB_NAME
              value: postgres
```

Important

Do not expect the pods to run successfully after completing this step. You will troubleshoot the deployment issue later in this exercise.

7.2. Click **Networking** → **Services** and then click **Create Service** to display the web console YAML editor. Update the YAML as follows and then click **Create**.

Note

You can copy the YAML from the **`~/DO280/labs/console-workloads/service.yaml`** file on **`workstation`**.

```
kind: Service
apiVersion: v1
metadata:
  name: books
  namespace: console-apps
spec:
  selector:
    app: books
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

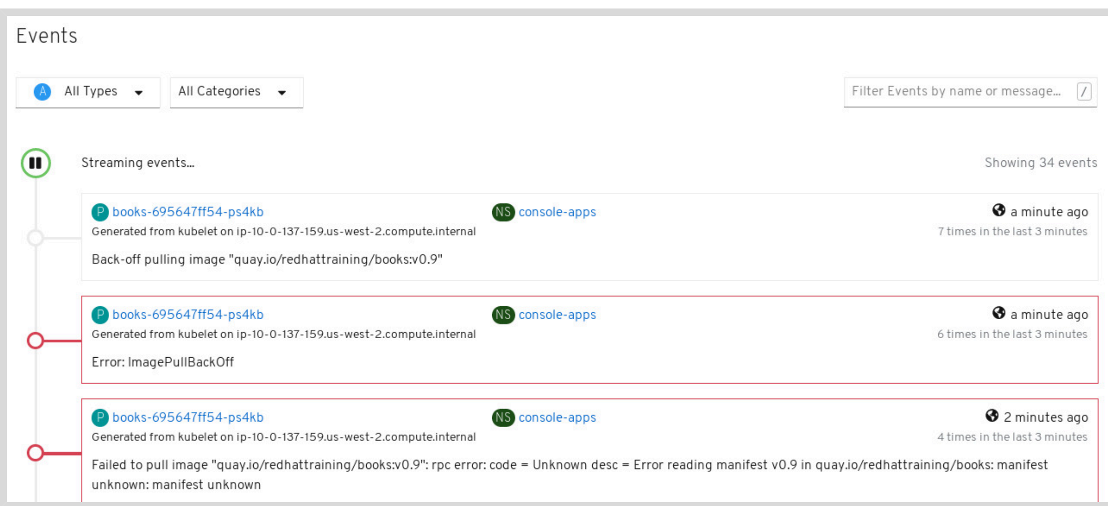
7.3. Click **Networking** → **Routes** and then click **Create Route**. Complete the page as follows, leaving the other fields unchanged, and then click **Create**.

Create Route Form

Field	Value
Name	**books**
Service	**books**
Target Port	**`8080 → 8080 (TCP)`**

8. Troubleshoot and fix the deployment issue.

8.1. Click **Home** → **Events** and notice the error events. Messages such as **`Failed to pull image "quay.io/redhattraining/books:v0.9"`** and **`Error: ImagePullBackOff`** indicate an issue with the image name or image tag.



8.2. Click **Workloads** → **Deployments** and then click the **books** deployment. Scroll to the bottom of the page to inspect the **Conditions** table. Notice that the **`Available`** condition type displays a **`False`** status.

Conditions

Type	Status	Updated	Reason	Message
Available	False	4 minutes ago	MinimumReplicas Unavailable	Deployment does not have minimum availability.
Progressing	True	4 minutes ago	ReplicaSetUpdated	ReplicaSet "books-695647ff54" is progressing.

8.3. Click the **Pods** tab at the top of the **Deployment Details** screen and locate the pod status. It displays **`ImagePullBackOff`**.

8.4. Click the **YAML** tab at the top of the **Deployment Details** page to navigate to the YAML editor and fix the issue. Update the **spec** image value to **`'quay.io/redhattraining/books:v1.4'`** and then click **Save**.

> **Note**
> When OpenShift updates a deployment resource while you are attempting to update it, the YAML editor will not allow you to save your changes without fetching the latest version first. If this happens, click **Reload**, perform the edit again, and then click **Save**.

8.5. Click the **Overview** tab at the top of the **Deployment Details** page and wait until the donut indicates that one pod is running.

8.6. Click **Networking** → **Routes** and then click the link in the **`Location`** column. Firefox will open a new tab rendering a list of books that were fetched from the database.

8.7. You will continue to use the new **`console-apps`** project and **`books`** deployment in the next section, so you do not need to delete them.

Finish

On the **`workstation`** machine, use the **`lab`** command to complete this exercise.

```
[student@workstation ~]$ lab console-workloads finish
```

> **Important**
> Do not delete the **`console-apps`** project or any of the work you performed in this section. It will be used in the next section.

This concludes the section.

Examining Cluster Metrics

Objectives

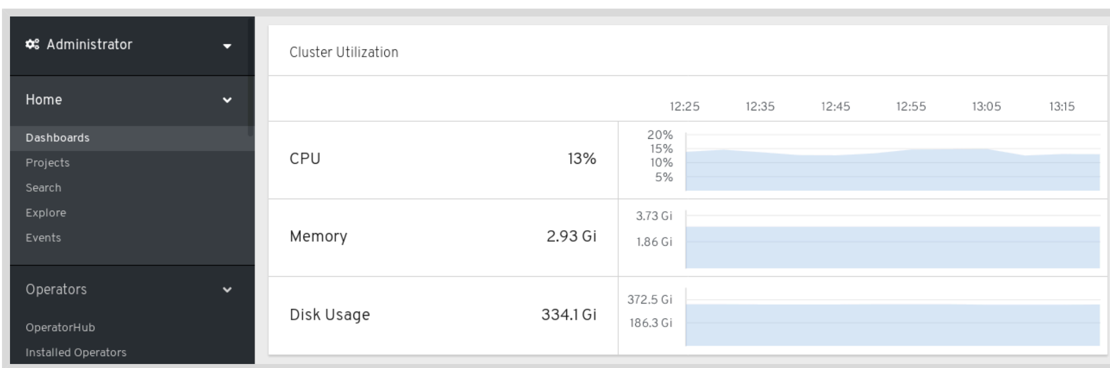
After completing this section, you should be able to examine the metrics page and dashboard within the web console.

Viewing Cluster Metrics

The OpenShift web console incorporates useful graphs to visualize cluster and resource analytics. The **Home** → **Dashboard** page, which displays a collection of cluster-wide metrics, provides a high-level view of the overall health of the cluster.

The dashboard includes the following metrics:

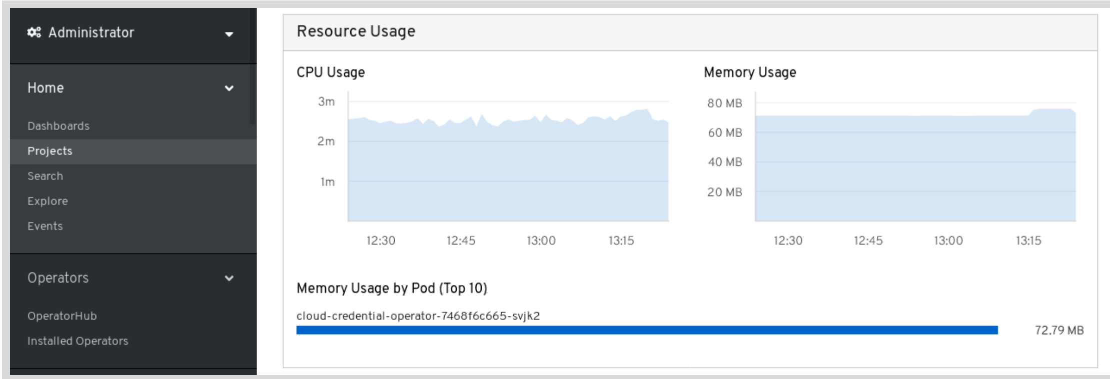
- Current cluster capacity based on CPU, memory, storage, and network usage.
- A time-series graph of total CPU, memory, and disk utilization.
- A list of top consuming nodes and pods by CPU, memory, or storage.



The **Top Consumers** page can be useful for identifying problematic pods or nodes. For example, a pod with an unexpected memory leak may appear on the top of the list.

Viewing Project Metrics

The **Project Details** page displays metrics that provide an overview of the resources used within the scope of a specific project. The **Resource Usage** page displays CPU and memory usage along with a top ten list of memory usage by pod.



All metrics are pulled from Prometheus. Click any graph to navigate to the Prometheus UI, view the executed query, and inspect the data further.

If a resource quota is created for the project, the current project request and limits appear on the **Project Details** page.

Viewing Resource Metrics

When troubleshooting, it is often useful to view metrics at a smaller granularity than the entire cluster or whole project. The **Pod Details** page displays time-series graphs of the CPU, memory, and file system usage for a specific pod.

A sudden change in these critical metrics, such as a CPU spike caused by high load, will be visible on this page.

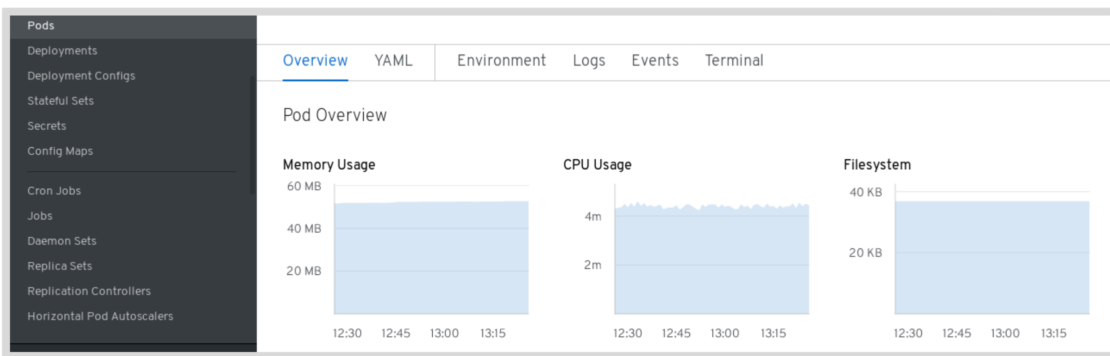


Figure 9.20: Time-series graphs showing various metrics for a pod.

Performing Prometheus Queries in the Web Console

The Prometheus UI is a feature-rich tool for visualizing metrics and configuring alerts. The OpenShift web console provides an interface for executing Prometheus queries directly from the web console.

To perform a query, navigate to the **Monitoring** → **Metrics** page, enter a Prometheus Query Language expression in the text field, and click **Run Queries**. The results of the query display as a time-series graph.

Figure 9.21: Using a Prometheus query to display a time-series graph.

Note

The Prometheus Query Language is not discussed in detail in this course. See the references below for a link to the official documentation.

References

For more information, refer to the *Red Hat OpenShift Container Platform 4.2 Monitoring* guide at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html-single/monitoring/index

Querying Prometheus
https://prometheus.io/docs/prometheus/latest/querying/basics/

Guided Exercise

Examining Cluster Metrics

In this exercise, you will examine the metrics page and dashboard within the web console.

Outcomes

You should be able to use the Red Hat OpenShift web console to:

- View cluster, project, pod, and node metrics.
- Identify a pod consuming large amounts of memory or CPU.

Before You Begin

On the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates the resources required for this exercise.

```
[student@workstation ~]$ lab console-metrics start
```

1. Open a web browser and navigate to the OpenShift web console. Log in as the **admin** localuser.

> **Note**
> Locating and accessing the OpenShift web console is discussed in detail in Guided Exercise: Performing Cluster Administration.

2. In this guided exercise, you will see how changes in load are displayed in the web console. Start by observing baseline healthy metrics on the Dashboard, Pod Details, and Project Details pages.

   2.1. Click **Home** → **Dashboards** to display the **Dashboards Overview** page. Notice the **Cluster Capacity** page, which displays the total available CPU, memory, storage, and network across the entire cluster. Scroll down to the **Cluster Utilization** section, which displays a time-series historical graph of the cluster's CPU, memory, and disk usage.

   2.2. In the **Top Consumers** pane select **By Memory** from the metrics list at the top. The pane will refresh its list of top consumers and display the pods consuming the most memory in the cluster.

   2.3. Click the top pod in the **Pods by Memory consumption** list to display the **Pod Details** page. Notice the **Memory Usage**, **CPU Usage**, and **Filesystem** time-series historical graphs at the top of the page.

   2.4. Click **Home** → **Projects** and then click **console-apps** to display the **console-apps Project Details** page.

Notice the **quota** resource quota section, which displays the current CPU and memory usage compared to the allotted quota. The workloads are running safely within their limits.

2.5. Scroll down to the **Resource Usage** section, which displays the metrics for the workloads running in the **`console-apps`** project.

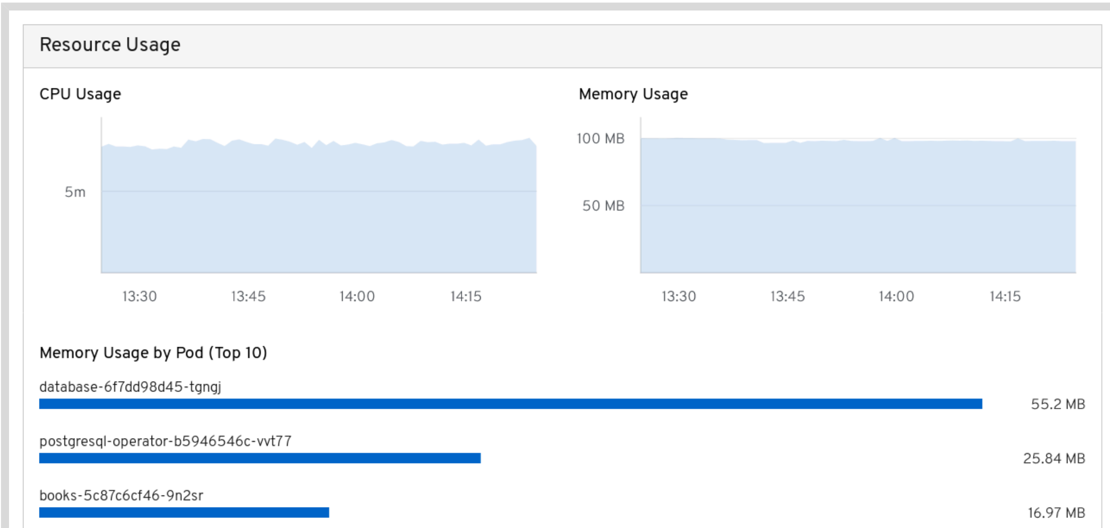


Figure 9.22: Time-series graphs showing CPU and memory usage for the selected project.

3. Find and review the baseline health metrics of a compute node.

3.1. Click **Compute** → **Nodes**, then click any of the nodes in the list.

3.2. On the **Node Overview** page, notice the time-series graphs that display the metrics for the individual node that you selected.

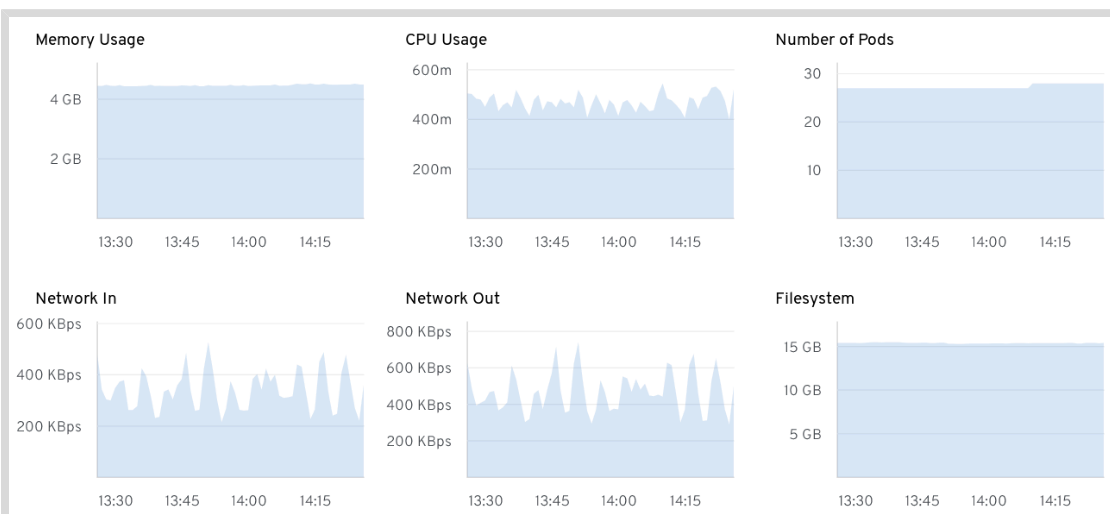


Figure 9.23: Time-series graphs showing various metrics for a node.

4. On **`workstation`**, execute the **`load.sh`** script to generate load on the example **books** deployment. The application intentionally contains a memory leak that consumes multiple megabytes of RAM with every request to its **`/leak`** path.

4.1. In a terminal on the **`workstation`** machine, run the following command.

```
[student@workstation ~]$ ~/DO280/labs/console-metrics/load.sh
```

5. In the OpenShift web console, observe the change in metrics and identify the problematic pod. The data displayed in the web console automatically refreshes, so there is no need to reload the page.

   5.1. Click **Home** → **Projects** and then click **console-apps** to display the **console-apps Project Details** page. Watch the **Memory Usage** time-series graph and the **Memory Usage by Pod (Top 10)** monitor for changes.

   The memory leak may take a minute or two before it is significant enough to be visible.

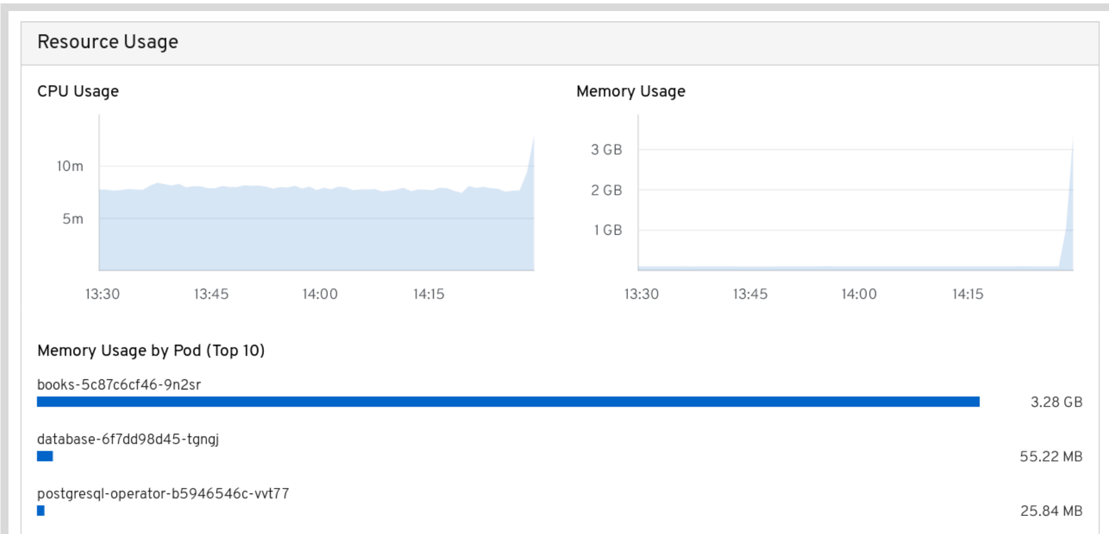
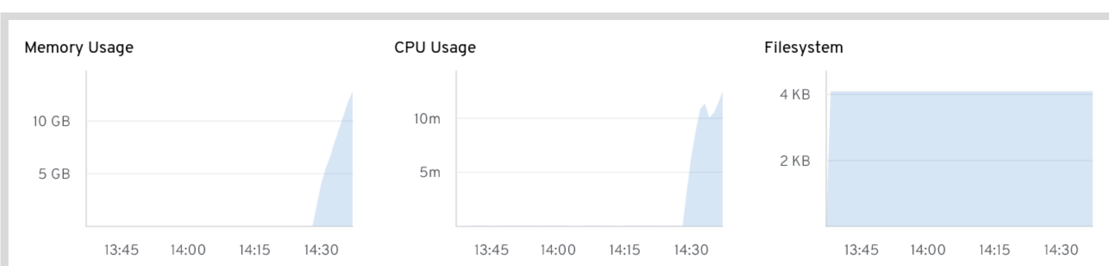


Figure 9.24: Resource usage graphs indicating a possible memory leak.

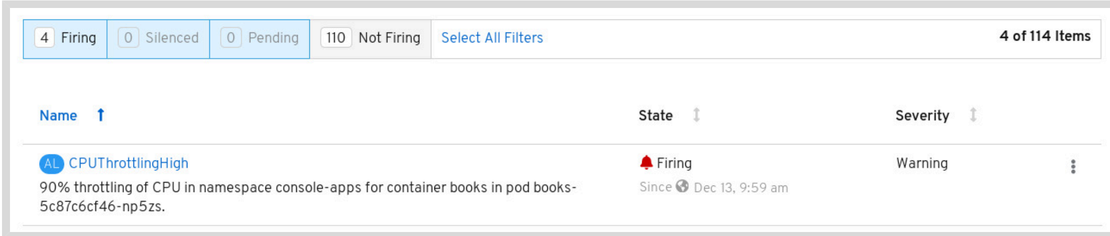
   5.2. Click **Home** → **Dashboards** to display the **Dashboards Overview** page. The memory consumed by the load test may be too small to notice across a large cluster, but the **Top Consumers** section provides a convenient list of pods using the most CPU or memory. Locate the **Top Consumers** section and select **`By Memory`** to display the **Pods by Memory consumption** list.

   The **books** pod appears at the top of the list. If its not on the list, you may need to wait a minute longer for the load script to complete.

   5.3. Click the **books** pod link in the **Top Consumers** section to navigate to the **Pod Details** page. Notice the climbing memory leak visible in the **Memory Usage** time-series graph.



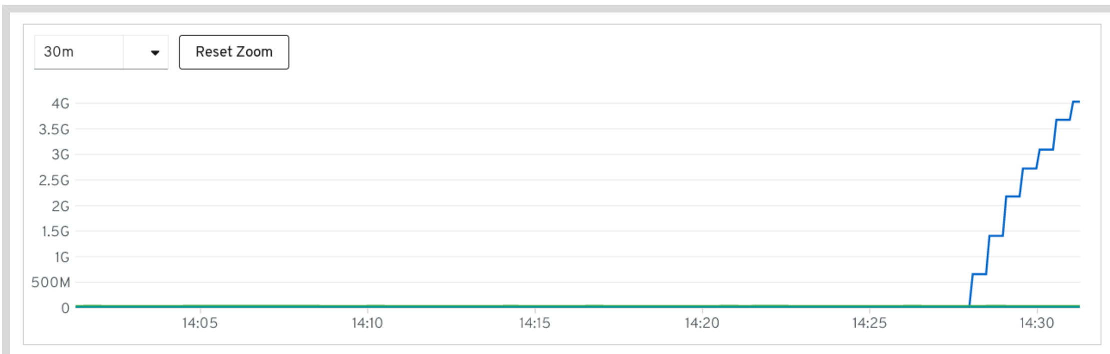
   5.4. Click **Monitoring** → **Alerting** to display the web console **Alerting** page. Observe the increased CPU alert caused by the high traffic load and low CPU limit.



5.5. Click **Monitoring** → **Metrics** to display the web console **Metrics** page. Type the following Prometheus query in the expression input field:

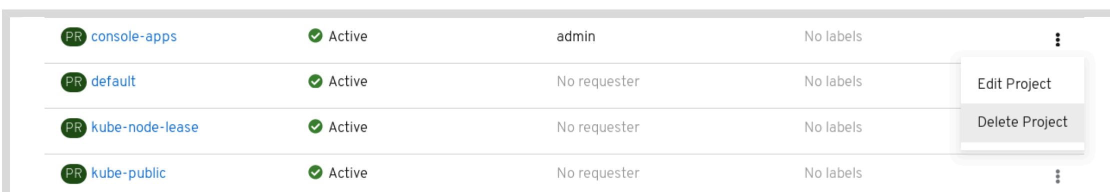
```
avg(container_memory_working_set_bytes{namespace='console-apps'}) BY (pod)
```

Click **Run Queries** to view the results in the OpenShift web console.



6. Delete the **console-apps** project and stop the load test.

6.1. Click **Home** → **Projects** and then click **Delete Project** in the menu at the end of the **console-apps** row.



6.2. In the **Delete Project** dialog box, type **console-apps** and then click **Delete**.

6.3. If **load.sh** is still running on the workstation terminal, press **Ctrl+C** in the terminal to stop the load test.

Finish

On the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab console-metrics finish
```

This concludes the section.

Lab

Managing the Cluster with the Web Console

In this lab, you will manage the OpenShift cluster using the web console.

Outcomes

You should be able to use the OpenShift web console to:

- Modify a secret to add htpasswd entries for new users.
- Configure a new project with role-based access controls and resource quotas.
- Use an OperatorHub operator to deploy a database.
- Create a deployment, service, and route for a web application.
- Troubleshoot an application using events and logs.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates a directory for the exercise files.

```
[student@workstation ~]$ lab console-review start
```

1. Log in to the OpenShift web console as the **admin** user.
2. Add **htpasswd** entries to the **localusers** secret for users named **dba** and **tester** using the **${RHT_OCP4_USER_PASSWD}** password.
3. Create a new **app-team** group that contains the **developer** and **dba** users.
4. Create a new **console-review** project with a **view** role binding for the **tester** user and an **edit** role binding for the **app-team** group. Set a resource quota that limits the project to three pods.
5. Install the Community CockroachDB operator for use in all namespaces.
6. Create a RoleBinding that allows the **dba** user to view resources in the **openshift-operators** project.
7. As the **dba** user, deploy a CockroachDB database instance into the **console-review** project using the OpenShift web console. This will bring the project's total pod count to three.
8. As the **developer** user, create a deployment, service, and route in the **console-review** project with issues that you will troubleshoot in the next step. Use the **quay.io/redhattraining/exoplanets:v1.0** image, and name all of the new resources

exoplanets. When correctly configured, the **exoplanets** application connects to the CockroachDB cluster and displays a list of planets located outside of our solar system.

> **Note**
> You can copy the deployment and service YAML resources from **~/DO280/labs/console-review** on **workstation**.

Specify the following environment variables in the deployment:

Deployment Environment Variables

Name	Value
DB_HOST	**localhost**
DB_PORT	**'26257'**
DB_USER	**root**
DB_NAME	**postgres**

> **Important**
> You will troubleshoot issues with the deployment in the next step.

9. Troubleshoot and fix the deployment issues.
10. Navigate to the exoplanets website in a browser and observe the working application.

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab console-review finish
```

This concludes the section.

Solution

Managing the Cluster with the Web Console

In this lab, you will manage the OpenShift cluster using the web console.

Outcomes

You should be able to use the OpenShift web console to:

- Modify a secret to add htpasswd entries for new users.
- Configure a new project with role-based access controls and resource quotas.
- Use an OperatorHub operator to deploy a database.
- Create a deployment, service, and route for a web application.
- Troubleshoot an application using events and logs.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and creates a directory for the exercise files.

```
[student@workstation ~]$ lab console-review start
```

1. Log in to the OpenShift web console as the **admin** user.

   1.1. Source the classroom variables file.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD} \
>     ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

   1.3. List the routes in the **openshift-console** namespace. Identify the **console** host and port.

```
[student@workstation ~]$ oc get routes -n openshift-console
NAME        HOST/PORT                                       ...  PORT   ...
console     console-openshift-console.apps.cluster.example.com  ...  https  ...
downloads   downloads-openshift-console.apps.cluster.example.com ...  http   ...
```

1.4. Echo the value of **${RHT_OCP4_USER_PASSWD}** to retrieve the **admin** password.

```
[student@workstation ~]$ echo ${RHT_OCP4_USER_PASSWD}
...output omitted...
```

1.5. Open a web browser and navigate to the console route URL discovered in a previous step.

> **Note**
> If prompted with an untrusted certificate message, click **Add Exception** and then click **Confirm Security Exception**.

1.6. Click **localusers** and log in as the **admin** user with the password from the **${RHT_OCP4_USER_PASSWD}** variable.

2. Add **htpasswd** entries to the **localusers** secret for users named **dba** and **tester** using the **${RHT_OCP4_USER_PASSWD}** password.

2.1. In the Red Hat OpenShift Container Platform web UI, click **Workloads** → **Secrets** and then select **openshift-config** from the **Project** filter list to display the secrets for the **openshift-config** project.

2.2. Scroll to the bottom of the page and click the **localusers** link to display the **localusers Secret Details**.

2.3. Click **Actions** → **Edit Secret** at the top of the page to navigate to the **Edit Key/Value Secret** tool.

2.4. Use the **workstation** terminal to generate an encrypted htpasswd entry for both users.

```
[student@workstation ~]$ htpasswd -n -b dba ${RHT_OCP4_USER_PASSWD}
dba:$apr1$YF4aCK.9$qhoOTHlWTC.cLByNEHDaV
[student@workstation ~]$ htpasswd -n -b tester ${RHT_OCP4_USER_PASSWD}
tester:$apr1$XdTSqET7$i0hkC5bIs7PhYUm2KhiI.0
```

2.5. Append the terminal output from the **htpasswd** commands to the **htpasswd** value in the OpenShift web console's secrets editor and then click **Save**.

```
admin:$apr1$Au9.fFr$0k5wvUBd3eeBt0baa77.dae
leader:$apr1$/abo4Hybn7a.tG5ZoOBn.QWefXckiy1
developer:$apr1$RjqTY4cv$xql3.BQfg42moSxwnTNkh.
dba:$apr1$YF4aCK.9$qhoOTHlWTC.cLByNEHDaV
tester:$apr1$XdTSqET7$i0hkC5bIs7PhYUm2KhiI.0
```

3. Create a new **app-team** group that contains the **developer** and **dba** users.

   3.1. Click **Home** → **Search** and then select **Group** from the **Select Resource** list.

   3.2. Click **Create Group** and use the YAML editor to define a Group resource as follows:

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  name: app-team
users:
  - developer
  - dba
```

   Click **Create** to add the new **app-team** group.

4. Create a new **console-review** project with a **view** role binding for the **tester** user and an **edit** role binding for the **app-team** group. Set a resource quota that limits the project to three pods.

   4.1. Click **Home** → **Projects** to view the **Projects** page, and then click **Create Project**. Type **console-review** in the **Name** field, and then provide an optional **Display Name** and **Description**. Click **Create**.

   4.2. Click **Role Bindings** and then click **Create Binding**. Complete the form as follows to create a namespaced Role Binding for the **app-team** group.

App Team Role Binding Form

Field	Value
Binding Type	**Namespace Role Binding (RoleBinding)**
Name	**app-team**
Namespace	**console-review**
Role Name	**edit**
Subject	**Group**
Subject Name	**app-team**

   Click **Create** to create the namespaced RoleBinding.

   4.3. Click the **Role Bindings** link to return to the **Role Bindings** page, and then click **Create Binding**. Complete the form as follows to create a namespaced Role Binding for the **tester** user.

Tester Role Binding Form

Field	Value
Binding Type	`Namespace Role Binding (RoleBinding)`
Name	`tester`
Namespace	`console-review`
Role Name	`view`
Subject	`User`
Subject Name	`tester`

Click **Create** to create the namespaced RoleBinding.

4.4. Click **Administration** → **Resource Quotas** and then click **Create Resource Quota**. Modify the YAML document to specify a limit of three pods as follows:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: console-review
spec:
  hard:
    pods: '3'
```

Remove the CPU and memory requests and limits, and then click **Create**.

5. Install the Community CockroachDB operator for use in all namespaces.

5.1. Click **Operators** → **OperatorHub** and then click **Database** to display the list of database operators available from OperatorHub.

5.2. Click the Community **CockroachDB** operator, and then click **Continue** to view the community operator page.

5.3. Click **Install** and then click **Subscribe** to install the operator for use in all namespaces.

6. Create a RoleBinding that allows the **dba** user to view resources in the **`openshift-operators`** project.

6.1. Click **Administration** → **Role Bindings** and then click **Create Binding**. Fill out the form as follows.

DBA OpenShift-Operators Role Binding Form

Field	Value
Binding Type	`Namespace Role Binding (RoleBinding)`
Name	`dba`
Namespace	`openshift-operators`
Role Name	`view`
Subject	`User`
Subject Name	`dba`

Click **Create** to add the namespaced RoleBinding.

7. As the **dba** user, deploy a CockroachDB database instance into the **`console-review`** project using the OpenShift web console. This will bring the project's total pod count to three.

   7.1. Click **admin** → **Log out**, and then log in as the **dba** user with the password from the **`${RHT_OCP4_USER_PASSWD}`** variable.

   7.2. Click **Home** → **Projects** and click the **console-review** project link to switch to the **`console-review`** project.

   7.3. Click **Operators** → **Installed Operators** and then click the **`CockroachDB`** operator name.

   7.4. In the **CockroachDB** API section, click **Create Instance** and then click **Create** to create the **`CockroachDB`** resource with the default settings.

8. As the **`developer`** user, create a deployment, service, and route in the **`console-review`** project with issues that you will troubleshoot in the next step. Use the **`quay.io/redhattraining/exoplanets:v1.0`** image, and name all of the new resources **`exoplanets`**. When correctly configured, the **`exoplanets`** application connects to the CockroachDB cluster and displays a list of planets located outside of our solar system.

> **Note**
> You can copy the deployment and service YAML resources from **`~/DO280/labs/console-review`** on **`workstation`**.

Specify the following environment variables in the deployment:

Deployment Environment Variables

Name	Value
DB_HOST	localhost
DB_PORT	'26257'
DB_USER	root
DB_NAME	postgres

> **Important**
> You will troubleshoot issues with the deployment in the next step.

8.1. Click **dba** → **Log out** and then log in as the **developer** user with the password from the **${RHT_OCP4_USER_PASSWD}** variable.

8.2. Click **Home** → **Projects** and then click the **console-review** project to switch to the **console-review** project.

8.3. Click **Workloads** → **Deployments** and then click **Create Deployment** to display the web console YAML editor. Update the YAML as follows and then click **Create**:

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: exoplanets
  namespace: console-review
spec:
  replicas: 1
  selector:
    matchLabels:
      app: exoplanets
  template:
    metadata:
      labels:
        app: exoplanets
    spec:
      containers:
        - name: exoplanets
          image: 'quay.io/redhattraining/exoplanets:v1.0'
          ports:
            - containerPort: 8080
              protocol: TCP
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8080
          env:
            - name: DB_HOST
              value: localhost
```

```
        - name: DB_PORT
          value: '26257'
        - name: DB_USER
          value: root
        - name: DB_NAME
          value: postgres
```

8.4. Click **Networking** → **Services** and then click **Create Service** to display the web console YAML editor. Update the YAML as follows and then click **Create**:

```
kind: Service
apiVersion: v1
metadata:
  name: exoplanets
  namespace: console-review
spec:
  selector:
    app: exoplanets
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

8.5. Click **Networking** → **Routes** and then click **Create Route**. Complete the form as follows, leaving the other fields unchanged, and then click **Create**:

Create Route Form

Field	Value
Name	**exoplanets**
Service	**exoplanets**
Target Port	**8080 → 8080 (TCP)**

9. Troubleshoot and fix the deployment issues.

9.1. Click **developer** → **Log out** and then log in as the **admin** user with the password from the **${RHT_OCP4_USER_PASSWD}** variable.

9.2. Click **Home** → **Events** and then select **console-review** from the project list filter at the top. Notice the exoplanets quota error:

```
(combined from similar events): Error creating: pods "exoplanets-5f88574546-lsnmx"
 is forbidden: exceeded quota: quota, requested: pods=1, used: pods=3, limited:
 pods=3
```

9.3. Click **Administration** → **Resource Quotas** and then select **console-review** from the **Project** filter list.

9.4. Click the **quota** link in the list of resource quotas, and then click the **YAML** tab. Modify the **spec** to specify a limit of five pods as follows, and then click **Save**.

```
kind: ResourceQuota
apiVersion: v1
metadata:
  name: quota
  namespace: console-review
  selfLink: /api/v1/namespaces/console-review/resourcequotas/quota
  uid: 61d1eafb-2106-11ea-bc81-06452abe2b2c
  resourceVersion: '5223946'
  creationTimestamp: '2019-12-17T19:49:48Z'
spec:
  hard:
    pods: '5'
status:
  hard:
    pods: '3'
  used:
    pods: '3'
```

> **Note**
> The project requires a pod for the exoplanet's specified replica and an additional pod in order to roll out a change.

9.5. Click **Workloads** → **Pods** and review the list of pods. The **exoplanets** pod may take a minute or two to appear on the list and then it displays a **CrashLoopBackOff** status.

9.6. Click the pod name and then click the **Logs** tab. Notice the connection error.

9.7. Click **Operators** → **Installed Operators** and then click the **CockroachDB** operator name. Click the **CockroachDB** tab and then click the **example** name to display the **Cockroachdb Details** page for the **exoplanets** cluster.

9.8. Click the **Resources** tab and click the service with the **-public** suffix. Copy the name of the service to the clipboard.

9.9. Click **Workloads** → **Deployments** and then click the **exoplanets** Deployment link. Click the **YAML** tab and change the **DB_HOST** value from **localhost** to the name of the CockroachDB service. Click **Save**.

9.10. Click the **Pods** tab to verify that the new exoplanets pod updates to a **Running** status.

10. Navigate to the exoplanets website in a browser and observe the working application.

10.1. Click **Networking** → **Routes**, click the **exoplanets** route name, and then click the link in the **Location** column. Firefox will open a new tab rendering a table of exoplanets.

This concludes the lab.

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab console-review finish
```

Summary

In this chapter, you learned that:

- The OpenShift web console provides a GUI for visualizing and managing OpenShift resources.
- Some resources feature a specialized page that makes creating and editing resources more convenient than writing YAML by hand, such as the **Edit Key/Value Secret** editor, which automatically handles Base64 encoding and decoding.
- You can install partner and community operators from the embedded **OperatorHub** page.
- Cluster-wide metrics such as CPU, memory, and storage usage are displayed on the **Dashboards** page.
- **Project Details** pages display metrics specific to the project, such as the top ten memory consumers by pod and the current resource quota usage.

Chapter 10

Comprehensive Review

Goal	Review tasks from *Red Hat OpenShift Administration I*
Objectives	• Review tasks from *Red Hat OpenShift Administration I*
Sections	• Comprehensive Review
Lab	Install, Manage, and Troubleshoot an OpenShift Cluster

Comprehensive Review

Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills learned in *Red Hat OpenShift Administration I*.

Reviewing Red Hat OpenShift Administration I

Before beginning the comprehensive review for this course, you should be comfortable with the topics covered in each chapter.

You can refer to earlier sections in the textbook for extra study.

Chapter 1, *Describing Red Hat OpenShift Container Platform*

Describe the architecture of Red Hat OpenShift Container Platform (RHOCP).

- Create an OpenShift cluster for use as a lab environment.
- Describe the typical use of RHOCP and its features.
- Describe the architecture of RHOCP.
- Describe what a cluster operator is, how it works, and name the major cluster operators.

Chapter 2, *Verifying a Cluster*

Review installation methods and verify the functionality of a newly installed cluster.

- Describe the various installation methods and platforms supported.
- Configure the OpenShift Cluster used for classroom exercises.
- Execute commands that assist in troubleshooting common problems.

Chapter 3, *Configuring Authentication*

Configure authentication with an identity provider

- Describe the resources associated with OpenShift authentication and authorization.
- Authenticate as a cluster administrator using the kubeconfig file or kubeadmin virtual user.
- Configure the HTPasswd identity provider for OpenShift authentication.
- Assign a user to a cluster role.

Chapter 4, *Controlling Access to OpenShift Resources*

Define and apply role-based access controls and protect sensitive information with secrets.

- Define role-based access controls and apply permissions to users.
- Create and apply secrets to manage sensitive information.

- Create service accounts and apply permissions to pods.

Chapter 5, *Configuring OpenShift Networking Components*

Identify the components of Red Hat OpenShift Container Platform software-defined networking and configure some of the components.

- Troubleshoot OpenShift software-defined networking using the command-line interface.
- Describe the ingress components of OpenShift and create a route.

Chapter 6, *Controlling Pod Scheduling*

Control the nodes on which a pod runs.

- Describe pod scheduling algorithms, the methods used to influence scheduling, and apply these methods.
- Limit the resources consumed by an application.
- Control the number of replicas of a pod.

Chapter 7, *Scaling an OpenShift Cluster*

Control the size of an OpenShift cluster.

- Issue commands to control the number of workers in a cluster.
- Define parameters to automatically control the size of a cluster.

Chapter 8, *Performing Cluster Updates*

Describe how to perform a cluster update.

Describe the cluster update process.

Chapter 9, *Managing a Cluster with the Web Console*

Manage a Red Hat OpenShift cluster using the web console.

- Perform cluster administration with the web console.
- Manage cluster workloads with the web console.
- Examine the metrics page and dashboard within the web console.

Lab

Install, manage, and troubleshoot an OpenShift cluster

In this review, you will manage and troubleshoot an OpenShift cluster for enterprise use.

Outcomes

You should be able to:

- Verify an OpenShift cluster installation.
- Use the HTPasswd identity provider for managing users.
- Manage RBAC and SCC for users and groups.
- Manage secrets for databases and applications.
- Manage application auto scale.
- Troubleshoot common problems.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab comprehensive-review start
```

Instructions

In this lab, you manage a new OpenShift cluster for a small set of teams: developers, QA engineers, and project leads. Before these teams can use the cluster, you must:

- Create user accounts for all users.
- Create groups for each team, and assign roles to the group that are appropriate for the assigned team.
- Configure resource management and troubleshoot existing issues.

Complete the following tasks:

1. Perform a smoke test of the cluster to verify basic cluster functionality.
    - As the **kubeadmin** user, create the **comprehensive-review** project. Perform all subsequent tasks in this project.
    - Build and deploy an application in this project.

      The application source code is located in the **hello-world-nginx** subdirectory of the `https://github.com/RedHatTraining/DO280-apps` repository. Name the application **hello-world-nginx**.

- Create a route for the application. Verify that the application responds to external requests.

2. Configure the cluster to use an HTPasswd identity provider. The name of the identity provider is **cluster-users**. The identity provider reads **htpasswd** credentials stored in the **compreview-users** secret.

   Ensure four user accounts exist: **admin**, **leader**, **developer**, and **qa-engineer**. All user account passwords are set to the value of the **RHT_OCP4_USER_PASSWD** variable. This variable is defined in the **/usr/local/etc/ocp4.config** file on **workstation**.

   Add the **cluster-admin** role to the **admin** user.

3. Create three user groups: **leaders**, **developers**, and **qa**.

   Assign the **leader** user to the **leaders** group, the **developer** user to the **developers** groups and the **qa-engineer** user to the **qa** group.

   Assign roles to each group:

   - Assign the **self-provisioner** role to the **leaders** group, which allows members to create projects. For this role to be effective, you must also remove the ability of any authenticated user to create new projects.
   - Assign the **edit** role to the **developers** group for the **comprehensive-review** project only, which allows members to create and delete project resources.
   - Assign the **view** role to the **qa** group for the **comprehensive-review** project only, which provides members with read access to project resources.

4. As the **developer** user, create a **mysql** application in the **comprehensive-review** project from the **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7** image. This application provides a shared database service for other project applications.

   Create a **mysql** secret to store the value of the root user password. Add a **password** variable with a value of **r3dh4t123** to the secret.

   Set the **MYSQL_ROOT_PASSWORD** environment variable from the value of the **password** variable in the **mysql** secret.

5. As the **developer** user create a **wordpress** application in the **comprehensive-review** project from the **docker.io/library/wordpress:5.3.0** image.

   Configure the **WORDPRESS_DB_HOST** environment variable to have a value of **mysql**. The application uses this variable to connect the **mysql** database service provided by the **mysql** application.

   Configure the **WORDPRESS_DB_NAME** environment variable to have a value of **wordpress**. The application uses this variable to identify the name of the database. If the database does not exist on the database server, then the application attempts to create a new database with this name.

   Set the **WORDPRESS_DB_USER** as **root**. Set the **WORDPRESS_DB_PASSWORD** variable value from the **password** key in the **mysql** secret, which contains the **mysql** root user password.

The **wordpress** application also requires the **anyuid** permission. Create a **wordpress-sa** service account, assign this permission to it, and then assign the service account to the **wordpress** deployment configuration.

If you correctly deploy the application, then an installation wizard displays when you access the application from a browser. Create a route for the application and verify thase a browser to navigate to the application. When you correctly deploy the application, a setup wizard displays in a browser.

The **wordpress** application also creates a **wordpress** database on the **mysql** database service.

6. As the developer user deploy the **famous-quotes** application in the **comprehensive-review** project using the **~/DO280/labs/comprehensive-review/deploy_famous-quotes.sh** script. This script creates the **defaultdb** database and the resources defined in the **~/DO280/labs/comprehensive-review/famous-quotes.yaml** file.

   The application pods do not initially deploy after you execute the script. The **famous-quotes** deployment configuration specifies a node selector, and there are no cluster nodes with a matching node label.

   Remove the node selector from the deployment configuration, which enables OpenShift to schedule application pods on any available node.

   Create a route for the **famous-quotes** application. Access the route and verify that the application responds with a list of quotes.

> **Note**
> The **mysql** application is an ephemeral database. If you delete it after creating the **wordpress** and **famous-quotes** applications, then these applications will fail to work as intended.

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab comprehensive-review finish
```

This concludes the comprehensive review.

Solution

Install, manage, and troubleshoot an OpenShift cluster

In this review, you will manage and troubleshoot an OpenShift cluster for enterprise use.

Outcomes

You should be able to:

- Verify an OpenShift cluster installation.
- Use the HTPasswd identity provider for managing users.
- Manage RBAC and SCC for users and groups.
- Manage secrets for databases and applications.
- Manage application auto scale.
- Troubleshoot common problems.

Before You Begin

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab comprehensive-review start
```

Instructions

In this lab, you manage a new OpenShift cluster for a small set of teams: developers, QA engineers, and project leads. Before these teams can use the cluster, you must:

- Create user accounts for all users.
- Create groups for each team, and assign roles to the group that are appropriate for the assigned team.
- Configure resource management and troubleshoot existing issues.

Complete the following tasks:

1. Perform a smoke test of the cluster to verify basic cluster functionality.
    - As the **kubeadmin** user, create the **comprehensive-review** project. Perform all subsequent tasks in this project.
    - Build and deploy an application in this project.

      The application source code is located in the **hello-world-nginx** subdirectory of the `https://github.com/RedHatTraining/DO280-apps` repository. Name the application **hello-world-nginx**.

- Create a route for the application. Verify that the application responds to external requests.

2. Configure the cluster to use an HTPasswd identity provider. The name of the identity provider is **cluster-users**. The identity provider reads **htpasswd** credentials stored in the **compreview-users** secret.

   Ensure four user accounts exist: **admin**, **leader**, **developer**, and **qa-engineer**. All user account passwords are set to the value of the **RHT_OCP4_USER_PASSWD** variable. This variable is defined in the **/usr/local/etc/ocp4.config** file on **workstation**.

   Add the **cluster-admin** role to the **admin** user.

3. Create three user groups: **leaders**, **developers**, and **qa**.

   Assign the **leader** user to the **leaders** group, the **developer** user to the **developers** groups and the **qa-engineer** user to the **qa** group.

   Assign roles to each group:

   - Assign the **self-provisioner** role to the **leaders** group, which allows members to create projects. For this role to be effective, you must also remove the ability of any authenticated user to create new projects.

   - Assign the **edit** role to the **developers** group for the **comprehensive-review** project only, which allows members to create and delete project resources.

   - Assign the **view** role to the **qa** group for the **comprehensive-review** project only, which provides members with read access to project resources.

4. As the **developer** user, create a **mysql** application in the **comprehensive-review** project from the **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7** image. This application provides a shared database service for other project applications.

   Create a **mysql** secret to store the value of the root user password. Add a **password** variable with a value of **r3dh4t123** to the secret.

   Set the **MYSQL_ROOT_PASSWORD** environment variable from the value of the **password** variable in the **mysql** secret.

5. As the **developer** user create a **wordpress** application in the **comprehensive-review** project from the **docker.io/library/wordpress:5.3.0** image.

   Configure the **WORDPRESS_DB_HOST** environment variable to have a value of **mysql**. The application uses this variable to connect the **mysql** database service provided by the **mysql** application.

   Configure the **WORDPRESS_DB_NAME** environment variable to have a value of **wordpress**. The application uses this variable to identify the name of the database. If the database does not exist on the database server, then the application attempts to create a new database with this name.

   Set the **WORDPRESS_DB_USER** as **root**. Set the **WORDPRESS_DB_PASSWORD** variable value from the **password** key in the **mysql** secret, which contains the **mysql** root user password.

The **wordpress** application also requires the **anyuid** permission. Create a **wordpress-sa** service account, assign this permission to it, and then assign the service account to the **wordpress** deployment configuration.

If you correctly deploy the application, then an installation wizard displays when you access the application from a browser. Create a route for the application and verify thase a browser to navigate to the application. When you correctly deploy the application, a setup wizard displays in a browser.

The **wordpress** application also creates a **wordpress** database on the **mysql** database service.

6. As the developer user deploy the **famous-quotes** application in the **comprehensive-review** project using the **~/DO280/labs/comprehensive-review/deploy_famous-quotes.sh** script. This script creates the **defaultdb** database and the resources defined in the **~/DO280/labs/comprehensive-review/famous-quotes.yaml** file.

   The application pods do not initially deploy after you execute the script. The **famous-quotes** deployment configuration specifies a node selector, and there are no cluster nodes with a matching node label.

   Remove the node selector from the deployment configuration, which enables OpenShift to schedule application pods on any available node.

   Create a route for the **famous-quotes** application. Access the route and verify that the application responds with a list of quotes.

> **Note**
> The **mysql** application is an ephemeral database. If you delete it after creating the **wordpress** and **famous-quotes** applications, then these applications will fail to work as intended.

Steps

1. Perform a smoke test of the cluster to verify basic cluster functionality.

   - As the **kubeadmin** user, create the **comprehensive-review** project. Perform all subsequent tasks in this project.
   - Build and deploy an application in this project.

     The application source code is located in the **hello-world-nginx** subdirectory of the `https://github.com/RedHatTraining/DO280-apps` repository. Name the application **hello-world-nginx**.
   - Create a route for the application. Verify that the application responds to external requests.

   1.1. Source the classroom configuration file that is accessible at **/usr/local/etc/ocp4.config** and log in as the **kubeadmin** user.

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
>    ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.2. Create the **comprehensive-review** project. Use the **oc new-app** command to create the **hello-world-nginx** application:

```
[student@workstation ~]$ oc new-project comprehensive-review
Now using project "comprehensive-review" on server
"https://api.cluster.domain.example.com:6443".
...output omitted...
[student@workstation ~]$ oc new-app --name hello-world-nginx \
>    https://github.com/RedhatTraining/DO280-apps \
>    --context-dir hello-world-nginx
...output omitted...
    service "hello-world-nginx" created
--> Success
...output omitted...
```

1.3. Expose the **hello-world-nginx** application.

```
[student@workstation ~]$ oc expose service hello-world-nginx
route.route.openshift.io/hello-world-nginx exposed
```

1.4. Wait until the application pod is running, and then verify access to the application.

```
[student@workstation ~]$ oc get pods
NAME                        READY   STATUS      RESTARTS   AGE
hello-world-nginx-1-build   0/1     Completed   0          2m59s
hello-world-nginx-1-deploy  0/1     Completed   0          108s
hello-world-nginx-1-m2lzr   1/1     Running     0          100s
[student@workstation ~]$ smoke_test_route=$(oc get routes \
>    hello-world-nginx -o jsonpath='{.spec.host}')
[student@workstation ~]$ echo $smoke_test_route
hello-world-nginx-comprehensive-review.apps.cluster.domain.example.com
[student@workstation ~]$ curl $smoke_test_route
...output omitted...
    <h1>Hello, world from nginx!</h1>
...output omitted...
```

2. Configure the cluster to use an HTPasswd identity provider. The name of the identity provider is **cluster-users**. The identity provider reads **htpasswd** credentials stored in the **compreview-users** secret.

   Ensure four user accounts exist: **admin**, **leader**, **developer**, and **qa-engineer**. All user account passwords are set to the value of the **RHT_OCP4_USER_PASSWD** variable. This variable is defined in the **/usr/local/etc/ocp4.config** file on **workstation**.

   Add the **cluster-admin** role to the **admin** user.

   2.1. Create an **htpasswd** authentication file with the required user and password values.

```
[student@workstation ~]$ pass_file=~/DO280/labs/comprehensive-review/cluster-users
[student@workstation ~]$ echo ${pass_file}
~/DO280/labs/comprehensive-review/cluster-users
[student@workstation ~]$ htpasswd -c -B -b \
>     ${pass_file} admin ${RHT_OCP4_USER_PASSWD}
Adding password for user admin
[student@workstation ~]$ for user in leader developer qa-engineer
> do
> htpasswd -B -b ${pass_file} ${user} ${RHT_OCP4_USER_PASSWD}
> done
Adding password for user leader
Adding password for user developer
Adding password for user qa-engineer
[student@workstation ~]$
```

2.2. Create a **compreview-users** secret from the **htpasswd** authentication file that you previously created:

```
[student@workstation ~]$ oc create secret generic compreview-users \
>     --from-file htpasswd=${pass_file} -n openshift-config
secret/compreview-users created
```

2.3. Export the existing OAuth resource to a YAML file.

```
[student@workstation ~]$ auth_file=~/DO280/labs/comprehensive-review/auth.yaml
[student@workstation ~]$ oc get -o yaml oauth cluster > ${auth_file}
```

2.4. Open the exported file with your preferred editor. Add a **HTPasswd** identity provider definition to the **identityProviders** list. Set the identity provider name to **cluster-users**, and set the **fileData** name to **compreview-users**.

After making these modifications, the file displays the following:

```
apiVersion: config.openshift.io/v1
kind: OAuth
...output omitted...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: compreview-users
    mappingMethod: claim
    name: cluster-users
    type: HTPasswd
```

Note

The **htpasswd**, **mappingMethod**, **name** and **type** keys all use the same indentation.

2.5. Replace the existing OAuth resource with the resource definition in the modified file:

```
[student@workstation ~]$ oc replace -f ${auth_file}
oauth.config.openshift.io/cluster replaced
```

2.6. Assign the **admin** user the **cluster-admin** role.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
>    cluster-admin admin
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "admin"
```

> **Warning**
>
> The **admin** user account is not immediately available after you replace the **cluster** OAuth resource.
>
> If the above command fails, then wait a few moments and try the command again. If the command continues to fail, ensure that there are no configuration errors in the **cluster** OAuth resource.

3. Create three user groups: **leaders**, **developers**, and **qa**.

   Assign the **leader** user to the **leaders** group, the **developer** user to the **developers** groups and the **qa-engineer** user to the **qa** group.

   Assign roles to each group:

   - Assign the **self-provisioner** role to the **leaders** group, which allows members to create projects. For this role to be effective, you must also remove the ability of any authenticated user to create new projects.
   - Assign the **edit** role to the **developers** group for the **comprehensive-review** project only, which allows members to create and delete project resources.
   - Assign the **view** role to the **qa** group for the **comprehensive-review** project only, which provides members with read access to project resources.

3.1. Log in as the **admin** user and remove the **self-provisioner** cluster role from to the **system:authenticated:oauth** group.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
>    self-provisioner system:authenticated:oauth
...output omitted...
```

3.2. Create the four user groups. Add each user to the appropriate group.

```
[student@workstation ~]$ for group in leaders developers qa
> do
> oc adm groups new ${group}
> done
...output omitted...
[student@workstation ~]$ oc adm groups add-users leaders leader
group.user.openshift.io/leaders added: "leader"
```

```
[student@workstation ~]$ oc adm groups add-users developers developer
group.user.openshift.io/developers added: "developer"
[student@workstation ~]$ oc adm groups add-users qa qa-engineer
group.user.openshift.io/qa added: "qa-engineer"
```

3.3. Allow members of the **leaders** group to create new projects:

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
>    self-provisioner leaders
```

3.4. Allow members of the **developers** group to create and delete resources in the **comprehensive-review** project:

```
[student@workstation ~]$ oc adm policy add-role-to-group edit developers
```

3.5. Only allow members of the **qa** group to view project resources:

```
[student@workstation ~]$ oc adm policy add-role-to-group view qa
```

4. As the **developer** user, create a **mysql** application in the **comprehensive-review** project from the **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7** image. This application provides a shared database service for other project applications.

Create a **mysql** secret to store the value of the root user password. Add a **password** variable with a value of **r3dh4t123** to the secret.

Set the **MYSQL_ROOT_PASSWORD** environment variable from the value of the **password** variable in the **mysql** secret.

4.1. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
```

4.2. Create a **mysql** secret with the root user password for the MySQL database.

```
[student@workstation ~]$ oc create secret generic mysql \
>   --from-literal password=r3dh4t123
secret/mysql created
```

4.3. Create a new application to deploy a **mysql** database server.

```
[student@workstation ~]$ oc new-app --name mysql \
  >   --docker-image registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "mysql" created
    deploymentconfig.apps.openshift.io "mysql" created
    service "mysql" created
--> Success
...output omitted...
```

4.4. Use the **mysql** secret to initialize environment variables on the **mysql** deployment.

```
[student@workstation ~]$ oc set env dc/mysql --prefix MYSQL_ROOT_ \
>   --from secret/mysql
deploymentconfig.apps.openshift.io/mysql updated
```

> **Note**
>
> It may take a while for the deployment to roll out successfully after setting the secret.

5. As the **developer** user create a **wordpress** application in the **comprehensive-review** project from the **docker.io/library/wordpress:5.3.0** image.

   Configure the **WORDPRESS_DB_HOST** environment variable to have a value of **mysql**. The application uses this variable to connect the **mysql** database service provided by the **mysql** application.

   Configure the **WORDPRESS_DB_NAME** environment variable to have a value of **wordpress**. The application uses this variable to identify the name of the database. If the database does not exist on the database server, then the application attempts to create a new database with this name.

   Set the **WORDPRESS_DB_USER** as **root**. Set the **WORDPRESS_DB_PASSWORD** variable value from the **password** key in the **mysql** secret, which contains the **mysql** root user password.

   The **wordpress** application also requires the **anyuid** permission. Create a **wordpress-sa** service account, assign this permission to it, and then assign the service account to the **wordpress** deployment configuration.

   If you correctly deploy the application, then an installation wizard displays when you access the application from a browser. Create a route for the application and verify thase a browser to navigate to the application. When you correctly deploy the application, a setup wizard displays in a browser.

   The **wordpress** application also creates a **wordpress** database on the **mysql** database service.

5.1. Deploy a **wordpress** application.

```
[student@workstation ~]$ oc new-app --name wordpress \
>    --docker-image docker.io/library/wordpress:5.3.0 \
>    -e WORDPRESS_DB_HOST=mysql -e WORDPRESS_DB_USER=root \
>    -e WORDPRESS_DB_NAME=wordpress
...output omitted...
-> Creating resources ...
    imagestream.image.openshift.io "wordpress" created
    deploymentconfig.apps.openshift.io "wordpress" created
    service "wordpress" created
--> Success
...output omitted...
```

5.2. Create a service account named **wordpress-sa**.

```
[student@workstation ~]$ oc create sa wordpress-sa
serviceaccount/wordpress-sa created
```

5.3. Log in to the cluster as the **admin** user and grant **anyuid** privileges to the **wordpress-sa** service account.

```
[student@workstation ~]$ oc login -u admin -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z wordpress-sa
securitycontextconstraints.security.openshift.io/anyuid added to:
 ["system:serviceaccount:authorization-review:wordpress-sa"]
```

5.4. Log in to the cluster as the **developer** user and set the **wordpress-sa** service account to the **wordpress** deployment configuration.

```
[student@workstation ~]$ oc login -u developer -p ${RHT_OCP4_USER_PASSWD}
Login successful.
...output omitted...
[student@workstation ~]$ oc set serviceaccount deploymentconfig wordpress \
>    wordpress-sa
deploymentconfig.apps.openshift.io/wordpress serviceaccount updated
```

5.5. Add the **WORDPRESS_DB_PASSWORD** environment variable to the **wordpress** deployment configuration. Use the value of the **password** key in the **mysql** secret as the value for the value of the variable.

```
[student@workstation ~]$ oc set env dc/wordpress --prefix WORDPRESS_DB_ \
>   --from secret/mysql
deploymentconfig.apps.openshift.io/wordpress updated
```

Note

It may take a while for the deployment to roll out successfully after setting the secret.

6. As the developer user deploy the **famous-quotes** application in the **comprehensive-review** project using the **~/DO280/labs/comprehensive-review/deploy_famous-quotes.sh** script. This script creates the **defaultdb** database and the resources defined in the **~/DO280/labs/comprehensive-review/famous-quotes.yaml** file.

   The application pods do not initially deploy after you execute the script. The **famous-quotes** deployment configuration specifies a node selector, and there are no cluster nodes with a matching node label.

   Remove the node selector from the deployment configuration, which enables OpenShift to schedule application pods on any available node.

   Create a route for the **famous-quotes** application. Access the route and verify that the application responds with a list of quotes.

   6.1. Run the **~/DO280/labs/comprehensive-review/deploy_famous-quotes.sh** script.

```
[student@workstation ~]$ ~/DO280/labs/comprehensive-review/deploy_famous-quotes.sh
Creating famous-quotes database
Deploying famous-quotes application
deploymentconfig.apps.openshift.io/famous-quotes created
service/famous-quotes created
```

6.2. Verify that the **famous-quotes** application pod is not scheduled for deployment.

```
[student@workstation ~]$ oc get pods
NAME                     READY    STATUS     RESTARTS     AGE
famous-quotes-1-deploy   0/1      Pending    0            41s
...output omitted...
[student@workstation ~]$ oc get events --sort-by='{.lastTimestamp}'
...output omitted...
34s    Warning    FailedScheduling      pod/famous-quotes-1-deploy    0/5 nodes are
 available: 5 node(s) didn't match node selector.
...output omitted...
```

6.3. Save the **famous-quotes** deployment configuration resource to a file. Use an editor to remove the **env=quotes** label from this file.

> **Note**
> The labels is located in the second **spec** section of the template.

```
[student@workstation ~]$ oc get -o yaml dc/famous-quotes > ~/famous-dc.yaml
[student@workstation ~]$ vi ~/famous-dc.yaml
...output omitted...
    spec:
      nodeSelector: {}
...output omitted...
```

6.4. Replace the **famous-quotes** deployment configuration with the modifications in the file:

```
[student@workstation ~]$ oc replace -f ~/famous-dc.yaml
...output omitted...
```

6.5. Wait a few moments and run **oc get pods** to ensure that the **famous-quotes** application is now running.

```
[student@workstation ~]$ oc get pods
NAME                    READY    STATUS     RESTARTS     AGE
famous-quotes-1-bjjx8   1/1      Running    0            41s
```

This concludes the lab.

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab comprehensive-review finish
```