



Red Hat Training and Certification

Student Workbook (ROLE)

OCP 4.5 DO285

**Containers, Kubernetes, and Red Hat
OpenShift Administration II**

Edition 1



Containers, Kubernetes, and Red Hat OpenShift Administration II



OCP 4.5 DO285
Containers, Kubernetes, and Red Hat OpenShift
Administration II
Edition 1 20201202
Publication date 20201202

Authors: Richard Allred, Joel Birchler, Christopher Caillouet, Ivan Chavero,
Zach Guterman, Andres Hernandez, Michael Jarrett,
Dan Kolepp, Fernando Lozano, Razique Mahroua, James Mighion,
Michael Phillips, Eduardo Ramirez Ronco, Jordi Sola Alaball
Editor: David O'Brien, Nicole Muller, Dave Sacco

Copyright © 2020 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are
Copyright © 2020 Red Hat, Inc.

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

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

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

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

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

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

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

The OpenStack® word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation in the United States and other countries, and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Forrest Taylor, Manuel Aude Morales, Jim Rigsbee

Document Conventions	ix
Introduction	xi
Containers, Kubernetes, and Red Hat OpenShift Administration II	xii
Orientation to the Classroom Environment	xiii
Internationalization	xix
1. Introducing Container Technology	1
Overview of Container Technology	2
Quiz: Overview of Container Technology	5
Overview of Container Architecture	9
Quiz: Overview of Container Architecture	12
Overview of Kubernetes and OpenShift	14
Quiz: Describing Kubernetes and OpenShift	17
Guided Exercise: Configuring the Classroom Environment	19
Summary	23
2. Creating Containerized Services	25
Provisioning Containerized Services	26
Guided Exercise: Creating a MySQL Database Instance	32
Summary	35
3. Managing Containers	37
Managing the Life Cycle of Containers	38
Guided Exercise: Managing a MySQL Container	46
Attaching Persistent Storage to Containers	49
Guided Exercise: Persisting a MySQL Database	52
Accessing Containers	55
Guided Exercise: Loading the Database	59
Lab: Managing Containers	62
Summary	71
4. Managing Container Images	73
Accessing Registries	74
Quiz: Working With Registries	80
Manipulating Container Images	84
Guided Exercise: Creating a Custom Apache Container Image	90
Lab: Managing Images	94
Summary	102
5. Creating Custom Container Images	103
Designing Custom Container Images	104
Quiz: Approaches to Container Image Design	108
Building Custom Container Images with Dockerfiles	110
Guided Exercise: Creating a Basic Apache Container Image	115
Lab: Creating Custom Container Images	119
Summary	126
6. Deploying Containerized Applications on OpenShift	127
Creating Kubernetes Resources	128
Guided Exercise: Deploying a Database Server on OpenShift	139
Creating Routes	144
Guided Exercise: Exposing a Service as a Route	148
Creating Applications with Source-to-Image	153
Guided Exercise: Creating a Containerized Application with Source-to-Image	163
Lab: Deploying Containerized Applications on OpenShift	169
Summary	174
7. Deploying Multi-Container Applications	175

Deploying a Multi-Container Application on OpenShift	176
Guided Exercise: Creating an Application with a Template	186
Lab: Containerizing and Deploying a Software Application	192
Summary	202
8. Describing Red Hat OpenShift Container Platform	203
Describing OpenShift Container Platform Features	204
Quiz: Describing OpenShift Container Platform Features	209
Describing the Architecture of OpenShift	213
Quiz: Describing the Architecture of OpenShift	216
Describing Cluster Operators	218
Quiz: Describing Cluster Operators	221
Summary	223
9. Verifying a Cluster	225
Describing Installation Methods	226
Quiz: Describing Installation Methods	228
Troubleshooting OpenShift Clusters and Applications	230
Guided Exercise: Troubleshooting OpenShift Clusters and Applications	238
Introducing OpenShift Dynamic Storage	245
Guided Exercise: Introducing OpenShift Dynamic Storage	249
Summary	254
10. Configuring Authentication and Authorization	255
Configuring Identity Providers	256
Guided Exercise: Configuring Identity Providers	263
Defining and Applying Permissions Using RBAC	272
Guided Exercise: Defining and Applying Permissions using RBAC	276
Lab: Configuring Authentication and Authorization	281
Summary	289
11. Configuring Application Security	291
Managing Sensitive Information with Secrets	292
Guided Exercise: Managing Sensitive Information With Secrets	297
Controlling Application Permissions with Security Context Constraints	303
Guided Exercise: Controlling Application Permissions with Security Context Constraints ..	306
Lab: Configuring Application Security	310
Summary	317
12. Configuring OpenShift Networking Components	319
Troubleshooting OpenShift Software-defined Networking	320
Guided Exercise: Troubleshooting OpenShift Software-defined Networking	327
Exposing Applications for External Access	336
Guided Exercise: Exposing Applications for External Access	342
Configuring Network Policies	352
Guided Exercise: Configuring Network Policies	356
Lab: Configuring OpenShift Networking for Applications	364
Summary	377
13. Controlling Pod Scheduling	379
Controlling Pod Scheduling Behavior	380
Guided Exercise: Controlling Pod Scheduling Behavior	387
Limiting Resource Usage by an Application	393
Guided Exercise: Limiting Resource Usage by an Application	403
Scaling an Application	413
Guided Exercise: Scaling an Application	417
Lab: Controlling Pod Scheduling	423
Summary	431

14. Describing Cluster Updates	433
Describing the Cluster Update Process	434
Quiz: Describing the Cluster Update Process	444
Summary	448
15. Managing a Cluster with the Web Console	449
Performing Cluster Administration	450
Guided Exercise: Performing Cluster Administration	454
Managing Workloads and Operators	461
Guided Exercise: Managing Workloads and Operators	466
Examining Cluster Metrics	475
Guided Exercise: Examining Cluster Metrics	479
Lab: Managing a Cluster with the Web Console	484
Summary	494
16. Comprehensive Review	495
Comprehensive Review	496
Lab: Troubleshoot an OpenShift Cluster and Applications	498
Lab: Configure a Project Template with Resource and Network Restrictions	512
A. Creating a GitHub Account	525
Creating a GitHub Account	526
B. Creating a Quay Account	529
Creating a Quay Account	530
Repositories Visibility	533
C. Useful Git Commands	537
Git Commands	538
D. Scaling an OpenShift Cluster	541
Manually Scaling an OpenShift Cluster	542
Automatically Scaling an OpenShift Cluster	546
Summary	550

Document Conventions



References

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



Note

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



Important

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



Warning

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

Introduction

Containers, Kubernetes, and Red Hat OpenShift Administration II

Containers, Kubernetes, and Red Hat OpenShift Administration II (DO285) helps you build core knowledge in building and managing Linux containers and Red Hat® OpenShift® Container Platform. This hands-on, lab-based course shows you how to deploy sample applications to either a local container runtime or an OpenShift cluster and then how to configure and manage OpenShift clusters to further understand how developers will use the platform. These skills are needed for multiple roles, including developers, administrators, and site reliability engineers.

Course Objectives

- After completing this course, you should be able to demonstrate the skills to create and manage local containers using Podman, establish a new OpenShift cluster, perform initial configuration of the cluster, and manage the cluster on a day-to-day basis. One major focus of the course is troubleshooting common problems that will be encountered beyond day one.

Audience

- System and Software Architects interested in understanding features and functionality of an OpenShift cluster.
- System Administrators interested in the initial establishment of a cluster.
- Cluster Operators interested in the ongoing maintenance of a cluster.
- Site Reliability Engineers interested in the ongoing maintenance and troubleshooting of a cluster.

Prerequisites

- Either attain the *Red Hat Certified System Administrator* certification (RHCSA), or have equivalent knowledge.

Orientation to the Classroom Environment

The Workstation Machine

In this course, the main computer system used for hands-on learning activities (exercises) is **workstation**.

The **workstation** machine has a standard user account, **student** with the password **student**. No exercise in this course requires that you log in as **root**, but if you must, the **root** password on the **workstation** machine is **redhat**.

It is from the **workstation** machine that you type **oc** commands to manage the OpenShift cluster, which comes preinstalled as part of your classroom environment.

It is also from the **workstation** machine that you run shell scripts and Ansible Playbooks required to complete exercises for this course.

If exercises require that you open a web browser to access any application or web site, then you are required to use the graphical console of the **workstation** machine and use the Firefox web browser from there.



Note

The first time you start your classroom environment, OpenShift clusters take a little longer to become fully available. The **lab** command at the beginning of each exercise checks and waits as required.

If you try to access your cluster using either the **oc** command or the web console without first running a **lab** command, then you might find that your cluster is not yet available. If that happens, then wait a few minutes and try again.

The Classroom Environment

This course, Containers, Kubernetes, and Red Hat OpenShift Administration II (DO285), includes content and labs from both Red Hat OpenShift I: Containers & Kubernetes (DO180) and OpenShift Container Platform Administration II (DO280). Every student gets a complete remote classroom environment. As part of that environment, every student gets a dedicated OpenShift cluster to perform administration tasks.

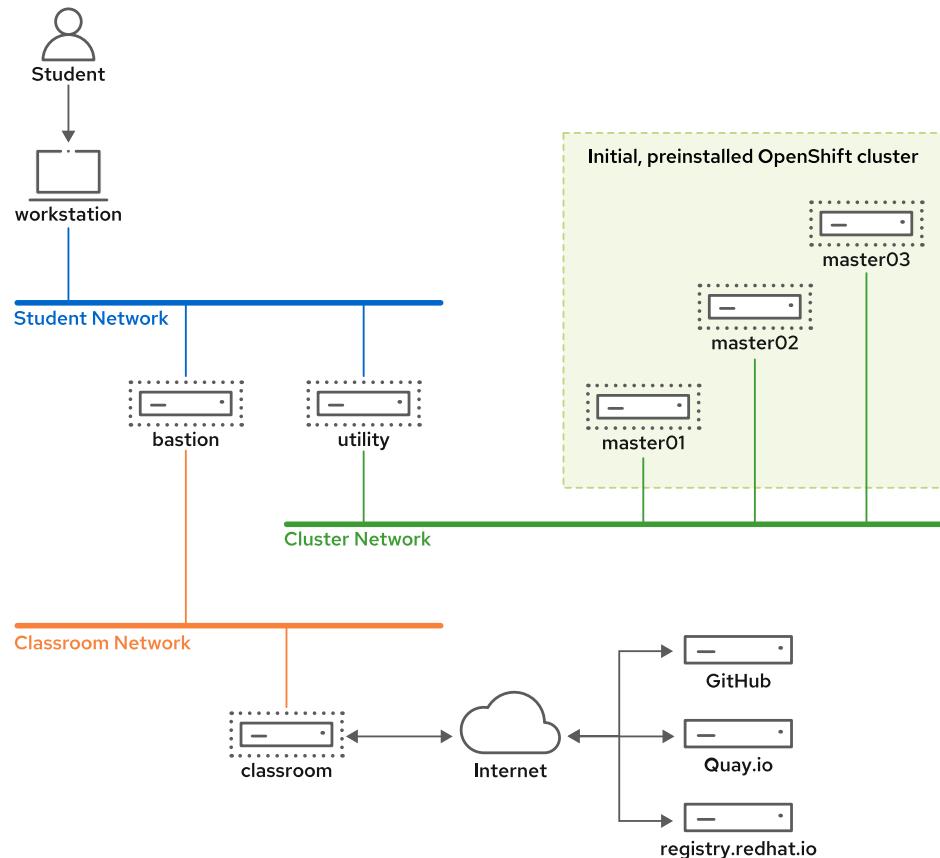
Many exercises from the DO180 portion of the course require that you complete the tasks in *Guided Exercise: Configuring the Classroom Environment*. This includes running the **lab-configure** command and cloning your fork of the <https://github.com/RedHatTraining/DO180-apps> git repository to **/home/student/DO180-apps**. The **lab-configure** command prompts for your GitHub and Quay.io user names and saves changes to the **/usr/local/etc/ocp4.config** file.

By default, the OpenShift cluster in the classroom environment uses the HTPasswd identity provider and allows the **developer** user access with the **developer** password. Because configuring OpenShift to use the HTPasswd identity provider is an objective of the DO280 portion of the course, some lab scripts remove the identity provider configuration. If you go back

to the DO180 portion of the course, then run the **lab-configure** command to ensure the OpenShift cluster allows access to the **developer** user with the **developer** password.

The classroom environment runs entirely as virtual machines in a large Red Hat OpenStack Platform cluster, which is shared among many students.

Red Hat Training maintains many OpenStack clusters, in different data centers across the globe, to provide lower latency to students from many countries.



All machines on the Student, Classroom, and Cluster Networks run Red Hat Enterprise Linux 8 (RHEL 8), except those machines that are nodes of the OpenShift cluster. These machines run RHEL CoreOS.

The systems called **bastion**, **utility**, and **classroom** must always be running. They provide infrastructure services required by the classroom environment and its OpenShift cluster. You are not expected to interact with any of these systems directly.

Usually, the **lab** commands from exercises access these machines when there is a requirement to set up your environment for the exercise, and will require no further action from you.

All systems in the *Student Network* are in the **lab.example.com** DNS domain, and all systems in the *Classroom Network* are in the **example.com** DNS domain.

The systems called **masterXX** are nodes of the OpenShift 4 cluster that is part of your classroom environment.

All systems in the *Cluster Network* are in the **ocp4.example.com** DNS domain.

Classroom Machines

Machine name	IP addresses	Role
<code>workstation.lab.example.com</code>	172.25.250.9	Graphical workstation used for system administration.
<code>classroom.example.com</code>	172.25.254.254	Router linking the Classroom Network to the Internet.
<code>bastion.lab.example.com</code>	172.25.250.254	Router linking the Student Network to the Classroom Network.
<code>utility.lab.example.com</code>	172.25.250.253	Router linking the Student Network to the Cluster Network and also storage server.
<code>master01.ocp4.example.com</code>	192.168.50.10	Control plane and compute node
<code>master02.ocp4.example.com</code>	192.168.50.11	Control plane and compute node
<code>master03.ocp4.example.com</code>	192.168.50.12	Control plane and compute node

Dependencies on Internet Services

Red Hat OpenShift Container Platform 4 requires access to two container registries to download container images for operators, S2I builders, and other cluster services. These registries are:

- `registry.redhat.io`
- `quay.io`

If either registry is unavailable when starting the classroom environment, then the OpenShift cluster might not start or could enter a degraded state.

If these container registries experiences an outage while the classroom environment is up and running, then it might not be possible to complete exercises until the outage is resolved.

The Dedicated OpenShift Cluster

The Red Hat OpenShift Container Platform 4 cluster inside the classroom environment is preinstalled using the pre-existing infrastructure installation method; all nodes are treated as bare metal servers, even though they are actually virtual machines in an OpenStack cluster.

OpenShift cloud-provider integration capabilities are not enabled and a few features that depend on that integration, such as machine sets and autoscaling of cluster nodes, are not available.

Restoring Access to your OpenShift Cluster

The start function for most lab scripts ensures that the appropriate prerequisites for the exercise have been completed. Because DO285 combines lab scripts from two separate courses, you might encounter problems if you do not perform the exercises sequentially. If you experience problems with the DO180 portion of the course, then run the `lab-configure` command to ensure the OpenShift cluster allows access to the `developer` user with the `developer` password. Running `lab-configure -d` allows you to enter new GitHub and Quay.io user names.

Introduction

Most exercises in the DO280 portion of the course provide access to the **admin** and **developer** OpenShift users. If you suspect that you cannot log in to your OpenShift cluster as the **admin** user because you incorrectly changed your cluster authentication settings, then run the **lab finish** command from your current exercise and restart the exercise by running its **lab start** command.

For labs that expect the **admin** and **developer** users, the **lab** command resets cluster authentication settings and restores passwords so that the **admin** user has a password of **redhat** and the **developer** user has a password of **developer**.

If running a **lab** command is not sufficient, then you can follow instructions in the next section to use the **utility** machine to access your OpenShift cluster.

Troubleshooting Access to your OpenShift Cluster

The **utility** machine was used to run the OpenShift installer inside your classroom environment, and it is a useful resource to troubleshoot cluster issues. You can view the installer manifests in the **/home/lab/ocp4** folder of the **utility** machine.

Logging in to the **utility** server is not required to perform exercises. If it looks like your OpenShift cluster is taking too long to start, or is in a degraded state, then you can log in to the **utility** machine as the **lab** user to troubleshoot your classroom environment.

The **student** user on the **workstation** machine is already configured with SSH keys that enable logging in to the **utility** machine without a password.

```
[student@workstation ~]$ ssh lab@utility
```

In the **utility** machine, the **lab** user is preconfigured with a **.kube/config** file that grants access as **system:admin** without first requiring **oc login**.

This allows you to run troubleshooting commands, such as **oc get node**, if they fail from the **workstation** machine.

You should not require SSH access to your OpenShift cluster nodes for regular administration tasks because OpenShift 4 provides the **oc debug** command; if necessary, the **lab** user on the **utility** server is preconfigured with SSH keys to access all cluster nodes. For example:

```
[lab@utility ~]$ ssh -i ~/.ssh/lab_rsa core@master01.ocp4.example.com
```

In the preceding example, replace **master01** with the name of the desired cluster node.

Approving Node Certificates on your OpenShift Cluster

Red Hat OpenShift Container Platform clusters are designed to run continuously until they are decommissioned. Unlike a production cluster, the classroom environment contains a cluster that was stopped after installation, and will be stopped and restarted a few times before you finish this course. This scenario requires special handling that would not be required by a production cluster.

The control plane and compute nodes in an OpenShift cluster communicate frequently with each other. All communication between cluster nodes is protected by mutual authentication based on per-node TLS certificates.

The OpenShift installer handles creating and approving TLS certificate signing requests (CSRs) for the full-stack automation installation method. The system administrator is expected to manually approve these CSRs for the pre-existing infrastructure installation method.

Introduction

All per-node TLS certificates have a short expiration life of 24 hours (the first time) and 30 days (after renewal). When they are about to expire, the affected cluster nodes create new CSRs and the control plane automatically approves them. If the control plane is offline when the TLS certificate of a node expires, then a cluster administrator is required to approve the pending CSR.

The **utility** machine includes a system service that approves CSRs from the cluster when you start your classroom to ensure that your cluster is ready when you begin the exercises. If you create or start your classroom and begin an exercise too quickly, then you might find that your cluster is not yet ready. If so, wait a few minutes while the **utility** machine handles CSRs and then try again.

Sometimes, the **utility** machine fails to approve all required CSRs, for example, because the cluster took too long to generate all required CSRs requests and the system service did not wait long enough. It's also possible that some OpenShift cluster nodes did not wait long enough for their CSRs to be approved, issuing new CSRs that superseded previous ones.

If these issues arise, then you will notice that your cluster is taking too long to come up and your **oc login** or **lab** commands keep failing. To resolve the problem, you can log in to the **utility** machine, as explained previously, and run the **sign.sh** script to approve any additional and pending CSRs.

```
[lab@utility ~]$ ./sign.sh
```

The **sign.sh** script loops a few times just in case your cluster nodes issue new CSRs that supersede the ones it approved.

After either you approve or the system service in the **utility** machine approves all CSRs, then OpenShift must restart a few cluster operators; it takes a few moments before your OpenShift cluster is ready to answer requests from clients. To help you handle this scenario, the **utility** machine provides the **wait.sh** script that waits until your OpenShift cluster is ready to accept authentication and API requests from remote clients.

```
[lab@utility ~]$ ./wait.sh
```

Although unlikely, if neither the service on the **utility** machine nor running the **sign.sh** and **wait.sh** scripts make your OpenShift cluster available to begin exercises, then open a customer support ticket.



Note

You can run troubleshooting commands from the **utility** machine at any time, even if you have control plane nodes that are not ready. Some useful commands include:

- **oc get node** to verify if all of your cluster nodes are ready.
- **oc get csr** to verify if your cluster still has any pending, unapproved CSRs.
- **oc get co** to verify if any of your cluster operators are unavailable, in a degraded state, or progressing through configuration and rolling out pods.

If these fail, you can try destroying and recreating your classroom as a last resort before creating a customer support ticket.

Controlling Your Systems

You 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>]. You should log in to this site using your 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.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. You can log in directly to the virtual machine and run commands. In most cases, you 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.

Button or Action	Description
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 you 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 **+** to add an additional hour to the timer. Set the number of hours until the classroom should automatically stop. Note that there is a maximum time of ten hours.

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.

Introduction

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
```

Introduction

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

```
[root@host ~]# yum repoquery --whatsonplements 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

Introducing Container Technology

Goal

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

Objectives

- Describe the difference between container applications and traditional deployments.
- Describe the basics of container architecture.
- Describe the benefits of orchestrating applications and OpenShift Container Platform.

Sections

- Overview of Container Technology (and Quiz)
- Overview of Container Architecture (and Quiz)
- Overview of Kubernetes and OpenShift (and Quiz)
- Guided Exercise: Configuring the Classroom Environment

Overview of Container Technology

Objectives

After completing this section, students should be able to describe the difference between container applications and traditional deployments.

Containerized Applications

Software applications typically depend on other libraries, configuration files, or services that are provided by the runtime environment. The traditional runtime environment for a software application is a physical host or virtual machine, and application dependencies are installed as part of the host.

For example, consider a Python application that requires access to a common shared library that implements the TLS protocol. Traditionally, a system administrator installs the required package that provides the shared library before installing the Python application.

The major drawback to traditionally deployed software application is that the application's dependencies are entangled with the runtime environment. An application may break when any updates or patches are applied to the base operating system (OS).

For example, an OS update to the TLS shared library removes TLS 1.0 as a supported protocol. This breaks the deployed Python application because it is written to use the TLS 1.0 protocol for network requests. This forces the system administrator to roll back the OS update to keep the application running, preventing other applications from using the benefits of the updated package. Therefore, a company developing traditional software applications may require a full set of tests to guarantee that an OS update does not affect applications running on the host.

Furthermore, a traditionally deployed application must be stopped before updating the associated dependencies. To minimize application downtime, organizations design and implement complex systems to provide high availability of their applications. Maintaining multiple applications on a single host often becomes cumbersome, and any deployment or update has the potential to break one of the organization's applications.

Figure 1.1 describes the difference between applications running as containers and applications running on the host operating system.

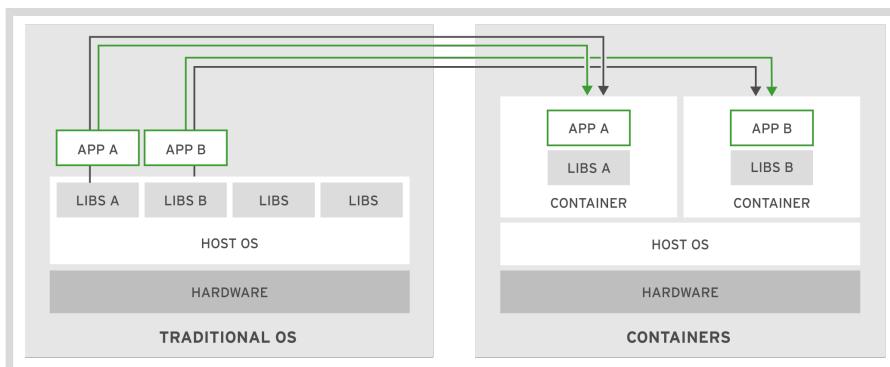


Figure 1.1: Container versus operating system differences

Chapter 1 | Introducing Container Technology

Alternatively, a software application can be deployed using a *container*. A container is a set of one or more processes that are isolated from the rest of the system. Containers provide many of the same benefits as virtual machines, such as security, storage, and network isolation. Containers require far fewer hardware resources and are quick to start and terminate. They also isolate the libraries and the runtime resources (such as CPU and storage) for an application to minimize the impact of any OS update to the host OS, as described in *Figure 1.1*.

The use of containers not only helps with the efficiency, elasticity, and reusability of the hosted applications, but also with application portability. The *Open Container Initiative* provides a set of industry standards that define a container runtime specification and a container image specification. The image specification defines the format for the bundle of files and metadata that form a container image. When you build an application as a container image, which complies with the OCI standard, you can use any OCI-compliant container engine to execute the application.

There are many *container engines* available to manage and execute individual containers, including Rocket, Drawbridge, LXC, Docker, and Podman. Podman is available in Red Hat Enterprise Linux 7.6 and later, and is used in this course to start, manage, and terminate individual containers.

The following are other major advantages to using containers:

Low hardware footprint

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

Environment isolation

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

Quick deployment

Containers deploy quickly because there is no need to install the entire underlying operating system. Normally, to support the isolation, a new OS installation is required on a physical host or VM, and any simple update might require a full OS restart. A container restart does not require stopping any services on the host OS.

Multiple environment deployment

In a traditional deployment scenario using a single host, any environment differences could break the application. Using containers, however, all application dependencies and environment settings are encapsulated in the container image.

Reusability

The same container can be reused without the need to set up a full OS. For example, the same database container that provides a production database service can be used by each developer to create a development database during application development. Using containers, there is no longer a need to maintain separate production and development database servers. A single container image is used to create instances of the database service.

Often, a software application with all of its dependent services (databases, messaging, file systems) are made to run in a single container. This can lead to the same problems associated

with traditional software deployments to virtual machines or physical hosts. In these instances, a multicontainer deployment may be more suitable.

Furthermore, containers are an ideal approach when using microservices for application development. Each service is encapsulated in a lightweight and reliable container environment that can be deployed to a production or development environment. The collection of containerized services required by an application can be hosted on a single machine, removing the need to manage a machine for each service.

In contrast, many applications are not well suited for a containerized environment. For example, applications accessing low-level hardware information, such as memory, file systems, and devices may be unreliable due to container limitations.



References

Home - Open Containers Initiative

<https://www.opencontainers.org/>

► Quiz

Overview of Container Technology

Choose the correct answers to the following questions:

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

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

► 2. Which two of the following use cases are best suited for containers? (Choose two.)

- a. A software provider needs to distribute software that can be reused by other companies in a fast and error-free way.
- b. A company is deploying applications on a physical host and would like to improve its performance by using containers.
- c. Developers at a company need a disposable environment that mimics the production environment so that they can quickly test the code they develop.
- d. A financial company is implementing a CPU-intensive risk analysis tool on their own containers to minimize the number of processors needed.

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

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

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

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

► Solution

Overview of Container Technology

Choose the correct answers to the following questions:

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

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

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

- a. A software provider needs to distribute software that can be reused by other companies in a fast and error-free way.
- b. A company is deploying applications on a physical host and would like to improve its performance by using containers.
- c. Developers at a company need a disposable environment that mimics the production environment so that they can quickly test the code they develop.
- d. A financial company is implementing a CPU-intensive risk analysis tool on their own containers to minimize the number of processors needed.

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

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

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

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

Overview of Container Architecture

Objectives

After completing this section, students should be able to:

- Describe the architecture of Linux containers.
- Install the **podman** utility to manage containers.

Introducing Container History

Containers have quickly gained popularity in recent years. However, the technology behind containers has been around for a relatively long time. In 2001, Linux introduced a project named VServer. VServer was the first attempt at running complete sets of processes inside a single server with a high degree of isolation.

From VServer, the idea of isolated processes further evolved and became formalized around the following features of the Linux kernel:

Namespaces

The kernel can isolate specific system resources, usually visible to all processes, by placing the resources within a namespace. Inside a namespace, only processes that are members of that namespace can see those resources. Namespaces can include resources like network interfaces, the process ID list, mount points, IPC resources, and the system's host name information.

Control groups (cgroups)

Control groups partition sets of processes and their children into groups to manage and limit the resources they consume. Control groups place restrictions on the amount of system resources processes might use. Those restrictions keep one process from using too many resources on the host.

Seccomp

Developed in 2005 and introduced to containers circa 2014, Seccomp limits how processes could use system calls. Seccomp defines a security profile for processes, whitelisting the system calls, parameters and file descriptors they are allowed to use.

SELinux

SELinux (Security-Enhanced Linux) is a mandatory access control system for processes. Linux kernel uses SELinux to protect processes from each other and to protect the host system from its running processes. Processes run as a confined SELinux type that has limited access to host system resources.

All of these innovations and features focus around a basic concept: enabling processes to run isolated while still accessing system resources. This concept is the foundation of container technology and the basis for all container implementations. Nowadays, containers are processes in Linux kernel making use of those security features to create an isolated environment. This environment forbids isolated processes from misusing system or other container resources.

A common use case of containers is having several replicas of the same service (for example, a database server) in the same host. Each replica has isolated resources (file system, ports,

memory), so there is no need for the service to handle resource sharing. Isolation guarantees that a malfunctioning or harmful service does not impact other services or containers in the same host, nor in the underlying system.

Describing Linux Container Architecture

From the Linux kernel perspective, a container is a process with restrictions. However, instead of running a single binary file, a container runs an *image*. An image is a file-system bundle that contains all dependencies required to execute a process: files in the file system, installed packages, available resources, running processes, and kernel modules.

Like executable files are the foundation for running processes, images are the foundation for running containers. Running containers use an immutable view of the image, allowing multiple containers to reuse the same image simultaneously. As images are files, they can be managed by versioning systems, improving automation on container and image provisioning.

Container images need to be locally available for the container runtime to execute them, but the images are usually stored and maintained in an *image repository*. An image repository is just a service - public or private - where images can be stored, searched and retrieved. Other features provided by image repositories are remote access, image metadata, authorization or image version control.

There are many different image repositories available, each one offering different features:

- Red Hat Container Catalog [<https://registry.redhat.io>]
- Docker Hub [<https://hub.docker.com>]
- Red Hat Quay [<https://quay.io/>]
- Google Container Registry [<https://cloud.google.com/container-registry/>]
- Amazon Elastic Container Registry [<https://aws.amazon.com/ecr/>]

This course uses the public image registry Quay, so students can operate with images without worrying about interfering with each other.

Managing Containers with Podman

Containers, images, and image registries need to be able to interact with each other. For example, you need to be able to build images and put them into image registries. You also need to be able to retrieve an image from the image registry and build a container from that image.

Podman is an open source tool for managing containers and container images and interacting with image registries. It offers the following key features:

- It uses image format specified by the Open Container Initiative [<https://www.opencontainers.org>] (OCI). Those specifications define a standard, community-driven, non-proprietary image format.
- Podman stores local images in local file-system. Doing so avoids unnecessary client/server architecture or having daemons running on local machine.
- Podman follows the same command patterns as the Docker CLI, so there is no need to learn a new toolset.
- Podman is compatible with Kubernetes. Kubernetes can use Podman to manage its containers.

Currently, Podman is only available on Linux systems. To install Podman in Red Hat Enterprise Linux, Fedora or similar RPM-based systems, run **sudo yum install podman** or **sudo dnf install podman**.



References

Red Hat Quay Container Registry

<https://quay.io>

Podman site

<https://podman.io/>

Open Container Initiative

<https://www.opencontainers.org>

► Quiz

Overview of Container Architecture

Choose the correct answers to the following questions:

- ▶ **1. Which three of the following Linux features are used for running containers? (Choose three.)**
 - a. Namespaces
 - b. Integrity Management
 - c. Security-Enhanced Linux
 - d. Control Groups

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

- ▶ **3. Which three of the following components are common across container architecture implementations? (Choose three.)**
 - a. Container runtime
 - b. Container permissions
 - c. Container images
 - d. Container registries

- ▶ **4. What is a container in relation to the Linux kernel?**
 - a. A virtual machine.
 - b. An isolated process with regulated resource access.
 - c. A set of file-system layers exposed by UnionFS.
 - d. An external service providing container images.

► Solution

Overview of Container Architecture

Choose the correct answers to the following questions:

- ▶ 1. **Which three of the following Linux features are used for running containers? (Choose three.)**
 - a. Namespaces
 - b. Integrity Management
 - c. Security-Enhanced Linux
 - d. Control Groups

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

- ▶ 3. **Which three of the following components are common across container architecture implementations? (Choose three.)**
 - a. Container runtime
 - b. Container permissions
 - c. Container images
 - d. Container registries

- ▶ 4. **What is a container in relation to the Linux kernel?**
 - a. A virtual machine.
 - b. An isolated process with regulated resource access.
 - c. A set of file-system layers exposed by UnionFS.
 - d. An external service providing container images.

Overview of Kubernetes and OpenShift

Objectives

After completing this section, students should be able to:

- Identify the limitations of Linux containers and the need for container orchestration.
- Describe the Kubernetes container orchestration tool.
- Describe Red Hat OpenShift Container Platform (RHOC).

Limitations of Containers

Containers provide an easy way to package and run services. As the number of containers managed by an organization grows, the work of manually starting them rises exponentially along with the need to quickly respond to external demands.

When using containers in a production environment, enterprises often require:

- Easy communication between a large number of services.
- Resource limits on applications regardless of the number of containers running them.
- Respond to application usage spikes to increase or decrease running containers.
- React to service deterioration.
- Gradually roll out a new release to a set of users.

Enterprises often require a container orchestration technology because container runtimes (such as Podman) do not adequately address the above requirements.

Kubernetes Overview

Kubernetes is an orchestration service that simplifies the deployment, management, and scaling of containerized applications.

The smallest unit manageable in Kubernetes is a pod. A pod consists of one or more containers with its storage resources and IP address that represent a single application. Kubernetes also uses pods to orchestrate the containers inside it and to limit its resources as a single unit.

Kubernetes Features

Kubernetes offers the following features on top of a container infrastructure:

Service discovery and load balancing

Kubernetes enables inter-service communication by assigning a single DNS entry to each set of containers. This way, the requesting service only needs to know the target's DNS name, allowing the cluster to change the container's location and IP address, leaving the service unaffected. This permits load-balancing the request across the pool of containers providing the service. For example, Kubernetes can evenly split incoming requests to a MySQL service taking into account the availability of the pods.

Horizontal scaling

Applications can scale up and down manually or automatically with configuration set either with the Kubernetes command-line interface or the web UI.

Self-healing

Kubernetes can use user-defined health checks to monitor containers to restart and reschedule them in case of failure.

Automated rollout

Kubernetes can gradually roll updates out to your application's containers while checking their status. If something goes wrong during the rollout, Kubernetes can roll back to the previous iteration of the deployment.

Secrets and configuration management

You can manage configuration settings and secrets of your applications without rebuilding containers. Application secrets can be user names, passwords, and service endpoints; any configuration settings that need to be kept private.

Operators

Operators are packaged Kubernetes applications that also bring the knowledge of the application's life cycle into the Kubernetes cluster. Applications packaged as Operators use the Kubernetes API to update the cluster's state reacting to changes in the application state.

OpenShift Overview

Red Hat OpenShift Container Platform (RHOCP) is a set of modular components and services built on top of a Kubernetes container infrastructure. RHOCP adds the capabilities to provide a production PaaS platform such as remote management, multitenancy, increased security, monitoring and auditing, application life-cycle management, and self-service interfaces for developers.

Beginning with Red Hat OpenShift v4, hosts in an OpenShift cluster all use Red Hat Enterprise Linux CoreOS as the underlying operating system.

Throughout this course, the terms RHOCP and OpenShift are used to refer to the Red Hat OpenShift Container Platform.

OpenShift Features

OpenShift adds the following features to a Kubernetes cluster:

Integrated developer workflow

RHOCP integrates a built-in container registry, CI/CD pipelines, and S2I; a tool to build artifacts from source repositories to container images.

Routes

Easily expose services to the outside world.

Metrics and logging

Include built-in and self-analyzing metrics service and aggregated logging.

Unified UI

OpenShift brings unified tools and a UI to manage all the different capabilities.



References

Production-Grade Container Orchestration - Kubernetes

<https://kubernetes.io/>

OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes

<https://www.openshift.com/>

► Quiz

Describing Kubernetes and OpenShift

Choose the correct answers to the following questions:

- ▶ **1. Which three of the following statements are correct regarding container limitations? (Choose three.)**
 - a. Containers are easily orchestrated in large numbers.
 - b. Lack of automation increases response time to problems.
 - c. Containers do not manage application failure inside them.
 - d. Containers are not load-balanced.
 - e. Containers are heavily isolated packaged applications.

- ▶ **2. Which two of the following statements are correct regarding Kubernetes? (Choose two.)**
 - a. Kubernetes is a container.
 - b. Kubernetes can only use Docker containers.
 - c. Kubernetes is a container orchestration system.
 - d. Kubernetes simplifies management, deployment, and scaling of containerized applications.
 - e. Applications managed in a Kubernetes cluster are harder to maintain.

- ▶ **3. Which three of the following statements are true regarding Red Hat OpenShift v4? (Choose three.)**
 - a. OpenShift provides additional features to a Kubernetes infrastructure.
 - b. Kubernetes and OpenShift are mutually exclusive.
 - c. OpenShift hosts use Red Hat Enterprise Linux as the base operating system.
 - d. OpenShift simplifies development incorporating a Source-to-Image technology and CI/CD pipelines.
 - e. OpenShift simplifies routing and load balancing.

- ▶ **4. What features does OpenShift offer that extend Kubernetes capabilities? (choose two.)**
 - a. Operators and the Operator Framework.
 - b. Routes to expose services to the outside world.
 - c. An integrated development workflow.
 - d. Self-healing and health checks.

► Solution

Describing Kubernetes and OpenShift

Choose the correct answers to the following questions:

- ▶ **1. Which three of the following statements are correct regarding container limitations? (Choose three.)**
 - a. Containers are easily orchestrated in large numbers.
 - b. Lack of automation increases response time to problems.
 - c. Containers do not manage application failure inside them.
 - d. Containers are not load-balanced.
 - e. Containers are heavily isolated packaged applications.

- ▶ **2. Which two of the following statements are correct regarding Kubernetes? (Choose two.)**
 - a. Kubernetes is a container.
 - b. Kubernetes can only use Docker containers.
 - c. Kubernetes is a container orchestration system.
 - d. Kubernetes simplifies management, deployment, and scaling of containerized applications.
 - e. Applications managed in a Kubernetes cluster are harder to maintain.

- ▶ **3. Which three of the following statements are true regarding Red Hat OpenShift v4? (Choose three.)**
 - a. OpenShift provides additional features to a Kubernetes infrastructure.
 - b. Kubernetes and OpenShift are mutually exclusive.
 - c. OpenShift hosts use Red Hat Enterprise Linux as the base operating system.
 - d. OpenShift simplifies development incorporating a Source-to-Image technology and CI/CD pipelines.
 - e. OpenShift simplifies routing and load balancing.

- ▶ **4. What features does OpenShift offer that extend Kubernetes capabilities? (choose two.)**
 - a. Operators and the Operator Framework.
 - b. Routes to expose services to the outside world.
 - c. An integrated development workflow.
 - d. Self-healing and health checks.

► Guided Exercise

Configuring the Classroom Environment

In this exercise, you will configure the **workstation** machine for the DO180 portion of the course.

Outcomes

You should be able to:

- Configure your **workstation** machine to access an OpenShift cluster, a container image registry, and a Git repository used throughout the course.
- Fork the Red Hat DO180 sample applications repository to your personal GitHub account.
- Clone the DO180 sample applications repository from your personal GitHub account to your **workstation** machine.

Before You Begin

To perform this exercise, ensure you have:

- A personal, free GitHub account. If you need to register to GitHub, see the instructions in *Appendix A, Creating a GitHub Account*.
- A personal, free Quay.io account. If you need to register to Quay.io, see the instructions in *Appendix B, Creating a Quay Account*.

- 1. Open a terminal on your **workstation** machine and execute the **lab-configure** command. Answer the interactive prompts. You can interrupt the command at any time using **Ctrl+C** and start over.

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

The **lab-configure** command prompts for your GitHub and Quay.io account names. The command exits with an error message if one of the entered account names does not exist.

- Enter the GitHub account name: **your_github_username**
Verifying GitHub account name: **your_github_username**
- Enter the Quay.io account name: **your_quay_username**
Verifying Quay.io account name: **your_quay_username**

If both account names exist, then the **lab-configure** command configures the **/usr/local/etc/ocp4.config** file with variables required by some exercises.

- Configuring RHT_OCP4_GITHUB_USER variable: SUCCESS
- Configuring RHT_OCP4_QUAY_USER variable: SUCCESS

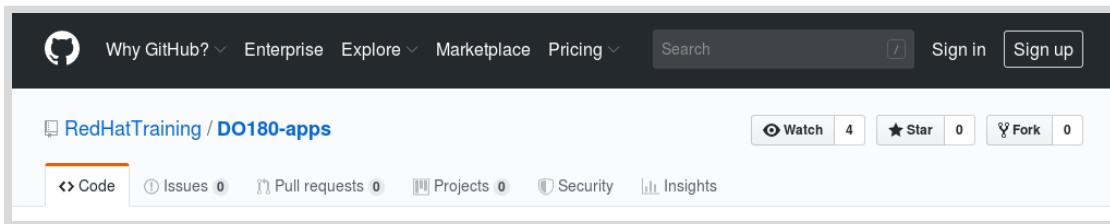
If you previously completed the **lab-configure** command, then running the command again displays your GitHub and Quay.io account names. Additionally, the command ensures

that the **developer** user can log in to the OpenShift cluster. The command configures the OpenShift cluster to use the HTPasswd identity provider if necessary. This is useful if you return to DO180 exercises after completing some DO280 exercises.

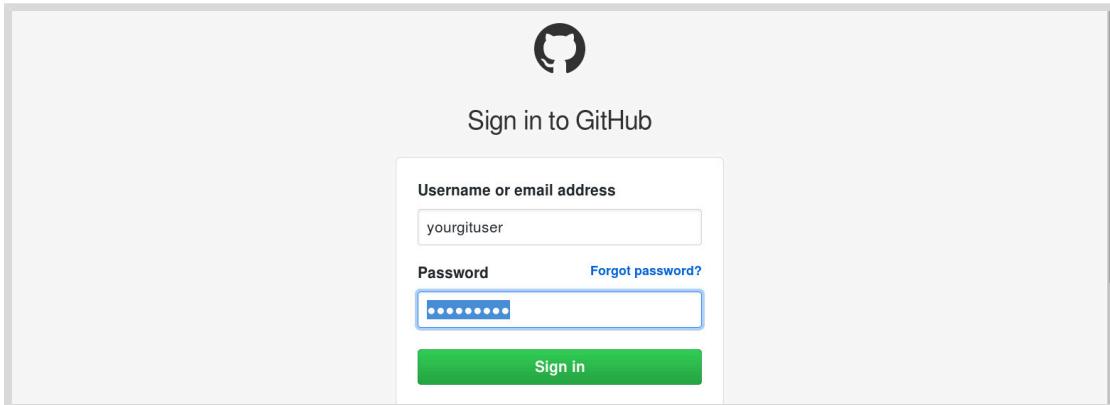
- . GitHub account name: your_github_username
- . Quay.io account name: your_quay_username
- . To reconfigure, run: lab-configure -d
- . Ensuring user 'developer' can log in to the OpenShift cluster.

► 2. Some DO180 exercises require a forked version of the DO180 sample applications saved to your personal GitHub account. Perform the following steps:

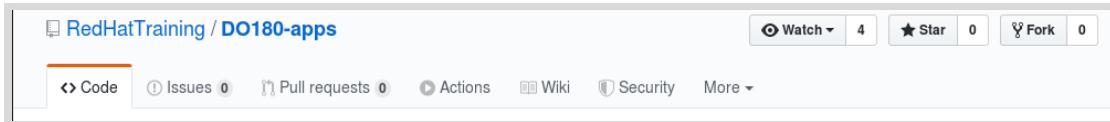
- 2.1. Open a web browser and navigate to <https://github.com/RedHatTraining/DO180-apps>. If you are not logged in to GitHub, click **Sign in** in the upper-right corner.



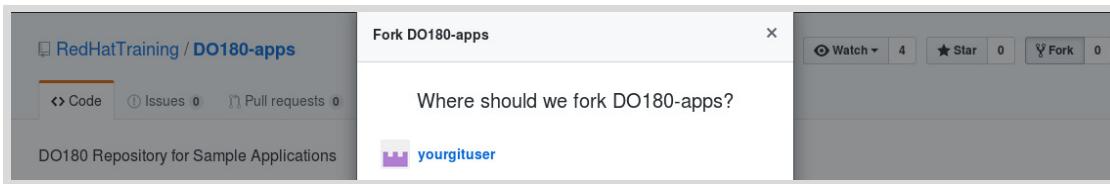
- 2.2. Log in to GitHub using your personal user name and password.



- 2.3. Return to the **RedHatTraining/DO180-apps** repository and click **Fork** in the upper-right corner.

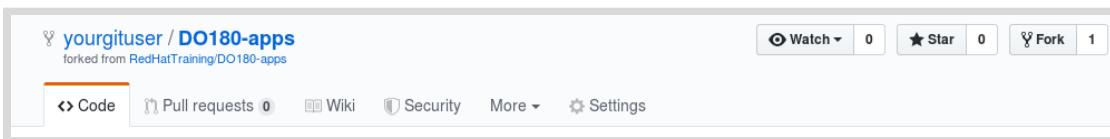


- 2.4. In the **Fork DO180-apps** window, click **yourgituser** to select your personal GitHub project.

**Important**

While it is possible to rename your personal fork of the <https://github.com/RedHatTraining/DO180-apps> repository, grading scripts, helper scripts, and the example output in this course assume that you retain the name **DO180-apps** when your fork the repository.

- 2.5. After a few minutes, the GitHub web interface displays your new repository **yourgituser/DO180-apps**.



- ▶ 3. Some DO180 exercises require a copy of the DO180 sample applications from your personal GitHub account cloned to the **/home/student/DO180-apps** directory. Perform the following steps:

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

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

- 3.2. Run the following command to clone the DO180 sample applications repository.

```
[student@workstation ~]$ git clone \
> https://github.com/${RHT_OCP4_GITHUB_USER}/DO180-apps
Cloning into 'DO180-apps'...
...output omitted...
```

- 3.3. Verify that the **/home/student/DO180-apps** directory is a Git repository:

```
[student@workstation ~]$ cd ~/DO180-apps
[student@workstation DO180-apps]$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

- 3.4. Verify that the **/home/student/DO180-apps** directory contains the DO180 sample applications.

```
[student@workstation DO180-apps]$ head README.md  
# DO180-apps  
DO180 Repository for Sample Applications
```

3.5. Change back to the **/home/student** directory.

```
[student@workstation DO180-apps]$ cd  
[student@workstation ~]$
```

- ▶ **4.** Now that you have a local clone of the **DO180-apps** repository on your **workstation** machine, and you have executed the **lab-configure** command successfully, you are ready to start exercises from the DO180 portion of the course.

During this course, all exercises that build applications from source start from the **master** branch of the **DO180-apps** Git repository. Exercises that make changes to source code require you to create new branches to host your changes, so that the **master** branch always contains a known good starting point. If for some reason you need to pause or restart an exercise, and you need to either save or discard changes you made to your Git branches, refer to *Appendix C, Useful Git Commands*.

This concludes the guided exercise.

Summary

In this chapter, you learned:

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

Chapter 2

Creating Containerized Services

Goal

Provision a service using container technology.

Objectives

- Create a database server from a container image.

Sections

- Provisioning Containerized Services (and Guided Exercise)

Provisioning Containerized Services

Objectives

After completing this section, students should be able to:

- Search for and fetch container images with Podman.
- Run and configure containers locally.
- Use the Red Hat Container Catalog.

Fetching Container Images with Podman

Applications can run inside containers as a way to provide them with an isolated and controlled execution environment. Running a containerized application, that is, running an application inside a container, requires a container image, a file system bundle providing all application files, libraries, and dependencies the application needs to run. Container images can be found in image registries: services that allow users to search and retrieve container images. Podman users can use the **search** subcommand to find available images from remote or local registries:

```
[student@workstation ~]$ sudo podman search rhel
INDEX      NAME          DESCRIPTION  STARS OFFICIAL AUTOMATED
redhat.com  registry.access.redhat.com/rhel This plat... 0
...output omitted...
```

After you have found an image, you can use Podman to download it. When using the **pull** subcommand, Podman fetches the image and saves it locally for future use:

```
[student@workstation ~]$ sudo podman pull rhel
Trying to pull registry.access.redhat.com/rhel...Getting image source signatures
Copying blob sha256: ...output omitted...
  72.25 MB / 72.25 MB [=====] 8s
Copying blob sha256: ...output omitted...
  1.20 KB / 1.20 KB [=====] 0s
Copying config sha256: ...output omitted...
  6.30 KB / 6.30 KB [=====] 0s
Writing manifest to image destination
Storing signatures
699d44bc6ea2b9fb23e7899bd4023d3c83894d3be64b12e65a3fe63e2c70f0ef
```

Container images are named based on the following syntax:

registry_name/user_name/image_name:tag

- First **registry_name**, the name of the registry storing the image. It is usually the FQDN of the registry.
- **user_name** stands for the user or organization the image belongs to.
- The **image_name** should be unique in user namespace.

- The **tag** identifies the image version. If the image name includes no image tag, **latest** is assumed.

**Note**

This classroom's Podman installation uses several publicly available registries, like Quay.io and Red Hat Container Catalog.

After retrieval, Podman stores images locally and you can list them with the **images** subcommand:

```
[student@workstation ~]$ sudo podman images
REPOSITORY           TAG      IMAGE ID      CREATED       SIZE
registry.access.redhat.com/rhel   latest   699d44bc6ea2  4 days ago  214MB
...output omitted...
```

Running Containers

The **podman run** command runs a container locally based on an image. At a minimum, the command requires the name of the image to execute in the container.

The container image specifies a process that starts inside the container known as the *entry point*. The **podman run** command uses all parameters after the image name as the entry point command for the container. The following example starts a container from a Red Hat Enterprise Linux image. It sets the entry point for this container to the **echo "Hello world"** command.

```
[student@workstation ~]$ sudo podman run ubi7/ubi:7.7 echo 'Hello!'
Hello world
```

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

```
[student@workstation ~]$ sudo podman run -d rhscl/httpd-24-rhel7:2.4-36.8
ff4ec6d74e9b2a7b55c49f138e56f8bc46fe2a09c23093664fea7febc3dfa1b2
[student@workstation ~]$ sudo podman inspect -l \
> -f "{{.NetworkSettings.IPAddress}}"
10.88.0.68
[student@workstation ~]$ curl http://10.88.0.68:8080
...output omitted...
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...output omitted...
<title>
Test Page for the Apache HTTP Server on Red Hat Enterprise Linux
</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<style type="text/css">
...output omitted...
```

The previous example ran a containerized Apache HTTP server in the background. Then, the example uses the **podman inspect** command to retrieve the container's internal IP address from container metadata. Finally, it uses the IP address to fetch the root page from Apache HTTP server. This response proves the container is still up and running after the **podman run** command.

**Note**

Most Podman subcommands accept the **-l** flag (**l** for latest) as a replacement for the container id. This flag applies the command to the latest used container in any Podman command.

**Note**

If the image to be executed is not available locally when using the **podman run** command, Podman automatically uses **pull** to download the image.

When referencing the container, Podman recognizes a container either with the container name or the generated container id. Use the **--name** option to set the container name when running the container with Podman. Container names must be unique. If the **podman run** command includes no container name, Podman generates a unique random name.

If the images require interacting with the user with console input, Podman can redirect container input and output streams to the console. The **run** subcommand requires the **-t** and **-i** flags (or, in short, **-it** flag) to enable interactivity.

**Note**

Many Podman flags also have an alternative long form; some of these are explained below.

- **-t** is equivalent to **--tty**, meaning a **pseudo-tty** (pseudo-terminal) is to be allocated for the container.
- **-i** is the same as **--interactive**. When used, standard input is kept open into the container.
- **-d**, or its long form **--detach**, means the container runs in the background (detached). Podman then prints the container id.

See the Podman documentation for the complete list of flags.

The following example starts a Bash terminal *inside* the container, and interactively runs some commands in it:

```
[student@workstation ~]$ sudo podman run -it ubi7/ubi:7.7 /bin/bash
bash-4.2# ls
...output omitted...
bash-4.2# whoami
root
bash-4.2# exit
exit
[student@workstation ~]$
```

Some containers need or can use external parameters provided at startup. The most common approach for providing and consuming those parameters is through environment variables. Podman can inject environment variables into containers at startup by adding the **-e** flag to the **run** subcommand:

```
[student@workstation ~]$ sudo podman run -e GREET=Hello -e NAME=RedHat \
> rhel7:7.5 printenv GREET NAME
Hello
RedHat
[student@workstation ~]$
```

The previous example starts a RHEL image container that prints the two environment variables provided as parameters. Another use case for environment variables is setting up credentials into a MySQL database server:

```
[root@workstation ~]# sudo podman run --name mysql-custom \
> -e MYSQL_USER=redhat -e MYSQL_PASSWORD=r3dh4t \
> -d rhmap47/mysql:5.5
```

Using the Red Hat Container Catalog

Red Hat maintains its repository of finely tuned container images. Using this repository provides customers with a layer of protection and reliability against known vulnerabilities, which could potentially be caused by untested images. The standard **podman** command is compatible with the Red Hat Container Catalog. The Red Hat Container Catalog provides a user-friendly interface for searching and exploring container images from the Red Hat repository.

The Container Catalog also serves as a single interface, providing access to different aspects of all the available container images in the repository. It is useful in determining the best image among multiple versions of container images given health index grades. The health index grade indicates how current an image is, and whether it contains the latest security updates.

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

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

The screenshot shows the Red Hat Container Catalog search results for the term 'Apache'. The main heading is 'Container images' with a subtext: 'Container images offer lightweight and self-contained software to enable deployment at scale.' Below the heading, there's a breadcrumb navigation: 'Home > Software > Container images'. A search bar contains the query 'Apache' with a red 'Search' button. To the right of the search bar are pagination controls showing '1 - 15 of 43' and arrows. On the left, there are two filter dropdowns: 'Provider' (listing ATS, DataStax Inc., IBM, Red Hat, Inc., and Splunk) and 'Category'. The search results are displayed in a grid of cards. The first card is for 'Apache httpd 2.4' (by Red Hat, Inc.) with a description: 'Platform for running Apache httpd 2.4 or building httpd-based application'. The second card is for 'Apache Hive' (by Red Hat, Inc.) with the Red Hat OpenShift logo. The third card is for 'Apache Hadoop' (by Red Hat, Inc.) with the Red Hat OpenShift logo and a description: 'Apache Hadoop is a collection of open-source software utilities that facilitate using a'.

Figure 2.1: Red Hat Container Catalog search page

As displayed above, searching for **Apache** in the search box of the Container Catalog displays a suggested list of products and image repositories matching the search pattern. To access the **Apache httpd 2.4** image page, select **rhscl/httpd-24-rhel7** from the suggested list.

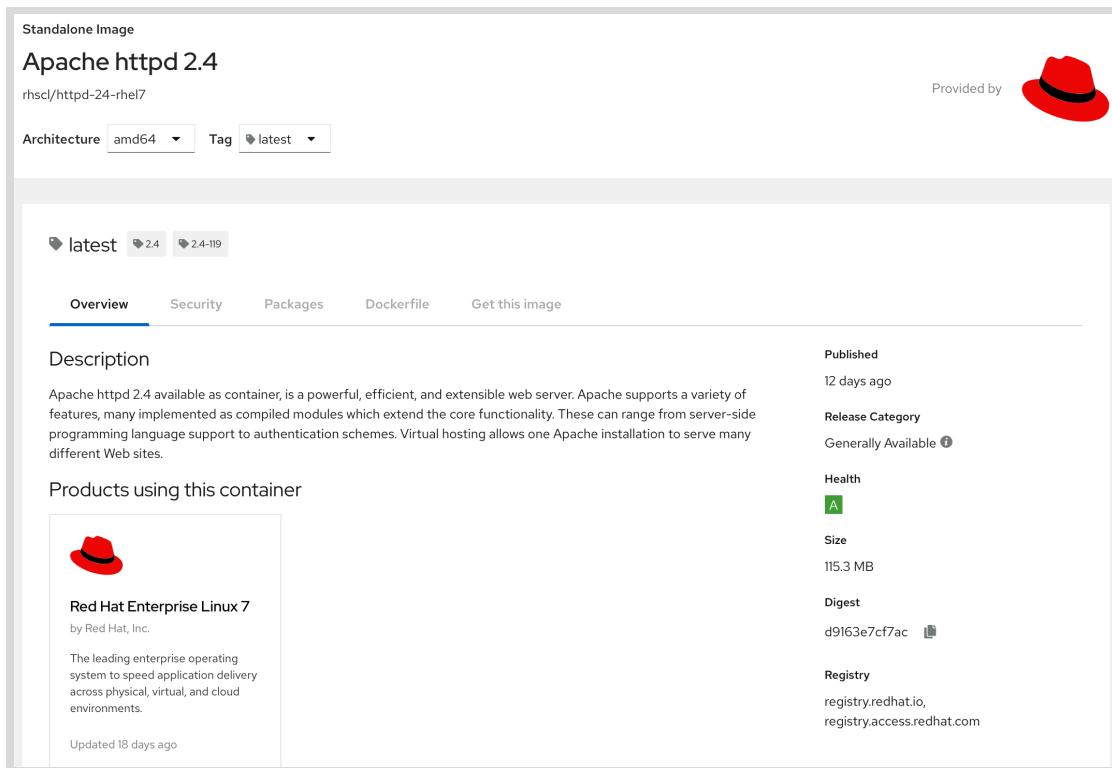


Figure 2.2: Apache httpd 2.4 (rhscl/httpd-24-rhel7) overview image page

The *Apache httpd 2.4* panel displays image details and several tabs. This page states that Red Hat maintains the image repository. Under the *Overview* tab, there are other details:

- *Description*: A summary of the image's capabilities.
- *Products using this container*: It indicates that Red Hat Enterprise Linux uses this image repository.
- *Most Recent Tag*: When the image received its latest update, the latest tag applied to the image, the health of the image, and more.

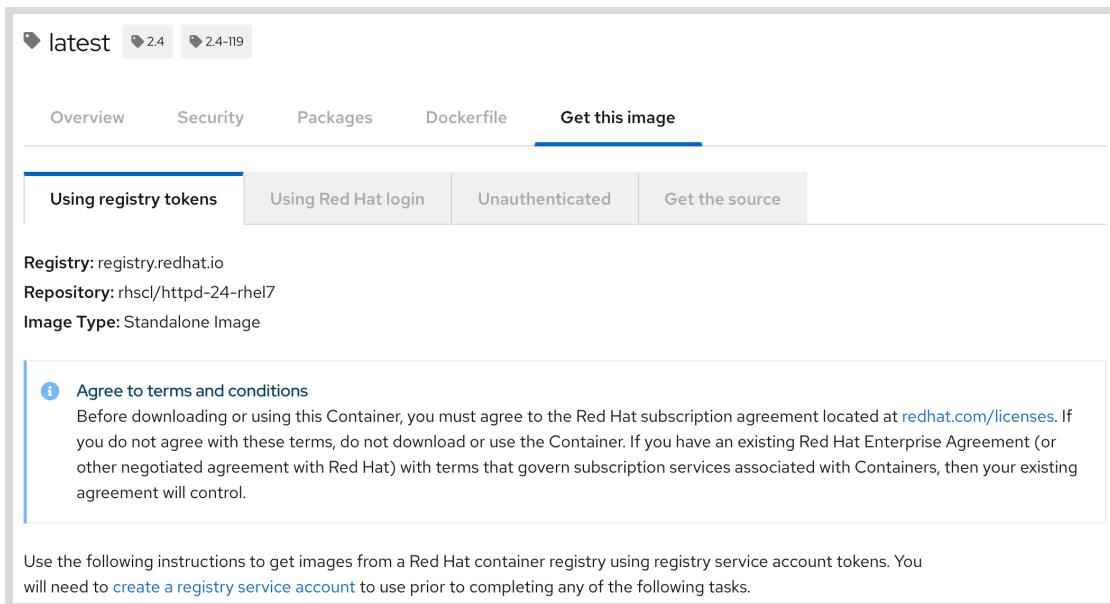


Figure 2.3: Apache httpd 2.4 (rhscl/httpd-24-rhel7) latest image page

The *Get this image* tab provides the procedure to get the most current version of the image. The page provides different options to retrieve the image. Choose your preferred procedure in the tabs, and the page provides the appropriate instructions to retrieve the image.

 **References**

Red Hat Container Catalog
<https://registry.redhat.io>

Quay.io website
<https://quay.io>

► Guided Exercise

Creating a MySQL Database Instance

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

Outcomes

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

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab container-create start
```

- 1. Create a MySQL container instance.

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

```
[student@workstation ~]$ sudo podman run --name mysql-basic \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> -d rh scl/mysql-57-rhel7:5.7-3.14
Trying to pull ...output omitted...
Copying blob sha256:e373541...output omitted...
  69.66 MB / 69.66 MB [=====] 8s
Copying blob sha256:c5d2e94...output omitted...
  1.20 KB / 1.20 KB [=====] 0s
Copying blob sha256:b3949ae...output omitted...
  62.03 MB / 62.03 MB [=====] 8s
Writing manifest to image destination
Storing signatures
92eaa6b67da0475745b2beffa7e0895391ab34ab3bf1ded99363bb09279a24a0
```

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

- 1.2. Verify that the container started without errors.

```
[student@workstation ~]$ sudo podman ps --format "{{.ID}} {{.Image}} {{.Names}}"
92eaa6b67da0 registry.access.redhat.com/rh scl/mysql-57-rhel7:5.7-3.14 mysql-basic
```

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

```
[student@workstation ~]$ sudo podman exec -it mysql-basic /bin/bash
bash-4.2$
```

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

► 3. Add data to the database.

3.1. Connect to MySQL as the database administrator user (root).

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

```
bash-4.2$ mysql -uroot
Welcome to the MySQL monitor. Commands end with ; or \g.
...output omitted...
mysql>
```

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

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| mysql          |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.01 sec)
```

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

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

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

```
mysql> CREATE TABLE Projects (id int(11) NOT NULL,
-> name varchar(255) DEFAULT NULL,
-> code varchar(255) DEFAULT NULL,
-> PRIMARY KEY (id));
Query OK, 0 rows affected (0.01 sec)
```

You can optionally use the [~/DO180/solutions/container-create/create_table.txt](#) file to copy and paste the **CREATE TABLE** MySQL statement as given above.

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

```
mysql> show tables;
+-----+
| Tables_in_items      |
+-----+
| Projects            |
+-----+
1 row in set (0.00 sec)
```

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

```
mysql> insert into Projects (id, name, code) values (1,'DevOps','D0180');
Query OK, 1 row affected (0.02 sec)
```

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

```
mysql> select * from Projects;
+----+-----+-----+
| id | name      | code   |
+----+-----+-----+
| 1  | DevOps    | D0180 |
+----+-----+
1 row in set (0.00 sec)
```

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

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

Finish

On **workstation**, run the **lab container-create finish** script to complete this lab.

```
[student@workstation ~]$ lab container-create finish
```

This concludes the exercise.

Summary

In this chapter, you learned:

- Podman allows users to search for and download images from local or remote registries.
- The **podman run** command creates and starts a container from a container image.
- Containers are executed in the background by using the **-d** flag, or interactively by using the **-it** flag.
- Some container images require environment variables that are set using the **-e** option from the **podman run** command.
- Red Hat Container Catalog assists in searching, exploring, and analyzing container images from Red Hat's official container image repository.

Chapter 3

Managing Containers

Goal

Modify prebuilt container images to create and manage containerized services.

Objectives

- Manage a container's life cycle from creation to deletion.
- Save container application data with persistent storage.
- Describe how to use port forwarding to access a container.

Sections

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

Lab

Managing Containers

Managing the Life Cycle of Containers

Objectives

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

Container Life Cycle Management with Podman

In previous chapters you learned how to use Podman to create a containerized service. Now you will dive deeper into commands and strategies that you can use to manage a container's life cycle. Podman allows you not only to run containers, but also to make them run in the background, execute new processes inside them, and provide them with resources such as file system volumes or a network.

Podman, implemented by the **podman** command, provides a set of subcommands to create and manage containers. Developers use those subcommands to manage container and container images life cycle. The following figure shows a summary of the most commonly used subcommands that change container and image state.

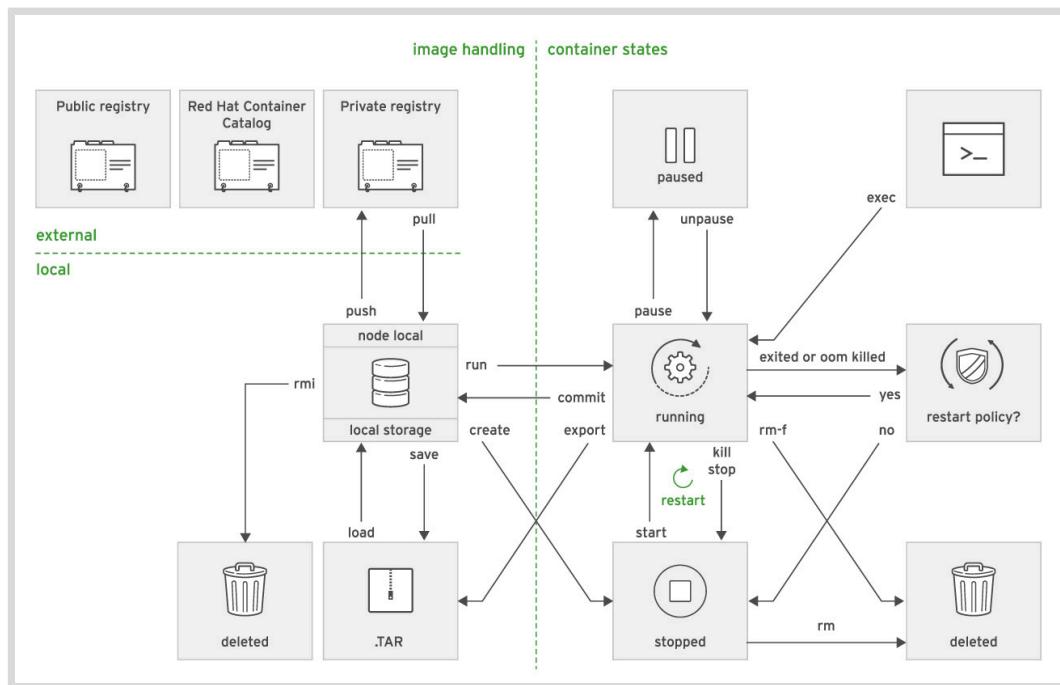


Figure 3.1: Podman managing subcommands

Podman also provides a set of useful subcommands to obtain information about running and stopped containers. You can use those subcommands to extract information from containers and images for debugging, updating, or reporting purposes. The following figure shows a summary of the most commonly used subcommands that query information from containers and images.

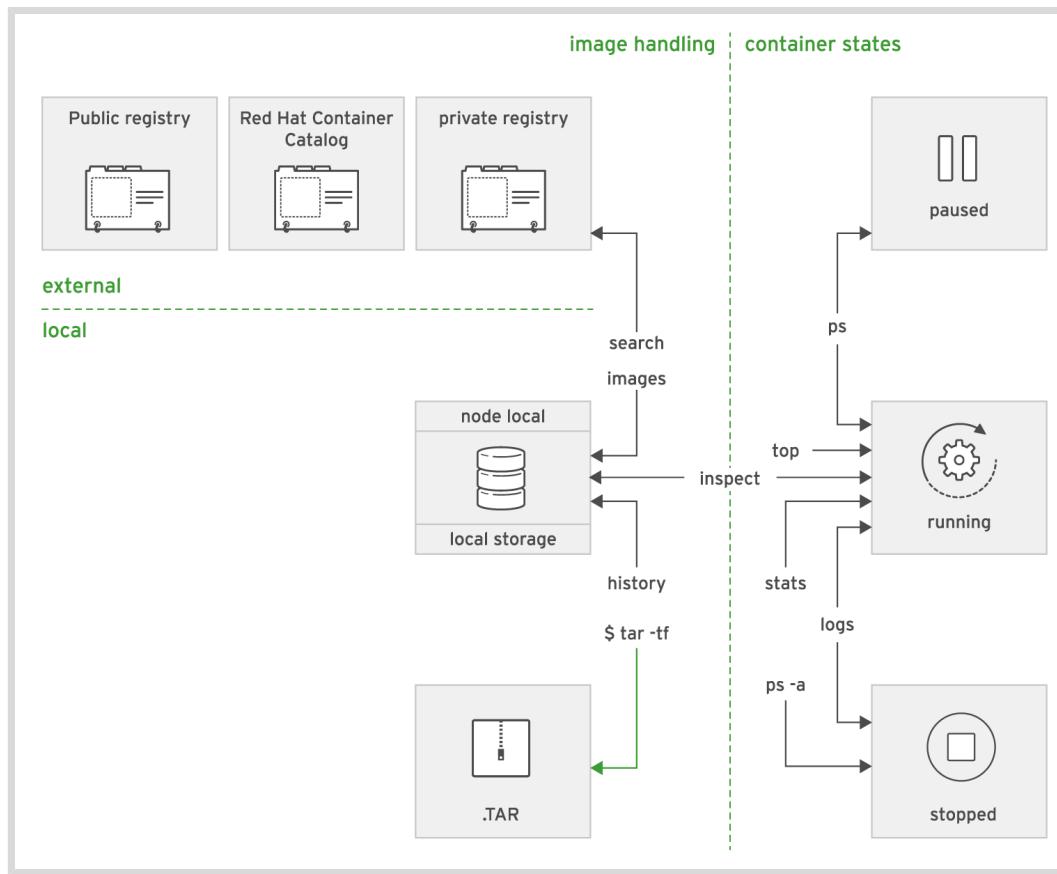


Figure 3.2: Podman query subcommands

Use these two figures as a reference while you learn about Podman subcommands in this course.

Creating Containers

The **podman run** command creates a new container from an image and starts a process inside the new container. If the container image is not available locally, this command attempts to download the image using the configured image repository:

```
[student@workstation ~]$ sudo podman run rhscl/httpd-24-rhel7
Trying to pull registry...httpd-24-rhel7:latest...Getting image source signatures
Copying blob sha256:23113...b0be82
72.21 MB / 72.21 MB [=====] 7s
...output omitted...AH00094: Command line: 'httpd -D FOREGROUND'
^C
```

In the previous output sample, the container was started with a non interactive process (without the **-it** option) and is running in the foreground because it was not started with the **-d** option. Stopping the resulting process with **Ctrl+C (SIGINT)** therefore stops both the container process as well as the container itself.

Podman identifies containers by a unique container ID or container name. The **podman ps** command displays the container ID and names for all actively running containers:

```
[student@workstation ~]$ sudo podman ps
CONTAINER ID      IMAGE                  COMMAND               ... NAMES
47c9aad6049①    rhscl/httpd-24-rhel7  "httpd -D FOREGROUND" ... focused_fermat②
```

- ① The container ID is unique and generated automatically.
- ② The container name can be manually specified, otherwise it is generated automatically. This name must be unique or the **run** command fails.

The **podman run** command automatically generates a unique, random ID. It also generates a random container name. To define the container name explicitly, use the **--name** option when running a container:

```
[student@workstation ~]$ sudo podman run --name my-httpd-container
rhscl/httpd-24-rhel7
...output omitted...AH00094: Command line: 'httpd -D FOREGROUND'
```



Note

The name must be unique. Podman throws an error if the name is already in use, including stopped containers.

Another important feature is the ability to run the container as a daemon process in the background. The **-d** option is responsible for running in detached mode. When using this option, Podman returns the container ID on the screen, allowing you to continue to run commands in the same terminal while the container runs in the background:

```
[student@workstation ~]$ sudo podman run --name my-httpd-container -d
rhscl/httpd-24-rhel7
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

The container image specifies the command to run to start the containerized process, known as the entry point. The **podman run** command can override this entry point by including the command after the container image:

```
[student@workstation ~]$ sudo podman run rhscl/httpd-24-rhel7 ls /tmp
anaconda-post.log
ks-script-1j4CXN
yum.log
```

The specified command must be executable inside the container image.



Note

Because a specified command appears in the previous example, the container skips the entry point for the **httpd** image. Hence, the **httpd** service does not start.

Some containers need to run as an interactive shell or process. This includes containers running processes that need user input (such as entering commands), and processes that generate output through standard output. The following example starts an interactive **bash** shell in a **rhscl/httpd-24-rhel7** container:

```
[student@workstation ~]$ sudo podman run -it rhscl/httpd-24-rhel7 /bin/bash  
bash-4.2#
```

The **-t** and **-i** options enable terminal redirection for interactive text-based programs. The **-t** option allocates a **pseudo-tty** (a terminal) and attaches it to the standard input of the container. The **-i** option keeps the container's standard input open, even if it was detached, so the main process can continue waiting for input.

Running Commands in a Container

When a container starts, it executes the entry point command. However, it may be necessary to execute other commands to manage the running container. Some typical use case are shown below:

- Executing an interactive shell in an already running container.
- Running processes that update or display the container's files.
- Starting new background processes inside the container.

The **podman exec** command starts an additional process inside an already running container:

```
[student@workstation ~]$ sudo podman exec 7ed6e671a600 cat /etc/hostname  
7ed6e671a600
```

The previous example uses the container ID to execute the command.

Podman remembers the last container used in any command. Developers can skip writing this container's ID or name in later Podman commands by replacing the container id by the **-l** option:

```
[student@workstation ~]$ sudo podman exec my-httpd-container cat /etc/hostname  
7ed6e671a600  
[student@workstation ~]$ sudo podman exec -l cat /etc/hostname  
7ed6e671a600
```

Managing Containers

Creating and starting a container is just the first step of the container's life cycle. A container's life cycle also includes stopping, restarting, or finally removing it. Users can also examine the container status and metadata for debugging, updating, or reporting purposes.

Podman provides the following commands for managing containers:

- **podman ps**: This command lists running containers:

```
[student@workstation ~]$ sudo podman ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
77d4b7b8ed1f① rhscl/httpd-24-rhel7② "httpd... "③ ...ago④ Up...⑤ 80/tcp⑥ my-  
htt...⑦
```

- ➊ Each container, when created, gets a **container ID**, which is a hexadecimal number. This ID looks like an image ID but is unrelated.
- ➋ Container image that was used to start the container.
- ➌ Command executed when the container started.

- ④ Date and time the container was started.
- ⑤ Total container uptime, if still running, or time since terminated.
- ⑥ Ports that were exposed by the container or any port forwarding that might be configured.
- ⑦ The container name.

Podman does not discard stopped containers immediately. Podman preserves their local file systems and other states for facilitating *postmortem* analysis. Option **-a** lists all containers, including stopped ones:

```
[student@workstation ~]$ sudo podman ps -a
CONTAINER ID  IMAGE          COMMAND       CREATED      STATUS        PORTS     NAMES
4829d82ffbf  rhscl/httpd-24-rhel7 "httpd..." ...ago      Exited (0)...   my-
httpd...
```



Note

While creating containers, Podman aborts if the container name is already in use, even if the container is in a “stopped” status. This option can help to avoid duplicated container names.

- **podman inspect**: This command lists metadata about a running or stopped container. The command produces **JSON** output:

```
[student@workstation ~]$ sudo podman inspect my-httpd-container
[
{
  "Id": "980e45...76c8be",
  ...output omitted...
  "NetworkSettings": {
    "Bridge": "",
    "EndpointID": "483fc9...5d801a",
    "Gateway": "172.17.42.1",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "HairpinMode": false,
    "IPAddress": "172.17.0.9",
  ...output omitted...
}
```

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

```
[student@workstation ~]$ sudo podman inspect \
> -f '{{ .NetworkSettings.IPAddress }}' my-httpd-container
172.17.0.9
```

- **podman stop**: This command stops a running container gracefully:

```
[student@workstation ~]$ sudo podman stop my-httpd-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

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

- **podman kill:** This command sends Unix signals to the main process in the container. If no signal is specified, it sends the **SIGKILL** signal, terminating the main process and the container.

```
[student@workstation ~]$ sudo podman kill my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

You can specify the signal with the **-s** option:

```
[student@workstation ~]$ sudo podman kill -s SIGKILL my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

Any Unix signal can be sent to the main process. Podman accepts either the signal name and number. The following table shows several useful signals:

Signal	Value	Default Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process

Signal	Value	Default Action	Comment
SIGTTOU	22,22,27	Stop	tty output for background process

 **Note**
Term

Terminate the process.

Core

Terminate the process and generate a core dump.

Ign

Signal is ignored.

Stop

Stop the process.

- **podman restart**: This command restarts a stopped container:

```
[student@workstation ~]$ sudo podman restart my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

The **podman restart** command creates a new container with the same container ID, reusing the stopped container state and file system.

- **podman rm**: This command deletes a container and discards its state and file system:

```
[student@workstation ~]$ sudo podman rm my-httdp-container
77d4b7b8ed1fd57449163bcb0b78d205e70d2314273263ab941c0c371ad56412
```

The **-f** option of the **rm** subcommand instructs Podman to remove the container even if not stopped. This option terminates the container forcefully and then removes it. Using **-f** option is equivalent to **podman kill** and **podman rm** commands together.

You can delete all containers at the same time. Many **podman** subcommands accept the **-a** option. This option indicates using the subcommand on all available containers or images. The following example removes all containers:

```
[student@workstation ~]$ sudo podman rm -a
5fd8e98ec7eab567eabe84943fe82e99fdfc91d12c65d99ec760d5a55b8470d6
716fd687f65b0957edac73b84b3253760e915166d3bc620c4aec8e5f4eadfe8e
86162c906b44f4cb63ba2e3386554030dc6abedbce9e9fcad60aa9f8b2d5d4
```

Before deleting all containers, all running containers must be in a “stopped” status. You can use the following command to stop all containers:

```
[student@workstation ~]$ sudo podman stop -a
5fd8e98ec7eab567eabe84943fe82e99fdfc91d12c65d99ec760d5a55b8470d6
716fd687f65b0957edac73b84b3253760e915166d3bc620c4aec8e5f4eadfe8e
86162c906b44f4cb63ba2e3386554030dc6abedbce9e9fcad60aa9f8b2d5d4
```



Note

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



References

Unix Posix Signals man page

<http://man7.org/linux/man-pages/man7/signal.7.html>

► Guided Exercise

Managing a MySQL Container

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

Outcomes

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

Before You Begin

Make sure that **workstation** has the **podman** command available and is correctly set up by running the following command from a terminal window:

```
[student@workstation ~]$ lab manage-lifecycle start
```

- 1. Download the MySQL database container image, and attempt to start it. The container does not start because several environment variables must be provided to the image.

```
[student@workstation ~]$ sudo podman run --name mysql-db rhscl/mysql-57-rhel7
Trying to pull ...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
You must either specify the following environment variables:
  MYSQL_USER (regex: '^[a-zA-Z0-9_-]+$')
  MYSQL_PASSWORD (regex: '^[a-zA-Z0-9_-!@#$%^&*()-=<>, .?;:|]+$')
  MYSQL_DATABASE (regex: '^[a-zA-Z0-9_-]+$')
Or the following environment variable:
  MYSQL_ROOT_PASSWORD (regex: '^[a-zA-Z0-9_-!@#$%^&*()-=<>, .?;:|]+$')
Or both.
Optional Settings:
...output omitted...

For more information, see https://github.com/sclorg/mysql-container
```



Note

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

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

- 2. Create a new container named **mysql**, and specify each required variable using the **-e** parameter.

**Note**

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

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

The command displays the container ID for the **mysql** container. Below is an example of the output.

```
a49dba9ff17f2b5876001725b581fdd331c9ab8b9eda21cc2a2899c23f078509
```

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

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}} {{.Status}}"
a49dba9ff17f    mysql    Up About a minute ago
```

The command only shows the first 12 characters of the container ID displayed in the previous command.

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

```
[student@workstation ~]$ sudo podman inspect \
> -f '{{ .NetworkSettings.IPAddress }}' mysql
10.88.0.6
```

The IP address of your container may differ from the one shown above (**10.88.0.6**).

**Note**

The **podman inspect** command provides additional important information. For example, the **Env** section displays environment variables, such as the **MYSQL_ROOT_PASSWORD** variable.

- 5. Populate the **items** database with the **Projects** table:

```
[student@workstation ~]$ mysql -uuser1 -h 10.88.0.6 \
> -pmypa55 items < /home/student/D0180/labs/manage-lifecycle/db.sql
```

- 6. Create another container using the same container image as the previous container. Interactively enter the **/bin/bash** shell instead of using the default command for the container image.

```
[student@workstation ~]$ sudo podman run --name mysql-2 \
> -it rhsc1/mysql-57-rhel7 /bin/bash
bash-4.2$
```

- 7. Try to connect to the MySQL database in the new container:

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

The following error displays:

```
ERROR 2002 (HY000): Can't connect to local MySQL ...output omitted...
```

The MySQL database server is not running, because the container executed the **/bin/bash** command rather than starting the MySQL server.

- 8. Exit the container:

```
bash-4.2$ exit
```

- 9. Verify that the **mysql-2** container is not running:

```
[student@workstation ~]$ sudo podman ps -a \
> --format="{{.ID}} {{.Names}} {{.Status}}"
2871e392af02 mysql-2 Exited (1) 19 seconds ago
a49dba9ff17f mysql Up 10 minutes ago
c053c7e09c21 mysql-db Exited (1) 44 minutes ago
```

- 10. Query the **mysql** container to list all rows in the **Projects** table. The command instructs the **bash** shell to query the **items** database using a **mysql** command.

```
[student@workstation ~]$ sudo podman exec mysql /bin/bash \
> -c 'mysql -uuser1 -pmypa55 -e "select * from items.Projects;"'
mysql: [Warning] Using a password on the command line interface can be insecure.
id      name      code
1       DevOps    D0180
```

Finish

On **workstation**, run the **lab manage-lifecycle finish** script to complete this exercise.

```
[student@workstation ~]$ lab manage-lifecycle finish
```

This concludes the exercise.

Attaching Persistent Storage to Containers

Objectives

After completing this section, students should be able to:

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

Preparing Permanent Storage Locations

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

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

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

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

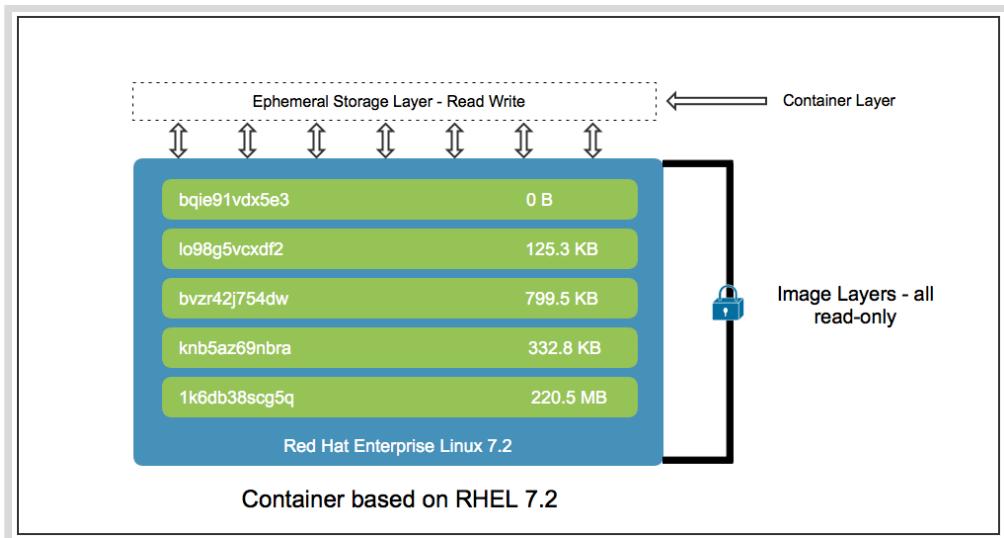


Figure 3.3: Container layers

Containerized applications should not try to use the container storage to store persistent data, because they cannot control how long its contents will be preserved. Even if it were possible to keep container storage indefinitely, the layered file system does not perform well for intensive I/O workloads and would not be adequate for most applications requiring persistent storage.

Reclaiming Storage

Podman keeps old stopped container storage available to be used by troubleshooting operations, such as reviewing failed container logs for error messages.

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

Preparing the Host Directory

Podman can mount host directories inside a running container. The containerized application sees these host directories as part of the container storage, much like regular applications see a remote network volume as if it were part of the host file system. But these host directories' contents are not reclaimed after the container is stopped, and they can be mounted to new containers whenever needed.

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

A container runs as a host operating system process, under a host operating system user and group ID, so the host directory needs to be configured with ownership and permissions allowing access to the container. In RHEL, the host directory also needs to be configured with the appropriate SELinux context, which is `container_file_t`. Podman uses the `container_file_t` SELinux context to restrict which files of the host system the container is allowed to access. This avoids information leakage between the host system and the applications running inside containers.

One way to set up the host directory is described below:

1. Create a directory with owner and group **root**:

```
[student@workstation ~]$ sudo mkdir /var/dbfiles
```

2. The user running processes in the container must be capable of writing files to the directory. If the host machine does not have exactly the same user defined, the permission should be defined with the numeric user ID (UID) from the container. In the case of the Red Hat-provided MySQL service, the UID is 27:

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

3. Apply the **container_file_t** context to the directory (and all subdirectories) to allow containers access to all of its contents.

```
[student@workstation ~]$ sudo semanage fcontext -a -t container_file_t '/var/dbfiles(/.*)?'
```

4. Apply the SELinux container policy that you set up in the first step to the newly created directory:

```
[student@workstation ~]$ sudo restorecon -Rv /var/dbfiles
```

The host directory must be configured *before* starting the container that uses the directory.

Mounting a Volume

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

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

```
[student@workstation ~]$ sudo podman run -v /var/dbfiles:/var/lib/mysql  
rhmap47/mysql
```

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

► Guided Exercise

Persisting a MySQL Database

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

Outcomes

You should be able to deploy container with a persistent database.

Before You Begin

The **workstation** should not have any container images running. Run the following command on **workstation**:

```
[student@workstation ~]$ lab manage-storage start
```

- 1. Create the **/var/local/mysql** directory with the correct SELinux context and permissions.

- 1.1. Create the **/var/local/mysql** directory.

```
[student@workstation ~]$ sudo mkdir -pv /var/local/mysql  
mkdir: created directory '/var/local/mysql'
```

- 1.2. Add the appropriate SELinux context for the **/var/local/mysql** directory and its contents.

```
[student@workstation ~]$ sudo semanage fcontext -a \  
> -t container_file_t '/var/local/mysql(/.*)?'
```

- 1.3. Apply the SELinux policy to the newly created directory.

```
[student@workstation ~]$ sudo restorecon -R /var/local/mysql
```

- 1.4. Verify that the SELinux context type for the **/var/local/mysql** directory is **container_file_t**.

```
[student@workstation ~]$ ls -ldZ /var/local/mysql  
drwxr-xr-x. root root unconfined_u:object_r:container_file_t:s0 /var/local/mysql
```

- 1.5. Change the owner of the **/var/local/mysql** directory to the **mysql** user and **mysql** group:

```
[student@workstation ~]$ sudo chown -Rv 27:27 /var/local/mysql  
changed ownership of '/var/local/mysql' from root:root to 27:27
```

**Note**

The user running processes in the container must be capable of writing files to the directory. If the host machine does not have exactly the same user defined, the permission should be defined with the numeric user ID (UID) from the container. For the MySQL service provided by Red Hat, the UID is 27.

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

- 2.1. Pull the MySQL container image:

```
[student@workstation ~]$ sudo podman pull rhscl/mysql-57-rhel7
Trying to pull ...output omitted...rhscl/mysql-57-rhel7...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
4ae3a3f4f409a8912cab9fbf71d3564d011ed2e68f926d50f88f2a3a72c809c5
```

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

```
[student@workstation ~]$ sudo podman run --name persist-db \
> -d -v /var/local/mysql:/var/lib/mysql/data \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhscl/mysql-57-rhel7
6e0ef134315b510042ca757faf869f2ba19df27790c601f95ec2fd9d3c44b95d
```

This command mounts the **/var/local/mysql** directory from the host to the **/var/lib/mysql/data** directory in the container. By default, the MySQL database stores data in the **/var/lib/mysql/data** directory.

- 2.3. Verify that the container started correctly.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}} {{.Status}}"
6e0ef134315b  persist-db  Up 3 minutes ago
```

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

```
[student@workstation ~]$ ls -ld /var/local/mysql/items
drwxr-x---. 2 27 27 20 Nov 13 12:55 /var/local/mysql/items
```

The **items** directory stores data related to the **items** database that was created by this container. If the **items** directory does not exist, then the mount point was not defined correctly during container creation.

Finish

On **workstation**, run the **lab manage-storage finish** script to complete this lab.

```
[student@workstation ~]$ lab manage-storage finish
```

This concludes the exercise.

Accessing Containers

Objectives

After completing this section, students should be able to:

- Describe the basics of networking with containers.
- Remotely connect to services within a container.

Introducing Networking with Containers

The Cloud Native Computing Foundation (CNCF) sponsors the *Container Networking Interface (CNI)* open source project. The CNI project aims to standardize the network interface for containers in cloud native environments, such as Kubernetes and Red Hat OpenShift Container Platform.

Podman uses the CNI project to implement a *software-defined network (SDN)* for containers on each host. Podman attaches each container to a virtual bridge and assigns each container a private IP address. The configuration file that specifies CNI settings for Podman is **/etc/cni/net.d/87-podman-bridge.conf**.

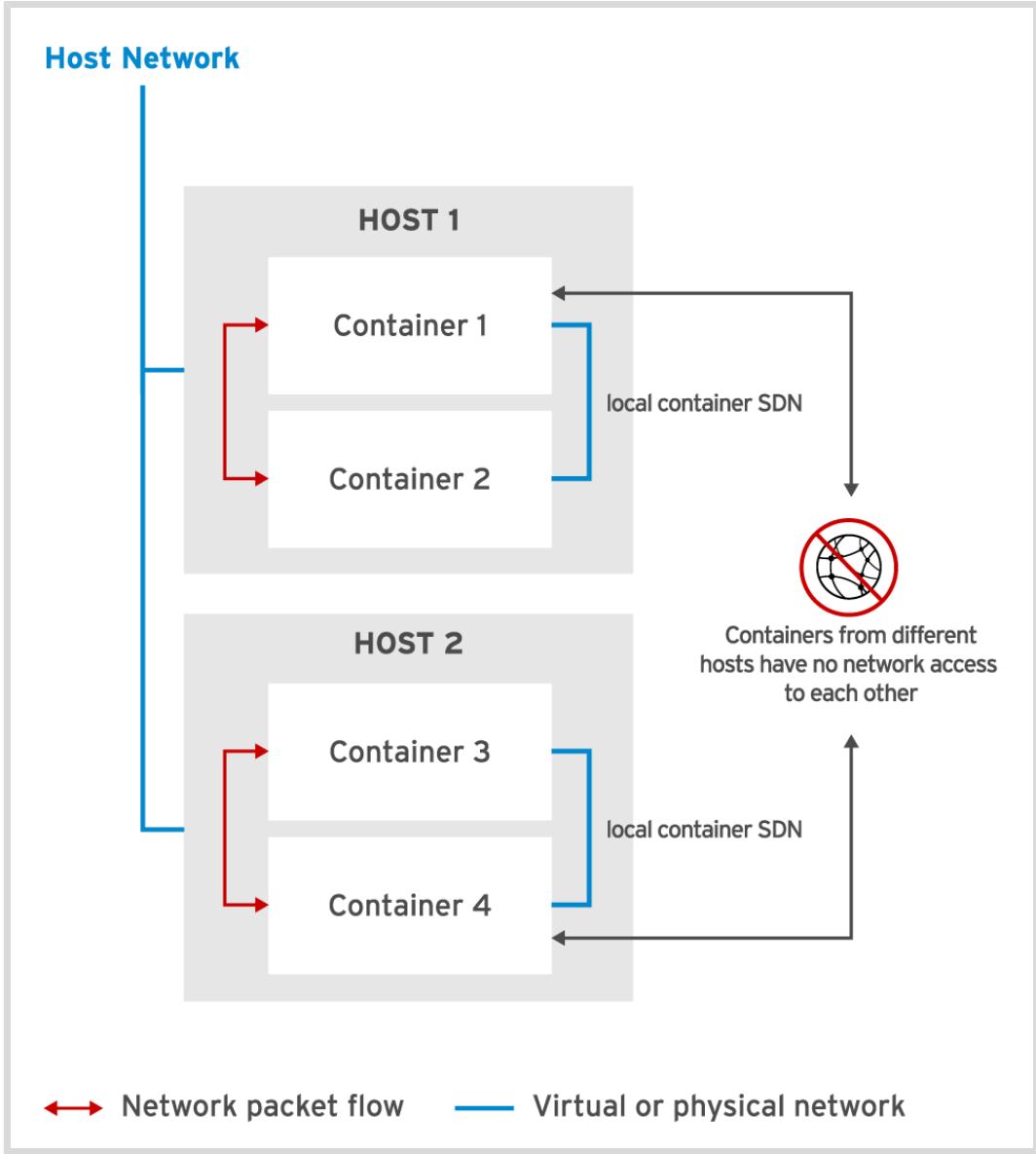


Figure 3.4: Basic Linux container networking

When Podman creates containers on the same host, it assigns each container a unique IP address and connects them all to the same software-defined network. These containers can communicate freely with each other by IP address.

Containers created with Podman running on different hosts belong to different software-defined networks. Each SDN is isolated, which prevents a container in one network from communicating with a container in a different network. Because of network isolation, a container in one SDN can have the same IP address as a container in a different SDN.

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

Mapping Network Ports

Accessing a container from the host network can be a challenge. A container is assigned an IP address from a pool of available addresses. When a container is destroyed, the container's address is released back to the pool of available addresses. Another problem is that the container software-defined network is only accessible from the container host.

To solve these problems, define port forwarding rules to allow external access to a container service. Use the `-p [<IP address>:]<host port>:<container port>` option with the `podman run` command to create an externally accessible container. Consider the following example:

```
[student@workstation ~]$ sudo podman run -d --name apache1 -p 8080:80
rh scl/httpd-24-rhel7:2.4
```

The value **8080:80** specifies that any requests to port 8080 on the host are forwarded to port 80 within the container.

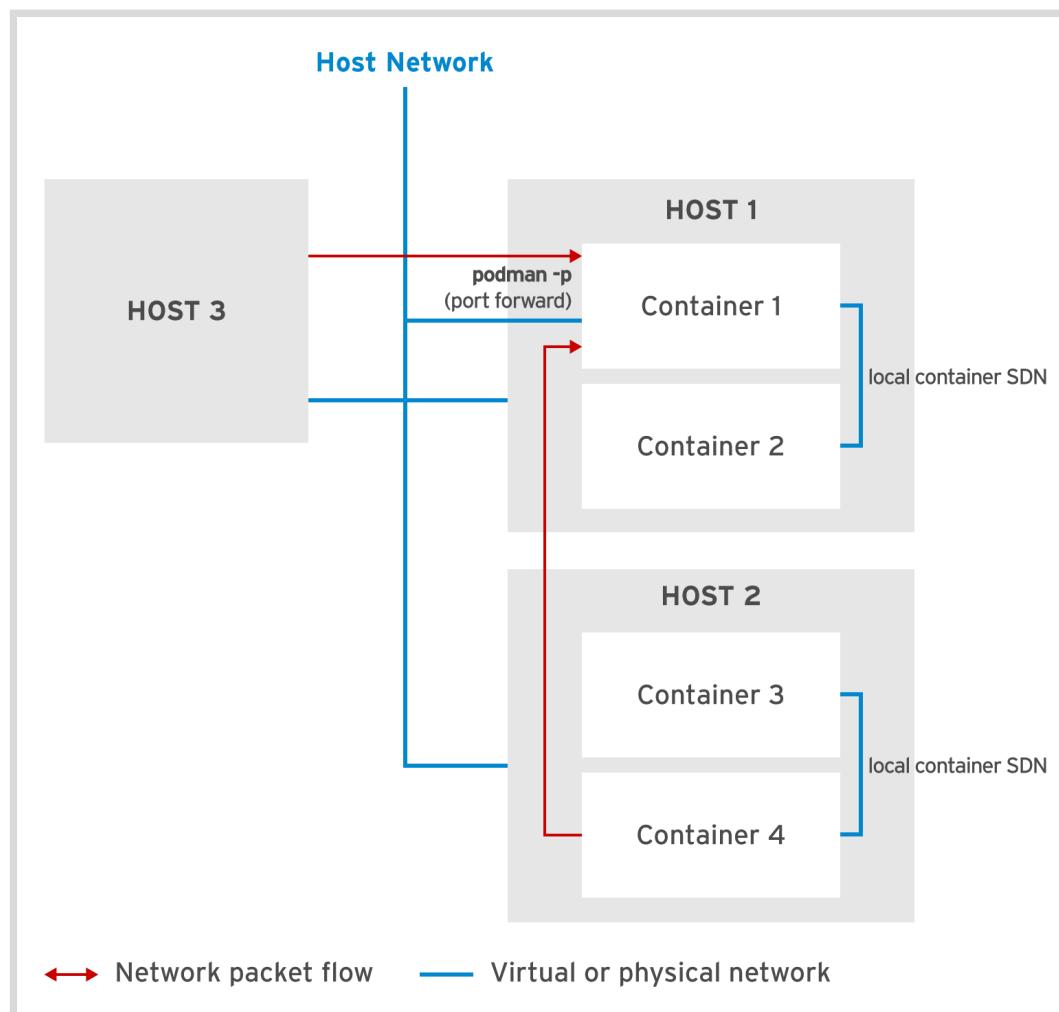


Figure 3.5: Allowing external accesses to Linux containers

You can also use the `-p` option to only forward requests to a container if those requests originate from a specified IP address:

```
[student@workstation ~]$ sudo podman run -d --name apache2 \
> -p 127.0.0.1:8081:80 rhsc1/httpd-24-rhel7:2.4
```

The example above limits external access to the **apache2** container to requests from **localhost** to host port 8081. These requests are forwarded to port 80 in the **apache2** container.

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

```
[student@workstation ~]$ sudo podman run -d --name apache3 -p 127.0.0.1::80
rhsc1/httpd-24-rhel7:2.4
```

To see the port assigned by Podman, use the **podman port <container name>** command:

```
[student@workstation ~]$ sudo podman port apache3
80/tcp -> 127.0.0.1:35134
[student@workstation ~]$ curl 127.0.0.1:35134
<html><body><h1>It works!</h1></body></html>
```

If only a container port is specified with the **-p** option, a random available host port is assigned to container. Requests to this assigned host port from any IP address are forwarded to the container port.

```
[student@workstation ~]$ sudo podman run -d --name apache4 -p 80
rhsc1/httpd-24-rhel7:2.4
[student@workstation ~]$ sudo podman port apache4
80/tcp -> 0.0.0.0:37068
```

In the above example, any routable request to host port **37068** is forwarded to the port 80 in the container.



References

Container Network Interface - networking for Linux containers

<https://github.com/containernetworking/cni>

Cloud Native Computing Foundation

<https://www.cncf.io/>

► Guided Exercise

Loading the Database

In this exercise, you will create a MySQL database container with port forwarding enabled. After populating a database with a SQL script, you verify the database content using three different methods.

Outcomes

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

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab manage-networking start
```

This ensures the **/var/local/mysql** directory exists and is configured with the correct permissions to enable persistent storage for the MySQL container.

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

```
[student@workstation ~]$ sudo podman run --name mysqldb-port \
> -d -v /var/local/mysql:/var/lib/mysql/data -p 13306:3306 \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhscl/mysql-57-rhel7
Trying to pull ...output omitted...
Copying blob sha256:1c9f515...output omitted...
  72.70 MB / ? [=====] 5s
Copying blob sha256:1d2c4ce...output omitted...
  1.54 KB / ? [=====] 0s
Copying blob sha256:f1e961f...output omitted...
  6.85 MB / ? [=====] 0s
Copying blob sha256:9f1840c...output omitted...
  62.31 MB / ? [=====] 7s
Copying config sha256:60726...output omitted...
```

```
6.85 KB / 6.85 KB [=====] 0s
Writing manifest to image destination
Storing signatures
066630d45cb902ab533d503c83b834aa6a9f9cf88755cb68eedb8a3e8edbc5aa
```

The last line of your output will differ from that shown above, as well as the time needed to download each image layer.

The **-p** option configures port forwarding so that port 13306 on the local host forwards to container port 3306.



Note

The start script creates the **/var/local/mysql** directory with the appropriate ownership and SELinux context required by the containerized database.

- ▶ 2. Verify that the **mysqldb-port** container started successfully and enables port forwarding.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}} {{.Ports}}"
9941da2936a5  mysql ldb-port  0.0.0.0:13306->3306/tcp
```

- ▶ 3. Populate the database using the provided file. If there are no errors, then the command does not return any output.

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
> -P13306 items < /home/student/D0180/labs/manage-networking/db.sql
```

There are multiple ways to verify that the database loaded successfully. The next steps show three different methods. You only need to complete one of the methods.

- ▶ 4. Verify that the database loaded successfully by executing a non-interactive command inside the container.

```
[student@workstation ~]$ sudo podman exec -it mysqldb-port \
> /opt/rh/rh-mysql57/root/usr/bin/mysql -uroot items -e "SELECT * FROM Item"
+-----+
| id | description      | done |
+-----+
| 1  | Pick up newspaper |  0  |
| 2  | Buy groceries     |  1  |
+-----+
```



Note

Because the **mysql** command is not found in a directory defined in the container **PATH** variable, you must use an absolute path.

- ▶ 5. Verify that the database loaded successfully by using port forwarding from the local host. This alternate method is optional.

```
[student@workstation ~]$ mysql -uuser1 -h 127.0.0.1 -pmypa55 \
> -P13306 items -e "SELECT * FROM Item"
+---+-----+---+
| id | description | done |
+---+-----+---+
| 1 | Pick up newspaper | 0 |
| 2 | Buy groceries | 1 |
+---+-----+---+
```

- ▶ 6. Verify that the database loaded successfully by opening an interactive terminal session inside the container. This alternate method is optional.

- 6.1. Open a Bash shell inside the container.

```
[student@workstation ~]$ sudo podman exec -it mysql-pod /bin/bash
bash-4.2$
```

- 6.2. Verify that the database contains data:

```
bash-4.2$ mysql -uroot items -e "SELECT * FROM Item"
+---+-----+---+
| id | description | done |
+---+-----+---+
| 1 | Pick up newspaper | 0 |
| 2 | Buy groceries | 1 |
+---+-----+---+
```

- 6.3. Exit from the container:

```
bash-4.2$ exit
```

Finish

On **workstation**, run the **lab manage-networking finish** script to complete this lab.

```
[student@workstation ~]$ lab manage-networking finish
```

This concludes the exercise.

► Lab

Managing Containers

Performance Checklist

In this lab, you will deploy a container that saves the MySQL database data into a folder on the host and then you will load the database in another container.

Outcomes

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

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab manage-review start
```

1. Create the **/var/local/mysql** directory with the correct SELinux context and permissions.
 - 1.1. Create the **/var/local/mysql** directory.
 - 1.2. Add the appropriate SELinux context for the **/var/local/mysql** directory and its contents. With the correct context, you can mount this directory in a running container.
 - 1.3. Apply the SELinux policy to the newly created directory.
 - 1.4. Change the owner of the **/var/local/mysql** directory to match the **mysql** user and **mysql** group for the **rhscl/mysql-57-rhel7** container image:
2. Deploy a MySQL container instance using the following characteristics:
 - **Name:** **mysql-1**
 - **Run as daemon:** yes
 - **Volume:** from **/var/local/mysql** host folder to **/var/lib/mysql/data** container folder
 - **Container image:** **rhscl/mysql-57-rhel7**
 - **Port forward:** no
 - **Environment variables:**
 - **MYSQL_USER:** **user1**
 - **MYSQL_PASSWORD:** **mypa55**
 - **MYSQL_DATABASE:** **items**

- **MYSQL_ROOT_PASSWORD: r00tpa55**
3. Load the **items** database using the **/home/student/D0180/labs/manage-review/db.sql** script.
 - 3.1. Get the container IP address.
 - 3.2. Load the database using the SQL commands in **/home/student/D0180/labs/manage-review/db.sql**. Use the IP address you find in the previous step as the database server's host IP.

**Note**

You can import all the commands in the above file using the less-than operator (<) after the **mysql** command, instead of typing them. Also, you need to add the **-h CONTAINER_IP** parameter to the **mysql** command to connect to the correct container.

- 3.3. Use an SQL **SELECT** statement to output all rows of the Item table to verify that the Items database is loaded.

**Note**

You can add the **-e SQL** parameter to the **mysql** command to execute an SQL instruction.

4. Stop the container gracefully.

**Important**

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

5. Create a new container with the following characteristics:
 - **Name:** **mysql-2**
 - **Run as a daemon:** yes
 - **Volume:** from **/var/local/mysql** host folder to **/var/lib/mysql/data** container folder
 - **Container image:** **rhscl/mysql-57-rhel7**
 - **Port forward:** yes, from host port 13306 to container port 3306
 - **Environment variables:**
 - **MYSQL_USER:** **user1**
 - **MYSQL_PASSWORD:** **mypa55**
 - **MYSQL_DATABASE:** **items**
 - **MYSQL_ROOT_PASSWORD:** **r00tpa55**

6. Save the list of all containers (including stopped ones) to the `/tmp/my-containers` file.
7. Access the Bash shell inside the container and verify that the `items` database and the `Item` table are still available. Confirm also that the table contains data.
 - 7.1. Access the Bash shell inside the container.
 - 7.2. Connect to the MySQL server.
 - 7.3. List all databases and confirm that the `items` database is available.
 - 7.4. List all tables from the `items` database and verify that the `Item` table is available.
 - 7.5. View the data from the table.
 - 7.6. Exit from the MySQL client and from the container shell.
8. Using port forwarding, insert a new row into the `Item` table. The row should have a `description` value of `Finished lab`, and a `done` value of `1`.
 - 8.1. Connect to the MySQL database.
 - 8.2. Insert the new row.
 - 8.3. Exit from the MySQL client.
9. Because the first container is not required any more, remove it to release resources.

Evaluation

Grade your work by running the `lab manage-review grade` command from your `workstation` machine. Correct any reported failures and rerun the script until successful.

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

Finish

On `workstation`, run the `lab manage-review finish` command to complete this lab.

```
[student@workstation ~]$ lab manage-review finish
```

This concludes the lab.

► Solution

Managing Containers

Performance Checklist

In this lab, you will deploy a container that saves the MySQL database data into a folder on the host and then you will load the database in another container.

Outcomes

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

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab manage-review start
```

1. Create the **/var/local/mysql** directory with the correct SELinux context and permissions.

- 1.1. Create the **/var/local/mysql** directory.

```
[student@workstation ~]$ sudo mkdir -pv /var/local/mysql
mkdir: created directory '/var/local/mysql'
```

- 1.2. Add the appropriate SELinux context for the **/var/local/mysql** directory and its contents. With the correct context, you can mount this directory in a running container.

```
[student@workstation ~]$ sudo semanage fcontext -a \
> -t container_file_t '/var/local/mysql(/.*)?'
```

- 1.3. Apply the SELinux policy to the newly created directory.

```
[student@workstation ~]$ sudo restorecon -R /var/local/mysql
```

- 1.4. Change the owner of the **/var/local/mysql** directory to match the **mysql** user and **mysql** group for the **rhscl/mysql-57-rhel7** container image:

```
[student@workstation ~]$ sudo chown -Rv 27:27 /var/local/mysql
changed ownership of '/var/local/mysql' from root:root to 27:27
```

2. Deploy a MySQL container instance using the following characteristics:

- **Name: mysql-1**

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

2.1. Create and start the container.

```
[student@workstation ~]$ sudo podman run --name mysql-1 \
> -d -v /var/local/mysql:/var/lib/mysql/data \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhscl/mysql-57-rhel7
Trying to pull ...output omitted...
...output omitted...
Writing manifest to image destination
Storing signatures
6l6azfaa55x866e7eb18b1fd0423a5461d8f24c147b1ad8668b76e6167587cdd
```

2.2. Verify that the container was started correctly.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}}"
6l6azfaa55x8    mysql-1
```

3. Load the `items` database using the `/home/student/D0180/labs/manage-review/db.sql` script.

3.1. Get the container IP address.

```
[student@workstation ~]$ sudo podman inspect \
> -f '{{ .NetworkSettings.IPAddress }}' mysql-1
10.88.0.6
```

3.2. Load the database using the SQL commands in `/home/student/D0180/labs/manage-review/db.sql`. Use the IP address you find in the previous step as the database server's host IP.

**Note**

You can import all the commands in the above file using the less-than operator (<) after the **mysql** command, instead of typing them. Also, you need to add the **-h CONTAINER_IP** parameter to the **mysql** command to connect to the correct container.

```
[student@workstation ~]$ mysql -uuser1 -h CONTAINER_IP \
> -pmypa55 items < /home/student/D0180/labs/manage-review/db.sql
```

- 3.3. Use an SQL **SELECT** statement to output all rows of the Item table to verify that the Items database is loaded.

**Note**

You can add the **-e SQL** parameter to the **mysql** command to execute an SQL instruction.

```
[student@workstation ~]$ mysql -uuser1 -h CONTAINER_IP -pmypa55 items \
> -e "SELECT * FROM Item"
+---+-----+-----+
| id | description | done |
+---+-----+-----+
| 1 | Pick up newspaper | 0 |
| 2 | Buy groceries | 1 |
+---+-----+-----+
```

4. Stop the container gracefully.

**Important**

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

Use the following command to stop the container:

```
[student@workstation ~]$ sudo podman stop mysql-1
616azfaa55x866e7eb18b1fd0423a5461d8f24c147b1ad8668b76e6167587cdd
```

5. Create a new container with the following characteristics:

- **Name:** **mysql-2**
- **Run as a daemon:** yes
- **Volume:** from **/var/local/mysql** host folder to **/var/lib/mysql/data** container folder
- **Container image:** **rhscl/mysql-57-rhel7**

- **Port forward:** yes, from host port 13306 to container port 3306
- **Environment variables:**
 - **MYSQL_USER:** user1
 - **MYSQL_PASSWORD:** mypa55
 - **MYSQL_DATABASE:** items
 - **MYSQL_ROOT_PASSWORD:** r00tpa55

5.1. Create and start the container.

```
[student@workstation ~]$ sudo podman run --name mysql-2 \
> -d -v /var/local/mysql:/var/lib/mysql/data \
> -p 13306:3306 \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \
> rhel7/mysql-5.7-rhel7
281c0e2790e54cd5a0b8e2a8cb6e3969981b85cde8ac611bf7ea98ff78bdffbb
```

5.2. Verify that the container was started correctly.

```
[student@workstation ~]$ sudo podman ps --format="{{.ID}} {{.Names}}"
281c0e2790e5    mysql-2
```

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

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

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

7.1. Access the Bash shell inside the container.

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

7.2. Connect to the MySQL server.

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

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

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| mysql          |
```

```
| performance_schema |
| sys                |
+-----+
5 rows in set (0.03 sec)
```

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

```
mysql> use items;
Database changed
mysql> show tables;
+-----+
| Tables_in_items |
+-----+
| Item           |
+-----+
1 row in set (0.01 sec)
```

- 7.5. View the data from the table.

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

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

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

8. Using port forwarding, insert a new row into the **Item** table. The row should have a **description** value of **Finished lab**, and a **done** value of **1**.

- 8.1. Connect to the MySQL database.

```
[student@workstation ~]$ mysql -uuser1 -h workstation.lab.example.com \
> -pmypa55 -P13306 items
...output omitted...

Welcome to the MariaDB monitor. Commands end with ; or \g.
...output omitted...

MySQL [items]>
```

- 8.2. Insert the new row.

```
MySQL[items]> insert into Item (description, done) values ('Finished lab', 1);
Query OK, 1 row affected (0.00 sec)
```

8.3. Exit from the MySQL client.

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

9. Because the first container is not required any more, remove it to release resources.

Use the following command to remove the container:

```
[student@workstation ~]$ sudo podman rm mysql-1  
6l6azfaa55x866e7eb18b1fd0423a5461d8f24c147b1ad8668b76e6167587cdd
```

Evaluation

Grade your work by running the **lab manage-review grade** command from your **workstation** machine. Correct any reported failures and rerun the script until successful.

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

Finish

On **workstation**, run the **lab manage-review finish** command to complete this lab.

```
[student@workstation ~]$ lab manage-review finish
```

This concludes the lab.

Summary

In this chapter, you learned:

- Podman has subcommands to: create a new container (**run**), delete a container (**rm**), list containers (**ps**), stop a container (**stop**), and start a process in a container (**exec**).
- Default container storage is ephemeral, meaning its contents are not present after the container restarts or is removed.
 - Containers can use a folder from the host file system to work with persistent data.
 - Podman mounts volumes in a container with the **-v** option in the **podman run** command.
- The **podman exec** command starts an additional process inside a running container.
- Podman maps local ports to container ports by using the **-p** option in the **run** subcommand.

Chapter 4

Managing Container Images

Goal

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

Objectives

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

Sections

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

Lab

Managing Images

Accessing Registries

Objectives

After completing this section, students should be able to:

- Search for and pull images from remote registries using Podman commands and the registry REST API.
- List the advantages of using a certified public registry to download secure images.
- Customize the configuration of Podman to access alternative container image registries.
- List images downloaded from a registry to the local file system.
- Manage tags to pull tagged images.

Public Registries

Image registries are services offering container images to download. They allow image creators and maintainers to store and distribute container images to public or private audiences.

Podman searches for and downloads container images from public and private registries. Red Hat Container Catalog is the public image registry managed by Red Hat. It hosts a large set of container images, including those provided by major open source projects, such as Apache, MySQL, and Jenkins. All images in the Container Catalog are vetted by the Red Hat internal security team, meaning they are trustworthy and secured against security flaws.

Red Hat container images provide the following benefits:

- *Trusted source*: All container images comprise sources known and trusted by Red Hat.
- *Original dependencies*: None of the container packages have been tampered with, and only include known libraries.
- *Vulnerability-free*: Container images are free of known vulnerabilities in the platform components or layers.
- *Runtime protection*: All applications in container images run as non-root users, minimizing the exposure surface to malicious or faulty applications.
- *Red Hat Enterprise Linux (RHEL) compatible*: Container images are compatible with all RHEL platforms, from bare metal to cloud.
- *Red Hat support*: Red Hat commercially supports the complete stack.

Quay.io is another public image repository sponsored by Red Hat. Quay.io introduces several exciting features, such as server-side image building, fine-grained access controls, and automatic scanning of images for known vulnerabilities.

While Red Hat Container Catalog images are trusted and verified, Quay.io offers live images regularly updated by creators. Quay.io users can create their namespaces, with fine-grained access control, and publish the images they create to that namespace. Container Catalog users rarely or never push new images, but consume trusted images generated by the Red Hat team.

Private Registries

Some image creators or maintainers want to make their images public available. Other image creators, however, prefer to keep their images private due to:

- Company privacy and secret protection.
- Legal restrictions and laws.
- Avoidance of publishing images in development.

Private registries give image creators the control about their images placement, distribution and usage.

Configuring Registries in Podman

To configure registries for the **podman** command, you need to update the **/etc/containers/registries.conf** file. Edit the **registries** entry in the **[registries.search]** section, adding an entry to the values list.

```
[registries.search]
registries = ["registry.access.redhat.com", "quay.io"]
```



Note

Use an FQDN and port number to identify a registry. A registry that does not include a port number has a default port number of 5000. If the registry uses a different port, it must be specified. Indicate port numbers by appending a colon (:) and the port number after the FQDN.

Secure connections to a registry require a trusted certificate. To support insecure connections, add the registry name to the **registries** entry in **[registries.insecure]** section of **/etc/containers/registries.conf** file:

```
[registries.insecure]
registries = ['localhost:5000']
```

Accessing Registries

Searching for Images in Registries

The **podman search** command finds images by image name, user name, or description from all the registries listed in the **/etc/containers/registries.conf** configuration file. The syntax for the **podman search** command is shown below:

```
[student@workstation ~]$ sudo podman search [OPTIONS] <term>
```

The following table shows some useful options available for the **search** subcommand:

Option	Description
--limit <number>	Limits the number of listed images per registry.

Option	Description
<code>--filter <filter=value></code>	<p>Filter output based on conditions provided. Supported filters are:</p> <ul style="list-style-type: none"> • stars=<number>: Show only images with at least this number of stars. • is-automated=<true false>: Show only images automatically built. • is-official=<true false>: Show only images flagged as official.
<code>--tls-verify <true false></code>	Enables or disables HTTPS certificate validation for all used registries. true

Registry HTTP API

A remote registry exposes web services that provide an application programming interface (API) to the registry. Podman uses these interfaces to access and interact with remote repositories.

Many registries conform to the **Docker Registry HTTP API v2** specification, which exposes a standardized REST interface for registry interaction. You can use this REST interface to directly interact with a registry, instead of using Podman.

Some samples using this API with **curl** commands are shown below:

To list all repositories available in a registry, use the **/v2/_catalog** endpoint. The **n** parameter is used to limit the number of repositories to return.

```
[student@workstation ~]$ curl -Ls https://myserver/v2/_catalog?n=3
{"repositories":["centos/httpd","do180/custom-httpd","hello-openshift"]}
```



If Python is available, use it to format the JSON response:

```
[student@workstation ~]$ curl -Ls https://myserver/v2/_catalog?n=3 \
> | python -m json.tool
{
  "repositories": [
    "centos/httpd",
    "do180/custom-httpd",
    "hello-openshift"
  ]
}
```

The **/v2/<name>/tags/list** endpoint provides the list of tags available for a single image:

```
[student@workstation ~]$ curl -Ls \
> https://quay.io/v2/redhattraining/httpd-parent/tags/list \
> | python -m json.tool
{
```

```

    "name": "redhattraining/httpd-parent",
    "tags": [
        "latest",
        "2.4"
    ]
}

```

**Note**

Quay.io offers a dedicated API to interact with repositories beyond what is specified in Docker Repository API. See <https://docs.quay.io/api/> for details.

Registry Authentication

Some container image registries require access authorization. The **podman login** command allows username and password authentication to a registry:

```
[student@workstation ~]$ sudo podman login -u username \
> -p password registry.access.redhat.com
Login Succeeded!
```

The registry HTTP API requires authentication credentials. First, use the Red Hat Single Sign On (SSO) service to obtain an access token:

```
[student@workstation ~]$ curl -u username:password -Ls \
> "https://sso.redhat.com/auth/realms/rhcc/protocol/redhat-docker-v2/auth?
service=docker-registry"
>{"token":"eyJh...o5G8",
"access_token":"eyJh...mgL4",
"expires_in":...output omitted...}[student@workstation ~]$
```

Then, include this token in a **Bearer** authorization header in subsequent requests:

```
[student@workstation ~]$ curl -H "Authorization: Bearer eyJh...mgL4" \
> -Ls https://registry.redhat.io/v2/rhscl/mysql-57-rhel7/tags/list \
> | python -mjson.tool
{
    "name": "rhscl/mysql-57-rhel7",
    "tags": [
        "5.7-3.9",
        "5.7-3.8",
        "5.7-3.4",
        "5.7-3.7",
    ...output omitted...
```

**Note**

Other registries may require different steps to provide credentials. If a registry adheres to the **Docker Registry HTTP v2 API**, authentication conforms to the RFC7235 scheme.

Pulling Images

To pull container images from a registry, use the **podman pull** command:

```
[student@workstation ~]$ sudo podman pull [OPTIONS] [REGISTRY[:PORT]/]NAME[:TAG]
```

The **podman pull** command uses the image name obtained from the **search** subcommand to pull an image from a registry. The **pull** subcommand allows adding the registry name to the image. This variant supports having the same image in multiple registries.

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

```
[student@workstation ~]$ sudo podman pull quay.io/bitnami/nginx
```



Note

If the image name does not include a registry name, Podman searches for a matching container image using the registries listed in the **/etc/containers/registries.conf** configuration file. Podman search for images in registries in the same order they appear in the configuration file.

Listing Local Copies of Images

Any container image downloaded from a registry is stored locally on the same host where the **podman** command is executed. This behavior avoids repeating image downloads and minimizes the deployment time for a container. Podman also stores any custom container images you build in the same local storage.



Note

By default, Podman stores container images in the **/var/lib/containers/storage/overlay-images** directory.

Podman provides an **images** subcommand to list all the container images stored locally.

```
[student@workstation ~]$ sudo podman images
REPOSITORY           TAG      IMAGE ID      CREATED     SIZE
registry.redhat.io/rhscl/mysql-57-rhel7  latest   c07bf25398f4  13 days ago  444MB
```

Image Tags

An image tag is a mechanism to support multiple releases of the same image. This feature is useful when multiple versions of the same software are provided, such as a production-ready container or the latest updates of the same software developed for community evaluation. Any Podman subcommand that requires a container image name accepts a tag parameter to differentiate between multiple tags. If an image name does not contain a tag, then the tag value defaults to **latest**. For example, to pull an image with the tag **5.7** from **rhscl/mysql-57-rhel7**, use the following command:

```
[student@workstation ~]$ sudo podman pull rhscl/mysql-57-rhel7:5.7
```

To start a new container based on the **rhscl/mysql-57-rhel7:5.7** image, use the following command:

```
[student@workstation ~]$ sudo podman run rhscl/mysql-57-rhel7:5.7
```



References

Red Hat Container Catalog

<https://registry.redhat.io>

Quay.io

<https://quay.io>

Docker Registry HTTP API V2

<https://github.com/docker/distribution/blob/master/docs/spec/api.md>

RFC7235 - HTTP/1.1: Authentication

<https://tools.ietf.org/html/rfc7235>

► Quiz

Working With Registries

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

Podman is available on a RHEL host with the following entry in `/etc/containers/registries.conf` file:

```
[registries.search]
registries = ["registry.redhat.io", "quay.io"]
```

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

Image names/tags per registry

Registry	Image
registry.redhat.io	<ul style="list-style-type: none"> nginx/1.0 mysql/5.6 httpd/2.2
quay.io	<ul style="list-style-type: none"> mysql/5.5 httpd/2.4

No images are locally available.

- 1. Which two commands display mysql images available for download from `registry.redhat.io`? (Choose two.)
 - a. `podman search registry.redhat.io/mysql`
 - b. `podman images`
 - c. `podman pull mysql`
 - d. `podman search mysql`

- 2. Which command is used to list all available image tags for the httpd container image?
 - a. `podman search httpd`
 - b. `podman images httpd`
 - c. `podman pull --all-tags=true httpd`
 - d. There is no podman command available to search for tags.

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

- a. `podman pull httpd:2.2`
- b. `podman pull httpd:latest`
- c. `podman pull quay.io/httpd`
- d. `podman pull registry.redhat.io/httpd:2.2`

► **4. When running the following commands, what container images will be downloaded?**

```
podman pull registry.redhat.io/httpd:2.2  
podman pull quay.io/mysql:5.6
```

- a. `quay.io/httpd:2.2`
`registry.redhat.io/mysql:5.6`
- b. `registry.redhat.io/httpd:2.2`
`registry.redhat.io/mysql:5.6`
- c. `registry.redhat.io/httpd:2.2`
No image will be downloaded for mysql.
- d. `quay.io/httpd:2.2`
No image will be downloaded for mysql.

► Solution

Working With Registries

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

Podman is available on a RHEL host with the following entry in `/etc/containers/registries.conf` file:

```
[registries.search]
registries = ["registry.redhat.io", "quay.io"]
```

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

Image names/tags per registry

Registry	Image
registry.redhat.io	<ul style="list-style-type: none"> nginx/1.0 mysql/5.6 httpd/2.2
quay.io	<ul style="list-style-type: none"> mysql/5.5 httpd/2.4

No images are locally available.

- 1. Which two commands display mysql images available for download from `registry.redhat.io`? (Choose two.)
 - a. `podman search registry.redhat.io/mysql`
 - b. `podman images`
 - c. `podman pull mysql`
 - d. `podman search mysql`

- 2. Which command is used to list all available image tags for the httpd container image?
 - a. `podman search httpd`
 - b. `podman images httpd`
 - c. `podman pull --all-tags=true httpd`
 - d. There is no podman command available to search for tags.

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

- a. **podman pull httpd:2.2**
- b. **podman pull httpd:latest**
- c. **podman pull quay.io/httpd**
- d. **podman pull registry.redhat.io/httpd:2.2**

► **4. When running the following commands, what container images will be downloaded?**

```
podman pull registry.redhat.io/httpd:2.2  
podman pull quay.io/mysql:5.6
```

- a. quay.io/httpd:2.2
registry.redhat.io/mysql:5.6
- b. registry.redhat.io/httpd:2.2
registry.redhat.io/mysql:5.6
- c. registry.redhat.io/httpd:2.2
No image will be downloaded for mysql.
- d. quay.io/httpd:2.2
No image will be downloaded for mysql.

Manipulating Container Images

Objectives

After completing this section, students should be able to:

- Save and load container images to local files.
- Delete images from the local storage.
- Create new container images from containers and update image metadata.
- Manage image tags for distribution purposes.

Introduction

There are various ways to manage image containers while adhering to DevOps principles. For example, a developer finishes testing a custom container in a machine and needs to transfer this container image to another host for another developer, or to a production server. There are two ways to do this:

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



Note

One of the ways a developer could have created this custom container is discussed later in this chapter (`podman commit`). However, in the following chapters we discuss the recommended way to do so using `Dockerfiles`.

Saving and Loading Images

Existing images from the Podman local storage can be saved to a `.tar` file using the `podman save` command. The generated file is not a regular TAR archive; it contains image metadata and preserves the original image layers. Using this file, Podman can recreate the original image exactly as it was.

The general syntax of the `save` subcommand is as follows:

```
[student@workstation ~]$ sudo podman save [-o FILE_NAME] IMAGE_NAME[:TAG]
```

Podman sends the generated image to the standard output as binary data. To avoid that, use the `-o` option.

The following example saves the previously downloaded MySQL container image from the Red Hat Container Catalog to the `mysql.tar` file:

```
[student@workstation ~]$ sudo podman save \
> -o mysql.tar registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7
```

Use the **.tar** files generated by the **save** subcommand for backup purposes. To restore the container image, use the **podman load** command. The general syntax of the command is as follows:

```
[student@workstation ~]$ sudo podman load [-i FILE_NAME]
```

For example, this command would load an image saved in a file named **mysql.tar**.

```
[student@workstation ~]$ sudo podman load -i mysql.tar
```

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



Note

To save disk space, compress the file generated by the **save** subcommand with Gzip using the **--compress** parameter. The **load** subcommand uses the **gunzip** command before importing the file to the local storage.

Deleting Images

Podman keeps any image downloaded in its local storage, even the ones currently unused by any container. However, images can become outdated, and should be subsequently replaced.



Note

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

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

```
[student@workstation ~]$ sudo podman rmi [OPTIONS] IMAGE [IMAGE...]
```

An image can be referenced using its name or its ID for removal purposes. Podman cannot delete images while containers are using that image. You must stop and remove all containers using that image before deleting it.

To avoid this, the **rmi** subcommand has the **--force** option. This option forces the removal of an image even if that the image is used by several containers or these containers are running. Podman stops and removes all containers using the forcefully removed image before removing it.

Deleting all Images

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

```
[student@workstation ~]$ sudo podman rmi -a
```

The command returns all the image IDs available in the local storage and passes them as parameters to the **podman rmi** command for removal. Images that are in use are not deleted. However, this does not prevent any unused images from being removed.

Modifying Images

Ideally, all container images should be built using a **Dockerfile**, in order to create a clean, lightweight set of image layers without log files, temporary files, or other artifacts created by the container customization. However, some users may provide container images as they are, without a **Dockerfile**. As an alternative approach to creating new images, change a running container in place and save its layers to create a new container image. The **podman commit** command provides this feature.



Warning

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

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

```
[student@workstation ~]$ sudo podman commit [OPTIONS] CONTAINER \
> [REPOSITORY[:PORT]/]IMAGE_NAME[:TAG]
```

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

Option	Description
--author ""	Identifies who created the container image.
--message ""	Includes a commit message to the registry.
--format	Selects the format of the image. Valid options are oci and docker .



Note

The **--message** option is not available in the default OCI container format.

To find the ID of a running container in Podman, run the **podman ps** command:

```
[student@workstation ~]$ sudo podman ps
CONTAINER ID IMAGE ... NAMES
87bdfcc7c656 mysql ...output omitted... mysql-basic
```

Eventually, administrators might customize the image and set the container to the desired state. To identify which files were changed, created, or deleted since the container was started, use the **diff** subcommand. This subcommand only requires the container name or container ID:

```
[student@workstation ~]$ sudo podman diff mysql-basic
C /run
C /run/mysql
A /run/mysql/mysql.pid
A /run/mysql/mysql.sock
A /run/mysql/mysql.sock.lock
A /run/secrets
```

The **diff** subcommand tags any added file with an **A**, any changed ones with a **C**, and any deleted file with a **D**.



Note

The **diff** command only reports added, changed, or deleted files to the container file system. Files that are mounted to a running container are not considered part of the container file system.

To retrieve the list of mounted files and directories for a running container, use the **podman inspect** command:

```
[student@workstation ~]$ sudo podman inspect \
> -f "{{range .Mounts}}{{println .Destination}}{{end}}" CONTAINER_NAME/ID
```

Any file in this list, or file under a directory in this list, is not shown in the output of the **podman diff** command.

To commit the changes to another image, run the following command:

```
[student@workstation ~]$ sudo podman commit mysql-basic mysql-custom
```

Tagging Images

A project with multiple images based on the same software could be distributed, creating individual projects for each image, but this approach requires more maintenance for managing and deploying the images to the correct locations.

Container image registries support tags to distinguish multiple releases of the same project. For example, a customer might use a container image to run with a MySQL or PostgreSQL database, using a tag as a way to differentiate which database is to be used by a container image.



Note

Usually, the tags are used by container developers to distinguish between multiple versions of the same software. Multiple tags are provided to identify a release easily. The official MySQL container image website uses the version as the tag's name (**5.5.16**). Also, the same image has a second tag with the minor version, such as 5.5, to minimize the need to get the latest release for a specific version.

To tag an image, use the **podman tag** command:

```
[student@workstation ~]$ sudo podman tag [OPTIONS] IMAGE[:TAG] \
> [REGISTRYHOST/]USERNAME[/:NAME[:TAG]]
```

The **IMAGE** argument is the image name with an optional tag, which is managed by Podman. The following argument refers to the new alternative name for the image. Podman assumes the latest version, as indicated by the **latest** tag, if the tag value is absent. For example, to tag an image, use the following command:

```
[student@workstation ~]$ sudo podman tag mysql-custom devops/mysql
```

The **mysql-custom** option corresponds to the image name in the container registry.

To use a different tag name, use the following command instead:

```
[student@workstation ~]$ sudo podman tag mysql-custom devops/mysql:snapshot
```

Removing Tags from Images

A single image can have multiple tags assigned using the **podman tag** command. To remove them, use the **podman rmi** command, as mentioned earlier:

```
[student@workstation ~]$ sudo podman rmi devops/mysql:snapshot
```



Note

Because multiple tags can point to the same image, to remove an image referred to by multiple tags, first remove each tag individually.

Best Practices for Tagging Images

Podman automatically adds the **latest** tag if you do not specify any tag, because Podman considers the image to be the latest build. However, this may not be true depending on how each project uses tags. For example, many open source projects consider the **latest** tag to match the most recent release, but not the latest build.

Moreover, multiple tags are provided to minimize the need to remember the latest release of a particular version of a project. Thus, if there is a project version release, for example, **2.1.10**, another tag called **2.1** can be created pointing to the same image from the **2.1.10** release. This simplifies pulling images from the registry.

Publishing Images to a Registry

To publish an image to a registry, it must reside in the Podman's local storage and be tagged for identification purposes. To push the image to the registry the syntax of the **push** subcommand is:

```
[student@workstation ~]$ sudo podman push [OPTIONS] IMAGE [DESTINATION]
```

For example, to push the **bitnami/nginx** image to its repository, use the following command:

```
[student@workstation ~]$ sudo podman push quay.io/bitnami/nginx
```



References

Podman site

<https://podman.io/>

► Guided Exercise

Creating a Custom Apache Container Image

In this guided exercise, you will create a custom Apache container image using the **podman commit** command.

Outcomes

You should be able to create a custom container image.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab image-operations start
```

- 1. Log into your Quay.io account and start a container by using the image available at **quay.io/redhattraining/httpd-parent**. The **-p** option allows you to specify a redirect port. In this case, Podman forwards incoming requests on TCP port 8180 of the host to TCP port 80 of the container.

```
[student@workstation ~]$ sudo podman login quay.io
Username: your_quay_username
Password: your_quay_password
Login Succeeded!
[student@workstation ~]$ sudo podman run -d --name official-httpd \
> -p 8180:80 redhattraining/httpd-parent
...output omitted...
Writing manifest to image destination
Storing signatures
3a6baecaff2b4e8c53b026e04847dda5976b773ade1a3a712b1431d60ac5915d
```

Your last line of output is different from the last line shown above. Note the first twelve characters.

- 2. Create an HTML page on the **official-httpd** container.
 - 2.1. Access the shell of the container by using the **podman exec** command and create an HTML page.

```
[student@workstation ~]$ sudo podman exec -it official-httpd /bin/bash
bash-4.4# echo "DO180 Page" > /var/www/html/do180.html
```

2.2. Exit from the container.

```
bash-4.4# exit
```

- 2.3. Ensure that the HTML file is reachable from the **workstation** VM by using the **curl** command.

```
[student@workstation ~]$ curl 127.0.0.1:8180/do180.html
```

You should see the following output:

D0180 Page

- 3. Use the **podman diff** command to examine the differences in the container between the image and the new layer created by the container.

```
[student@workstation ~]$ sudo podman diff official-httdp
C /etc
C /root
A /root/.bash_history
...output omitted...
C /tmp
C /var
C /var/log
C /var/log/httpd
A /var/log/httpd/access_log
A /var/log/httpd/error_log
C /var/www
C /var/www/html
A /var/www/html/do180.html
```



Note

Often, web server container images label the **/var/www/html** directory as a volume. In these cases, any files added to this directory are not considered part of the container file system, and would not show in the output of the **git diff** command.

The **redhattraining/httpd-parent** container image does not label the **/var/www/html** directory as a volume. As a result, the change to the **/var/www/html/do180.html** file is considered a change to the underlying container file system.

- 4. Create a new image with the changes created by the running container.

- 4.1. Stop the **official-httdp** container.

```
[student@workstation ~]$ sudo podman stop official-httdp
3a6baecaff2b4e8c53b026e04847dda5976b773ade1a3a712b1431d60ac5915d
```

- 4.2. Commit the changes to a new container image with a new name. Use your name as the author of the changes.

```
[student@workstation ~]$ sudo podman commit \
> -a 'Your Name' official-httdp do180-custom-httdp
Getting image source signatures
Skipping fetch of repeat blob sha256:071d8bd765171080d01682844524be57ac9883e...
```

```
...output omitted...
Copying blob sha256:1e19be875ce6f5b9dece378755eb9df96ee205abfb4f165c797f59a9...
  15.00 KB / 15.00 KB [=====] 0s
Copying config sha256:8049dc2e7d0a0b1a70fc0268ad236399d9f5fb686ad4e31c7482cc...
  2.99 KB / 2.99 KB [=====] 0s
Writing manifest to image destination
Storing signatures
31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeeb399befef2a23
```

- 4.3. List the available container images.

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

The expected output is similar to the following:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/do180-custom-httd	latest	31c3ac78e9d4
quay.io/redhattraining/httpd-parent	latest	2cc07fbb5000

The image ID matches the first 12 characters of the hash. The most recent images are listed at the top.

► 5. Publish the saved container image to the container registry.

- 5.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

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

- 5.2. To tag the image with the registry host name and tag, run the following command.

```
[student@workstation ~]$ sudo podman tag do180-custom-httd \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-custom-httd:v1.0
```

- 5.3. Run the **podman images** command to ensure that the new name has been added to the cache.

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

REPOSITORY	TAG	IMAGE ID	...
localhost/do180-custom-httd	latest	31c3ac78e9d4	...
quay.io/\${RHT_OCP4_QUAY_USER}/do180-custom-httd	v1.0	31c3ac78e9d4	...
quay.io/redhattraining/httpd-parent	latest	2cc07fbb5000	...

- 5.4. Publish the image to your Quay.io registry.

```
[student@workstation ~]$ sudo podman push \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-custom-httd:v1.0
Getting image source signatures
Copying blob sha256:071d8bd765171080d01682844524be57ac9883e53079b6ac66707e19...
```

```
200.44 MB / 200.44 MB [=====] 1m38s
...output omitted...
Copying config sha256:31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeef3...
2.99 KB / 2.99 KB [=====] 0s
Writing manifest to image destination
Copying config sha256:31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeef3...
0 B / 2.99 KB [-----] 0s
Writing manifest to image destination
Storing signatures
```



Note

Pushing the **do180-custom-httppd** image creates a private repository in your Quay.io account. Currently, private repositories are disallowed by Quay.io free plans. You can either create the public repository prior to pushing the image, or change the repository to public afterwards.

- 5.5. Verify that the image is available from Quay.io. The **podman search** command requires the image to be indexed by Quay.io. That may take some hours to occur, so use the **podman pull** command to fetch the image. This proves the availability for the image.

```
[student@workstation ~]$ sudo podman pull \
> -q quay.io/${RHT_OCP4_QUAY_USER}/do180-custom-httppd:v1.0
31c3ac78e9d4137c928da23762e7d32b00c428eb1036cab1caeef3
```

- ▶ 6. Create a container from the newly published image.

Use the **podman run** command to start a new container. Use **your_quay_username/do180-custom-httppd:v1.0** as the base image.

```
[student@workstation ~]$ sudo podman run -d --name test-httppd -p 8280:80 \
> ${RHT_OCP4_QUAY_USER}/do180-custom-httppd:v1.0
c0f04e906bb12bd0e514cbd0e581d2746e04e44a468dfbc85bc29ffcc5acd16c
```

- ▶ 7. Use the **curl** command to access the HTML page. Make sure you use port 8280.

This should display the HTML page created in the previous step.

```
[student@workstation ~]$ curl http://localhost:8280/do180.html
DO180 Page
```

Finish

On **workstation**, run the **lab image-operations finish** script to complete this lab.

```
[student@workstation ~]$ lab image-operations finish
```

This concludes the guided exercise.

► Lab

Managing Images

Performance Checklist

In this lab, you will create and manage container images.

Outcomes

You should be able to create a custom container image and manage container images.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab image-review start
```

1. Use the **podman pull** command to download the **quay.io/redhattraining/nginx:1.17** container image. This image is a copy of the official container image available at docker.io/library/nginx:1.17.
Ensure that you successfully downloaded the image.
2. Start a new container using the Nginx image, according to the specifications listed in the following list.
 - **Name:** **official-nginx**
 - **Run as daemon:** yes
 - **Container image:** **nginx**
 - **Port forward:** from host port 8080 to container port 80.
3. Log in to the container using the **podman exec** command. Replace the contents of the **index.html** file with **DO180**. The web server directory is located at **/usr/share/nginx/html**.
After the file has been updated, exit the container and use the **curl** command to access the web page.
4. Stop the running container and commit your changes to create a new container image. Give the new image a name of **do180/mynginx** and a tag of **v1.0-SNAPSHOT**. Use the following specifications:
 - Image name: **do180/mynginx**
 - Image tag: **v1.0-SNAPSHOT**
 - Author name: *your name*
5. Start a new container using the updated Nginx image, according to the specifications listed in the following list.
 - **Name:** **official-nginx-dev**

- **Run as daemon:** yes
 - **Container image:** **do180/mynginx:v1.0-SNAPSHOT**
 - **Port forward:** from host port 8080 to container port 80.
6. Log in to the container using the **podman exec** command, and introduce a final change. Replace the contents of the file **/usr/share/nginx/html/index.html** file with **DO180 Page**. After the file has been updated, exit the container and use the **curl** command to verify the changes.
7. Stop the running container and commit your changes to create the final container image. Give the new image a name of **do180/mynginx** and a tag of **v1.0**. Use the following specifications:
- Image name: **do180/mynginx**
 - Image tag: **v1.0**
 - Author name: *your name*
8. Remove the development image **do180/mynginx:v1.0-SNAPSHOT** from local image storage.
9. Use the image tagged **do180/mynginx:v1.0** to create a new container with the following specifications:
- Container name: **my-nginx**
 - Run as daemon: yes
 - Container image: **do180/mynginx:v1.0**
 - Port forward: from host port 8280 to container port 80

On **workstation**, use the **curl** command to access the web server, accessible from the port 8280.

Evaluation

Grade your work by running the **lab image-review grade** command on your **workstation** machine. Correct any reported failures and rerun the script until successful.

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

Finish

On **workstation**, run the **lab image-review finish** command to complete this lab.

```
[student@workstation ~]$ lab image-review finish
```

This concludes the lab.

► Solution

Managing Images

Performance Checklist

In this lab, you will create and manage container images.

Outcomes

You should be able to create a custom container image and manage container images.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab image-review start
```

1. Use the **podman pull** command to download the **quay.io/redhattraining/nginx:1.17** container image. This image is a copy of the official container image available at docker.io/library/nginx:1.17.

Ensure that you successfully downloaded the image.

- 1.1. Use the **podman pull** command to pull the Nginx container image.

```
[student@workstation ~]$ sudo podman pull quay.io/redhattraining/nginx:1.17
Trying to pull Trying to pull quay.io/redhattraining/nginx:1.17...
...output omitted...
Storing signatures
9beeba249f3ee158d3e495a6ac25c5667ae2de8a43ac2a8bfd2bf687a58c06c9
```

- 1.2. Ensure that the container image exists on the local system by running the **podman images** command.

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

This command produces output similar to the following:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
quay.io/redhattraining/nginx	1.17	9beeba249f3e	6 months ago	131MB

2. Start a new container using the Nginx image, according to the specifications listed in the following list.

- **Name:** **official-nginx**
- **Run as daemon:** yes
- **Container image:** **nginx**

- **Port forward:** from host port 8080 to container port 80.

- 2.1. On **workstation**, use the **podman run** command to create a container named **official-nginx**.

```
[student@workstation ~]$ sudo podman run --name official-nginx \
> -d -p 8080:80 quay.io/redhattraining/nginx:1.17
b9d5739af239914b371025c38340352ac1657358561e7ebbd5472dfd5ff97788
```

3. Log in to the container using the **podman exec** command. Replace the contents of the **index.html** file with **DO180**. The web server directory is located at **/usr/share/nginx/html**.

After the file has been updated, exit the container and use the **curl** command to access the web page.

- 3.1. Log in to the container by using the **podman exec** command.

```
[student@workstation ~]$ sudo podman exec -it official-nginx /bin/bash
root@b9d5739af239:/#
```

- 3.2. Update the **index.html** file located at **/usr/share/nginx/html**. The file should read **DO180**.

```
root@b9d5739af239:/# echo 'DO180' > /usr/share/nginx/html/index.html
```

- 3.3. Exit the container.

```
root@b9d5739af239:/# exit
```

- 3.4. Use the **curl** command to ensure that the **index.html** file is updated.

```
[student@workstation ~]$ curl 127.0.0.1:8080
DO180
```

4. Stop the running container and commit your changes to create a new container image. Give the new image a name of **do180/mynginx** and a tag of **v1.0-SNAPSHOT**. Use the following specifications:

- Image name: **do180/mynginx**
- Image tag: **v1.0-SNAPSHOT**
- Author name: *your name*

- 4.1. Use the **sudo podman stop** command to stop the **official-nginx** container.

```
[student@workstation ~]$ sudo podman stop official-nginx
b9d5739af239914b371025c38340352ac1657358561e7ebbd5472dfd5ff97788
```

- 4.2. Commit your changes to a new container image. Use your name as the author of the changes.

```
[student@workstation ~]$ sudo podman commit -a 'Your Name' \
> official-nginx do180/mynginx:v1.0-SNAPSHOT
Getting image source signatures
...output omitted...
Storing signatures
d6d10f52e258e4e88c181a56c51637789424e9261b208338404e82a26c960751
```

- 4.3. List the available container images to locate your newly created image.

```
[student@workstation ~]$ sudo podman images
REPOSITORY           TAG      IMAGE ID      CREATED     ...
localhost/do180/mynginx   v1.0-SNAPSHOT  d6d10f52e258  30 seconds ago ...
quay.io/redhattraining/nginx  1.17       9beeba249f3e  6 months ago ...
```

5. Start a new container using the updated Nginx image, according to the specifications listed in the following list.

- **Name:** `official-nginx-dev`
- **Run as daemon:** yes
- **Container image:** `do180/mynginx:v1.0-SNAPSHOT`
- **Port forward:** from host port 8080 to container port 80.

- 5.1. On **workstation**, use the `podman run` command to create a container named `official-nginx-dev`.

```
[student@workstation ~]$ sudo podman run --name official-nginx-dev \
> -d -p 8080:80 do180/mynginx:v1.0-SNAPSHOT
cfa21f02a77d0e46e438c255f83dea2dfb89bb5aa72413a28866156671f0bbbb
```

6. Log in to the container using the `podman exec` command, and introduce a final change. Replace the contents of the file `/usr/share/nginx/html/index.html` file with **DO180 Page**.

After the file has been updated, exit the container and use the `curl` command to verify the changes.

- 6.1. Log in to the container by using the `podman exec` command.

```
[student@workstation ~]$ sudo podman exec -it official-nginx-dev /bin/bash
root@cfa21f02a77d:/#
```

- 6.2. Update the `index.html` file located at `/usr/share/nginx/html`. The file should read **DO180 Page**.

```
root@cfa21f02a77d:/# echo 'DO180 Page' > /usr/share/nginx/html/index.html
```

- 6.3. Exit the container.

```
root@cfa21f02a77d:/# exit
```

- 6.4. Use the **curl** command to ensure that the **index.html** file is updated.

```
[student@workstation ~]$ curl 127.0.0.1:8080
DO180 Page
```

7. Stop the running container and commit your changes to create the final container image. Give the new image a name of **do180/mynginx** and a tag of **v1.0**. Use the following specifications:

- Image name: **do180/mynginx**
- Image tag: **v1.0**
- Author name: *your name*

- 7.1. Use the **sudo podman stop** command to stop the **official-nginx-dev** container.

```
[student@workstation ~]$ sudo podman stop official-nginx-dev
cfa21f02a77d0e46e438c255f83dea2dfb89bb5aa72413a28866156671f0bbbb
```

- 7.2. Commit your changes to a new container image. Use your name as the author of the changes.

```
[student@workstation ~]$ sudo podman commit -a 'Your Name' \
> official-nginx-dev do180/mynginx:v1.0
Getting image source signatures
...output omitted...
Storing signatures
90915976c33de534e06778a74d2a8969c25ef5f8f58c0c1ab7aeaac19abd18af
```

- 7.3. List the available container images in order to locate your newly created image.

```
[student@workstation ~]$ sudo podman images
REPOSITORY          TAG      IMAGE ID   CREATED     ...
localhost/do180/mynginx    v1.0    90915976c33d  6 seconds ago ...
localhost/do180/mynginx    v1.0-SNAPSHOT  d6d10f52e258  8 minutes ago ...
quay.io/redhattraining/nginx  1.17    9beeba249f3e  6 months ago ...
```

8. Remove the development image **do180/mynginx:v1.0-SNAPSHOT** from local image storage.

- 8.1. Despite being stopped, the **official-nginx-dev** is still present. Display the container with the **podman ps** command with the **-a** flag.

```
[student@workstation ~]$ sudo podman ps -a \
> --format="{{.ID}} {{.Names}} {{.Status}}"
cfa21f02a77d  official-nginx-dev  Exited (0) 9 minutes ago
b9d5739af239  official-nginx      Exited (0) 12 minutes ago
```

- 8.2. Remove the container with the **podman rm** command.

```
[student@workstation ~]$ sudo podman rm official-nginx-dev
cfa21f02a77d0e46e438c255f83dea2dfb89bb5aa72413a28866156671f0bbbb
```

- 8.3. Verify that the container is deleted by resubmitting the same **podman ps** command.

```
[student@workstation ~]$ sudo podman ps -a \
> --format="{{.ID}} {{.Names}} {{.Status}}"
b9d5739af239    official-nginx      Exited (0) 12 minutes ago
```

- 8.4. Use the **sudo podman rmi** command to remove the **do180/mynginx:v1.0-SNAPSHOT** image.

```
[student@workstation ~]$ sudo podman rmi do180/mynginx:v1.0-SNAPSHOT
Untagged: localhost/do180/mynginx:v1.0-SNAPSHOT
```

- 8.5. Verify that the image is no longer present by listing all images using the **podman images** command.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/do180/mynginx	v1.0	90915976c33d	5 minutes ago	131MB
quay.io/redhattraining/nginx	1.17	9beeba249f3e	6 months ago	131MB

9. Use the image tagged **do180/mynginx:v1.0** to create a new container with the following specifications:

- Container name: **my-nginx**
- Run as daemon: yes
- Container image: **do180/mynginx:v1.0**
- Port forward: from host port 8280 to container port 80

On **workstation**, use the **curl** command to access the web server, accessible from the port 8280.

- 9.1. Use the **sudo podman run** command to create the **my-nginx** container, according to the specifications.

```
[student@workstation ~]$ sudo podman run -d --name my-nginx \
> -p 8280:80 do180/mynginx:v1.0
51958c8ec8d2613bd26f85194c66ca96c95d23b82c43b23b0f0fb9fded74da20
```

- 9.2. Use the **curl** command to ensure that the **index.html** page is available and returns the custom content.

```
[student@workstation ~]$ curl 127.0.0.1:8280
DO180 Page
```

Evaluation

Grade your work by running the **lab image-review grade** command on your **workstation** machine. Correct any reported failures and rerun the script until successful.

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

Finish

On **workstation**, run the **lab image-review finish** command to complete this lab.

```
[student@workstation ~]$ lab image-review finish
```

This concludes the lab.

Summary

In this chapter, you learned:

- The Red Hat Container Catalog provides tested and certified images at `registry.redhat.io`.
- Podman can interact with remote container registries to search, pull, and push container images.
- Image tags are a mechanism to support multiple releases of a container image.
- Podman provides commands to manage container images both in local storage and as compressed files.
- Use the **podman commit** to create an image from a container.

Chapter 5

Creating Custom Container Images

Goal

Design and code a Dockerfile to build a custom container image.

Objectives

- Describe the approaches for creating custom container images.
- Create a container image using common Dockerfile commands.

Sections

- Designing Custom Container Images (and Quiz)
- Building Custom Container Images with Dockerfiles (and Guided Exercise)

Lab

Creating Custom Container Images

Designing Custom Container Images

Objectives

After completing this section, students should be able to:

- Describe the approaches for creating custom container images.
- Find existing Dockerfiles to use as a starting point for creating a custom container image.
- Define the role played by the Red Hat Software Collections Library (RHSCl) in designing container images from the Red Hat registry.
- Describe the Source-to-Image (S2I) alternative to Dockerfiles.

Reusing Existing Dockerfiles

One method of creating container images has been covered so far: create a container, modify it to meet the requirements of the application to run in it, and then commit the changes to an image. This option, although straightforward, is only suitable for using or testing very specific changes. It does not follow best software practices, like maintainability, automation of building, and repeatability.

Dockerfiles are another option for creating container images, addressing these limitations. Dockerfiles are easy to share, version control, reuse, and extend.

Dockerfiles also make it easy to extend one image, called a *child image*, from another image, called a *parent image*. A child image incorporates everything in the parent image and all changes and additions made to create it.

To share and reuse images, many popular applications, languages, and frameworks are already available in public image registries such as Quay .io. It is not trivial to customize an application configuration to follow recommended practices for containers, and so starting from a proven parent image usually saves a lot of work.

Using a high-quality parent image enhances maintainability, especially if the parent image is kept updated by its author to account for bug fixes and security issues.

Typical scenarios in creating a Dockerfile for building a child image from an existing container image include:

- Add new runtime libraries, such as database connectors.
- Include organization-wide customization such as SSL certificates and authentication providers.
- Add internal libraries to be shared as a single image layer by multiple container images for different applications.

Changing an existing Dockerfile to create a new image can also be a sensible approach in other scenarios. For example:

- Trim the container image by removing unused material (such as man pages, or documentation found in `/usr/share/doc`).

- Lock either the parent image or some included software package to a specific release to lower risk related to future software updates.

Two sources of container images to use either as parent images or for changing their Dockerfiles are Docker Hub and the Red Hat Software Collections Library (RHSCl).

Working with the Red Hat Software Collections Library

Red Hat Software Collections Library (RHSCl), or simply Software Collections, is Red Hat's solution for developers who need to use the latest development tools that usually do not fit the standard RHEL release schedule.

Red Hat Enterprise Linux (RHEL) provides a stable environment for enterprise applications. This requires RHEL to keep the major releases of upstream packages at the same level to prevent API and configuration file format changes. Security and performance fixes are backported from later upstream releases, but new features that would break backward-compatibility are not backported.

RHSCl allows software developers to use the latest version without impacting RHEL, because RHSCl packages do not replace or conflict with default RHEL packages. Default RHEL packages and RHSCl packages are installed side-by-side.



Note

All RHEL subscribers have access to the RHSCl. To enable a particular *software collection* for a specific user or application environment (for example, MySQL 5.7, which is named `rh-mysql157`), enable the RHSCl software Yum repositories and follow a few simple steps.

Finding Dockerfiles from the Red Hat Software Collections Library

RHSCl is the source of most container images provided by the Red Hat image registry for use by RHEL Atomic Host and OpenShift Container Platform customers.

Red Hat provides RHSCl Dockerfiles and related sources in the `rhscl-dockerfiles` package available from the RHSCl repository. Community users can get Dockerfiles for CentOS-based equivalent container images from <https://github.com/sclorg?q=-container>.



Note

Many RHSCl container images include support for *Source-to-Image (S2I)*, best known as an OpenShift Container Platform feature. Having support for S2I does not affect the use of these container images with Docker.

Container Images in Red Hat Container Catalog (RHCC)

Mission-critical applications require trusted containers. The *Red Hat Container Catalog* is a repository of reliable, tested, certified, and curated collection of container images built on versions of Red Hat Enterprise Linux (RHEL) and related systems. Container images available through RHCC have undergone a quality-assurance process. All components have been rebuilt by Red Hat to avoid known security vulnerabilities. They are upgraded on a regular basis so that they contain the required version of software even when a new image is not yet available. Using RHCC, you

can browse and search for images, and you can access information about each image, such as its version, contents, and usage.

Searching for Images Using Quay.io

Quay.io is an advanced container repository from CoreOS optimized for team collaboration. You can search for container images using <https://quay.io/search>.

Clicking on an image's name provides access to the image information page, including access to all existing tags for the image, and the command to pull the image.

Finding Dockerfiles on Docker Hub

Anyone can create a Docker Hub account and publish container images there. There are no general assurances about quality and security; images on Docker Hub range from professionally supported to one-time experiments. Each image has to be evaluated individually.

After searching for an image, the documentation page might provide a link to its Dockerfile. For example, the first result when searching for **mysql** is the documentation page for the **MySQL official** image at https://hub.docker.com/_/mysql.

On that page, the link for the **5.6/Dockerfile** image points to the **docker-library** GitHub project, which hosts **Dockerfiles** for images built by the Docker community automatic build system.

The direct URL for the Docker Hub MySQL 5.6 **Dockerfile** tree is <https://github.com/docker-library/mysql/blob/master/5.6>.

Describing How to use the OpenShift Source-to-Image Tool

Source-to-Image (S2I) provides an alternative to using Dockerfiles to create new container images and can be used either as a feature from OpenShift or as the standalone **s2i** utility. S2I allows developers to work using their usual tools, instead of learning Dockerfile syntax and using operating system commands such as **yum**, and usually creates slimmer images, with fewer layers.

S2I uses the following process to build a custom container image for an application:

1. Start a container from a base container image called the *builder image*, which includes a programming language runtime and essential development tools such as compilers and package managers.
2. Fetch the application source code, usually from a Git server, and send it to the container.
3. Build the application binary files inside the container.
4. Save the container, after some clean up, as a new container image, which includes the programming language runtime and the application binaries.

The builder image is a regular container image following a standard directory structure and providing scripts that are called during the S2I process. Most of these builder images can also be used as base images for Dockerfiles, outside the S2I process.

The **s2i** command is used to run the S2I process outside of OpenShift, in a Docker-only environment. It can be installed on a RHEL system from the *source-to-image* RPM package, and on other platforms, including Windows and Mac OS, from the installers available in the S2I project on GitHub.



References

Red Hat Software Collections Library (RHSCl)

<https://access.redhat.com/documentation/en/red-hat-software-collections/>

Red Hat Container Catalog (RHCC)

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

RHSCl Dockerfiles on GitHub

<https://github.com/sclorg?q=-container>

Using Red Hat Software Collections Container Images

<https://access.redhat.com/articles/1752723>

Quay.io

<https://quay.io/search>

Docker Hub

<https://hub.docker.com/>

Docker Library GitHub project

<https://github.com/docker-library>

The S2I GitHub project

<https://github.com/openshift/source-to-image>

► Quiz

Approaches to Container Image Design

Choose the correct answers to the following questions:

- ▶ 1. **Which method for creating container images is recommended by the containers community? (Choose one.)**
 - a. Run commands inside a basic OS container, commit the container, and save or export it as a new container image.
 - b. Run commands from a Dockerfile and push the generated container image to an image registry.
 - c. Create the container image layers manually from tar files.
 - d. Run the **podman build** command to process a container image description in YAML format.

- ▶ 2. **What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)**
 - a. Requires no additional tools apart from a basic Podman setup.
 - b. Creates smaller container images, having fewer layers.
 - c. Reuses high-quality builder images.
 - d. Automatically updates the child image as the parent image changes (for example, with security fixes).
 - e. Creates images compatible with OpenShift, unlike container images created from Docker tools.

- ▶ 3. **What are two typical scenarios for creating a Dockerfile to build a child image from an existing image? (Choose two.)**
 - a. Adding new runtime libraries.
 - b. Setting constraints to a container's access to the host machine's CPU.
 - c. Adding internal libraries to be shared as a single image layer by multiple container images for different applications.

► Solution

Approaches to Container Image Design

Choose the correct answers to the following questions:

- ▶ 1. **Which method for creating container images is recommended by the containers community? (Choose one.)**
 - a. Run commands inside a basic OS container, commit the container, and save or export it as a new container image.
 - b. Run commands from a Dockerfile and push the generated container image to an image registry.
 - c. Create the container image layers manually from tar files.
 - d. Run the `podman build` command to process a container image description in YAML format.
- ▶ 2. **What are two advantages of using the standalone S2I process as an alternative to Dockerfiles? (Choose two.)**
 - a. Requires no additional tools apart from a basic Podman setup.
 - b. Creates smaller container images, having fewer layers.
 - c. Reuses high-quality builder images.
 - d. Automatically updates the child image as the parent image changes (for example, with security fixes).
 - e. Creates images compatible with OpenShift, unlike container images created from Docker tools.
- ▶ 3. **What are two typical scenarios for creating a Dockerfile to build a child image from an existing image? (Choose two.)**
 - a. Adding new runtime libraries.
 - b. Setting constraints to a container's access to the host machine's CPU.
 - c. Adding internal libraries to be shared as a single image layer by multiple container images for different applications.

Building Custom Container Images with Dockerfiles

Objectives

After completing this section, students should be able to create a container image using common Dockerfile commands.

Building Base Containers

A **Dockerfile** is a mechanism to automate the building of container images. Building an image from a Dockerfile is a three-step process:

1. Create a working directory
2. Write the **Dockerfile**
3. Build the image with Podman

Create a Working Directory

The *working directory* is the directory containing all files needed to build the image. Creating an empty working directory is good practice to avoid incorporating unnecessary files into the image. For security reasons, the root directory, `/`, should never be used as a working directory for image builds.

Write the Dockerfile Specification

A **Dockerfile** is a text file that must exist in the working directory. This file contains the instructions needed to build the image. The basic syntax of a **Dockerfile** follows:

```
# Comment  
INSTRUCTION arguments
```

Lines that begin with a hash, or pound, symbol (#) are comments. *INSTRUCTION* states for any Dockerfile instruction keyword. Instructions are not case-sensitive, but the convention is to make instructions all uppercase to improve visibility.

The first non-comment instruction must be a **FROM** instruction to specify the base image. Dockerfile instructions are executed into a new container using this image and then committed to a new image. The next instruction (if any) executes into that new image. The execution order of instructions is the order of their appearance in the Dockerfile.



Note

The **ARG** instruction can appear before the **FROM** instruction, but **ARG** instructions are outside the objectives for this section.

Each Dockerfile instruction runs in an independent container using an intermediate image built from every previous command. This means each instruction is independent from other instructions in the Dockerfile.

The following is an example Dockerfile for building a simple Apache web server container:

```
# This is a comment line ①
FROM ubi7/ubi:7.7 ②
LABEL description="This is a custom httpd container image" ③
MAINTAINER John Doe <jdoe@xyz.com> ④
RUN yum install -y httpd ⑤
EXPOSE 80 ⑥
ENV LogLevel "info" ⑦
ADD http://someserver.com/filename.pdf /var/www/html ⑧
COPY ./src/ /var/www/html/ ⑨
USER apache ⑩
ENTRYPOINT ["/usr/sbin/httpd"] ⑪
CMD ["-D", "FOREGROUND"] ⑫
```

- ① Lines that begin with a hash, or pound, sign (#) are comments.
- ② The **FROM** instruction declares that the new container image extends **ubi7/ubi:7.7** container base image. Dockerfiles can use any other container image as a base image, not only images from operating system distributions. Red Hat provides a set of container images that are certified and tested and highly recommends using these container images as a base.
- ③ The **LABEL** is responsible for adding generic metadata to an image. A **LABEL** is a simple key-value pair.
- ④ The **MAINTAINER** indicates the **Author** field of the generated container image's metadata. You can use the **podman inspect** command to view image metadata.
- ⑤ **RUN** executes commands in a new layer on top of the current image. The shell that is used to execute commands is **/bin/sh**.
- ⑥ **EXPOSE** indicates that the container listens on the specified network port at runtime. The **EXPOSE** instruction defines metadata only; it does not make ports accessible from the host. The **-p** option in the **podman run** command exposes container ports from the host.
- ⑦ **ENV** is responsible for defining environment variables that are available in the container. You can declare multiple **ENV** instructions within the **Dockerfile**. You can use the **env** command inside the container to view each of the environment variables.
- ⑧ **ADD** instruction copies files or folders from a local or remote source and adds them to the container's file system. If used to copy local files, those must be in the working directory. **ADD** instruction unpacks local **.tar** files to the destination image directory.
- ⑨ **COPY** copies files from the working directory and adds them to the container's file system. It is not possible to copy a remote file using its URL with this Dockerfile instruction.
- ⑩ **USER** specifies the username or the UID to use when running the container image for the **RUN**, **CMD**, and **ENTRYPOINT** instructions. It is a good practice to define a different user other than **root** for security reasons.
- ⑪ **ENTRYPOINT** specifies the default command to execute when the image runs in a container. If omitted, the default **ENTRYPOINT** is **/bin/sh -c**.
- ⑫ **CMD** provides the default arguments for the **ENTRYPOINT** instruction. If the default **ENTRYPOINT** applies (**/bin/sh -c**), then **CMD** forms an executable command and parameters that run at container start.

CMD and ENTRYPOINT

ENTRYPOINT and **CMD** instructions have two formats:

- Exec form (using a JSON array):

```
ENTRYPOINT ["command", "param1", "param2"]
CMD ["param1", "param2"]
```

- Shell form:

```
ENTRYPOINT command param1 param2
CMD param1 param2
```

Exec form is the preferred form. Shell form wraps the commands in a `/bin/sh -c` shell, creating a sometimes unnecessary shell process. Also, some combinations are not allowed, or may not work as expected. For example, if **ENTRYPOINT** is `["ping"]` (exec form) and **CMD** is `localhost` (shell form), then the expected executed command is `ping localhost`, but the container tries `ping /bin/sh -c localhost`, which is a malformed command.

The **Dockerfile** should contain at most one **ENTRYPOINT** and one **CMD** instruction. If more than one of each is present, then only the last instruction takes effect. **CMD** can be present without specifying an **ENTRYPOINT**. In this case, the base image's **ENTRYPOINT** applies, or the default **ENTRYPOINT** if none is defined.

Podman can override the **CMD** instruction when starting a container. If present, all parameters for the **podman run** command after the image name form the **CMD** instruction. For example, the following instruction causes the running container to display the current time:

```
ENTRYPOINT ["/bin/date", "+%H:%M"]
```

The **ENTRYPOINT** defines both the command to be executed and the parameters. So the **CMD** instruction cannot be used. The following example provides the same functionality, with the added benefit of the **CMD** instruction being overwritable when a container starts:

```
ENTRYPOINT ["/bin/date"]
CMD ["+%H:%M"]
```

In both cases, when a container starts without providing a parameter, the current time is displayed:

```
[student@workstation ~]$ sudo podman run -it do180/rhel
11:41
```

In the second case, if a parameter appears after the image name in the **podman run** command, it overwrites the **CMD** instruction. The following command displays the current day of the week instead of the time:

```
[student@workstation demo-basic]$ sudo podman run -it do180/rhel +%
Tuesday
```

Another approach is using the default **ENTRYPOINT** and the **CMD** instruction to define the initial command. The following instruction displays the current time, with the added benefit of being able to be overridden at run time.

```
CMD ["date", "+%H:%M"]
```

ADD and COPY

The **ADD** and **COPY** instructions have two forms:

- The Shell form:

```
ADD <source>... <destination>
COPY <source>... <destination>
```

- The Exec form:

```
ADD [<source>, ... <destination>]
COPY [<source>, ... <destination>]
```

If the source is a file system path, it must be inside the working directory.

The **ADD** instruction also allows you to specify a resource using a URL:

```
ADD http://someserver.com/filename.pdf /var/www/html
```

If the source is a compressed file, then the **ADD** instruction decompresses the file to the *destination* folder. The **COPY** instruction does not have this functionality.



Warning

Both the **ADD** and **COPY** instructions copy the files, retaining permissions, with **root** as the owner, even if the **USER** instruction is specified. Red Hat recommends using a **RUN** instruction after the copy to change the owner and avoid "permission denied" errors.

Layering Image

Each instruction in a **Dockerfile** creates a new image layer. Having too many instructions in a **Dockerfile** causes too many layers, resulting in large images. For example, consider the following **RUN** instructions in a **Dockerfile**:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms"
RUN yum update -y
RUN yum install -y httpd
```

The previous example is not a good practice when creating container images. It creates three layers (one for each **RUN** instruction) while only the last is meaningful. Red Hat recommends minimizing the number of layers. You can achieve the same objective while creating a single layer by using the **&&** conjunction:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" && yum update -y && yum
install -y httpd
```

The problem with this approach is that the readability of the **Dockerfile** decays. Use the \ escape code to insert line breaks and improve readability. You can also indent lines to align the commands:

```
RUN yum --disablerepo=* --enablerepo="rhel-7-server-rpms" && \
yum update -y && \
yum install -y httpd
```

This example creates only one layer, and the readability improves. **RUN**, **COPY**, and **ADD** instructions create new image layers, but **RUN** can be improved this way.

Red Hat recommends applying similar formatting rules to other instructions accepting multiple parameters, such as **LABEL** and **ENV**:

```
LABEL version="2.0" \
      description="This is an example container image" \
      creationDate="01-09-2017"
```

```
ENV MYSQL_ROOT_PASSWORD="my_password" \
      MYSQL_DATABASE "my_database"
```

Building Images with Podman

The **podman build** command processes the **Dockerfile** and builds a new image based on the instructions it contains. The syntax for this command is as follows:

```
$ podman build -t NAME:TAG DIR
```

DIR is the path to the working directory, which must include the **Dockerfile**. It can be the current directory as designated by a dot (.) if the working directory is the current directory. *NAME:TAG* is a name with a tag given to the new image. If *TAG* is not specified, then the image is automatically tagged as **latest**.



References

Dockerfile Reference Guide

<https://docs.docker.com/engine/reference/builder/>

Creating base images

<https://docs.docker.com/engine/userguide/eng-image/baseimages/>

► Guided Exercise

Creating a Basic Apache Container Image

In this exercise, you will create a basic Apache container image.

Outcomes

You should be able to create a custom Apache container image built on a Red Hat Enterprise Linux 7.5 image.

Before You Begin

Run the following command to download the relevant lab files and to verify that Docker is running:

```
[student@workstation ~]$ lab dockerfile-create start
```

► 1. Create the Apache Dockerfile

- 1.1. Open a terminal on **workstation**. Use your preferred editor and create a new Dockerfile:

```
[student@workstation ~]$ vim /home/student/D0180/labs/dockerfile-create/Dockerfile
```

- 1.2. Use UBI 7.7 as a base image by adding the following **FROM** instruction at the top of the new Dockerfile:

```
FROM ubi7/ubi:7.7
```

- 1.3. Below the **FROM** instruction, include the **MAINTAINER** instruction to set the **Author** field in the new image. Replace the values to include your name and email address:

```
MAINTAINER Your Name <youremail>
```

- 1.4. Below the **MAINTAINER** instruction, add the following **LABEL** instruction to add description metadata to the new image:

```
LABEL description="A custom Apache container based on UBI 7"
```

- 1.5. Add a **RUN** instruction with a **yum install** command to install Apache on the new container:

```
RUN yum install -y httpd && \
    yum clean all
```

- 1.6. Add a **RUN** instruction to replace contents of the default HTTPD home page:

```
RUN echo "Hello from Dockerfile" > /var/www/html/index.html
```

- 1.7. Use the **EXPOSE** instruction below the **RUN** instruction to document the port that the container listens to at runtime. In this instance, set the port to 80, because it is the default for an Apache server:

```
EXPOSE 80
```

**Note**

The **EXPOSE** instruction does not actually make the specified port available to the host; rather, the instruction serves as metadata about which ports the container is listening on.

- 1.8. At the end of the file, use the following **CMD** instruction to set **httpd** as the default entry point:

```
CMD ["httpd", "-D", "FOREGROUND"]
```

- 1.9. Verify that your Dockerfile matches the following before saving and proceeding with the next steps:

```
FROM ubi7/ubi:7.7

MAINTAINER Your Name <youremail>

LABEL description="A custom Apache container based on UBI 7"

RUN yum install -y httpd && \
    yum clean all

RUN echo "Hello from Dockerfile" > /var/www/html/index.html

EXPOSE 80

CMD ["httpd", "-D", "FOREGROUND"]
```

► 2. Build and verify the Apache container image.

- 2.1. Use the following commands to create a basic Apache container image using the newly created Dockerfile:

```
[student@workstation ~]$ cd /home/student/D0180/labs/dockerfile-create
[student@workstation dockerfile-create]$ sudo podman build --layers=false \
> -t do180/apache .
STEP 1: FROM ubi7/ubi:7.7
Getting image source signatures ①
Copying blob sha256:...output omitted...
71.46 MB / 71.46 MB [=====] 18s
...output omitted...
```

```

Storing signatures
STEP 2: MAINTAINER Your Name <youremail>
STEP 3: LABEL description="A custom Apache container based on UBI 7"
STEP 4: RUN yum install -y httpd &&      yum clean all
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
...output omitted...
STEP 5: RUN echo "Hello from Dockerfile" > /var/www/html/index.html
STEP 6: EXPOSE 80
STEP 7: CMD ["httpd", "-D", "FOREGROUND"]
ERRO[0109] HOSTNAME is not supported for OCI image format...output omitted... ②
STEP 8: COMMIT ...output omitted... localhost/do180/apache:latest
Getting image source signatures
...output omitted...
Storing signatures
--> b49375fa8ee1e549dc1b72742532f01c13e0ad5b4a82bb088e5befbe59377bcf

```

- ① The container image listed in the **FROM** instruction is only downloaded if not already present in local storage.
- ② This error is benign, and can be ignored. It is a known issue of Podman (see Bug 1634806 [https://bugzilla.redhat.com/show_bug.cgi?id=1634806]) and will disappear in later versions.



Note

Podman creates many anonymous intermediate images during the build process. They are not be listed unless **-a** is used. Use the **--layers=false** option of the **build** subcommand to instruct Podman to delete intermediate images.

- 2.2. After the build process has finished, run **podman images** to see the new image in the image repository:

```
[student@workstation dockerfile-create]$ sudo podman images
REPOSITORY           TAG      IMAGE ID   CREATED       SIZE
localhost/do180/apache    latest   b49375fa8ee1  11 seconds ago  247MB
registry.access.redhat.com/ubi7/ubi    7.7     0355cd652bd1  8 months ago   215MB
```

3. Run the Apache container.

- 3.1. Use the following command to run a container using the Apache image:

```
[student@workstation dockerfile-create]$ sudo podman run --name lab-apache \
> -d -p 10080:80 do180/apache
fa1d1c450e8892ae085dd8bbf763edac92c41e6ffaa7ad6ec6388466809bb391
```

- 3.2. Run the **podman ps** command to see the running container:

```
[student@workstation dockerfile-create]$ sudo podman ps
CONTAINER ID IMAGE           COMMAND      ...output omitted...
fa1d1c450e888 localhost/do180/apache:latest httpd -D FOREGROU...output omitted...
```

- 3.3. Use the **curl** command to verify that the server is running:

```
[student@workstation dockerfile-create]$ curl 127.0.0.1:10080
Hello from Dockerfile
```

Finish

On **workstation**, run the **lab dockerfile-create finish** script to complete this lab.

```
[student@workstation ~]$ lab dockerfile-create finish
```

This concludes the guided exercise.

▶ Lab

Creating Custom Container Images

Performance Checklist

In this lab, you will create a Dockerfile to build a custom Apache Web Server container image. The custom image will be based on a RHEL 7.7 UBI image and serve a custom **index.html** page.

Outcomes

You should be able to create a custom Apache Web Server container that hosts static HTML files.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab dockerfile-review start
```

- Review the provided Dockerfile stub in the **/home/student/D0180/labs/dockerfile-review/** folder. Edit the **Dockerfile** and ensure that it meets the following specifications:
 - The base image is **ubi7/ubi:7.7**
 - Sets the desired author name and email ID with the **MAINTAINER** instruction
 - Sets the environment variable **PORT** to 8080
 - Install Apache (*httpd* package).
 - Change the Apache configuration file **/etc/httpd/conf/httpd.conf** to listen to port 8080 instead of the default port 80.
 - Change ownership of the **/etc/httpd/logs** and **/run/httpd** folders to user and group **apache** (UID and GID are 48).
 - So that container users know how to access the Apache Web Server, expose the value set in the **PORT** environment variable.
 - Copy the contents of the **src/** folder in the lab directory to the Apache **DocumentRoot** file (**/var/www/html/**) inside the container.

The **src** folder contains a single **index.html** file that prints a **Hello World!** message.

- Start the Apache **httpd** daemon in the foreground using the following command:

```
httpd -D FOREGROUND
```

- Build the custom Apache image with the name **d0180/custom-apache**.
- Create a new container in detached mode with the following characteristics:

- Name: **dockerfile**
- Container image: **do180/custom-apache**
- Port forward: from host port 20080 to container port 8080
- Run as a daemon: yes

Verify that the container is ready and running.

4. Verify that the server is serving the HTML file.

Evaluation

Grade your work by running the **lab dockerfile-review grade** command from your **workstation** machine. Correct any reported failures and rerun the script until successful.

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

Finish

From **workstation**, run the **lab dockerfile-review finish** command to complete this lab.

```
[student@workstation ~]$ lab dockerfile-review finish
```

This concludes the lab.

► Solution

Creating Custom Container Images

Performance Checklist

In this lab, you will create a Dockerfile to build a custom Apache Web Server container image. The custom image will be based on a RHEL 7.7 UBI image and serve a custom **index.html** page.

Outcomes

You should be able to create a custom Apache Web Server container that hosts static HTML files.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab dockerfile-review start
```

1. Review the provided Dockerfile stub in the **/home/student/D0180/labs/dockerfile-review/** folder. Edit the **Dockerfile** and ensure that it meets the following specifications:
 - The base image is **ubi7/ubi:7.7**
 - Sets the desired author name and email ID with the **MAINTAINER** instruction
 - Sets the environment variable **PORT** to 8080
 - Install Apache (*httpd* package).
 - Change the Apache configuration file **/etc/httpd/conf/httpd.conf** to listen to port 8080 instead of the default port 80.
 - Change ownership of the **/etc/httpd/logs** and **/run/httpd** folders to user and group **apache** (UID and GID are 48).
 - So that container users know how to access the Apache Web Server, expose the value set in the **PORT** environment variable.
 - Copy the contents of the **src/** folder in the lab directory to the Apache **DocumentRoot** file (**/var/www/html/**) inside the container.

The **src** folder contains a single **index.html** file that prints a **Hello World!** message.

- Start the Apache **httpd** daemon in the foreground using the following command:

```
httpd -D FOREGROUND
```

- 1.1. Use your preferred editor to modify the Dockerfile located in the **/home/student/D0180/labs/dockerfile-review/** folder.

```
[student@workstation ~]$ cd /home/student/D0180/labs/dockerfile-review/
[student@workstation dockerfile-review]$ vim Dockerfile
```

- 1.2. Set the base image for the Dockerfile to **ubi7/ubi:7.7**.

```
FROM ubi7/ubi:7.7
```

- 1.3. Set your name and email with a **MAINTAINER** instruction.

```
MAINTAINER Your Name <youremail>
```

- 1.4. Create an environment variable called **PORT** and set it to 8080.

```
ENV PORT 8080
```

- 1.5. Install Apache server.

```
RUN yum install -y httpd && \
    yum clean all
```

- 1.6. Change the Apache HTTP Server configuration file to listen to port 8080 and change ownership of the server working folders with a single **RUN** instruction.

```
RUN sed -ri -e "/^Listen 80/c\Listen ${PORT}" /etc/httpd/conf/httpd.conf && \
    chown -R apache:apache /etc/httpd/logs/ && \
    chown -R apache:apache /run/httpd/
```

- 1.7. Use the **USER** instruction to run the container as the **apache** user. Use the **EXPOSE** instruction to document the port that the container listens to at runtime. In this instance, set the port to the **PORT** environment variable, which is the default for an Apache server.

```
USER apache

# Expose the custom port that you provided in the ENV var
EXPOSE ${PORT}
```

- 1.8. Copy all the files from the **src** folder to the Apache **DocumentRoot** path at **/var/www/html**.

```
# Copy all files under src/ folder to Apache DocumentRoot (/var/www/html)
COPY ./src/ /var/www/html/
```

- 1.9. Finally, insert a **CMD** instruction to run **httpd** in the foreground, and then save the Dockerfile.

```
# Start Apache in the foreground
CMD ["httpd", "-D", "FOREGROUND"]
```

2. Build the custom Apache image with the name **do180/custom-apache**.

2.1. Verify the Dockerfile for the custom Apache image.

The Dockerfile for the custom Apache image should look like the following:

```
FROM ubi7/ubi:7.7

MAINTAINER Your Name <youremail>

ENV PORT 8080

RUN yum install -y httpd && \
    yum clean all

RUN sed -ri -e "/^Listen 80/c\Listen ${PORT}" /etc/httpd/conf/httpd.conf && \
    chown -R apache:apache /etc/httpd/logs/ && \
    chown -R apache:apache /run/httpd/

USER apache

# Expose the custom port that you provided in the ENV var
EXPOSE ${PORT}

# Copy all files under src/ folder to Apache DocumentRoot (/var/www/html)
COPY ./src/ /var/www/html/

# Start Apache in the foreground
CMD ["httpd", "-D", "FOREGROUND"]
```

2.2. Run a **sudo podman build** command to build the custom Apache image and name it **do180/custom-apache**.

```
[student@workstation dockerfile-review]$ sudo podman build --layers=false \
> -t do180/custom-apache .
STEP 1: FROM ubi7/ubi:7.7
...output omitted...
STEP 2: MAINTAINER username <username@example.com>
STEP 3: ENV PORT 8080
STEP 4: RUN yum install -y httpd &&      yum clean all
...output omitted...
STEP 5: RUN sed -ri -e "/^Listen 80/c\Listen ${PORT}" /etc/httpd/conf/httpd.conf \
&&      chown -R apache:apache /etc/httpd/logs/ &&      chown -R apache:apache / \
run/httpd/
STEP 6: USER apache
STEP 7: EXPOSE ${PORT}
STEP 8: COPY ./src/ /var/www/html/
STEP 9: CMD ["httpd", "-D", "FOREGROUND"]
ERRO[0109] HOSTNAME is not supported for OCI image format...output omitted...
STEP 10: COMMIT ...output omitted... localhost/do180/custom-apache:latest
...output omitted...
```

2.3. Run the **podman images** command to verify that the custom image is built successfully.

```
[student@workstation dockerfile-review]$ sudo podman images
REPOSITORY                                     TAG      IMAGE ID      ...
localhost/do180/custom-apache                 latest   da92b9426325 ...
registry.access.redhat.com/ubi7/ubi           7.7     6fcccc91c83 ...
```

3. Create a new container in detached mode with the following characteristics:

- Name: **dockerfile**
- Container image: **do180/custom-apache**
- Port forward: from host port 20080 to container port 8080
- Run as a daemon: yes

Verify that the container is ready and running.

- 3.1. Create and run the container.

```
[student@workstation dockerfile-review]$ sudo podman run -d \
> --name dockerfile -p 20080:8080 do180/custom-apache
367823e35c4a...
```

- 3.2. Verify that the container is ready and running.

```
[student@workstation dockerfile-review]$ sudo podman ps
... IMAGE          COMMAND          ... PORTS          NAMES
... do180/custom... "httpd -D ..." ... 0.0.0.0:20080->8080/tcp dockerfile
```

4. Verify that the server is serving the HTML file.

Run a **curl** command on 127.0.0.1:20080

```
[student@workstation dockerfile-review]$ curl 127.0.0.1:20080
```

The output should be as follows:

```
<html>
<header><title>DO180 Hello!</title></header>
<body>
  Hello World! The dockerfile-review lab works!
</body>
</html>
```

Evaluation

Grade your work by running the **lab dockerfile-review grade** command from your **workstation** machine. Correct any reported failures and rerun the script until successful.

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

Finish

From **workstation**, run the **lab dockerfile-review finish** command to complete this lab.

```
[student@workstation ~]$ lab dockerfile-review finish
```

This concludes the lab.

Summary

In this chapter, you learned:

- A **Dockerfile** contains instructions that specify how to construct a container image.
- Container images provided by Red Hat Container Catalog or Quay.io are a good starting point for creating custom images for a specific language or technology.
- Building an image from a Dockerfile is a three-step process:
 1. Create a working directory.
 2. Specify the build instructions in a **Dockerfile** file.
 3. Build the image with the **podman build** command.
- The Source-to-Image (S2I) process provides an alternative to Dockerfiles. S2I implements a standardized container image build process for common technologies from application source code. This allows developers to focus on application development and not Dockerfile development.

Chapter 6

Deploying Containerized Applications on OpenShift

Goal

Deploy single container applications on OpenShift Container Platform.

Objectives

- Describe the architecture of Kubernetes and Red Hat OpenShift Container Platform.
- Create standard Kubernetes resources.
- Create a route to a service.
- Build an application using the Source-to-Image facility of OpenShift Container Platform.
- Create an application using the OpenShift web console.

Sections

- Creating Kubernetes Resources (and Guided Exercise)
- Creating Routes (and Guided Exercise)
- Creating Applications with Source-to-Image (and Guided Exercise)

Lab

Deploying Containerized Applications on OpenShift

Creating Kubernetes Resources

Objectives

After completing this section, students should be able to create standard Kubernetes resources.

The Red Hat OpenShift Container Platform (RHOC) Command-line Tool

The main method of interacting with an RHOC cluster is using the **oc** command. The basic usage of the command is through its subcommands in the following syntax:

```
$> oc <command>
```

Before interacting with a cluster, most operations require a logged-in user. The syntax to log in is shown below:

```
$> oc login <clusterUrl>
```

Describing Pod Resource Definition Syntax

RHOC runs containers inside Kubernetes pods, and to create a pod from a container image, OpenShift needs a *pod resource definition*. This can be provided either as a JSON or YAML text file, or can be generated from defaults by the **oc new-app** command or the OpenShift web console.

A pod is a collection of containers and other resources. An example of a WildFly application server pod definition in YAML format is shown below:

```
apiVersion: v1
kind: Pod1
metadata:
  name: wildfly2
  labels:
    name: wildfly3
spec:
  containers:
    - resources:
        limits:
          cpu: 0.5
      image: do276/todojee
      name: wildfly
      ports:
        - containerPort: 80804
          name: wildfly
    env:5
      - name: MYSQL_ENV_MYSQL_DATABASE
        value: items
```

```

- name: MYSQL_ENV_MYSQL_USER
  value: user1
- name: MYSQL_ENV_MYSQL_PASSWORD
  value: mypa55

```

- ➊ Declares a Kubernetes pod resource type.
- ➋ A unique name for a pod in Kubernetes that allows administrators to run commands on it.
- ➌ Creates a label with a key named **name** that other resources in Kubernetes, usually as service, can use to find it.
- ➍ A container-dependent attribute identifying which port on the container is exposed.
- ➎ Defines a collection of environment variables.

Some pods may require environment variables that can be read by a container. Kubernetes transforms all the **name** and **value** pairs to environment variables. For instance, the **MYSQL_ENV_MYSQL_USER** variable is declared internally by the Kubernetes runtime with a value of **user1**, and is forwarded to the container image definition. Because the container uses the same variable name to get the user's login, the value is used by the WildFly container instance to set the username that accesses a MySQL database instance.

Describing Service Resource Definition Syntax

Kubernetes provides a virtual network that allows pods from different workers to connect. But, Kubernetes provides no easy way for a pod to discover the IP addresses of other pods.

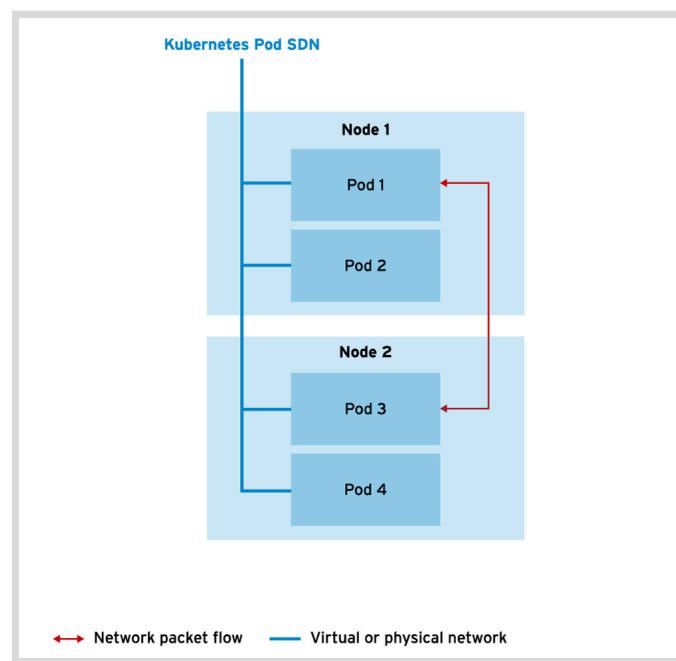


Figure 6.1: Basic Kubernetes networking

Services are essential resources to any OpenShift application. They allow containers in one pod to open network connections to containers in another pod. A pod can be restarted for many reasons, and it gets a different internal IP address each time. Instead of a pod having to discover the IP address of another pod after each restart, a service provides a stable IP address for other pods to use, no matter what worker node runs the pod after each restart.

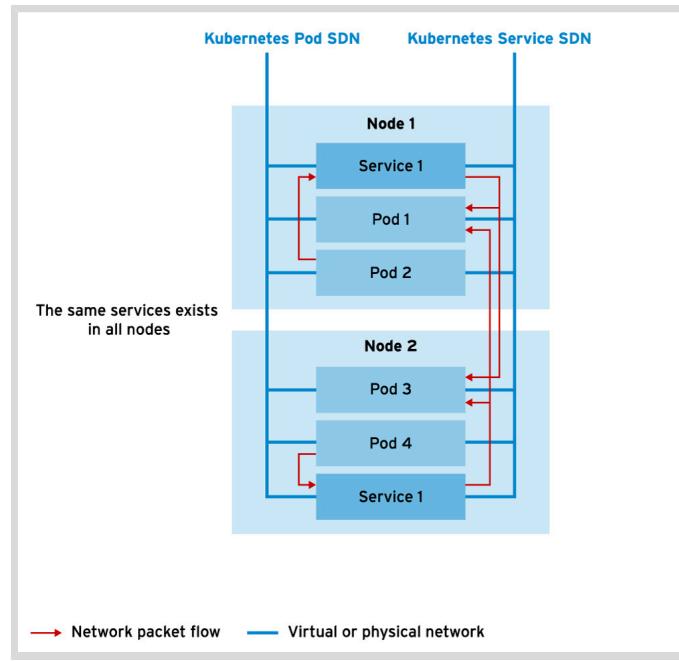


Figure 6.2: Kubernetes services networking

Most real-world applications do not run as a single pod. They need to scale horizontally, so many pods run the same containers from the same pod resource definition to meet growing user demand. A service is tied to a set of pods, providing a single IP address for the whole set, and a load-balancing client request among member pods.

The set of pods running behind a service is managed by a *DeploymentConfig* resource. A *DeploymentConfig* resource embeds a *ReplicationController* that manages how many pod copies (replicas) have to be created, and creates new ones if any of them fail. *DeploymentConfig* and *ReplicationController* resources are explained later in this chapter.

The following example shows a minimal service definition in JSON syntax:

```
{
  "kind": "Service", ①
  "apiVersion": "v1",
  "metadata": {
    "name": "quotedb" ②
  },
  "spec": {
    "ports": [ ③
      {
        "port": 3306,
        "targetPort": 3306
      }
    ],
    "selector": {
      "name": "mysqldb" ④
    }
  }
}
```

① The kind of Kubernetes resource. In this case, a Service.

- ② A unique name for the service.
- ③ **ports** is an array of objects that describes network ports exposed by the service. The **targetPort** attribute has to match a **containerPort** from a pod container definition, and the **port** attribute is the port that is exposed by the service. Clients connect to the service port and the service forwards packets to the pod **targetPort**.
- ④ **selector** is how the service finds pods to forward packets to. The target pods need to have matching labels in their metadata attributes. If the service finds multiple pods with matching labels, it load balances network connections between them.

Each service is assigned a unique IP address for clients to connect to. This IP address comes from another internal OpenShift SDN, distinct from the pods' internal network, but visible only to pods. Each pod matching the **selector** is added to the service resource as an endpoint.

Discovering Services

An application typically finds a service IP address and port by using environment variables. For each service inside an OpenShift project, the following environment variables are automatically defined and injected into containers for all pods inside the same project:

- **SVC_NAME_SERVICE_HOST** is the service IP address.
- **SVC_NAME_SERVICE_PORT** is the service TCP port.



Note

The SVC_NAME part of the variable is changed to comply with DNS naming restrictions: letters are capitalized and underscores (_) are replaced by dashes (-).

Another way to discover a service from a pod is by using the OpenShift internal DNS server, which is visible only to pods. Each service is dynamically assigned an SRV record with an FQDN of the form:

SVC_NAME.PROJECT_NAME.svc.cluster.local

When discovering services using environment variables, a pod has to be created and started only after the service is created. If the application was written to discover services using DNS queries, however, it can find services created after the pod was started.

There are two ways for an application to access the service from outside an OpenShift cluster:

1. **NodePort** type: This is an older Kubernetes-based approach, where the service is exposed to external clients by binding to available ports on the worker node host, which then proxies connections to the service IP address. Use the **oc edit svc** command to edit service attributes and specify **NodePort** as the value for **type**, and provide a port value for the **nodePort** attribute. OpenShift then proxies connections to the service via the public IP address of the worker node host and the port value set in **nodePort**.
2. OpenShift Routes: This is the preferred approach in OpenShift to expose services using a unique URL. Use the **oc expose** command to expose a service for external access or expose a service from the OpenShift web console.

Figure 6.3 illustrates how **NodePort** services allow external access to Kubernetes services. OpenShift routes are covered in more detail later in this course.

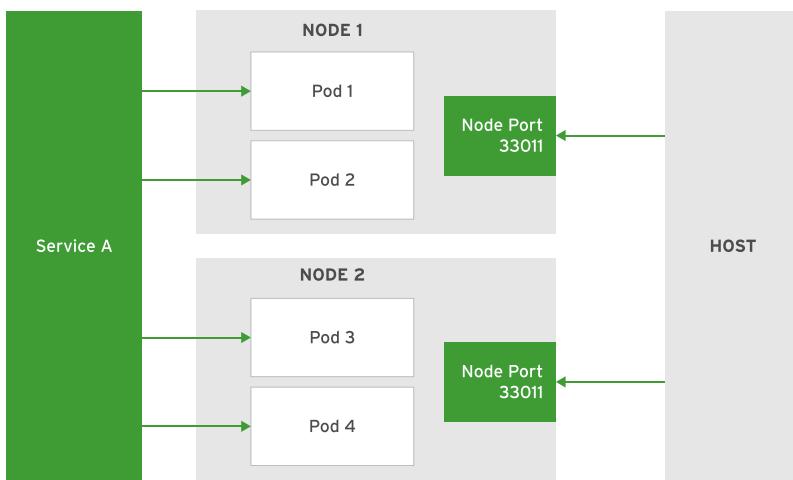


Figure 6.3: Alternative method for external access to a Kubernetes service

OpenShift provides the **oc port-forward** command for forwarding a local port to a pod port. This is different from having access to a pod through a service resource:

- The port-forwarding mapping exists only on the workstation where the **oc** client runs, while a service maps a port for all network users.
- A service load-balances connections to potentially multiple pods, whereas a port-forwarding mapping forwards connections to a single pod.



Note

Red Hat discourages the use of the **NodePort** approach to avoid exposing the service to direct connections. Mapping via port-forwarding in OpenShift is considered a more secure alternative.

The following example demonstrates the use of the **oc port-forward** command:

```
[student@workstation ~]$ oc port-forward mysql-openshift-1-glqrp 3306:3306
```

The previous command forwards port 3306 from the developer machine to port 3306 on the **db** pod, where a MySQL server (inside a container) accepts network connections.



Note

When running this command, make sure you leave the terminal window running. Closing the window or canceling the process stops the port mapping.

Creating New Applications

Simple applications, complex multtier applications, and microservice applications can be described by a single resource definition file. This single file would contain many pod definitions, service definitions to connect the pods, replication controllers or DeploymentConfigs to horizontally scale the application pods, PersistentVolumeClaims to persist application data, and anything else needed that can be managed by OpenShift.



Important

In OpenShift 4.5 the **oc new-app** command now produces Deployment resources instead of DeploymentConfig resources by default. This version of DO180 does not cover the Deployment resource, only DeploymentConfigs. To create DeploymentConfig resources, you can pass the **--as-deployment-config** flag when invoking **oc new-app**, which is done throughout this version of the course. For more information, see Understanding Deployments and DeploymentConfigs [<https://docs.openshift.com/container-platform/4.5/applications/deployments/what-deployments-are.html#what-deployments-are>].

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

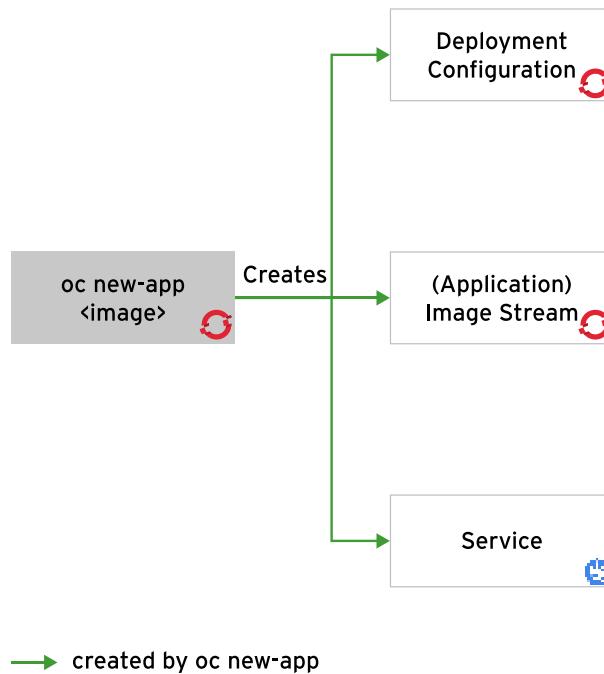
The **oc new-app** command can create application pods to run on OpenShift in many different ways. It can create pods from existing docker images, from Dockerfiles, and from raw source code using the Source-to-Image (S2I) process.

Run the **oc new-app -h** command to understand all the different options available for creating new applications on OpenShift.

The following command creates an application based on an image, **mysql**, from Docker Hub, with the label set to **db=mysql**:

```
[student@workstation ~]$ oc new-app mysql --as-deployment-config \
> MYSQL_USER=user MYSQL_PASSWORD=pass MYSQL_DATABASE=testdb -l db=mysql
```

The following figure shows the Kubernetes and OpenShift resources created by the **oc new-app** command when the argument is a container image:

**Figure 6.4: Resources created for a new application**

The following command creates an application based on an image from a private Docker image registry:

```
oc new-app --docker-image=myregistry.com/mycompany/myapp --name=myapp --as-deployment-config
```

The following command creates an application based on source code stored in a Git repository:

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

You will learn more about the Source-to-Image (S2I) process, its associated concepts, and more advanced ways to use `oc new-app` to build applications for OpenShift in the next section.

Managing OpenShift Resources at the Command Line

There are several essential commands used to manage OpenShift resources as described below.

Use the `oc get` command to retrieve information about resources in the cluster. Generally, this command outputs only the most important characteristics of the resources and omits more detailed information.

The `oc get RESOURCE_TYPE` command displays a summary of all resources of the specified type. The following illustrates example output of the `oc get pods` command.

NAME	READY	STATUS	RESTARTS	AGE
nginx-1-5r583	1/1	Running	0	1h
myapp-1-l44m7	1/1	Running	0	1h

oc get all

Use the **oc get all** command to retrieve a summary of the most important components of a cluster. This command iterates through the major resource types for the current project and prints out a summary of their information.

NAME	DOCKER REPO	TAGS	UPDATED	
is/nginx	172.30.1.1:5000/basic-kubernetes/nginx	latest	About an hour ago	
NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY
dc/nginx	1	1	1	config,image(nginx:latest)
NAME	DESIRED	CURRENT	READY	AGE
rc/nginx-1	1	1	1	1h
NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/nginx	172.30.72.75	<none>	80/TCP, 443/TCP	1h
NAME	READY	STATUS	RESTARTS	AGE
po/nginx-1-ypp8t	1/1	Running	0	1h

oc describe RESOURCE_TYPE RESOURCE_NAME

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

```
Name: mysql-openshift-1-glqrp
Namespace: mysql-openshift
Priority: 0
PriorityClassName: none
Node: cluster-worker-1/172.25.250.52
Start Time: Fri, 15 Feb 2019 02:14:34 +0000
Labels: app=mysql-openshift
        deployment=mysql-openshift-1
        deploymentconfig=mysql-openshift
Annotations: openshift.io/deployment-config.latest-version: 1
            openshift.io/deployment-config.name: mysql-openshift
            openshift.io/deployment.name: mysql-openshift-1
            openshift.io/generated-by: OpenShiftNewApp
            openshift.io/scc: restricted
Status: Running
IP: 10.129.0.85
```

oc get RESOURCE_TYPE RESOURCE_NAME -o yaml

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

oc create

This command creates resources from a resource definition. Typically, this is paired with the **oc get RESOURCE_TYPE RESOURCE_NAME -o yaml** command for editing definitions.

oc edit

This command allows the user to edit resources of a resource definition. By default, this command opens a **vi** buffer for editing the resource definition.

oc delete RESOURCE_TYPE name

The **oc delete** command removes a resource from an OpenShift cluster. Note that a fundamental understanding of the OpenShift architecture is needed here, because deleting managed resources such as pods results in new instances of those resources being automatically created. When a project is deleted, it deletes all of the resources and applications contained within it.

oc exec CONTAINER_ID options command

The **oc exec** command executes commands inside a container. You can use this command to run interactive and noninteractive batch commands as part of a script.

Labelling resources

When working with many resources in the same project, it is often useful to group those resources by application, environment, or some other criteria. To establish these groups, you define labels for the resources in your project. Labels are part of the **metadata** section of a resource, and are defined as key/value pairs, as shown in the following example:

```
apiVersion: v1
kind: Service
metadata:
...contents omitted...
labels:
  app: nexus
  template: nexus-persistent-template
  name: nexus
...contents omitted...
```

Many **oc** subcommands support a **-l** option to process resources from a label specification. For the **oc get** command, the **-l** option acts as a selector to only retrieve objects that have a matching label:

\$ oc get svc,dc -l app=nexus						
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
service/nexus	ClusterIP	172.30.29.218	<none>	8081/TCP	4h	
NAME			REVISION	DESIRED	CURRENT	...
deploymentconfig.apps.openshift.io/nexus			1	1	1	...

**Note**

Although any label can appear in resources, both the **app** and **template** keys are common for labels. By convention, the **app** key indicates the application related to this resource. The **template** key labels all resources generated by the same template with the template's name.

When using templates to generate resources, labels are especially useful. A template resource has a **labels** section separated from the **metadata.labels** section. Labels defined in the **labels** section do not apply to the template itself, but are added to every resource generated by the template.

```
apiVersion: template.openshift.io/v1
kind: Template
labels:
  app: nexus
  template: nexus-persistent-template
metadata:
  ...contents omitted...
labels:
  maintainer: redhat
  name: nexus-persistent
  ...contents omitted...
objects:
- apiVersion: v1
  kind: Service
  metadata:
    name: nexus
  labels:
    version: 1
  ...contents omitted...
```

The previous example defines a template resource with a single label: **maintainer: redhat**. The template generates a service resource with three labels: **app: nexus**, **template: nexus-persistent-template**, and **version: 1**.



References

Additional information about pods and services is available in the *Pods and Services* section of the OpenShift Container Platform documentation:

Architecture

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/architecture/index

Additional information about creating images is available in the OpenShift Container Platform documentation:

Creating Images

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/images/index

Labels and label selectors details are available in *Working with Kubernetes Objects* section for the Kubernetes documentation:

Labels and Selectors

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

► Guided Exercise

Deploying a Database Server on OpenShift

In this exercise, you will create and deploy a MySQL database pod on OpenShift using the **oc new-app** command.

Outcomes

You should be able to create and deploy a MySQL database pod on OpenShift.

Before You Begin

On **workstation**, run the following command to set up the environment:

```
[student@workstation ~]$ lab openshift-resources start
```

► 1. Prepare the lab environment.

- 1.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

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

- 1.2. Log in to the OpenShift cluster.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 1.3. Create a new project that contains your RHOCP developer username for the resources you create during this exercise:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-mysql-openshift
Now using project ...output omitted...
```

► 2. Create a new application from the **rhscl/mysql-57-rhel7** container image using the **oc new-app** command.

This image requires that you use the **-e** option to set the **MYSQL_USER**, **MYSQL_PASSWORD**, **MYSQL_DATABASE**, and **MYSQL_ROOT_PASSWORD** environment variables.

Use the **--docker-image** option with the **oc new-app** command to specify the classroom private registry URI so that OpenShift does not try and pull the image from the internet:

```
[student@workstation ~]$ oc new-app --as-deployment-config \
> --docker-image=registry.access.redhat.com/rhscl/mysql-57-rhel7:latest \
> --name=mysqlOpenshift \
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 -e MYSQL_DATABASE=testdb \
> -e MYSQL_ROOT_PASSWORD=r00tpa55
--> Found Docker image b48e700 (5 weeks old) from registry.access.redhat.com for
"registry.access.redhat.com/rhscl/mysql-57-rhel7:latest"
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "mysqlOpenshift" created
deploymentconfig.apps.openshift.io "mysqlOpenshift" created
service "mysqlOpenshift" 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/mysqlOpenshift'
Run 'oc status' to view your app.
```

- ▶ 3. Verify that the MySQL pod was created successfully and view the details about the pod and its service.
- 3.1. Run the **oc status** command to view the status of the new application and verify that the deployment of the MySQL image was successful:

```
[student@workstation ~]$ oc status
In project ${RHT_OCP4_DEV_USER}-mysqlOpenshift on server ...

svc/mysqlOpenshift - 172.30.114.39:3306
dc/mysqlOpenshift deploys istag/mysqlOpenshift:latest
    deployment #1 running for 11 seconds - 0/1 pods
...output omitted...
```

- 3.2. List the pods in this project to verify that the MySQL pod is ready and running:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysqlOpenshift-1-deploy  0/1     Completed   0          111s
mysqlOpenshift-1-xg665   1/1     Running    0          109s
```



Note

Notice the name of the running pod. You need this information to be able to log in to the MySQL database server later.

- 3.3. Use the **oc describe** command to view more details about the pod:

```
[student@workstation ~]$ oc describe pod mysqlOpenshift-1-xg665
Name:         mysqlOpenshift-1-xg665
Namespace:    ${RHT_OCP4_DEV_USER}-mysqlOpenshift
Priority:    0
Node:        master01/192.168.50.10
```

```

Start Time: Fri, 13 Nov 2020 08:50:03 -0500
Labels: deployment=mysql-openshift-1
        deploymentconfig=mysql-openshift
...output omitted...
Status: Running
IP: 10.10.0.34
...output omitted...

```

- 3.4. List the services in this project and verify that the service to access the MySQL pod was created:

```
[student@workstation ~]$ oc get svc
NAME            TYPE      CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
mysql-openshift ClusterIP  172.30.121.55 <none>       3306/TCP   10m
```

- 3.5. Retrieve the details of the **mysql-openshift** service using the **oc describe** command and note that the service type is **ClusterIP** by default:

```
[student@workstation ~]$ oc describe service mysql-openshift
Name: mysql-openshift
Namespace: ${RHT_OCP4_DEV_USER}-mysql-openshift
Labels: app=mysql-openshift
        app.kubernetes.io/component=mysql-openshift
        app.kubernetes.io/instance=mysql-openshift
Annotations: openshift.io/generated-by: OpenShiftNewApp
Selector: deploymentconfig=mysql-openshift
Type: ClusterIP
IP: 172.30.121.55
Port: 3306-tcp 3306/TCP
TargetPort: 3306/TCP
Endpoints: 10.10.0.34:3306
Session Affinity: None
Events: <none>
```

- 3.6. View details about the deployment configuration (**dc**) for this application:

```
[student@workstation ~]$ oc describe dc mysql-openshift
Name: mysql-openshift
Namespace: ${RHT_OCP4_DEV_USER}-mysql-openshift
Created: 15 minutes ago
Labels: app=mysql-openshift
        app.kubernetes.io/component=mysql-openshift
        app.kubernetes.io/instance=mysql-openshift
...output omitted...
Deployment #1 (latest):
  Name: mysql-openshift-1
  Created: 15 minutes ago
  Status: Complete
  Replicas: 1 current / 1 desired
  Selector: deployment=mysql-openshift-1,deploymentconfig=mysql-openshift
```

```
Labels:           app.kubernetes.io/component=mysqlOpenshift, app.kubernetes.io/
instance=mysqlOpenshift, app=mysqlOpenshift, openshift.io/deployment-
config.name=mysqlOpenshift
Pods Status:    1 Running / 0 Waiting / 0 Succeeded / 0 Failed
...output omitted...
```

- 3.7. Expose the service creating a route with a default name and a fully qualified domain name (FQDN):

```
[student@workstation ~]$ oc expose service mysqlOpenshift
route.route.openshift.io/mysqlOpenshift exposed
[student@workstation ~]$ oc get routes
NAME            HOST/PORT          ... PORT
mysqlOpenshift  mysqlOpenshift-$RHT_OCP4_DEV_USER}-mysql...  ... 3306-tcp
```

- ▶ 4. Connect to the MySQL database server and verify that the database was created successfully.
- 4.1. From the **workstation** machine, configure port forwarding between **workstation** and the database pod running on OpenShift using port 3306. The terminal will hang after executing the command.

```
[student@workstation ~]$ oc port-forward mysqlOpenshift-1-xg665 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
```

- 4.2. From the **workstation** machine open another terminal and connect to the MySQL server using the MySQL client.

```
[student@workstation ~]$ mysql -uuser1 -pmypa55 --protocol tcp -h localhost
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

- 4.3. Verify the creation of the **testdb** database.

```
MySQL [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| testdb        |
+-----+
2 rows in set (0.00 sec)
```

- 4.4. Exit from the MySQL prompt:

```
MySQL [(none)]> exit  
Bye
```

Close the terminal and return to the previous one. Finish the port forwarding process by pressing **Ctrl+C**.

```
Forwarding from 127.0.0.1:3306 -> 3306  
Forwarding from [::1]:3306 -> 3306  
Handling connection for 3306  
^C
```

- 5. Delete the project to remove all the resources within the project:

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-mysql-openshift
```

Finish

On **workstation**, run the **lab openshift-resources finish** script to complete this lab.

```
[student@workstation ~]$ lab openshift-resources finish
```

This concludes the exercise.

Creating Routes

Objectives

After completing this section, students should be able to expose services using OpenShift routes.

Working with Routes

Services allow for network access between pods inside an OpenShift instance, and routes allow for network access to pods from users and applications outside the OpenShift instance.

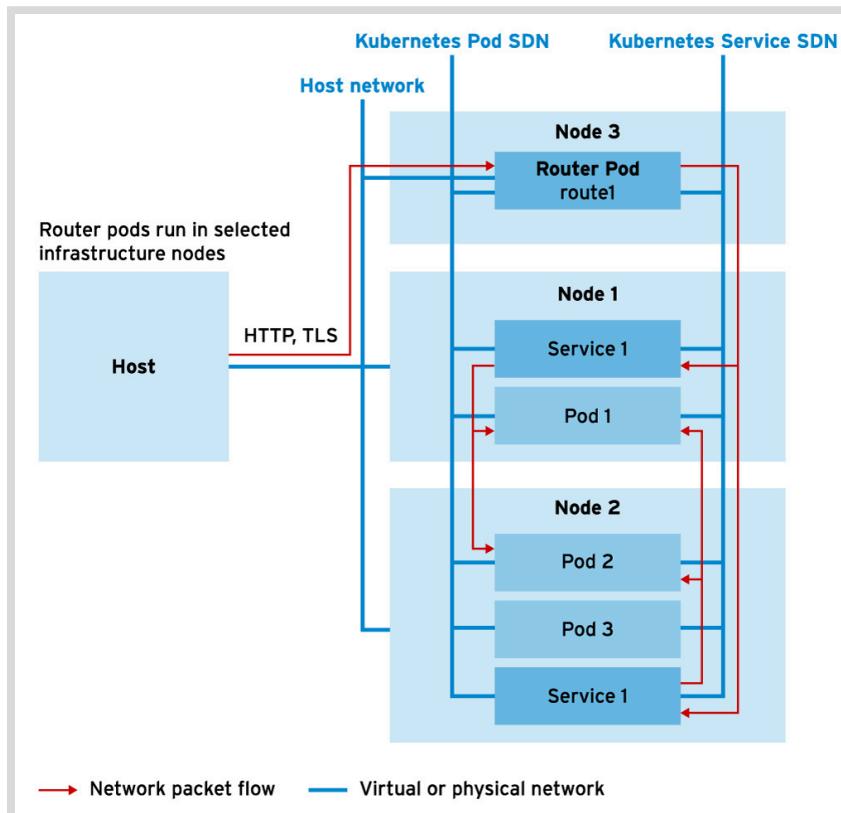


Figure 6.5: OpenShift routes and Kubernetes services

A route connects a public-facing IP address and DNS host name to an internal-facing service IP. It uses the service resource to find the endpoints; that is, the ports exposed by the service.

OpenShift routes are implemented by a cluster-wide router service, which runs as a containerized application in the OpenShift cluster. OpenShift scales and replicates router pods like any other OpenShift application.



Note

In practice, to improve performance and reduce latency, the OpenShift router connects directly to the pods using the internal pod software-defined network (SDN).

The router service uses *HAProxy* as the default implementation.

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

The following example shows a minimal route defined using JSON syntax:

```
{  
    "apiVersion": "v1",  
    "kind": "Route",  
    "metadata": {  
        "name": "quoteapp"  
    },  
    "spec": {  
        "host": "quoteapp.apps.example.com",  
        "to": {  
            "kind": "Service",  
            "name": "quoteapp"  
        }  
    }  
}
```

The **apiVersion**, **kind**, and **metadata** attributes follow standard Kubernetes resource definition rules. The **Route** value for **kind** shows that this is a route resource, and the **metadata.name** attribute gives this particular route the identifier **quoteapp**.

As with pods and services, the main part is the **spec** attribute, which is an object containing the following attributes:

- **host** is a string containing the FQDN associated with the route. DNS must resolve this FQDN to the IP address of the OpenShift router. The details to modify DNS configuration are outside the scope of this course.
- **to** is an object stating the resource this route points to. In this case, the route points to an OpenShift Service with the **name** set to **quoteapp**.



Note

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



Important

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

Creating Routes

Use the **oc create** command to create route resources, just like any other OpenShift resource. You must provide a JSON or YAML resource definition file, which defines the route, to the **oc create** command.

The **oc new-app** command does not create a route resource when building a pod from container images, Dockerfiles, or application source code. After all, **oc new-app** does not know if the pod is intended to be accessible from outside the OpenShift instance or not.

Another way to create a route is to use the **oc expose service** command, passing a service resource name as the input. The **--name** option can be used to control the name of the route resource. For example:

```
$ oc expose service quotedb --name quote
```

By default, routes created by **oc expose** generate DNS names of the form:

route-name-project-name.default-domain

Where:

- *route-name* is the name assigned to the route. If no explicit name is set, OpenShift assigns the route the same name as the originating resource (for example, the service name).
- *project-name* is the name of the project containing the resource.
- *default-domain* is configured on the OpenShift master and corresponds to the wildcard DNS domain listed as a prerequisite for installing OpenShift.

For example, creating a route named **quote** in project named **test** from an OpenShift instance where the wildcard domain is **cloudapps.example.com** results in the FQDN **quote-test.cloudapps.example.com**.



Note

The DNS server that hosts the wildcard domain knows nothing about route host names. It merely resolves any name to the configured IP addresses. Only the OpenShift router knows about route host names, treating each one as an HTTP virtual host. The OpenShift router blocks invalid wildcard domain host names that do not correspond to any route and returns an HTTP 404 error.

Leveraging the Default Routing Service

The default routing service is implemented as an **HAProxy** pod. Router pods, containers, and their configuration can be inspected just like any other resource in an OpenShift cluster:

```
$ oc get pod --all-namespaces -l app=router
NAMESPACE      NAME          READY   STATUS    RESTARTS   AGE
openshift-ingress  router-default-746b5cfb65-f6sdm  1/1     Running   1          4d
```

By default, router is deployed in **openshift-ingress** project. Use **oc describe pod** command to get the routing configuration details:

```
$ oc describe pod router-default-746b5cfb65-f6sdm
Name:           router-default-746b5cfb65-f6sdm
Namespace:      openshift-ingress
...output omitted...
Containers:
  router:
```

```
...output omitted...
Environment:
  STATS_PORT:          1936
  ROUTER_SERVICE_NAMESPACE:  openshift-ingress
  DEFAULT_CERTIFICATE_DIR: /etc/pki/tls/private
  ROUTER_SERVICE_NAME:    default
  ROUTER_CANONICAL_HOSTNAME: apps.cluster.lab.example.com
...output omitted...
```

The subdomain, or default domain to be used in all default routes, takes its value from the **ROUTER_CANONICAL_HOSTNAME** entry.



References

Additional information about the architecture of routes in OpenShift is available in the *Architecture* and *Developer Guide* sections of the **OpenShift Container Platform documentation**.

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

► Guided Exercise

Exposing a Service as a Route

In this exercise, you will create, build, and deploy an application on an OpenShift cluster and expose its service as a route.

Outcomes

You should be able to expose a service as a route for a deployed OpenShift application.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab openshift-routes start
```

► 1. Prepare the lab environment.

1.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

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

1.2. Log in to the OpenShift cluster.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

1.3. Create a new project that contains your RHOCP developer username for the resources you create during this exercise:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-route
```

► 2. Create a new PHP application using Source-to-Image from the **php-helloworld** directory in the Git repository at [http://github.com/\\${RHT_OCP4_GITHUB_USER}/D0180-apps/](http://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps/)

2.1. Use the **oc new-app** command to create the PHP application.



Important

The following example uses a backslash (\) to indicate that the second line is a continuation of the first line. If you wish to ignore the backslash, you can type the entire command in one line.

```
[student@workstation ~]$ oc new-app --as-deployment-config \
> php:7.3~https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps \
> --context-dir php-helloworld --name php-helloworld
--> Found image fbe3911 (13 days old) in image stream "openshift/php" under tag
"7.3" for "php:7.3"
...output omitted...
--> Creating resources ...
...output omitted...
--> Success
Build scheduled, use 'oc logs -f bc/php-helloworld' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/php-helloworld'
Run 'oc status' to view your app.
```

- 2.2. Wait until the application finishes building and deploying by monitoring the progress with the **oc get pods -w** command:

```
[student@workstation ~]$ oc get pods -w
NAME           READY   STATUS    RESTARTS   AGE
php-helloworld-1-build  0/1     Init:0/2   0          2s
php-helloworld-1-build  0/1     Init:0/2   0          4s
php-helloworld-1-build  0/1     Init:1/2   0          5s
php-helloworld-1-build  0/1     PodInitializing   0          6s
php-helloworld-1-build  1/1     Running   0          7s
php-helloworld-1-deploy  0/1     Pending   0          0s
php-helloworld-1-deploy  0/1     Pending   0          0s
php-helloworld-1-deploy  0/1     ContainerCreating   0          0s
php-helloworld-1-build  0/1 Completed  0          5m8s
php-helloworld-1-cnphm  0/1     Pending   0          0s
php-helloworld-1-cnphm  0/1     Pending   0          1s
php-helloworld-1-deploy  1/1     Running   0          4s
php-helloworld-1-cnphm  0/1     ContainerCreating   0          1s
php-helloworld-1-cnphm  1/1 Running  0          62s
php-helloworld-1-deploy  0/1     Completed  0          65s
php-helloworld-1-deploy  0/1     Terminating  0          66s
php-helloworld-1-deploy  0/1     Terminating  0          66s
^C
```

Your exact output may differ in names, status, timing, and order. Look for the **Completed** container with the **deploy** suffix: That means the application is deployed successfully. The container in **Running** status with a random suffix (**cnphm** in the example) contains the application and shows it is up and running.

Alternatively, monitor the build and deployment logs with the **oc logs -f bc/php-helloworld** and **oc logs -f dc/php-helloworld** commands, respectively. Press **Ctrl+C** to exit the command if necessary.

```
[student@workstation ~]$ oc logs -f bc/php-helloworld
Cloning "https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps" ...
Commit: f7cd8963ef353d9173c3a21dcccf402f3616840b (Initial commit, including all
apps previously in course)
...output omitted...
```

```

STEP 7: USER 1001
STEP 8: RUN /usr/libexec/s2i/assemble
---> Installing application source...
...output omitted...
Push successful
[student@workstation ~]$ oc logs -f dc/php-helloworld
-> Cgroups memory limit is set, using HTTPD_MAX_REQUEST_WORKERS=136
=> sourcing 20-copy-config.sh ...
...output omitted...
[core:notice] [pid 1] AH00094: Command line: 'httpd -D FOREGROUND'
^C

```

Your exact output may differ.

- 2.3. Review the service for this application using the **oc describe** command:

```

[student@workstation ~]$ oc describe svc/php-helloworld
Name:          php-helloworld
Namespace:     ${RHT_OCP4_DEV_USER}-route
Labels:         app=php-helloworld
                app.kubernetes.io/component=php-helloworld
                app.kubernetes.io/instance=php-helloworld
Annotations:   openshift.io/generated-by: OpenShiftNewApp
Selector:      deploymentconfig=php-helloworld
Type:          ClusterIP
IP:            172.30.228.124
Port:          8080-tcp  8080/TCP
TargetPort:    8080/TCP
Endpoints:    10.10.0.35:8080
Port:          8443-tcp  8443/TCP
TargetPort:    8443/TCP
Endpoints:    10.10.0.35:8443
Session Affinity: None
Events:        <none>

```

The IP address displayed in the output of the command may differ.

- 3. Expose the service, which creates a route. Use the default name and fully qualified domain name (FQDN) for the route:

```

[student@workstation ~]$ oc expose svc/php-helloworld
route.route.openshift.io/php-helloworld exposed
[student@workstation ~]$ oc describe route
Name:          php-helloworld
Namespace:     ${RHT_OCP4_DEV_USER}-route
Created:       5 seconds ago
Labels:         app=php-helloworld
                app.kubernetes.io/component=php-helloworld
                app.kubernetes.io/instance=php-helloworld
Annotations:   openshift.io/host.generated=true
Requested Host: php-helloworld-${RHT_OCP4_DEV_USER}-route.${RHT_OCP4_WILDCARD}...
               exposed on router default (host ${RHT_OCP4_WILDCARD_DOMAIN}) 4 seconds ago
Path:          <none>
TLS Termination: <none>
Insecure Policy: <none>

```

```
Endpoint Port: 8080-tcp  
  
Service: php-helloworld  
Weight: 100 (100%)  
Endpoints: 10.10.0.35:8443, 10.10.0.35:8080
```

- ▶ 4. Access the service from a host external to the cluster to verify that the service and route are working.

```
[student@workstation ~]$ curl \  
> php-helloworld-${RHT_OCP4_DEV_USER}-route.${RHT_OCP4_WILDCARD_DOMAIN}  
Hello, World! php version is 7.3.11
```



Note

The output of the PHP application depends on the actual code in the Git repository. It may be different if you updated the code in previous sections.

Notice the FQDN is comprised of the application name and project name by default. The remainder of the FQDN, the subdomain, is defined when OpenShift is installed.

- ▶ 5. Replace this route with a route named **xyz**.

5.1. Delete the current route:

```
[student@workstation ~]$ oc delete route/php-helloworld  
route.route.openshift.io "php-helloworld" deleted
```



Note

Deleting the route is optional. You can have multiple routes for the same service, provided they have different names.

5.2. Create a route for the service with a name of **\${RHT_OCP4_DEV_USER}-xyz**.

```
[student@workstation ~]$ oc expose svc/php-helloworld \  
> --name=${RHT_OCP4_DEV_USER}-xyz  
route.route.openshift.io/${RHT_OCP4_DEV_USER}-xyz exposed  
[student@workstation ~]$ oc describe route  
Name: ${RHT_OCP4_DEV_USER}-xyz  
Namespace: ${RHT_OCP4_DEV_USER}-route  
Created: 5 seconds ago  
Labels:  
    app=php-helloworld  
    app.kubernetes.io/component=php-helloworld  
    app.kubernetes.io/instance=php-helloworld  
Annotations: openshift.io/host.generated=true  
Requested Host: ${RHT_OCP4_DEV_USER}-xyz-${RHT_OCP4_DEV_USER}-route.${RHT_...  
    exposed on router default (host ${RHT_OCP4_WILDCARD_DOMAIN}) 4 seconds ago  
Path: <none>  
TLS Termination: <none>  
Insecure Policy: <none>
```

```
Endpoint Port: 8080-tcp  
  
Service: php-helloworld  
Weight: 100 (100%)  
Endpoints: 10.10.0.35:8443, 10.10.0.35:8080
```

Note the new FQDN that was generated based on the new route name. Both the route name and the project name contain your user name, hence it appears twice in the route FQDN.

5.3. Make an HTTP request using the FQDN on port 80:

```
[student@workstation ~]$ curl \  
> ${RHT_OCP4_DEV_USER}-xyz-${RHT_OCP4_DEV_USER}-route.${RHT_OCP4_WILDCARD_DOMAIN}  
Hello, World! php version is 7.3.11
```

Finish

On **workstation**, run the **lab openshift-routes finish** script to complete this exercise.

```
[student@workstation ~]$ lab openshift-routes finish
```

This concludes the guided exercise.

Creating Applications with Source-to-Image

Objectives

After completing this section, students should be able to deploy an application using the Source-to-Image (S2I) facility of OpenShift Container Platform.

The Source-to-Image (S2I) Process

Source-to-Image (S2I) is a tool that makes it easy to build container images from application source code. This tool takes an application's source code from a Git repository, injects the source code into a base container based on the language and framework desired, and produces a new container image that runs the assembled application.

Figure 6.6 shows the resources created by the `oc new-app` command when the argument is an application source code repository. Notice that S2I also creates a Deployment Configuration and all its dependent resources.

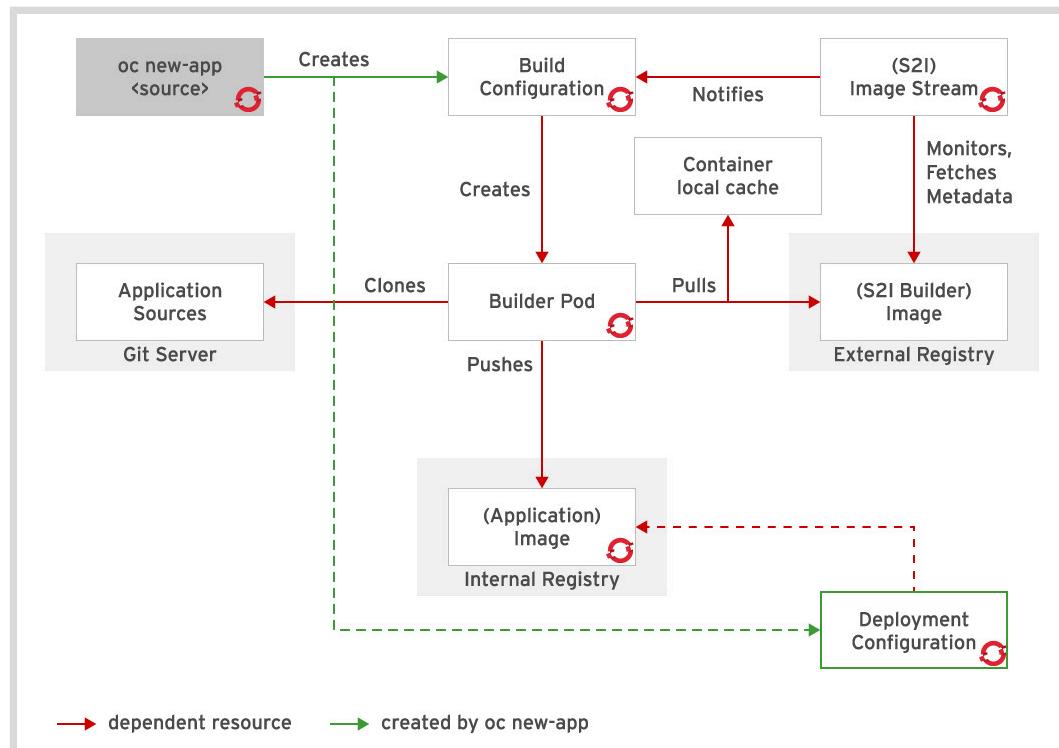


Figure 6.6: Deployment Configuration and dependent resources

S2I is the primary strategy used for building applications in OpenShift Container Platform. The main reasons for using source builds are:

- User efficiency: Developers do not need to understand Dockerfiles and operating system commands such as `yum install`. They work using their standard programming language tools.

- Patching: S2I allows for rebuilding all the applications consistently if a base image needs a patch due to a security issue. For example, if a security issue is found in a PHP base image, then updating this image with security patches updates all applications that use this image as a base.
- Speed: With S2I, the assembly process can perform a large number of complex operations without creating a new layer at each step, resulting in faster builds.
- Ecosystem: S2I encourages a shared ecosystem of images where base images and scripts can be customized and reused across multiple types of applications.

Describing Image Streams

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

Even though it is evident that the container image needs to be updated when application code changes, it may not be evident that the deployed pods also need to be updated should the builder image change.

The *image stream resource* is a configuration that names specific container images associated with *image stream tags*, an alias for these container images. OpenShift builds applications against an image stream. The OpenShift installer populates several image streams by default during installation. To determine available image streams, use the **oc get** command, as follows:

```
$ oc get is -n openshift
NAME          IMAGE REPOSITORY           TAGS
cli           ...svc:5000/openshift/cli   latest
dotnet        ...svc:5000/openshift/dotnet 2.0,2.1,latest
dotnet-runtime ...svc:5000/openshift/dotnet-runtime 2.0,2.1,latest
httpd         ...svc:5000/openshift/httpd  2.4,latest
jenkins       ...svc:5000/openshift/jenkins 1,2
mariadb       ...svc:5000/openshift/mariadb 10.1,10.2,latest
mongodb       ...svc:5000/openshift/mongodb 2.4,2.6,3.2,3.4,3.6,latest
mysql         ...svc:5000/openshift/mysql  5.5,5.6,5.7,latest
nginx         ...svc:5000/openshift/nginx  1.10,1.12,1.8,latest
nodejs        ...svc:5000/openshift/nodejs 0.10,10,11,4,6,8,latest
perl          ...svc:5000/openshift/perl   5.16,5.20,5.24,5.26,latest
php           ...svc:5000/openshift/php   5.5,5.6,7.0,7.1,latest
postgresql    ...svc:5000/openshift/postgresql 10,9.2,9.4,9.5,9.6,latest
python        ...svc:5000/openshift/python  2.7,3.3,3.4,3.5,3.6,latest
redis         ...svc:5000/openshift/redis  3.2,latest
ruby          ...svc:5000/openshift/ruby   2.0,2.2,2.3,2.4,2.5,latest
wildfly       ...svc:5000/openshift/wildfly 10.0,10.1,11.0,12.0,...
```



Note

Your OpenShift instance may have more or fewer image streams depending on local additions and OpenShift point releases.

OpenShift detects when an image stream changes and takes action based on that change. If a security issue arises in the **nodejs-010-rhel7** image, it can be updated in the image repository, and OpenShift can automatically trigger a new build of the application code.

It is likely that an organization chooses several supported base S2I images from Red Hat, but may also create their own base images.

Building an Application with S2I and the CLI

Building an application with S2I can be accomplished using the OpenShift CLI.

An application can be created using the S2I process with the **oc new-app** command from the CLI.

```
$ oc new-app 1 --as-deployment-config 2 php~http://my.git.server.com/my-app3  
--name=myapp4
```

- 2** Create a DeploymentConfig instead of a Deployment
- 2** The image stream used in the process appears to the left of the tilde (~).
- 3** The URL after the tilde indicates the location of the source code's Git repository.
- 4** Sets the application name.



Note

Instead of using the tilde, you can set the image stream by using the **-i** option.

```
$ oc new-app --as-deployment-config -i php http://services.lab.example.com/app  
--name=myapp
```

The **oc new-app** command allows creating applications using source code from a local or remote Git repository. If only a source repository is specified, **oc new-app** tries to identify the correct image stream to use for building the application. In addition to application code, S2I can also identify and process Dockerfiles to create a new image.

The following example creates an application using the Git repository in the current directory.

```
$ oc new-app --as-deployment-config .
```



Important

When using a local Git repository, the repository must have a remote origin that points to a URL accessible by the OpenShift instance.

It is also possible to create an application using a remote Git repository and a context subdirectory:

```
$ oc new-app --as-deployment-config \  
https://github.com/openshift/sti-ruby.git \  
--context-dir=2.0/test/puma-test-app
```

Finally, it is possible to create an application using a remote Git repository with a specific branch reference:

```
$ oc new-app --as-deployment-config \
https://github.com/openshift/ruby-hello-world.git#beta4
```

If an image stream is not specified in the command, **new-app** attempts to determine which language builder to use based on the presence of certain files in the root of the repository:

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

After a language is detected, the **new-app** command searches for image stream tags that have support for the detected language, or an image stream that matches the name of the detected language.

Create a JSON resource definition file by using the **-o json** parameter and output redirection:

```
$ oc -o json new-app --as-deployment-config \
> php-http://services.lab.example.com/app \
> --name=myapp > s2i.json
```

This JSON definition file creates a list of resources. The first resource is the image stream:

```
...output omitted...
{
  "kind": "ImageStream", ❶
  "apiVersion": "image.openshift.io/v1",
  "metadata": {
    "name": "myapp", ❷
    "creationTimestamp": null
    "labels": {
      "app": "myapp"
    },
    "annotations": {
      "openshift.io/generated-by": "OpenShiftNewApp"
    }
  },
  "spec": {
    "lookupPolicy": {
      "local": false
    }
  },
  "status": {
    "dockerImageRepository": ""
  }
}
```

```

    },
},
...output omitted...

```

- ➊ Define a resource type of image stream.
- ➋ Name the image stream **myapp**.

The build configuration (**bc**) is responsible for defining input parameters and triggers that are executed to transform the source code into a runnable image. The **BuildConfig** (BC) is the second resource, and the following example provides an overview of the parameters used by OpenShift to create a runnable image.

```

...output omitted...
{
  "kind": "BuildConfig", ➊
  "apiVersion": "build.openshift.io/v1",
  "metadata": {
    "name": "myapp", ➋
    "creationTimestamp": null,
    "labels": {
      "app": "myapp"
    },
    "annotations": {
      "openshift.io/generated-by": "OpenShiftNewApp"
    }
  },
  "spec": {
    "triggers": [
      {
        "type": "GitHub",
        "github": {
          "secret": "S5_4BZpPabM6KrIuPBvI"
        }
      },
      {
        "type": "Generic",
        "generic": {
          "secret": "3q8K8JNDoRzhjoz1KgMz"
        }
      },
      {
        "type": "ConfigChange"
      },
      {
        "type": "ImageChange",
        "imageChange": {}
      }
    ],
    "source": {
      "type": "Git",
      "git": {
        "uri": "http://services.lab.example.com/app" ➌
      }
    }
  }
}

```

```

"strategy": {
    "type": "Source", ④
    "sourceStrategy": {
        "from": {
            "kind": "ImageStreamTag",
            "namespace": "openshift",
            "name": "php:7.3" ⑤
        }
    },
    "output": {
        "to": {
            "kind": "ImageStreamTag",
            "name": "myapp:latest" ⑥
        }
    },
    "resources": {},
    "postCommit": {},
    "nodeSelector": null
},
"status": {
    "lastVersion": 0
}
},
...output omitted...

```

- ① Define a resource type of **BuildConfig**.
- ② Name the **BuildConfig** **myapp**.
- ③ Define the address to the source code Git repository.
- ④ Define the strategy to use S2I.
- ⑤ Define the builder image as the php:7.3 image stream.
- ⑥ Name the output image stream **myapp:latest**.

The third resource is the deployment configuration that is responsible for customizing the deployment process in OpenShift. It may include parameters and triggers that are necessary to create new container instances, and are translated into a replication controller from Kubernetes. Some of the features provided by **DeploymentConfig** objects are:

- User customizable strategies to transition from the existing deployments to new deployments.
- Rollbacks to a previous deployment.
- Manual replication scaling.

```

...output omitted...
{
    "kind": "DeploymentConfig", ①
    "apiVersion": "apps.openshift.io/v1",
    "metadata": {
        "name": "myapp", ②
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {

```

```

        "openshift.io/generated-by": "OpenShiftNewApp"
    }
},
"spec": {
    "strategy": {
        "resources": {}
    },
    "triggers": [
        {
            "type": "ConfigChange" ③
        },
        {
            "type": "ImageChange", ④
            "imageChangeParams": {
                "automatic": true,
                "containerNames": [
                    "myapp"
                ],
                "from": {
                    "kind": "ImageStreamTag",
                    "name": "myapp:latest"
                }
            }
        }
    ],
    "replicas": 1,
    "test": false,
    "selector": {
        "app": "myapp",
        "deploymentconfig": "myapp"
    },
    "template": {
        "metadata": {
            "creationTimestamp": null,
            "labels": {
                "app": "myapp",
                "deploymentconfig": "myapp"
            },
            "annotations": {
                "openshift.io/generated-by": "OpenShiftNewApp"
            }
        },
        "spec": {
            "containers": [
                {
                    "name": "myapp",
                    "image": "myapp:latest", ⑤
                    "ports": [ ⑥
                        {
                            "containerPort": 8080,
                            "protocol": "TCP"
                        },
                        {
                            "containerPort": 8443,
                            "protocol": "TCP"
                        }
                    ]
                }
            ]
        }
    }
}

```

```

        }
    ],
    "resources": {}
}
]
}
},
"status": {
    "latestVersion": 0,
    "observedGeneration": 0,
    "replicas": 0,
    "updatedReplicas": 0,
    "availableReplicas": 0,
    "unavailableReplicas": 0
}
},
...output omitted...

```

- ➊ Define a resource type of **DeploymentConfig**.
- ➋ Name the **DeploymentConfig** **myapp**.
- ➌ A configuration change trigger causes a new deployment to be created any time the replication controller template changes.
- ➍ An image change trigger causes the creation of a new deployment each time a new version of the **myapp:latest** image is available in the repository.
- ➎ Defines the container image to deploy: **myapp:latest**.
- ➏ Specifies the container ports.

The last item is the service, already covered in previous chapters:

```

...output omitted...
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "myapp",
        "creationTimestamp": null,
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "ports": [
            {
                "name": "8080-tcp",
                "protocol": "TCP",
                "port": 8080,
                "targetPort": 8080
            },
            {
                "name": "8443-tcp",

```

```

        "protocol": "TCP",
        "port": 8443,
        "targetPort": 8443
    }
],
"selector": {
    "app": "myapp",
    "deploymentconfig": "myapp"
}
},
"status": {
    "loadBalancer": {}
}
}
}

```

**Note**

By default, the **oc new-app** command does not create a route. You can create a route after creating the application. However, a route is automatically created when using the web console because it uses a template.

After creating a new application, the build process starts. Use the **oc get builds** command to see a list of application builds:

\$ oc get builds					
NAME	TYPE	FROM	STATUS	STARTED	DURATION
php-helloworld-1	Source	Git@9e17db8	Running	13 seconds ago	

OpenShift allows viewing the build logs. The following command shows the last few lines of the build log:

```
$ oc logs build/myapp-1
```

**Important**

If the build is not **Running** yet, or OpenShift has not deployed the **s2i-build** pod yet, the above command throws an error. Just wait a few moments and retry it.

Trigger a new build with the **oc start-build build_config_name** command:

\$ oc get buildconfig			
NAME	TYPE	FROM	LATEST
myapp	Source	Git	1

```
$ oc start-build myapp
build "myapp-2" started
```

Relationship Between Build and Deployment Configurations

The **BuildConfig** pod is responsible for creating the images in OpenShift and pushing them to the internal container registry. Any source code or content update typically requires a new build to guarantee the image is updated.

The **DeploymentConfig** pod is responsible for deploying pods to OpenShift. The outcome of a **DeploymentConfig** pod execution is the creation of pods with the images deployed in the internal container registry. Any existing running pod may be destroyed, depending on how the **DeploymentConfig** resource is set.

The **BuildConfig** and **DeploymentConfig** resources do not interact directly. The **BuildConfig** resource creates or updates a container image. The **DeploymentConfig** reacts to this new image or updated image event and creates pods from the container image.



References

Source-to-Image (S2I) Build

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/builds/build-strategies#build-strategy-s2i_build-strategies

S2I GitHub repository

<https://github.com/openshift/source-to-image>

► Guided Exercise

Creating a Containerized Application with Source-to-Image

In this exercise, you will build an application from source code and deploy the application to an OpenShift cluster.

Outcomes

You should be able to:

- Build an application from source code using the OpenShift command-line interface.
- Verify the successful deployment of the application using the OpenShift command-line interface.

Before You Begin

Run the following command to download the relevant lab files and configure the environment:

```
[student@workstation ~]$ lab openshift-s2i start
```

- 1. Inspect the PHP source code for the sample application and create and push a new branch named **s2i** to use during this exercise.

- 1.1. Enter your local clone of the **D0180-apps** Git repository and checkout the **master** branch of the course's repository to ensure you start this exercise from a known good state:

```
[student@workstation openshift-s2i]$ cd ~/D0180-apps
[student@workstation D0180-apps]$ git checkout master
...output omitted...
```

- 1.2. Create a new branch to save any changes you make during this exercise:

```
[student@workstation D0180-apps]$ git checkout -b s2i
Switched to a new branch 's2i'
[student@workstation D0180-apps]$ git push -u origin s2i
...output omitted...
 * [new branch]      s2i -> s2i
Branch s2i set up to track remote branch s2i from origin.
```

- 1.3. Review the PHP source code of the application, inside the the **php-helloworld** folder.
Open the **index.php** file in the **/home/student/D0180-apps/php-helloworld** folder:

```
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
?>
```

The application implements a simple response which returns the PHP version it is running.

► 2. Prepare the lab environment.

- 2.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

```
[student@workstation DO180-apps]$ source /usr/local/etc/ocp4.config
```

- 2.2. Log in to the OpenShift cluster.

```
[student@workstation DO180-apps]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 2.3. Create a new project that contains your RHOCP developer username for the resources you create during this exercise:

```
[student@workstation DO180-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-s2i
```

► 3. Create a new PHP application using Source-to-Image from the **php-helloworld** directory using the **s2i** branch you created in the previous step in your fork of the DO180-apps Git repository.

- 3.1. Use the **oc new-app** command to create the PHP application.



Important

The following example uses the number sign (#) to select a specific branch from the git repository, in this case the **s2i** branch created in the previous step.

```
[student@workstation DO180-apps]$ oc new-app --as-deployment-config php:7.3 \
> --name=php-helloworld \
> https://github.com/${RHT_OCP4_GITHUB_USER}/DO180-apps#s2i \
> --context-dir php-helloworld
```

- 3.2. Wait for the build to complete and the application to deploy. Verify that the build process starts with the **oc get pods** command.

```
[student@workstation openshift-s2i]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
php-helloworld-1-build   1/1     Running   0          5s
```

- 3.3. Examine the logs for this build. Use the build pod name for this build, **php-helloworld-1-build**.

```
[student@workstation D0180-apps]$ oc logs --all-containers \
> -f php-helloworld-1-build
Cloning "https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps" ...
Commit: f7cd8963ef353d9173c3a21dccc402f3616840b (Initial commit, including all
apps previously in course)

...output omitted...

Writing manifest to image destination
Storing signatures
Generating dockerfile with builder image image-registry.openshift-image-...
php@sha256:f3c9...7546
STEP 1: FROM image-registry.openshift-image-registry.svc:5000/...

...output omitted...

Pushing image ...openshift-image-registry.svc:5000/s2i/php-helloworld:latest...
Getting image source signatures
...output omitted...
STEP 8: RUN /usr/libexec/s2i/assemble
--> Installing application source...
...output omitted...
Copying config sha256:6ce5730f48d9c746e7cbd7ea7b8ed0f15b83932444d1d2bd7711d7...
21.45 KiB / 21.45 KiB 0s
Writing manifest to image destination
Storing signatures
Successfully pushed .../php-helloworld:latest@sha256:63e757a4c0edaeda497dab7...
Push successful
```

Notice the clone of the Git repository as the first step of the build. Next, the Source-to-Image process built a new container called **s2i/php-helloworld:latest**. The last step in the build process is to push this container to the OpenShift private registry.

- 3.4. Review the **DeploymentConfig** for this application:

```
[student@workstation D0180-apps]$ oc describe dc/php-helloworld
Name:          php-helloworld
Namespace:     ${RHT_OCP4_DEV_USER}-s2i
Created:       About a minute ago
Labels:        app=php-helloworld
               app.kubernetes.io/component=php-helloworld
               app.kubernetes.io/instance=php-helloworld
Annotations:   openshift.io/generated-by=OpenShiftNewApp
Latest Version: 1
Selector:      deploymentconfig=php-helloworld
Replicas:      1
Triggers:      Config, Image(phi-helloworld@latest, auto=true)
Strategy:      Rolling
Template:
Pod Template:
```

```

Labels:      deploymentconfig=php-helloworld
Annotations: openshift.io/generated-by: OpenShiftNewApp
Containers:
  php-helloworld:
    Image:          image-registry.openshift-image-
registry.svc:5000/${RHT_OCP4_DEV_USER}-s2i/php-
helloworld@sha256:ae3ec890625f153c0163812501f2522fddc96431036374aa2472ddd52bc7ccb4
    Ports:         8080/TCP, 8443/TCP
    Host Ports:   0/TCP, 0/TCP
    Environment: <none>
    Mounts:       <none>
    Volumes:      <none>

Deployment #1 (latest):
  Name:        php-helloworld-1
  Created:     about a minute ago
  Status:      Complete
  Replicas:    1 current / 1 desired
  Selector:    deployment=php-helloworld-1,deploymentconfig=php-helloworld
  Labels:      app.kubernetes.io/component=php-helloworld,app.kubernetes.io/
instance=php-helloworld,app=php-helloworld,openshift.io/deployment-
config.name=php-helloworld
  Pods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 Failed
...output omitted...

```

3.5. Add a route to test the application:

```
[student@workstation D0180-apps]$ oc expose service php-helloworld \
> --name ${RHT_OCP4_DEV_USER}-helloworld
route.route.openshift.io/${RHT_OCP4_DEV_USER}-helloworld exposed
```

3.6. Find the URL associated with the new route:

```
[student@workstation D0180-apps]$ oc get route -o jsonpath='{..spec.host}{"\n"}'
${RHT_OCP4_DEV_USER}-helloworld-${RHT_OCP4_DEV_USER}-s2i.
${RHT_OCP4_WILDCARD_DOMAIN}
```

3.7. Test the application by sending an HTTP GET request to the URL you obtained in the previous step:

```
[student@workstation D0180-apps]$ curl -s \
> ${RHT_OCP4_DEV_USER}-helloworld-${RHT_OCP4_DEV_USER}-s2i.\
> ${RHT_OCP4_WILDCARD_DOMAIN}
Hello, World! php version is 7.3.11
```

- ▶ 4. Explore starting application builds by changing the application in its Git repository and executing the proper commands to start a new Source-to-Image build.

4.1. Enter the source code directory.

```
[student@workstation D0180-apps]$ cd ~/D0180-apps/php-helloworld
```

- 4.2. Edit the **index.php** file as shown below:

```
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
print "A change is a coming!\n";
?>
```

Save the file.

- 4.3. Commit the changes and push the code back to the remote Git repository:

```
[student@workstation php-helloworld]$ git add .
[student@workstation php-helloworld]$ git commit -m 'Changed index page contents.'
[s2i b1324aa] changed index page contents
 1 file changed, 1 insertion(+)
[student@workstation php-helloworld]$ git push origin s2i
...output omitted...
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 417 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/${RHT_OCP4_GITHUB_USER}/D0180-apps
 f7cd896..b1324aa s2i -> s2i
```

- 4.4. Start a new Source-to-Image build process and wait for it to build and deploy:

```
[student@workstation php-helloworld]$ oc start-build php-helloworld
build.build.openshift.io/php-helloworld-2 started
[student@workstation php-helloworld]$ oc logs php-helloworld-2-build -f
...output omitted...

Successfully pushed .../php-helloworld:latest@sha256:74e757a4c0edaeda497dab7...
Push successful
```



Note

Logs may take some seconds to be available after the build starts. If the previous command fails, wait a bit and try again.

- 4.5. After the second build has completed use the **oc get pods** command to verify that the new version of the application is running.

NAME	READY	STATUS	RESTARTS	AGE
php-helloworld-1-build	0/1	Completed	0	11m
php-helloworld-1-deploy	0/1	Completed	0	10m
php-helloworld-2-build	0/1	Completed	0	45s
php-helloworld-2-deploy	0/1	Completed	0	16s
php-helloworld-2-wq9wz	1/1	Running	0	13s

- 4.6. Test that the application serves the new content:

```
[student@workstation php-helloworld]$ curl -s \  
> ${RHT_OCP4_DEV_USER}-helloworld-${RHT_OCP4_DEV_USER}-s2i.\ \  
> ${RHT_OCP4_WILDCARD_DOMAIN}  
Hello, World! php version is 7.3.11  
A change is a coming!
```

Finish

On **workstation**, run the **lab openshift-s2i finish** script to complete this lab.

```
[student@workstation php-helloworld]$ lab openshift-s2i finish
```

This concludes the guided exercise.

► Lab

Deploying Containerized Applications on OpenShift

Performance Checklist

In this lab, you will create an application using the OpenShift Source-to-Image facility.

Outcomes

You should be able to create an OpenShift application and access it through a web browser.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab openshift-review start
```

1. Prepare the lab environment.

- 1.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

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

- 1.2. Log in to the OpenShift cluster.
 - 1.3. Create a new project named "\${RHT_OCP4_DEV_USER}-ocp" for the resources you create during this exercise:
2. Create a temperature converter application written in PHP using the **php:7.3** image stream tag. The source code is in the Git repository at <https://github.com/RedHatTraining/D0180-apps/> in the **temps** directory. You may use the OpenShift command-line interface or the web console to create the application.
Expose the application's service to make the application accessible from a web browser.
3. Verify that you can access the application in a web browser at `http://temps-${RHT_OCP4_DEV_USER}-ocp.${RHT_OCP4_WILDCARD_DOMAIN}`.

Evaluation

On **workstation**, run the **lab openshift-review grade** command to grade your work. Correct any reported failures and rerun the script until successful.

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

Finish

On **workstation**, run the **lab openshift-review finish** command to complete this lab.

```
[student@workstation ~]$ lab openshift-review finish
```

This concludes the lab.

► Solution

Deploying Containerized Applications on OpenShift

Performance Checklist

In this lab, you will create an application using the OpenShift Source-to-Image facility.

Outcomes

You should be able to create an OpenShift application and access it through a web browser.

Before You Begin

Open a terminal on **workstation** as the **student** user and run the following command:

```
[student@workstation ~]$ lab openshift-review start
```

1. Prepare the lab environment.

- 1.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

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

- 1.2. Log in to the OpenShift cluster.

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p \
> ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful
...output omitted...
```

- 1.3. Create a new project named "\${RHT_OCP4_DEV_USER}-ocp" for the resources you create during this exercise:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-ocp
```

2. Create a temperature converter application written in PHP using the **php:7.3** image stream tag. The source code is in the Git repository at <https://github.com/RedHatTraining/D0180-apps/> in the **temps** directory. You may use the OpenShift command-line interface or the web console to create the application.

Expose the application's service to make the application accessible from a web browser.

- 2.1. If using the command-line interface, run the following commands:

```
[student@workstation ~]$ oc new-app --as-deployment-config \
> php:7.3-https://github.com/RedHatTraining/D0180-apps \
> --context-dir temps --name temps
--> Found image fbe3911 (2 weeks old) in image stream "openshift/php" under tag
"7.3" for "php:7.3"

Apache 2.4 with PHP 7.3
-----
PHP 7.3 available as container is a base platform ...output omitted...
...output omitted...

--> Creating resources ...
imagestream.image.openshift.io "temps" created
buildconfig.build.openshift.io "temps" created
deploymentconfig.apps.openshift.io "temps" created
service "temps" created
--> Success
Build scheduled, use 'oc logs -f bc/temps' to track its progress.
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose svc/temps'
Run 'oc status' to view your app.
```

2.2. Monitor progress of the build.

```
[student@workstation ~]$ oc logs -f bc/temps
Cloning "https://github.com/RedHatTraining/D0180-apps" ...
Commit: f7cd8963ef353d9173c3a21dcccf402f3616840b (Initial commit, including all
apps previously in course)
...output omitted...
Successfully pushed image-registry.openshift-
image-registry.svc:5000/${RHT_OCP4_DEV_USER}-temps/
temps@sha256:59f713adfacdbc2a3ca81c4ef4af46517dfffa3f0029372f86fbcaf571416a74
Push successful
```

2.3. Verify that the application is deployed.

```
[student@workstation ~]$ oc get pods -w
NAME        READY   STATUS    RESTARTS   AGE
temps-1-build  0/1     Completed   0          91s
temps-1-deploy  0/1     Completed   0          60s
temps-1-p4zjc   1/1     Running    0          58s
```

Press **Ctrl+C** to exit the **oc get pods -w** command.

2.4. Expose the **temps** service to create an external route for the application.

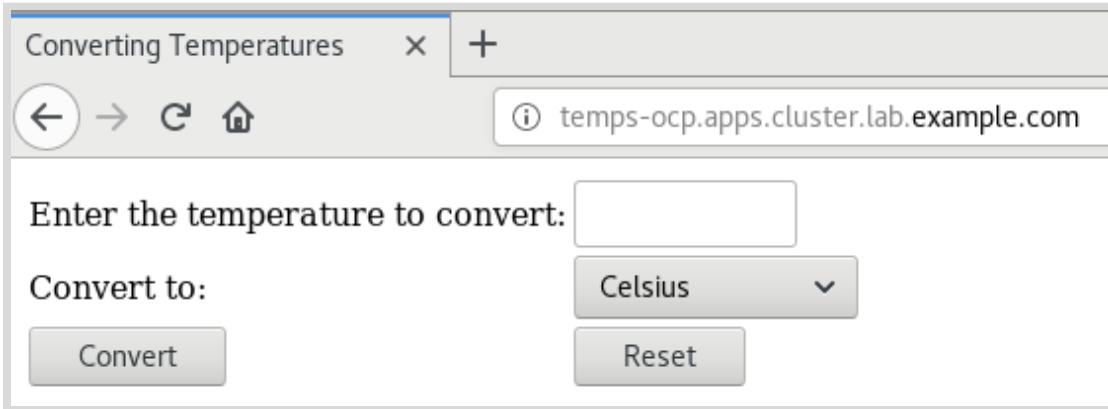
```
[student@workstation ~]$ oc expose svc/temps
route.route.openshift.io/temps exposed
```

3. Verify that you can access the application in a web browser at `http://temps-$ {RHT_OCP4_DEV_USER} - ocp . ${RHT_OCP4_WILDCARD_DOMAIN}`.

- 3.1. Determine the URL for the route.

```
[student@workstation ~]$ oc get route/temps
NAME      HOST/PORT
temps     temps-$ {RHT_OCP4_DEV_USER} - ocp . ${RHT_OCP4_WILDCARD_DOMAIN} ...
```

- 3.2. Verify that the temperature converter application works by opening a web browser and navigating to the URL displayed in the previous step.



Evaluation

On **workstation**, run the **lab openshift-review grade** command to grade your work. Correct any reported failures and rerun the script until successful.

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

Finish

On **workstation**, run the **lab openshift-review finish** command to complete this lab.

```
[student@workstation ~]$ lab openshift-review finish
```

This concludes the lab.

Summary

In this chapter, you learned:

- Use the OpenShift command-line client **oc** to:
 - Create, change, and delete projects.
 - Create application resources inside a project.
 - Delete, inspect, edit, and export resources inside a project.
 - Check logs from application pods, deployments, and build operations.
- The **oc new-app** command can create application pods in many different ways: from an existing container image hosted on an image registry, from Dockerfiles, and from source code using the Source-to-Image (S2I) process.
- Source-to-Image (S2I) is a tool that makes it easy to build a container image from application source code. This tool retrieves source code from a Git repository, injects the source code into a selected container image based on a desired language or technology, and produces a new container image that runs the assembled application.
- A **Route** connects a public-facing IP address and DNS host name to an internal-facing service IP. While services allow for network access between pods inside an OpenShift instance, routes allow for network access to pods from users and applications outside the OpenShift instance.

Chapter 7

Deploying Multi-Container Applications

Goal

Deploy applications that are containerized using multiple container images.

Objectives

- Deploy a multicontainer application on OpenShift using a template.

Sections

- Deploying a Multi-Container Applications on OpenShift (and Guided Exercise)

Lab

Containerizing and Deploying a Software Application

Deploying a Multi-Container Application on OpenShift

Objectives

After completing this section, students should be able to deploy a multicontainer application on OpenShift using a template.

Examining the Skeleton of a Template

Deploying an application on OpenShift Container Platform often requires creating several related resources within a Project. For example, a web application may require a **BuildConfig**, **DeploymentConfig**, **Service**, and **Route** resource to run in an OpenShift project. Often the attributes of these resources have the same value, such as a resource's name attribute.

OpenShift *templates* provide a way to simplify the creation of resources that an application requires. A template defines a set of related resources to be created together, as well as a set of application parameters. The attributes of template resources are typically defined in terms of the template parameters, such as a resource's name attribute.

For example, an application might consist of a front-end web application and a database server. Each consists of a service resource and a deployment configuration resource. They share a set of credentials (parameters) for the front end to authenticate to the back end. The template can be processed by specifying parameters or by allowing them to be automatically generated (for example, for a unique database password) in order to instantiate the list of resources in the template as a cohesive application.

The OpenShift installer creates several templates by default in the **openshift** namespace. Run the **oc get templates** command with the **-n openshift** option to list these preinstalled templates:

```
[student@workstation ~]$ oc get templates -n openshift
NAME                  DESCRIPTION
cakephp-mysql-example   An example CakePHP application ...
cakephp-mysql-persistent  An example CakePHP application ...
dancer-mysql-example     An example Dancer application with a MySQL ...
dancer-mysql-persistent   An example Dancer application with a MySQL ...
django-psql-example      An example Django application with a PostgreSQL ...
...output omitted...
rails-psql-persistent     An example Rails application with a PostgreSQL ...
rails-postgresql-example  An example Rails application with a PostgreSQL ...
redis-ephemeral          Redis in-memory data structure store, ...
redis-persistent          Redis in-memory data structure store, ...
```

The following shows a YAML template definition:

```
[student@workstation ~]$ oc get template mysql-persistent -n openshift -o yaml
apiVersion: template.openshift.io/v1
kind: Template
labels: ...value omitted...
message: ...message omitted ...
```

```

metadata:
  annotations:
    description: ...description omitted...
    iconClass: icon-mysql-database
    openshift.io/display-name: MySQL
    openshift.io/documentation-url: ...value omitted...
    openshift.io/long-description: ...value omitted...
    openshift.io/provider-display-name: Red Hat, Inc.
    openshift.io/support-url: https://access.redhat.com
    tags: database,mysql ①
    labels: ...value omitted...
    name: mysql-persistent ②
objects: ③
- apiVersion: v1
  kind: Secret
  metadata:
    annotations: ...annotations omitted...
    name: ${DATABASE_SERVICE_NAME} ④
    stringData: ...stringData omitted...
- apiVersion: v1
  kind: Service
  metadata:
    annotations: ...annotations omitted...
    name: ${DATABASE_SERVICE_NAME}
    spec: ...spec omitted...
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: ${DATABASE_SERVICE_NAME}
    spec: ...spec omitted...
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
    annotations: ...annotations omitted...
    name: ${DATABASE_SERVICE_NAME}
    spec: ...spec omitted...
parameters: ⑤
- ...MEMORY_LIMIT parameter omitted...
- ...NAMESPACE parameter omitted...
- description: The name of the OpenShift Service exposed for the database.
  displayName: Database Service Name
  name: DATABASE_SERVICE_NAME ⑥
  required: true
  value: mysql
- ...MYSQL_USER parameter omitted...
- description: Password for the MySQL connection user.
  displayName: MySQL Connection Password
  from: '[a-zA-Z0-9]{16}' ⑦
  generate: expression
  name: MYSQL_PASSWORD
  required: true
- ...MYSQL_ROOT_PASSWORD parameter omitted...

```

```
- ...MYSQL_DATABASE parameter omitted...
- ...VOLUME_CAPACITY parameter omitted...
- ...MYSQL_VERSION parameter omitted...
```

- ➊ Defines a list of arbitrary tags to associate with this template. Enter any of these tags in the UI to find this template.
- ➋ Defines the template name.
- ➌ The **objects** section defines the list of OpenShift resources for this template. This template creates four resources: a **Secret**, a **Service**, a **PersistentVolumeClaim**, and a **DeploymentConfig**.
- ➍ All four resource objects have their names set to the value of the **DATABASE_SERVICE_NAME** parameter.
- ➎➏ The **parameters** section contains a list of nine parameters. Template resources often define their attributes using the values of these parameters, as demonstrated with the **DATABASE_SERVICE_NAME** parameter.
- ➐ If you do not specify a value for the **MYSQL_PASSWORD** parameter when you create an application with this template, OpenShift generates a password that matches this regular expression.

You can publish a new template to the OpenShift cluster so that other developers can build an application from the template.

Assume you have an task list application named **todo** that requires an OpenShift **DeploymentConfig**, **Service**, and **Route** object for deployment. You create a YAML template definition file that defines attributes for these OpenShift resources, along with definitions for any required parameters. Assuming the template is defined in the **todo-template.yaml** file, use the **oc create** command to publish the application template:

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.yaml
template.template.openshift.io/todonodejs-persistent created
```

By default, the template is created under the current project unless you specify a different one using the **-n** option, as shown in the following example:

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.yaml \
> -n openshift
```



Important

Any template created under the **openshift** namespace (OpenShift project) is available in the web console under the dialog box accessible in the **Catalog** → **Developer Catalog** menu item. Moreover, any template created under the current project is accessible from that project.

Parameters

Templates define a set of *parameters*, which are assigned values. OpenShift resources defined in the template can get their configuration values by referencing *named parameters*. Parameters in a template can have default values, but they are optional. Any default value can be replaced when processing the template.

Each parameter value can be set either explicitly by using the **oc process** command, or generated by OpenShift according to the parameter configuration.

There are two ways to list available parameters from a template. The first one is using the **oc describe** command:

```
$ oc describe template mysql-persistent -n openshift
Name: mysql-persistent
Namespace: openshift
Created: 12 days ago
Labels: samplesoperator.config.openshift.io/managed=true
Description: MySQL database service, with ...description omitted...
Annotations: iconClass=icon-mysql-database
  openshift.io/display-name=MySQL
  ...output omitted...
tags=database,mysql

Parameters:
  Name: MEMORY_LIMIT
  Display Name: Memory Limit
  Description: Maximum amount of memory the container can use.
  Required: true
  Value: 512Mi

  Name: NAMESPACE
  Display Name: Namespace
  Description: The OpenShift Namespace where the ImageStream resides.
  Required: false
  Value: openshift

  ...output omitted...

  Name: MYSQL_VERSION
  Display Name: Version of MySQL Image
  Description: Version of MySQL image to be used (5.7, or latest).
  Required: true
  Value: 5.7

Object Labels: template=mysql-persistent-template

Message: ...output omitted... in your project: ${DATABASE_SERVICE_NAME}.

  Username: ${MYSQL_USER}
  Password: ${MYSQL_PASSWORD}
  Database Name: ${MYSQL_DATABASE}
  Connection URL: mysql://${DATABASE_SERVICE_NAME}:3306/

For more information about using this template, ...output omitted...

Objects:
  Secret      ${DATABASE_SERVICE_NAME}
  Service     ${DATABASE_SERVICE_NAME}
  PersistentVolumeClaim ${DATABASE_SERVICE_NAME}
  DeploymentConfig    ${DATABASE_SERVICE_NAME}
```

The second way is by using the **oc process** with the **--parameters** option:

```
$ oc process --parameters mysql-persistent -n openshift
NAME          DESCRIPTION      GENERATOR      VALUE
MEMORY_LIMIT  Maximum a...    expression      512Mi
NAMESPACE     The OpenS...    expression      openshift
DATABASE_SERVICE_NAME The name ...
MYSQL_USER    Username ...   expression      user[A-Z0-9]{3}
MYSQL_PASSWORD Password ...  expression      [a-zA-Z0-9]{16}
MYSQL_ROOT_PASSWORD Password ... expression      [a-zA-Z0-9]{16}
MYSQL_DATABASE Name of t...   expression      sampledb
VOLUME_CAPACITY Volume sp...  expression      1Gi
MYSQL_VERSION Version o...   expression      5.7
```

Processing a Template Using the CLI

When you process a template, you generate a list of resources to create a new application. To process a template, use the **oc process** command:

```
$ oc process -f <filename>
```

The previous command processes a template file, in either JSON or YAML format, and returns the list of resources to standard output. The format of the output resource list is JSON. To output the resource list in YAML format, use the **-o yaml** with the **oc process** command:

```
$ oc process -o yaml -f <filename>
```

Another option is to process a template from the current project or the **openshift** project:

```
$ oc process <uploaded-template-name>
```



Note

The **oc process** command returns a list of resources to standard output. This output can be redirected to a file:

```
$ oc process -o yaml -f filename > myapp.yaml
```

Templates often generate resources with configurable attributes that are based on the template parameters. To override a parameter, use the **-p** option followed by a **<name>=<value>** pair.

```
$ oc process -o yaml -f mysql.yaml \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi > mysqlProcessed.yaml
```

To create the application, use the generated YAML resource definition file:

```
$ oc create -f mysqlProcessed.yaml
```

Alternatively, it is possible to process the template and create the application without saving a resource definition file by using a UNIX pipe:

```
$ oc process -f mysql.yaml -p MYSQL_USER=dev \
> -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi | oc create -f -
```

To use a template in the **openshift** project to create an application in your project, first export the template:

```
$ oc get template mysql-persistent -o yaml \
> -n openshift > mysql-persistent-template.yaml
```

Next, identify appropriate values for the template parameters and process the template:

```
$ oc process -f mysql-persistent-template.yaml \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi | oc create -f -
```

You can also use two slashes (//) to provide the namespace as part of the template name:

```
$ oc process openshift//mysql-persistent \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi | oc create -f -
```

Alternatively, it is possible to create an application using the **oc new-app** command passing the template name as the **--template** option argument:

```
$ oc new-app --template=mysql-persistent \
> -p MYSQL_USER=dev -p MYSQL_PASSWORD=$P4SSD -p MYSQL_DATABASE=bank \
> -p VOLUME_CAPACITY=10Gi \
> --as-deployment-config
```

Configuring Persistent Storage for OpenShift Applications

OpenShift Container Platform manages persistent storage as a set of pooled, cluster-wide resources. To add a storage resource to the cluster, an OpenShift administrator creates a **PersistentVolume** object that defines the necessary metadata for the storage resource. The metadata describes how the cluster accesses the storage, as well as other storage attributes such as capacity or throughput.

To list the **PersistentVolume** objects in a cluster, use the **oc get pv** command:

```
[admin@host ~]$ oc get pv
NAME      CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS     CLAIM      ...
pv0001    1Mi        RWO          Retain        Available   ...
pv0002    10Mi       RWX          Recycle       Available   ...
...output omitted...
```

To see the YAML definition for a given **PersistentVolume**, use the **oc get** command with the **-o yaml** option:

```
[admin@host ~]$ oc get pv pv0001 -o yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  creationTimestamp: ...value omitted...
  finalizers:
  - kubernetes.io/pv-protection
  labels:
    type: local
  name: pv0001
  resourceVersion: ...value omitted...
  selfLink: /api/v1/persistentvolumes/pv0001
  uid: ...value omitted...
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 1Mi
  hostPath:
    path: /data/pv0001
    type: ""
  persistentVolumeReclaimPolicy: Retain
status:
  phase: Available
```

To add more **PersistentVolume** objects to a cluster, use the **oc create** command:

```
[admin@host ~]$ oc create -f pv1001.yaml
```



Note

The above **pv1001.yaml** file must contain a persistent volume definition, similar in structure to the output of the **oc get pv *pv-name* -o yaml** command.

Requesting Persistent Volumes

When an application requires storage, you create a **PersistentVolumeClaim** (PVC) object to request a dedicated storage resource from the cluster pool. The following content from a file named **pvc.yaml** is an example definition for a PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

The PVC defines storage requirements for the application, such as capacity or throughput. To create the PVC, use the `oc create` command:

```
[admin@host ~]$ oc create -f pvc.yaml
```

After you create a PVC, OpenShift attempts to find an available **PersistentVolume** resource that satisfies the PVC's requirements. If OpenShift finds a match, it binds the PersistentVolume object to the PersistentVolumeClaim object. To list the PVCs in a project, use the `oc get pvc` command:

```
[admin@host ~]$ oc get pvc
NAME      STATUS    VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
myapp     Bound     pv0001   1Gi       RWO          6s
```

The output indicates whether a persistent volume is bound to the PVC, along with attributes of the PVC (such as capacity).

To use the persistent volume in an application pod, define a volume mount for a container that references the **PersistentVolumeClaim** object. The application pod definition below references a **PersistentVolumeClaim** object to define a volume mount for the application:

```
apiVersion: "v1"
kind: "Pod"
metadata:
  name: "myapp"
  labels:
    name: "myapp"
spec:
  containers:
    - name: "myapp"
      image: openshift/myapp
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/var/www/html"
          name: "pvol" ①
  volumes:
    - name: "pvol" ②
  persistentVolumeClaim:
    claimName: "myapp" ③
```

- ① This section declares that the **pvol** volume mounts at **/var/www/html** in the container file system.
- ② This section defines the **pvol** volume.
- ③ The **pvol** volume references the **myapp** PVC. If OpenShift associates an available persistent volume to the **myapp** PVC, then the **pvol** volume refers to this associated volume.

Configuring Persistent Storage with Templates

Templates are often used to simplify the creation of applications requiring persistent storage. Many of these templates have a suffix of **-persistent**:

```
[student@workstation ~]$ oc get templates -n openshift | grep persistent
cakephp-mysql-persistent   An example CakePHP application with a MySQL data...
dancer-mysql-persistent    An example Dancer application with a MySQL datab...
django-psql-persistent     An example Django application with a PostgreSQL ...
dotnet-psql-persistent     An example .NET Core application with a Postgres...
jenkins-persistent         Jenkins service, with persistent storage....
mariadb-persistent         MariaDB database service, with persistent storag...
mongodb-persistent         MongoDB database service, with persistent storag...
mysql-persistent            MySQL database service, with persistent storage...
nodejs-mongo-persistent   An example Node.js application with a MongoDB da...
postgresql-persistent      PostgreSQL database service, with persistent sto...
rails-psql-persistent      An example Rails application with a PostgreSQL d...
redis-persistent            Redis in-memory data structure store, with persi...
```

The following example template defines a **PersistentVolumeClaim** object along with a **DeploymentConfig** object:

```
apiVersion: template.openshift.io/v1
kind: Template
labels:
  template: myapp-persistent-template
metadata:
  name: myapp-persistent
  namespace: openshift
objects:
- apiVersion: v1
  kind: PersistentVolumeClaim ①
  metadata:
    name: ${APP_NAME}
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: ${VOLUME_CAPACITY}
- apiVersion: v1
  kind: DeploymentConfig ②
  metadata:
    name: ${APP_NAME}
  spec:
    replicas: 1
    selector:
      name: ${APP_NAME}
    strategy:
      type: Recreate
    template:
      metadata:
        labels:
          name: ${APP_NAME}
      spec:
        containers:
          - image: 'openshift/myapp'
            name: myapp
```

```

volumeMounts:
  - mountPath: /var/lib/myapp/data
    name: ${APP_NAME}-data ③
volumes:
  - name: ${APP_NAME}-data ④
    persistentVolumeClaim:
      claimName: ${APP_NAME}
parameters:
  - description: The name for the myapp application.
    displayName: Application Name
    name: APP_NAME ⑤
    required: true
    value: myapp
  - description: Volume space available for data, e.g. 512Mi, 2Gi.
    displayName: Volume Capacity
    name: VOLUME_CAPACITY ⑥
    required: true
    value: 1Gi

```

①② The template defines a **PersistentVolumeClaim** and **DeploymentConfig** object. Both objects have names matching the value of the **APP_NAME** parameter. The persistent volume claim defines a capacity corresponding the value of the **VOLUME_CAPACITY** parameter.

③④ The **DeploymentConfig** object defines a volume mount referencing the **PersistentVolumeClaim** created by the template.

⑤⑥ The template defines two parameters: **APP_NAME** and **VOLUME_CAPACITY**. The template uses these parameters to specify the value of attributes for the **PersistentVolumeClaim** and **DeploymentConfig** objects.

With this template, you only need to specify the **APP_NAME** and **VOLUME_CAPACITY** parameters to deploy the **myapp** application with persistent storage:

```
[student@workstation ~]$ oc create myapp-template.yaml
template.template.openshift.io/myapp-persistent created
[student@workstation ~]$ oc process myapp-persistent \
> -p APP_NAME=myapp-dev -p VOLUME_CAPACITY=1Gi \
> | oc create -f -
deploymentconfig/myapp created
persistentvolumeclaim/myapp created
```



References

Developer information about templates can be found in the *Using Templates* section of the OpenShift Container Platform documentation:

Developer Guide

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/images/index#using-templates

► Guided Exercise

Creating an Application with a Template

In this exercise, you will deploy the To Do List application in OpenShift Container Platform using a template to define resources your application needs to run.

Outcomes

You should be able to build and deploy an application in OpenShift Container Platform using a provided JSON template.

Before You Begin

You must have the To Do List application source code and lab files on **workstation**. To download the lab files and verify the status of the OpenShift cluster, run the following command in a new terminal window.

```
[student@workstation ~]$ lab multicontainer-openshift start
```

- 1. Use the **Dockerfile** in the **images/mysql** subdirectory to build the database container. Publish the container image to quay.io with a tag of **do180-mysql-57-rhel7**.

- 1.1. Build the MySQL database image.

```
[student@workstation ~]$ cd ~/D0180/labs/multicontainer-openshift/images/mysql
[student@workstation mysql]$ sudo podman build -t do180-mysql-57-rhel7 .
STEP 1: FROM rhscl/mysql-57-rhel7
Getting image source signatures
Copying blob sha256:e373541...output omitted...
  69.66 MB / 69.66 MB [=====] 6s
Copying blob sha256:c5d2e94...output omitted...
  1.20 KB / 1.20 KB [=====] 0s
Copying blob sha256:b3949ae...output omitted...
  62.03 MB / 62.03 MB [=====] 5s
Writing manifest to image destination
Storing signatures
STEP 2: ADD root /
ERRO[0001] HOSTNAME is not supported for OCI image format ...output omitted...
--> b628...a079
STEP 3: COMMIT do180-mysql-57-rhel7
```



Note

The **ERRO** error message is a known issue with an early version of Podman. You can ignore this message.

- 1.2. To make the image available for OpenShift, tag it and push it up to quay.io. To do so, run the following commands in the terminal window.

```
[student@workstation mysql]$ source /usr/local/etc/ocp4.config
[student@workstation mysql]$ sudo podman login quay.io -u ${RHT_OCP4_QUAY_USER}
Password: your_quay_password
Login Succeeded!
[student@workstation mysql]$ sudo podman tag \
> do180-mysql-57-rhel7 quay.io/${RHT_OCP4_QUAY_USER}/do180-mysql-57-rhel7
[student@workstation mysql]$ sudo podman push \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-mysql-57-rhel7
Getting image source signatures
Copying blob ssha256:04dbea...b21619
205.17 MB / 205.17 MB [=====] 1m19s
...output omitted...
Writing manifest to image destination
Storing signatures
```

**Note**

The **config sha256:** value in the above output may differ from your output.

**Warning**

Make sure the repository is public in quay.io so OpenShift can get the image from it. Refer to the **Repositories Visibility** section of the Appendix C to read details about how to change repository visibility.

- ▶ 2. Build the base image for the To Do List application using the Node.js Dockerfile, located in the exercise subdirectory **images/nodejs**. Tag the image as **do180-nodejs**. Do not publish this image to the registry.

```
[student@workstation mysql]$ cd ~/D0180/labs/multicontainer-openshift
[student@workstation multicontainer-openshift]$ cd images/nodejs
[student@workstation nodejs]$ sudo podman build --layers=false -t do180-nodejs .
STEP 1: ubi7/ubi:7.7
Getting image source signatures
Copying blob sha256:5d92fc...1ce84e
...output omitted...
Storing signatures
STEP 2: MAINTAINER username <username@example.com>
...output omitted...
STEP 8: CMD ["echo", "You must create your own container from this one."]
...output omitted...
STEP 9: COMMIT ...output omitted...localhost/do180-nodejs:latest
...output omitted...
```

- ▶ 3. Use the **build.sh** script in the **deploy/nodejs** subdirectory to build the To Do List application. Publish the application image to quay.io with an image tag of **do180-todonodejs**.
 - 3.1. Go to the **~/D0180/labs/multicontainer-openshift/deploy/nodejs** directory and run the **build.sh** command to build the child image.

```
[student@workstation nodejs]$ cd ~/DO180/labs/multicontainer-openshift
[student@workstation multicontainer-openshift]$ cd deploy/nodejs
[student@workstation nodejs]$ ./build.sh
Preparing build folder
STEP 1: FROM do180-nodejs
STEP 2: ARG NEXUS_BASE_URL
STEP 3: MAINTAINER username <username@example.com>
...output omitted...
STEP 7: CMD ["scl","enable","rh-nodejs8","./run.sh"]
STEP 8: COMMIT containers-storage:...output omitted...
```

3.2. Push the image to quay.io.

In order to make the image available for OpenShift to use in the template, tag it and push it to the private registry. To do so, run the following commands in the terminal window.

```
[student@workstation nodejs]$ sudo podman tag do180/todonodejs \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-todonodejs
[student@workstation nodejs]$ sudo podman push \
> quay.io/${RHT_OCP4_QUAY_USER}/do180-todonodejs
Getting image source signatures
Copying blob sha256:24a5c62...output omitted...
...output omitted...
Copying blob sha256:5f70bf1...output omitted...
1024 B / 1024 B [=====] 0s
Copying config sha256:c43306d...output omitted...
7.26 KB / 7.26 KB [=====] 0s
Writing manifest to image destination
Copying config sha256:c43306d...output omitted...
0 B / 7.26 KB [-----] 0s
Writing manifest to image destination
Storing signatures
```



Warning

Make sure the repository is public in quay.io so OpenShift can get the image from it. Refer to the **Repositories Visibility** section of the Appendix C to read details about how to change repository visibility.

▶ 4. Create the To Do List application from the provided JSON template.

4.1. Log in to OpenShift Container Platform.

```
[student@workstation nodejs]$ oc login -u ${RHT_OCP4_DEV_USER} \
> -p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.

...output omitted...

Using project "default".
```

If the **oc login** command prompts about using insecure connections, answer **y** (yes).

- 4.2. Create a new project *template* in OpenShift to use for this exercise. Run the following command to create the **template** project.

```
[student@workstation nodejs]$ oc new-project ${RHT_OCP4_DEV_USER}-template
Now using project ...output omitted...
```

- 4.3. Review the template.

Using your preferred editor, open and examine the template located at **/home/student/D0180/labs/multicontainer-openshift/todo-template.json**. Notice the following resources defined in the template and review their configurations.

- The **todoapi** pod definition defines the Node.js application.
- The **mysql** pod definition defines the MySQL database.
- The **todoapi** service provides connectivity to the Node.js application pod.
- The **mysql** service provides connectivity to the MySQL database pod.
- The **dbinit** persistent volume claim definition defines the MySQL **/var/lib/mysql/init** volume.
- The **dbclaim** persistent volume claim definition defines the MySQL **/var/lib/mysql/data** volume.

- 4.4. Process the template and create the application resources.

Use the **oc process** command to process the template file. This template requires the Quay.io namespace to retrieve the container images, as the **RHT_OCP4_QUAY_USER** parameter. Use the **pipe** command to send the result to the **oc create** command.

Run the following command in the terminal window:

```
[student@workstation nodejs]$ cd /home/student/D0180/labs/multicontainer-openshift
[student@workstation multicontainer-openshift]$ oc process \
> -f todo-template.json -p RHT_OCP4_QUAY_USER=${RHT_OCP4_QUAY_USER} \
> | oc create -f -
pod/mysql created
pod/todoapi created
service/todoapi created
service/mysql created
persistentvolumeclaim/dbinit created
persistentvolumeclaim/dbclaim created
```

- 4.5. Review the deployment.

Review the status of the deployment using the **oc get pods** command with the **-w** option to continue to monitor the pod status. Wait until both the containers are running. It may take some time for both pods to start.

```
[student@workstation multicontainer-openshift]$ oc get pods -w
NAME      READY   STATUS            RESTARTS   AGE
mysql     0/1    ContainerCreating   0          27s
todoapi   1/1    Running           0          27s
mysql     1/1    Running           0          27s
```

Press **Ctrl+C** to exit the command.

► 5. Expose the Service.

To allow the To Do List application to be accessible through the OpenShift router and to be available as a public FQDN, use the **oc expose** command to expose the **todoapi** service.

Run the following command in the terminal window.

```
[student@workstation multicontainer-openshift]$ oc expose service todoapi
route.route.openshift.io/todoapi exposed
```

► 6. Test the application.

- 6.1. Find the FQDN of the application by running the **oc status** command and note the FQDN for the app.

Run the following command in the terminal window.

```
[student@workstation multicontainer-openshift]$ oc status | grep -o "http:.*com"
http://todoapi-${{RHT_OCP4_QUAY_USER}}-template.${{RHT_OCP4_WILDCARD_DOMAIN}}
```

- 6.2. Use **curl** to test the REST API for the To Do List application.

```
[student@workstation multicontainer-openshift]$ curl -w "\n" \
> $(oc status | grep -o "http:.*com")/todo/api/items/1
{"id":1,"description":"Pick up newspaper","done":false}
```

The **-w "\n"** option with **curl** command lets the shell prompt appear at the next line rather than merging with the output in the same line.

- 6.3. Open Firefox on **workstation** and point your browser to **http://todoapi-\${{RHT_OCP4_QUAY_USER}}-template.\${{RHT_OCP4_WILDCARD_DOMAIN}}/todo/** and you should see the To Do List application.



Note

The trailing slash in the URL mentioned above is necessary. If you do not include that in the URL, you may encounter issues with the application.

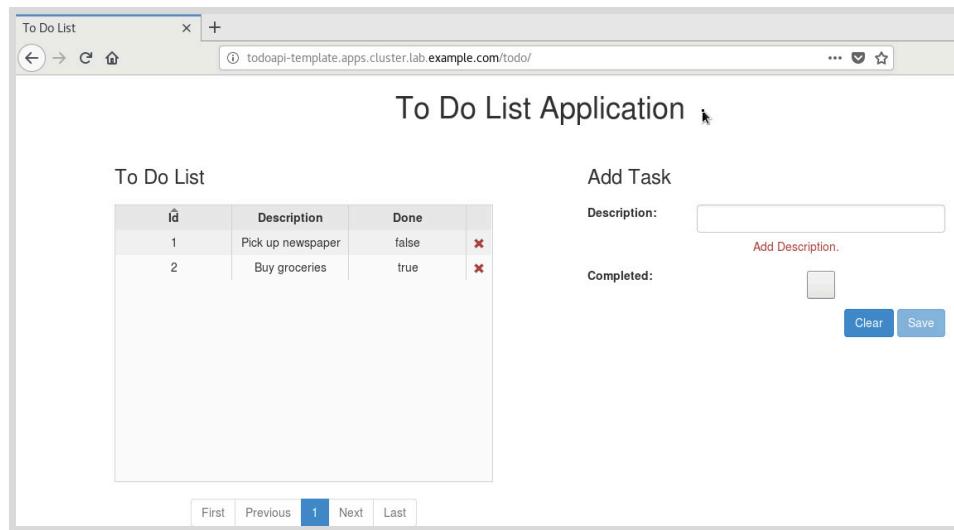


Figure 7.1: To Do List application

Finish

On **workstation**, run the **lab multicontainer-openshift finish** script to complete this lab.

```
[student@workstation ~]$ lab multicontainer-openshift finish
```

This concludes the guided exercise.

► Lab

Containerizing and Deploying a Software Application

In this review, you will containerize a Nexus server, build and test it using Podman, and deploy it to an OpenShift cluster.

Outcomes

You should be able to:

- Write a Dockerfile that successfully containerizes a Nexus server.
- Build a Nexus server container image and deploy it using Podman.
- Deploy the Nexus server container image to an OpenShift cluster.

Before You Begin

Run the set-up script for this comprehensive review.

```
[student@workstation ~]$ lab comprehensive-review start
```

The lab files are located in the **/home/student/D0180/labs/comprehensive-review** directory. The solution files are located in the **/home/student/D0180/solutions/comprehensive-review** directory.

Instructions

Use the following steps to create and test a containerized Nexus server both locally and in OpenShift:

Steps

1. Create a container image that starts an instance of a Nexus server:
 - The **/home/student/D0180/labs/comprehensive-review/image** directory contains files for building the container image. Execute the **get-nexus-bundle.sh** script to retrieve the Nexus server files.
 - Write a Dockerfile that containerizes the Nexus server. The Dockerfile must be located in the **/home/student/D0180/labs/comprehensive-review/image** directory. The Dockerfile must also:
 - Use a base image of **ubi7/ubi:7.7** and set an arbitrary maintainer.
 - Set the environment variable **NEXUS_VERSION** to **2.14.3-02**, and set **NEXUS_HOME** to **/opt/nexus**.
 - Install the *java-1.8.0-openjdk-devel* package

The RPM repositories are configured in the provided **training.repo** file. Be sure to add this file to the container in the **/etc/yum.repos.d** directory.

- Run a command to create a **nexus** user and group. They both have a UID and GID of **1001**.

- Unpack the **nexus-2.14.3-02-bundle.tar.gz** file to the **\${NEXUS_HOME} / \${NEXUS_HOME}/nexus2**, to create a symlink in the container. Run a command to recursively change the ownership of the Nexus home directory to **nexus:nexus**.

Run a command, **ln -s \${NEXUS_HOME}/nexus-\$NEXUS_VERSION \${NEXUS_HOME}/nexus2**, to create a symlink in the container. Run a command to recursively change the ownership of the Nexus home directory to **nexus:nexus**.

- Make the container run as the **nexus** user, and set the working directory to **/opt/nexus**.
- Define a volume mount point for the **/opt/nexus/sonatype-work** container directory. The Nexus server stores data in this directory.
- Set the default container command to **nexus-start.sh**.

There are two ***.snippet** files in the **/home/student/D0180/labs/comprehensive-review/images** directory that provide the commands needed to create the nexus account and install Java. Use the files to assist you in writing the Dockerfile.

- Build the container image with the name **nexus**.
2. Build and test the container image using Podman with a volume mount:
 - Use the script **/home/student/D0180/labs/comprehensive-review/deploy/local/run-persistent.sh** to start a new container with a volume mount.
 - Review the container logs to verify that the server is started and running.
 - Test access to the container service using the URL: **http://<container IP address>:8081/nexus**.
 - Remove the test container.
 3. Deploy the Nexus server container image to the OpenShift cluster. You must:
 - Tag the Nexus server container image as **quay.io/\${RHT_OCP4_QUAY_USER}/nexus:latest**, and push it to the private registry.
 - Create an OpenShift project with a name of **\${RHT_OCP4_DEV_USER}-review**.
 - Process the **deploy/openshift/resources/nexus-template.json** template and create the Kubernetes resources.
 - Create a route for the Nexus service. Verify that you can access **http://nexus- \${RHT_OCP4_DEV_USER}-review.\${RHT_OCP4_WILDCARD_DOMAIN}/nexus/** from **workstation**.

Evaluation

After deploying the Nexus server container image to the OpenShift cluster, verify your work by running the lab grading script:

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

Finish

On **workstation**, run the **lab comprehensive-review finish** command to complete this lab.

```
[student@workstation ~]$ lab comprehensive-review finish
```

This concludes the lab.

► Solution

Containerizing and Deploying a Software Application

In this review, you will containerize a Nexus server, build and test it using Podman, and deploy it to an OpenShift cluster.

Outcomes

You should be able to:

- Write a Dockerfile that successfully containerizes a Nexus server.
- Build a Nexus server container image and deploy it using Podman.
- Deploy the Nexus server container image to an OpenShift cluster.

Before You Begin

Run the set-up script for this comprehensive review.

```
[student@workstation ~]$ lab comprehensive-review start
```

The lab files are located in the **/home/student/D0180/labs/comprehensive-review** directory. The solution files are located in the **/home/student/D0180/solutions/comprehensive-review** directory.

Instructions

Use the following steps to create and test a containerized Nexus server both locally and in OpenShift:

Steps

1. Create a container image that starts an instance of a Nexus server:
 - The **/home/student/D0180/labs/comprehensive-review/image** directory contains files for building the container image. Execute the **get-nexus-bundle.sh** script to retrieve the Nexus server files.
 - Write a Dockerfile that containerizes the Nexus server. The Dockerfile must be located in the **/home/student/D0180/labs/comprehensive-review/image** directory. The Dockerfile must also:
 - Use a base image of **ubi7/ubi:7.7** and set an arbitrary maintainer.
 - Set the environment variable **NEXUS_VERSION** to **2.14.3-02**, and set **NEXUS_HOME** to **/opt/nexus**.
 - Install the *java-1.8.0-openjdk-devel* package

The RPM repositories are configured in the provided **training.repo** file. Be sure to add this file to the container in the **/etc/yum.repos.d** directory.

- Run a command to create a **nexus** user and group. They both have a UID and GID of **1001**.

- Unpack the **nexus-2.14.3-02-bundle.tar.gz** file to the **/\${NEXUS_HOME}/nexus-\${NEXUS_VERSION}** directory. Add the **nexus-start.sh** to the same directory.

Run a command, **ln -s \${NEXUS_HOME}/nexus-\${NEXUS_VERSION} \${NEXUS_HOME}/nexus2**, to create a symlink in the container. Run a command to recursively change the ownership of the Nexus home directory to **nexus:nexus**.

- Make the container run as the **nexus** user, and set the working directory to **/opt/nexus**.
- Define a volume mount point for the **/opt/nexus/sonatype-work** container directory. The Nexus server stores data in this directory.
- Set the default container command to **nexus-start.sh**.

There are two ***.snippet** files in the **/home/student/D0180/labs/comprehensive-review/images** directory that provide the commands needed to create the nexus account and install Java. Use the files to assist you in writing the Dockerfile.

- Build the container image with the name **nexus**.

- 1.1. Execute the **get-nexus-bundle.sh** script to retrieve the Nexus server files.

```
[student@workstation ~]$ cd /home/student/D0180/labs/comprehensive-review/image
[student@workstation image]$ ./get-nexus-bundle.sh
#####
# Nexus bundle download successful
```

- 1.2. Write a Dockerfile that containerizes the Nexus server. Go to the **/home/student/D0180/labs/comprehensive-review/image** directory and create the Dockerfile.

- 1.2.1. Specify the base image to use:

```
FROM ubi7/ubi:7.7
```

- 1.2.2. Enter an arbitrary name and email as the maintainer:

```
FROM ubi7/ubi:7.7

MAINTAINER username <username@example.com>
```

- 1.2.3. Set a build argument for **NEXUS_VERSION** and an environment variable for **NEXUS_HOME**:

```
FROM ubi7/ubi:7.7

MAINTAINER username <username@example.com>

ARG NEXUS_VERSION=2.14.3-02
ENV NEXUS_HOME=/opt/nexus
```

- 1.2.4. Add the **training.repo** repository to the **/etc/yum.repos.d** directory.
Install the Java package using **yum** command.

```
...
ARG NEXUS_VERSION=2.14.3-02
ENV NEXUS_HOME=/opt/nexus

RUN yum install -y --setopt=tsflags=nodocs java-1.8.0-openjdk-devel && \
    yum clean all -y
```

- 1.2.5. Create the server home directory and service account and group. Make home directory owned by the service account.

```
...
RUN groupadd -r nexus -f -g 1001 && \
    useradd -u 1001 -r -g nexus -m -d ${NEXUS_HOME} -s /sbin/nologin \
    -c "Nexus User" nexus && \
    chown -R nexus:nexus ${NEXUS_HOME} && \
    chmod -R 755 ${NEXUS_HOME}
```

- 1.2.6. Make the container run as the **nexus** user.

```
...
USER nexus
```

- 1.2.7. Install the Nexus server software at **NEXUS_HOME** and add the startup script.
Note that the **ADD** directive will extract the Nexus files.

Create the **nexus2** symbolic link pointing to the Nexus server directory.
Recursively change the ownership of the **\${NEXUS_HOME}** directory to **nexus:nexus**.

```
...
ADD nexus-${NEXUS_VERSION}-bundle.tar.gz ${NEXUS_HOME}
ADD nexus-start.sh ${NEXUS_HOME}/

RUN ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} \
    ${NEXUS_HOME}/nexus2
```

- 1.2.8. Make **/opt/nexus** the current working directory:

```
...
WORKDIR ${NEXUS_HOME}
```

1.2.9. Define a volume mount point to store the Nexus server persistent data:

```
...
VOLUME ["/opt/nexus/sonatype-work"]
```

1.2.10. Set the **CMD** instruction to **["sh", "nexus-start.sh"]**. The completed Dockerfile reads as follows:

```
FROM ubi7/ubi:7.7

MAINTAINER username <username@example.com>

ARG NEXUS_VERSION=2.14.3-02
ENV NEXUS_HOME=/opt/nexus

RUN yum install -y --setopt=tsflags=nodocs java-1.8.0-openjdk-devel && \
    yum clean all -y

RUN groupadd -r nexus -f -g 1001 && \
    useradd -u 1001 -r -g nexus -m -d ${NEXUS_HOME} -s /sbin/nologin \
        -c "Nexus User" nexus && \
    chown -R nexus:nexus ${NEXUS_HOME} && \
    chmod -R 755 ${NEXUS_HOME}

USER nexus

ADD nexus-${NEXUS_VERSION}-bundle.tar.gz ${NEXUS_HOME}
ADD nexus-start.sh ${NEXUS_HOME}/

RUN ln -s ${NEXUS_HOME}/nexus-${NEXUS_VERSION} \
    ${NEXUS_HOME}/nexus2

WORKDIR ${NEXUS_HOME}

VOLUME ["/opt/nexus/sonatype-work"]

CMD ["sh", "nexus-start.sh"]
```

1.3. Build the container image with the name **nexus**.

```
[student@workstation image]$ sudo podman build --layers=false -t nexus .
STEP 1: FROM ubi7/ubi:7.7
Getting image source signatures
...output omitted...
STEP 14: COMMIT ...output omitted...localhost/nexus:latest
...output omitted...
```

2. Build and test the container image using Podman with a volume mount:

- Use the script **/home/student/D0180/labs/comprehensive-review/deploy/local/run-persistent.sh** to start a new container with a volume mount.
- Review the container logs to verify that the server is started and running.

- Test access to the container service using the URL: `http://<container IP address>:8081/nexus`.
 - Remove the test container.
- 2.1. Execute the `run-persistent.sh` script. Replace the container name as shown in the output of the `podman ps` command.

```
[student@workstation images]$ cd /home/student/D0180/labs/comprehensive-review
[student@workstation comprehensive-review]$ cd deploy/local
[student@workstation local]$ ./run-persistent.sh
80970007036bbb313d8eeb7621fada0ed3f0b4115529dc50da4dccef0da34533
```

- 2.2. Review the container logs to verify that the server is started and running.

```
[student@workstation local]$ sudo podman ps \
> --format="{{.ID}} {{.Names}} {{.Image}}"
81f480f21d47 inspiring_poincare localhost/nexus:latest
[student@workstation local]$ sudo podman logs inspiring_poincare
...output omitted...
... INFO [jetty-main-1] ...jetty.JettyServer - Running
... INFO [main] ...jetty.JettyServer - Started
```

- 2.3. Inspect the running container to determine its IP address. Provide this IP address to the `curl` command to test the container.

```
[student@workstation local]$ sudo podman inspect \
> -f '{{.NetworkSettings.IPAddress}}' inspiring_poincare
10.88.0.12
[student@workstation local]$ curl -v 10.88.0.12:8081/nexus/ 2>&1 \
> | grep -E 'HTTP|<title>' 
> GET /nexus/ HTTP/1.1
< HTTP/1.1 200 OK
<title>Nexus Repository Manager</title>
```

- 2.4. Remove the test container.

```
[student@workstation local]$ sudo podman kill inspiring_poincare
81f480f21d475af683b4b003ca6e002d37e6aaa581393d3f2f95a1a7b7eb768b
```

3. Deploy the Nexus server container image to the OpenShift cluster. You must:
- Tag the Nexus server container image as `quay.io/${RHT_OCP4_QUAY_USER}/nexus:latest`, and push it to the private registry.
 - Create an OpenShift project with a name of `${RHT_OCP4_DEV_USER}-review`.
 - Process the `deploy/openshift/resources/nexus-template.json` template and create the Kubernetes resources.
 - Create a route for the Nexus service. Verify that you can access `http://nexus- ${RHT_OCP4_DEV_USER}-review.${RHT_OCP4_WILDCARD_DOMAIN}/nexus/` from **workstation**.

- 3.1. Log in to your Quay.io account.

```
[student@workstation local]$ sudo podman login -u ${RHT_OCP4_QUAY_USER} quay.io
Password: your_quay_password
Login Succeeded!
```

- 3.2. Publish the Nexus server container image to your quay.io registry.

```
[student@workstation local]$ sudo podman push localhost/nexus:latest \
> quay.io/${RHT_OCP4_QUAY_USER}/nexus:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

- 3.3. Repositories created by pushing images to quay.io are private by default. Refer to the **Repositories Visibility** section of the Appendix C to read details about how change repository visibility.

- 3.4. Create the OpenShift project:

```
[student@workstation local]$ cd ~/DO180/labs/comprehensive-review/deploy/openshift
[student@workstation openshift]$ oc login -u ${RHT_OCP4_DEV_USER} \
> -p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
[student@workstation openshift]$ oc new-project ${RHT_OCP4_DEV_USER}-review
Now using project ...output omitted...
```

- 3.5. Process the template and create the Kubernetes resources:

```
[student@workstation openshift]$ oc process -f resources/nexus-template.json \
> -p RHT_OCP4_QUAY_USER=${RHT_OCP4_QUAY_USER} \
> | oc create -f -
service/nexus created
persistentvolumeclaim/nexus created
deploymentconfig.apps.openshift.io/nexus created
[student@workstation openshift]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
nexus-1-wk8rv   1/1     Running   1          1m
nexus-1-deploy   0/1     Completed  0          2m
```

- 3.6. Expose the service by creating a route:

```
[student@workstation openshift]$ oc expose svc/nexus
route.route.openshift.io/nexus exposed.
[student@workstation openshift]$ oc get route -o yaml
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
...output omitted...
```

```
spec:  
  host: nexus-${RHT_OCP4_DEV_USER}-review.${RHT_OCP4_WILDCARD_DOMAIN}  
  ...output omitted...
```

- 3.7. Use a browser to connect to the Nexus server web application at `http://nexus-${RHT_OCP4_DEV_USER}-review.${RHT_OCP4_WILDCARD_DOMAIN}/nexus/`.

Evaluation

After deploying the Nexus server container image to the OpenShift cluster, verify your work by running the lab grading script:

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

Finish

On `workstation`, run the `lab comprehensive-review finish` command to complete this lab.

```
[student@workstation ~]$ lab comprehensive-review finish
```

This concludes the lab.

Summary

In this chapter, you learned:

- Kubernetes and OpenShift create a software-defined network between all containers in a pod.
- Within the same project, Kubernetes injects a set of variables for each service into all pods.
- Kubernetes templates automate creating applications consisting of multiple resources.
Template parameters allow using the same values when creating multiple resources.

Chapter 8

Describing Red Hat OpenShift Container Platform

Goal

Describe the architecture of OpenShift Container Platform.

Objectives

- Describe the typical usage of the product and its features.
- Describe the architecture of Red Hat OpenShift Container Platform.
- Describe what a cluster operator is, how it works, and name the major cluster operators.

Sections

- Describing OpenShift Container Platform Features (and Quiz)
- Describing the Architecture of OpenShift (and Quiz)
- Describing Cluster Operators (and Quiz)

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), Google Cloud Platform (GCP), Microsoft Azure, or IBM Cloud. 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 OpenShift Kubernetes Engine

Provides a subset of the features present in OpenShift Container Platform, such as the Red Hat Enterprise Linux CoreOS lightweight transactional operating system, the CRI-O engine, the Kubernetes container orchestration platform, and the core cluster services (web console, Over-the-air updates, internal registry, and Operator Lifecycle Manager, among others).

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 RHOC, 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:

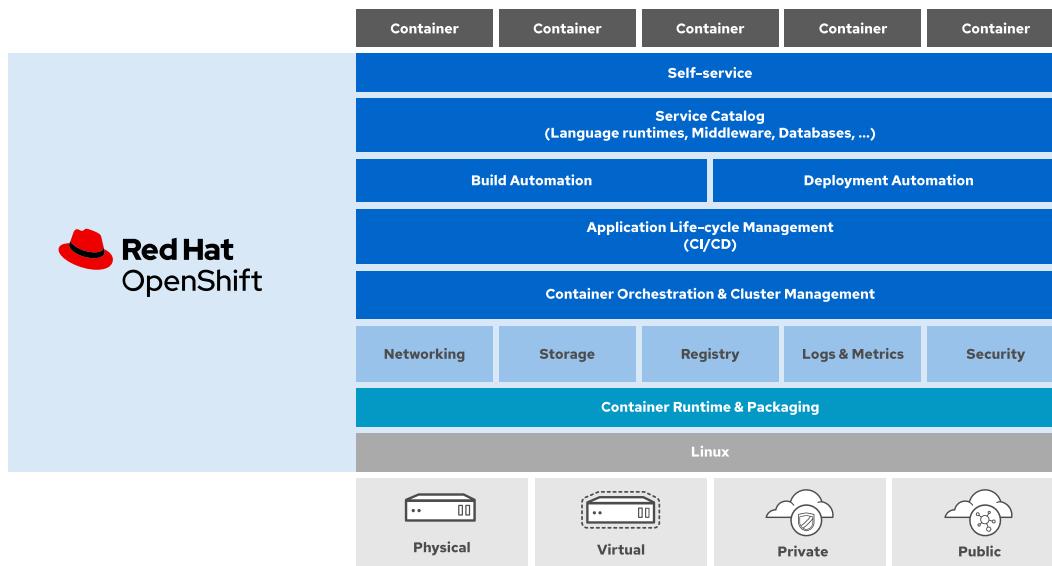


Figure 8.1: OpenShift services and features

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.

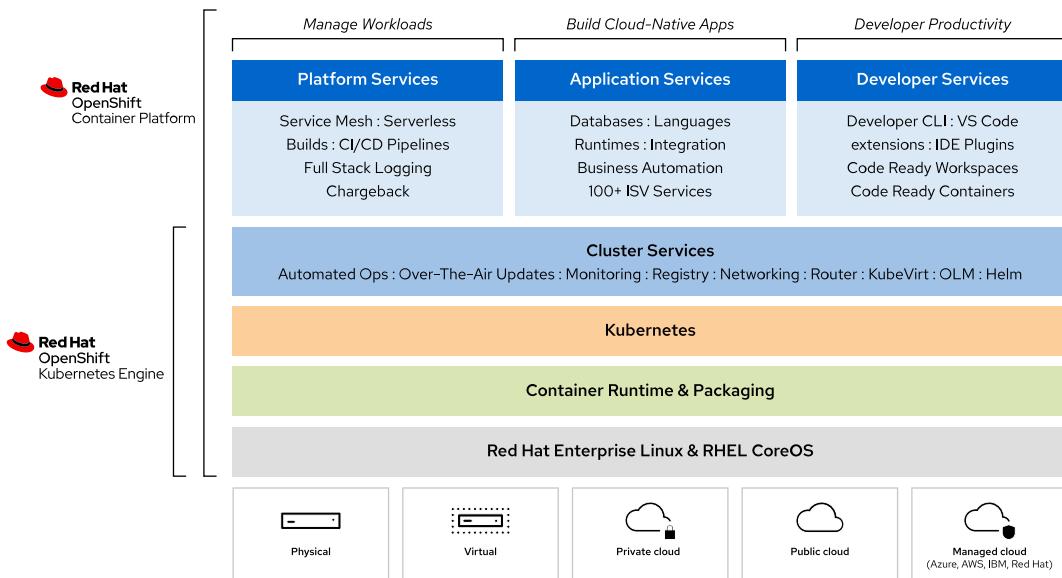


Figure 8.2: Feature comparison between OpenShift Container Platform and OpenShift Kubernetes Engine

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 to 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.

Red Hat Marketplace is a platform that allows access to certified software packaged as *Kubernetes operators* that can be deployed in an OpenShift cluster. The certified software includes automatic deployments and seamless upgrades for an integrated experience.



References

Further information is available in the Red Hat OpenShift Container Platform 4.5

product documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/

Red Hat OpenShift Kubernetes Engine

<https://www.openshift.com/products/kubernetes-engine>

► 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 the Architecture of OpenShift

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.

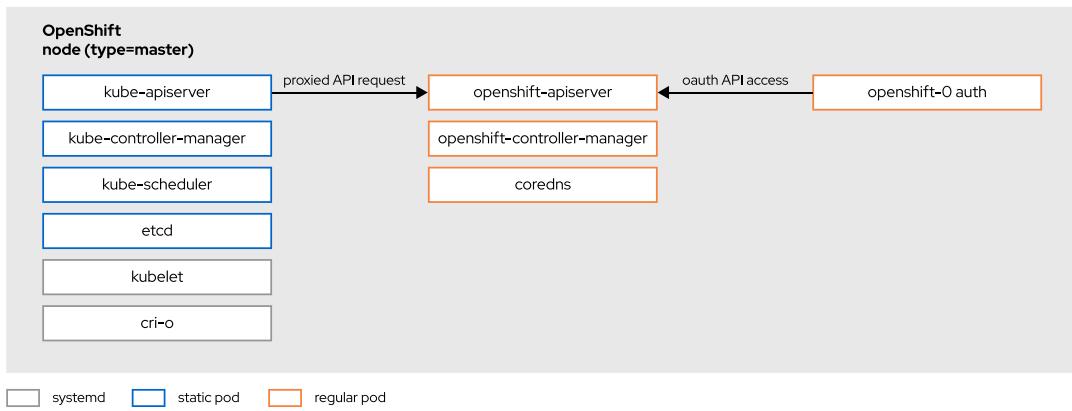
Kubernetes provides a REST API to manage these resources. All actions that an OpenShift user takes, either using the command-line interface or the web console, are performed by invoking this REST API.

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

The following graphic provides an overview of an OpenShift control plane node, illustrating the main processes that run in a regular node and in a control plane node, as either system services or containers.

**Figure 8.3: Architecture of an OpenShift control plane 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 available in the Red Hat OpenShift Container Platform 4.5 product documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/

► Quiz

Describing the Architecture of OpenShift

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 API components?**

- a. Worker nodes
- b. Nodes
- c. Control plane nodes

► 4. **Which of the 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 the Architecture of OpenShift

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 API components?**

- a. Worker nodes
- b. Nodes
- c. Control plane nodes

► 4. **Which of the 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 by 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 Red Hat Marketplace

Red Hat Marketplace is a platform that allows access to a curated set of enterprise grade operators that can be deployed on an OpenShift or a Kubernetes cluster. Operators available in the Red Hat Marketplace have gone through a certification process to ensure the software follows best practices and also the containers are scanned for vulnerabilities.

The Red Hat Marketplace Operator allows a seamless integration between an OpenShift cluster and the Red Hat Marketplace. This integration manages updates and consolidates billing and reporting, simplifying the deployment of certified operators. Vendors provide several pricing options for their operators, such as free trials, different editions, and discounts for large customers.

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 control plane 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.

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



References

Further information is available in the Red Hat OpenShift Container Platform 4.5 product documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/

Introducing the Operator Framework

<https://blog.openshift.com/introducing-the-operator-framework/>

Getting started with Red Hat Marketplace

<https://marketplace.redhat.com/en-us/documentation/getting-started>

► 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

Red Hat Marketplace

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.	
Platform that allows access to certified software packaged as Kubernetes operators that can be deployed in an OpenShift cluster.	

► 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
Platform that allows access to certified software packaged as Kubernetes operators that can be deployed in an OpenShift cluster.	Red Hat Marketplace

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 9

Verifying a Cluster

Goal

Describe OpenShift installation methods and verify the health of a newly installed cluster.

Objectives

- Describe the OpenShift installation process, full-stack automation, and pre-existing infrastructure installation methods.
- Execute commands that assist in troubleshooting common problems.
- Identify the components and resources of persistent storage and deploy an application that uses a persistent volume claim.

Sections

- Describing Installation Methods (and Quiz)
- Troubleshooting OpenShift Clusters and Applications (and Guided Exercise)
- Introducing OpenShift Dynamic Storage (and Guided Exercise)

Describing Installation Methods

Objectives

After completing this section, you should be able to describe the OpenShift installation process, full-stack automation, and pre-existing infrastructure installation methods.

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 4.5 release, the set of cloud providers supported for the full-stack automation method includes Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Red Hat OpenStack Platform using the standard Intel architecture (x86). Supported virtualization providers and architectures for full-stack automation include VMware, Red Hat Virtualization, IBM Power, and IBM System Z.

Every minor release of the 4.x stream adds more capabilities and more support for customizations, such as reusing precreated cloud resources.

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 set up 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:

- Defining custom storage classes for dynamic storage provisioning.
- 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 Red Hat OpenShift Container Platform 4.5 *Installing* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/installing/index

For more information on Installer Provisioned Infrastructure, refer to the Red Hat OpenShift Container Platform 4.5 *OpenShift 4.x Installation - Quick Overview (IPI Installation)* video at
<https://www.youtube.com/watch?v=uBsilb4cual>

For more information on User Provisioned Infrastructure, refer to the Red Hat OpenShift Container Platform 4.5 *OpenShift 4 User Provisioned Infrastructure with VMware vSphere* video at
<https://www.youtube.com/watch?v=TsAJEEDv-gg>

► 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 control plane 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 control plane 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.

Troubleshooting OpenShift Clusters and Applications

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

The OpenShift installer creates an **auth** directory containing the **kubeconfig** and **kubeadm-password** files. Run the **oc login** command to connect to the cluster with the **kubeadm** user. The password of the **kubeadm** user is in the **kubeadm-password** file.

```
[user@dem ~]$ oc login -u kubeadm -p MMTUC-TnXjo-NFyh3-aewmc
>     https://api.ocp4.example.com:6443
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, 4.5.4). Use this resource to declare the version of the cluster you want 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, and 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   4.5.4        True          False           4d23h     Cluster version is 4.5.4
```

- 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-4.5 ①
  Cluster ID:  f33267f8-260b-40c1-9cf3-ecc406ce035e ②
  Upstream:     https://api.openshift.com/api/upgrades_info/v1/graph ③
Status:
  Available Updates: ④
    Force:      false
    Image:     quay.io/openshift-release-dev/ocp-release@sha256:...
    Version:   4.5.5
  Conditions:
    Last Transition Time: 2020-08-05T18:35:08Z
    Message:      Done applying 4.5.4 ⑤
    Status:       True
    Type:        Available
...output omitted...
```

```

Desired:
  Force:    false
  Image:    quay.io/openshift-release-dev/ocp-release@sha256:...
  Version:  4.5.4
...output omitted...
History:
  Completion Time: 2020-08-05T18:35:08Z ⑥
  Image:          quay.io/openshift-release-dev/ocp-release@sha256:...
  Started Time:   2020-08-05T18:22:42Z
  State:          Completed ⑦
  Verified:       false
  Version:        4.5.4
  Observed Generation: 2
...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. Red Hat uses this identifier 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.
- 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 if 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  DEGRADED  SINCE
authentication ①      4.5.4     True       False        False     3h58m
cloud-credential 4.5.4     True       False        False     4d23h
cluster-autoscaler 4.5.4     True       False        False     4d23h
config-operator   4.5.4     True       False        False     4d23h
console          4.5.4     True       False        False     3h58m
csi-snapshot-controller 4.5.4     True       False        False     4d23h
dns              4.5.4     True       False        False     4d23h
etcd             4.5.4     True       False        False     4d23h
image-registry    4.5.4     True       False        False     4d23h
...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 **cricctl** 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 **--loglevel 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 --loglevel 6
```

```
[user@demo ~]$ oc get pod --loglevel 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, refer to the *Viewing system event information in an OpenShift Container Platform cluster* section in the *Working with clusters* chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-events

For more information about how to copy files to running containers, refer to the *Copying files to or from an OpenShift Container Platform container* section in the *Working with containers* chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-copying-files

For more information about how to execute commands on running containers, refer to the *Executing remote commands in an OpenShift Container Platform container* section in the *Working with containers* chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-remote-commands

For more information about how to forward local ports to running containers, refer to the *Using port forwarding to access applications in a container* section in the *Working with containers* chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-containers-port-forwarding

For more information about aggregated logging, refer to the Red Hat OpenShift Container Platform 4.5 *Logging* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/logging/index

ClusterOperator Custom Resource

<https://github.com/openshift/cluster-version-operator/blob/master/docs/dev/clusteroperator.md>

► Guided Exercise

Troubleshooting OpenShift Clusters and Applications

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

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 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_PWD} \
> https://api.ocp4.example.com:6443
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.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES          AGE     VERSION
master01   Ready     master,worker  2d      v1.18.3+012b3ec
master02   Ready     master,worker  2d      v1.18.3+012b3ec
master03   Ready     master,worker  2d      v1.18.3+012b3ec
```

- 1.4. Verify whether any of your nodes are close to using all of the CPU and memory available to them.

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
NAME      CPU(cores)   CPU%    MEMORY(bytes)   MEMORY%
master01   499m        14%     3235Mi         21%
master02   769m        21%     4933Mi         33%
master03   1016m       29%     6087Mi         40%
```

- 1.5. Use the **oc describe** command to verify that all of the conditions that might indicate problems are false.

```
[student@workstation ~]$ oc describe node master01
...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 1 -n openshift-image-registry \
>   image-registry-794dfc7978-w7w69
time="2019-09-25T21:18:06.827703727Z" level=info msg=response
go.version=g01.11.6 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 1 -u kubelet master01
-- Logs begin at Wed 2020-07-29 18:24:47 UTC, end at Fri 2020-07-31 20:55:53 UTC.
--
Jul 31 16:23:27.420689 master01 systemd[1]: kubelet.service: Consumed 11min
12.108s CPU time
-- Logs begin at Wed 2020-07-29 18:24:47 UTC, end at Fri 2020-07-31 20:55:53 UTC.
--
Jul 31 20:55:53.080133 master01 hyperkube[1524]: I0731 20:55:53.080122      1524
prober.go:133] Liveness probe for "ovs-22g2c_openshift-sdn(d118e034-2de7-447f-
a79b-fed5f9863840):openvswitch" succeeded
```

- 3. Start a shell session to the same 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.

Chapter 9 | Verifying a Cluster

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

```
[student@workstation ~]$ oc debug node/master01
Starting pod/master01-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
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 - MCO environment configuration
  Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
            └─10-mco-default-env.conf
  Active: active (running) since Fri 2020-07-31 16:26:57 UTC; 4h 32min 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
● cri-o.service - MCO environment configuration
  Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/crio.service.d
            └─10-mco-default-env.conf
  Active: active (running) since Fri 2020-07-31 16:26:50 UTC; 4h 35min 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   0           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.ocp4.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        ...
pgsql-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/pgsql 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/pgsql-7d4cc9d6d-m5r59   Successfully
assigned execute-troubleshoot/pgsql-7d4cc9d6d-m5r59 to ip-10-0-143-197.us-
west-1.compute.internal
21s          Normal    Pulling         pod/pgsql-7d4cc9d6d-m5r59   Pulling
image "registry.access.redhat.com/rhscl/postgresq-96-rhel7:1"
21s          Warning   Failed         pod/pgsql-7d4cc9d6d-m5r59   Failed to
pull image "registry.access.redhat.com/rhscl/postgresq-96-rhel7:1": rpc error:
code = Unknown desc = Error reading manifest 1 in registry.access.redhat.com/
rhscl/postgresq-96-rhel7: name unknown: Repo not found
21s          Warning   Failed         pod/pgsql-7d4cc9d6d-m5r59   Error:
ErrImagePull
8s           Normal    BackOff        pod/pgsql-7d4cc9d6d-m5r59   Back-off
pulling image "registry.access.redhat.com/rhscl/postgresq-96-rhel7:1"
```

Chapter 9 | Verifying a Cluster

```
8s      Warning  Failed          pod/sql-7d4cc9d6d-m5r59  Error:
ImagePullBackOff
112s    Normal   SuccessfulCreate replicaset/sql-7d4cc9d6d  Created pod:
sql-7d4cc9d6d-m5r59
112s    Normal   ScalingReplicaSet deployment/sql           Scaled up
replica set sql-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[0000] Error parsing image name "docker://registry.access.redhat.com/rhscl/
postgresq-96-rhel7:1": 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 **sql** deployment to correct the name of the container image. The **oc edit** command 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/sql
...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  
pgsql-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.

Introducing OpenShift Dynamic Storage

Objectives

After completing this section, you should be able to identify the components and resources of persistent storage and deploy an application that uses a persistent volume claim.

Persistent Storage Overview

Containers have ephemeral storage by default. For example, when a container is deleted, all the files and data inside it are deleted also. To preserve the files, containers offer two main ways of maintaining persistent storage: volumes and bind mounts. Volumes are the preferred OpenShift way of managing persistent storage. Volumes are managed manually by the administrator or dynamically through a storage class. Developers working with containers on a local system can mount a local directory into a container using a bind mount.

OpenShift cluster administrators use the Kubernetes persistent volume framework to manage persistent storage for the users of a cluster. There are two ways of provisioning storage for the cluster: static and dynamic. Static provisioning requires the cluster administrator to create persistent volumes manually. Dynamic provisioning uses storage classes to create the persistent volumes on demand.

The OpenShift Container Platform uses storage classes to allow administrators to provide persistent storage. Storage classes are a way to describe types of storage for the cluster and provision dynamic storage on demand.

Developers use persistent volume claims to add persistent volumes dynamically to their applications; it is not necessary for the developer to know details of the storage infrastructure. With static provisioning, developers use precreated PVs or ask a cluster administrator to manually create persistent volumes for their applications.

A persistent volume claim (PVC) belongs to a specific project. To create a PVC, you must specify the access mode and size, among other options. Once created, a PVC cannot be shared between projects. Developers use a PVC to access a persistent volume (PV). Persistent volumes are not exclusive to projects and are accessible across the entire OpenShift cluster. When a persistent volume binds to a persistent volume claim, the persistent volume cannot be bound to another persistent volume claim.

Persistent Volume and Persistent Volume Claim Life Cycle

Persistent volume claims request persistent volume resources. To be eligible, a PV must not be bound to another PVC. Additionally, the PV must provide the access mode specified in the PVC and it must be at least as large as the size requested in the PVC. A PVC can specify additional criteria, such as the name of a storage class. If a PVC cannot find a PV that matches all criteria, the PVC enters a pending state and waits until an appropriate PV becomes available. A cluster administrator can manually create the PV or a storage class can dynamically create the PV. A bound persistent volume can be mounted as a volume to a specific mount point in the pod (for example, /var/lib/pgsql for a PostgreSQL database).

Verifying the Dynamic Provisioned Storage

Use the **oc get storageclass** command to view available storage classes. The output identifies the default storage class. If a storage class exists, then persistent volumes are created dynamically to match persistent volume claims. A persistent volume claim that does not specify a storage class uses the default storage class.

```
[user@demo ~]$ oc get storageclass
NAME          PROVISIONER      ...
nfs-storage   (default)    nfs-storage-provisioner ...
```



Note

The classroom environment uses an external, open source NFS provisioner. The provisioner dynamically creates NFS persistent volumes from an existing NFS server. Red Hat does not recommend using this provisioner in production environments.

Deploying Dynamically Provisioned Storage

To add a volume to an application create a **PersistentVolumeClaim** resource and add it to the application as a volume. Create the persistent volume claim using either a Kubernetes manifest or the **oc set volumes** command. In addition to either creating a new persistent volume claim or using an existing persistent volume claim, the **oc set volumes** command can modify a deployment or a deployment configuration to mount the persistent volume claim as a volume within the pod.

To add a volume to an application, use the **oc set volumes** command:

```
[user@demo ~]$ oc set volumes deployment/example-application \
> --add --name example-storage --type pvc --claim-class nfs-storage \
> --claim-mode rwo --claim-size 15Gi --mount-path /var/lib/example-app \
> --claim-name example-storage
```

The command creates a persistent volume claim resource and adds it to the application as a volume within the pod.

The following YAML example specifies a persistent volume claim.

To create a **PersistentVolumeClaim** API object:

```
apiVersion: v1
kind: PersistentVolumeClaim ①
metadata:
  name: example-pv-claim ②
  labels:
    app: example-application
spec:
  accessModes:
    - ReadWriteOnce ③
```

```
resources:
  requests:
    storage: 15Gi ④
```

- ① Indicates that it is a persistent volume claim.
- ② The name to use in the **claimName** field of the **persistentVolumeClaim** element in the **volumes** section of a deployment manifest.
- ③ The storage class provisioner must provide this access mode. If persistent volumes are created statically, then an eligible persistent volume must provide this access mode.
- ④ The storage class will create a persistent volume matching this size request. If persistent volumes are created statically, then an eligible persistent volume must be at least the requested size.

OpenShift defines three access modes that are summarized in the following table.

Access Mode	CLI Abbreviation	Description
ReadWriteMany	RWX	Kubernetes can mount the volume as read-write on many nodes.
ReadOnlyMany	ROX	Kubernetes can mount the volume as read-only on many nodes.
ReadWriteOnce	RWO	Kubernetes can mount the volume as read-write on only a single node.

To add the PVC to the application:

```
...output omitted...
spec:
  volumes:
    - name: example-pv-storage
      persistentVolumeClaim:
        claimName: example-pv-claim
  containers:
    - name: example-application
      image: registry.redhat.io/rhel8/example-app
      ports:
        - containerPort: 1234
      volumeMounts:
        - mountPath: "/var/lib/example-app"
          name: example-pv-storage
...output omitted...
```

Deleting Persistent Volume Claims

To delete a volume, use the **oc delete** command to delete the persistent volume claim. The storage class will reclaim the volume after the PVC is removed.

```
[user@demo ~]$ oc delete pvc/example-pvc-storage
```



References

For more information on persistent storage, refer to the *Understanding persistent storage* chapter in the Red Hat OpenShift Container Platform 4.5 Storage documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/storage/index#understanding-persistent-storage

For more information on ephemeral storage, refer to the *Understanding ephemeral storage* chapter of the Red Hat OpenShift Container Platform 4.5 Storage documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/storage/index#understanding-ephemeral-storage

► Guided Exercise

Introducing OpenShift Dynamic Storage

In this exercise, you will deploy a PostgreSQL database using a persistent volume claim and identify its dynamically allocated volume.

Outcomes

You should be able to:

- Identify the default storage settings of an OpenShift cluster.
- Create persistent volume claims.
- Manage persistent volumes.

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 downloads files required for this exercise.

```
[student@workstation ~]$ lab install-storage start
```

► 1. Log in to the OpenShift cluster.

- 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. If prompted, accept the insecure certificate.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
> https://api.ocp4.example.com:6443
...output omitted...
```

► 2. Create a new project named **install-storage**.

```
[student@workstation ~]$ oc new-project install-storage
Now using project "install-storage" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

► 3. Verify the default storage class.

```
[student@workstation ~]$ oc get storageclass
NAME           PROVISIONER          RECLAIMPOLICY   ...
nfs-storage (default) nfs-storage-provisioner Delete     ...
```

- 4. Create a new database deployment using the container image located at `registry.redhat.io/rhel8/postgresql-12:1-43`.

```
[student@workstation ~]$ oc new-app --name postgresql-persistent \
>   --docker-image registry.redhat.io/rhel8/postgresql-12:1-43 \
>   -e POSTGRESQL_USER=redhat \
>   -e POSTGRESQL_PASSWORD=redhat123 \
>   -e POSTGRESQL_DATABASE=persistentdb
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "postgresql-persistent" created
deployment.apps "postgresql-persistent" created
service "postgresql-persistent" created
--> Success
...output omitted...
```

**Note**

For convenience, the `~/DO280/labs/install-storage/commands.txt` file contains some commands that you can copy and paste.

- 5. Add a persistent volume for the PostgreSQL database.

- 5.1. Create a new persistent volume claim to add a new volume to the **postgresql-persistent** deployment.

```
[student@workstation ~]$ oc set volumes deployment/postgresql-persistent \
>   --add --name postgresql-storage --type pvc --claim-class nfs-storage \
>   --claim-mode rwo --claim-size 10Gi --mount-path /var/lib/pgsql \
>   --claim-name postgresql-storage
deployment.apps/postgresql-persistent volume updated
```

- 5.2. Verify that you successfully created the new PVC.

```
[student@workstation ~]$ oc get pvc
NAME           STATUS  ...  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
postgresql-storage  Bound  ...  10Gi      RWO          nfs-storage  25s
```

- 5.3. Verify that you successfully created the new PV.

```
[student@workstation ~]$ oc get pv \
>   -o custom-columns=NAME:.metadata.name,CLAIM:.spec.claimRef.name
NAME           CLAIM
pvc-26cc804a-4ec2-4f52-b6e5-84404b4b9def  image-registry-storage
pvc-65c3cce7-45eb-482d-badf-a6469640bd75  postgresql-storage
```

- 6. Populate the database using the `~/DO280/labs/install-storage/init_data.sh` script.

- 6.1. Execute the `init_data.sh` script.

```
[student@workstation ~]$ cd ~/DO280/labs/install-storage
[student@workstation install-storage]$ ./init_data.sh
Populating characters table
CREATE TABLE
INSERT 0 5
```

- 6.2. Use the `~/DO280/labs/install-storage/check_data.sh` script to verify that the database was populated successfully.

```
[student@workstation install-storage]$ ./check_data.sh
Checking characters table
+-----+
| id | name | nationality |
+-----+
| 1 | Wolfgang Amadeus Mozart | Prince-Archbishopric of Salzburg |
| 2 | Ludwig van Beethoven | Bonn, Germany |
| 3 | Johann Sebastian Bach | Eisenach, Germany |
| 4 | José Pablo Moncayo | Guadalajara, México |
| 5 | Niccolò Paganini | Genoa, Italy |
(5 rows)
```

- 7. Remove the `postgresql-persistent` deployment and create another deployment named `postgresql-deployment2` that uses the same persistent volume; verify that the data persisted.

- 7.1. Delete all resources that contain the `app=postgresql-persistent` label.

```
[student@workstation install-storage]$ oc delete all -l app=postgresql-persistent
service "postgresql-persistent" deleted
deployment.apps "postgresql-persistent" deleted
imagestream.image.openshift.io "postgresql-persistent" deleted
```

- 7.2. Create the `postgresql-persistent2` deployment with the same initialization data as the `postgresql-persistent` deployment.

```
[student@workstation install-storage]$ oc new-app --name postgresql-persistent2 \
> --docker-image registry.redhat.io/rhel8/postgresql-12:1-43 \
> -e POSTGRESQL_USER=redhat \
> -e POSTGRESQL_PASSWORD=redhat123 \
> -e POSTGRESQL_DATABASE=persistentdb
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "postgresql-persistent2" created
deployment.apps "postgresql-persistent2" created
service "postgresql-persistent2" created
--> Success
...output omitted...
```

- 7.3. Use the `~/DO280/labs/install-storage/check_data.sh` script to verify that the database does not have the characters table.

```
[student@workstation install-storage]$ ./check_data.sh
Checking characters table
ERROR: 'characters' table does not exist
```

- 7.4. Add the existing **postgresql-persistent** persistent volume claim to the **postgresql-persistent2** deployment.

```
[student@workstation install-storage]$ oc set volumes \
>   deployment/postgresql-persistent2 \
>     --add --name postgresql-storage --type pvc \
>     --claim-name postgresql-storage --mount-path /var/lib/pgsql
deployment.apps/postgresql-persistent2 volume updated
```

- 7.5. Use the `~/DO280/labs/install-storage/check_data.sh` script to verify that the persistent volume was successfully added and that the **postgresql-persistent2** pod can access the previously created data.

```
[student@workstation install-storage]$ ./check_data.sh
Checking characters table
id | name | nationality
---+-----+-----
1 | Wolfgang Amadeus Mozart | Prince-Archbishopric of Salzburg
2 | Ludwig van Beethoven | Bonn, Germany
...output omitted...
```

► 8. Remove the **postgresql-persistent2** deployment and the persistent volume claim.

- 8.1. Delete all resources that contain the **app=postgresql-persistent2** label.

```
[student@workstation install-storage]$ oc delete all -l app=postgresql-persistent2
service "postgresql-persistent2" deleted
deployment.apps "postgresql-persistent2" deleted
imagestream.image.openshift.io "postgresql-persistent2" deleted
```

- 8.2. Delete the persistent volume by removing the **postgresql-storage** persistent volume claim, and then return to the home directory.

```
[student@workstation install-storage]$ oc delete pvc/postgresql-storage
persistentvolumeclaim "postgresql-storage" deleted
```

- 8.3. Return to the **/home/student** directory.

```
[student@workstation install-storage]$ cd
```

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 install-storage 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 infrastructure.
- 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 10

Configuring Authentication and Authorization

Goal

Configure authentication with the HTPasswd identity provider and assign roles to users and groups.

Objectives

- Configure the HTPasswd identity provider for OpenShift authentication.
- Define role-based access controls and apply permissions to users and groups.

Sections

- Configuring Identity Providers (and Guided Exercise)
- Defining and Applying Permissions using RBAC (and Guided Exercise)

Lab

Configuring Authentication and Authorization

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 several OpenShift resources related to authentication and authorization. The following is a list of the primary resource types and their definitions:

User

In the OpenShift Container Platform architecture, users are entities that interact with the API server. The user resource represents 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

The identity resource 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 to 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, then 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 defines 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

Authentication and authorization are the two security layers responsible for enabling user interaction with the cluster. When a user makes a request to the API, the API associates the user with the request. The authentication layer authenticates the user. Upon successful authentication, the authorization layer decides to either honor or reject the API request. The authorization layer uses *role-based access control (RBAC)* policies to determine user privileges.

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 for the user. OpenShift automatically creates identity and user resources after a successful login.

Introducing Identity Providers

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

HTPasswd

Validates user names and passwords against a secret that stores credentials generated using the **htpasswd** command.

Keystone

Enables shared authentication with an OpenStack Keystone v3 server.

LDAP

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

GitHub or GitHub Enterprise

Configures 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 newly-installed OpenShift cluster provides two ways to authenticate API requests with cluster administrator privileges:

- Authenticate as the **kubeadmin** virtual user. Successful authentication grants an OAuth access token.
- Use the **kubeconfig** file, which 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.509 Certificate

During installation, the OpenShift installer creates a unique **kubeconfig** file in the **auth** directory. The **kubeconfig** 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 of the **kubeconfig** file:

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



Note

In the classroom environment, the **utility** machine stores the **kubeconfig** file at **/home/lab/ocp4/auth/kubeconfig**.

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 **--kubeconfig** option of the **oc** command.

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

Authenticating Using the Virtual User

After installation completes, OpenShift creates the **kubeadmin** virtual user. The **kubeadmin** secret in the **kube-system** namespace contains the hashed password for the **kubeadmin** user. The **kubeadmin** user has cluster administrator privileges.

The OpenShift installer dynamically generates a unique **kubeadmin** password for the cluster. The installation logs provide the **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.ocp4.example.com
INFO Login to the console with user: kubeadmin, password: shdU_trbi_6ucX_edbu_aqop
```



Note

In the classroom environment, the **utility** machine stores the password for the **kubeadmin** user in the **/home/lab/ocp4/auth/kubeconfig** file.

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, then the only way you can administer your cluster is using the **kubeconfig** file. If you do not have a copy of this file in a safe location, then you cannot 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 contains user names and passwords generated with the **htpasswd** command from the Apache HTTP Server project. Only a cluster administrator can change the data inside the HTPasswd 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: htpasswd-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 data generated using the **htpasswd** command.

Updating the OAuth Custom Resource

To 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 oauth cluster -o yaml > 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 replace -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 **httpd-utils** package provides the **htpasswd** utility. The **httpd-utils** package must be installed and available on your system.

Create the **htpasswd** file.

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



Important

Use the **-c** option only when creating a new file. The **-c** option replaces all file content if the file already exists.

Add or update credentials.

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

Delete credentials.

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

Creating the HTPasswd Secret

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

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



Important

A secret used by the HTPasswd identity provider requires adding the **htpasswd=** prefix before specifying the path to the file.

Extracting Secret Data

When adding or removing users, an administrator cannot assume the validity of a local htpasswd file. Moreover, the administrator might not 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.

By default, the **oc extract** command saves each key within a configuration map or secret as a separate file. Alternatively, all data can then be redirected to a file or displayed as standard output. To extract data from the **htpasswd-secret** secret to the **/tmp/** directory, use the following command. The **--confirm** option replaces the file if it already exists.

```
[user@demo ~]$ oc extract secret/htpasswd-secret -n openshift-config \
> --to /tmp/ --confirm
/tmp/htpasswd
```

Updating the HTPasswd Secret

The secret must be updated after adding, changing, or deleting users. Use the **oc set data secret** command to update a secret. Unless the file name is **htpasswd**, you must specify **htpasswd=** to update the **htpasswd** key within the secret.

The following command updates the **htpasswd-secret** secret in the **openshift-config** namespace using the content of the **/tmp/htpasswd** file.

```
[user@demo ~]$ oc set data secret/htpasswd-secret \
> --from-file htpasswd=/tmp/htpasswd -n openshift-config
```

After updating the secret, the OAuth operator redeploys pods in the **openshift-authentication** namespace. Monitor the redeployment of the new OAuth pods by running:

```
[user@demo ~]$ watch oc get pods -n openshift-authentication
```

Test additions, changes, or deletions to the secret after the new pods finish deploying.

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, remove the user from the local htpasswd file, and then update the secret.

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

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

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

```
[user@demo ~]$ oc set data secret/htpasswd-secret \
> --from-file htpasswd=/tmp/htpasswd -n openshift-config
```

Remove the user resource with the following command:

```
[user@demo ~]$ oc delete user manager  
user.user.openshift.io "manager" deleted
```

Identity resources include the name of the identity provider. To delete the identity resource for the **manager** user, find the resource and then delete it.

```
[user@demo ~]$ oc get identities | grep manager  
my_htpasswd_provider:manager    my_htpasswd_provider    manager      manager    ...  
  
[user@demo ~]$ oc delete identity my_htpasswd_provider:manager  
identity.user.openshift.io "my_htpasswd_provider:manager" deleted
```

Assigning Administrative Privileges

The cluster-wide **cluster-admin** role grants cluster administration privileges to users and groups. This role enables the user to perform any action on any resources within the cluster. The following example assigns the **cluster-admin** role 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* chapter in the Red Hat OpenShift Container Platform 4.5 *Authentication and authorization* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#understanding-identity-provider

► 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

As the **student** user on the **workstation** machine, use the **lab** command to prepare your system for this 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 **admin** a password of **redhat** and **developer** a password of **developer**.

- 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 **htpasswd** in the **~/D0280/labs/auth-provider/** directory. Add the **admin** user with the password of **redhat**. The name of the file is arbitrary, but this exercise use the **~/D0280/labs/auth-provider/htpasswd** file.

Use the **htpasswd** command to populate the HTPasswd authentication file with the user names and encrypted passwords. The **-B** option uses bcrypt encryption. By default, the **htpasswd** command uses MD5 encryption when you do not specify an encryption option.

```
[student@workstation ~]$ htpasswd -c -B -b ~/D0280/labs/auth-provider/htpasswd \
>     admin redhat
Adding password for user admin
```

- 1.3. Add the **developer** user with a password of **developer** to the **~/D0280/labs/auth-provider/htpasswd** file.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-provider/htpasswd \
>     developer developer
Adding password for user developer
```

- 1.4. Review the contents of the `~/D0280/labs/auth-provider/htpasswd` file and verify that it includes two entries with hashed passwords: one for the **admin** user and another for the **developer** user.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-provider/htpasswd
admin:$2y$05$QPuzHdl06IDkJssT.tdkZuSmgjUHV1XeYU4FjxhQrFqKL7hs2ZUl6
developer:$apr1$ONzmc1rh$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} \
>     https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 2.2. Create a secret from the `/home/student/D0280/labs/auth-provider/htpasswd` 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/D0280/labs/auth-provider/htpasswd`.



Important

A secret that is used by the HTPasswd identity provider requires adding the `htpasswd=` prefix before specifying the path to the file.

```
[student@workstation ~]$ oc create secret generic localusers \
>     --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
>     -n openshift-config
secret/localusers created
```

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



Note

The output indicates that the **admin** user is not found and 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 **~/D0280/labs/auth-provider** directory.

```
[student@workstation ~]$ oc get oauth cluster \
> -o yaml > ~/D0280/labs/auth-provider/oauth.yaml
```

**Note**

An **oauth.yaml** file containing the completed custom resource file is downloaded to **~/D0280/solutions/auth-provider** for your convenience.

- 3.2. Edit the **~/D0280/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 **localusers** 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: localusers
      mappingMethod: claim
      name: myusers
      type: HTPasswd
```

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

```
[student@workstation ~]$ oc replace -f ~/D0280/labs/auth-provider/oauth.yaml
oauth.config.openshift.io/cluster replaced
```

**Note**

Pods in the **openshift-authentication** namespace will redeploy if the **oc replace** command succeeds. Provided the previously created secret was created correctly, you can log in using the HTPasswd identity provider.

- 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. The authentication operator takes some time to load the configuration changes from the previous step.

**Note**

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

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

- 4.2. Use the **oc get nodes** command to verify that the **admin** user has the **cluster-admin** role.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES          AGE     VERSION
master01   Ready     master,worker  2d2h    v1.18.3+012b3ec
master02   Ready     master,worker  2d2h    v1.18.3+012b3ec
master03   Ready     master,worker  2d2h    v1.18.3+012b3ec
```

- 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 developer
Login successful.
...output omitted...
```

- 4.4. Use the **oc get nodes** command to verify that the **developer** and **admin** users 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 redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc get users
NAME        UID                  FULL NAME  IDENTITIES
admin       31f6ccd2-6c58-47ee-978d-5e5e3c30d617
developer   d4e77b0d-9740-4f05-9af5-ecfc08a85101
```

- 4.6. Display the list of current identities.

```
[student@workstation ~]$ oc get identity
NAME          IDP NAME  IDP USER NAME  USER NAME  USER UID
myusers:admin  myusers  admin          admin      31f6ccd2-6c58-47...
myusers:developer  myusers  developer    developer  d4e77b0d-9740-4f...
```

- 5. As the **admin** user, create a new HTPasswd user named **manager** with a password of **redhat**.

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

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
>   --to ~/D0280/labs/auth-provider/ --confirm
/home/student/D0280/labs/auth-provider/htpasswd
```

- 5.2. Add an entry to your `~/D0280/labs/auth-provider/htpasswd` file for the additional user **manager** with a password of **redhat**.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-provider/htpasswd \
>   manager redhat
Adding password for user manager
```

- 5.3. Review the contents of your `~/D0280/labs/auth-provider/htpasswd` file and verify that it includes three entries with hashed passwords: one each for the **admin**, **developer** and **manager** users.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-provider/htpasswd
admin:$2y$05$QPuzHdl06IDkJssT.tdkZuSmgjUHV1XeYU4FjxhQrFqKL7hs2ZUL6
developer:$apr1$0Nzmc1rh$yGtne1k.JX6L5s5wNa2ye.
manager:$apr1$CJ/tpa6a$sLhjPkIIAy755ZArTT5EH/
```

- 5.4. You must update the secret after adding additional users. Use the **oc set data secret** command to update the secret. If you receive a failure, then rerun the command again after a few moments as the oauth operator might still be reloading.

```
[student@workstation ~]$ oc set data secret/localusers \
>   --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
>   -n openshift-config
secret/localusers data updated
```

- 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 redhat
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.ocp4.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 developer
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 the **admin** user and extract the file data from the secret to the **~/D0280/labs/auth-provider/htpasswd** file.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
> --to ~/D0280/labs/auth-provider/ --confirm
/home/student/D0280/labs/auth-provider/htpasswd
```

- 7.2. Generate a random user password and assign it to the **MANAGER_PASSWD** variable.

```
[student@workstation ~]$ MANAGER_PASSWD="$(openssl rand -hex 15)"
```

- 7.3. Update the **manager** user to use the password stored in the **MANAGER_PASSWD** variable.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-provider/htpasswd \
> manager ${MANAGER_PASSWD}
Updating password for user manager
```

- 7.4. Update the secret.

```
[student@workstation ~]$ oc set data secret/localusers \
> --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
> -n openshift-config
secret/localusers data updated
```

- 7.5. 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 the **admin** user and extract the file data from the secret to the **~/D0280/labs/auth-provider/htpasswd** file.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
> --to ~/D0280/labs/auth-provider/ --confirm
/home/student/D0280/labs/auth-provider/htpasswd
```

- 8.2. Delete the **manager** user from the **~/D0280/labs/auth-provider/htpasswd** file.

```
[student@workstation ~]$ htpasswd -D ~/D0280/labs/auth-provider/htpasswd manager
Deleting password for user manager
```

- 8.3. Update the secret.

```
[student@workstation ~]$ oc set data secret/localusers \
> --from-file htpasswd=/home/student/D0280/labs/auth-provider/htpasswd \
> -n openshift-config
secret/localusers data updated
```

- 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, attempts to log in as the **manager** user fail.

```
[student@workstation ~]$ oc login -u manager -p ${MANAGER_PASSWD}
Login failed (401 Unauthorized)
Verify you have provided correct credentials.
```

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

NAME	UID	FULL NAME	IDENTITIES
admin	31f6cccd2-6c58-47ee-978d-5e5e3c30d617		myusers:admin
developer	d4e77b0d-9740-4f05-9af5-ecfc08a85101		myusers:developer

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

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. Using **--to -** sends the secret to STDOUT rather than saving it to a file.

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config --to -  
# htpasswd  
admin:$2y$05$TizWp/2ct4Edn08gmeMBI09IXujpLqkKAJ0Nldxc/V2XYYMBf6wBy  
developer:$apr1$8Bc6txgb$bwHke4cGRGk9C8tQLg.hi1
```

- 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 OAuth:

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

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

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

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

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

- 9.4. Delete the **localusers** secret from the **openshift-config** namespace.

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

- 9.5. Delete all user resources.

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

- 9.6. Delete all identity resources.

```
[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.

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 user's level of responsibility: 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	Assignment of users or groups to a role.

RBAC Scope

Red Hat OpenShift Container Platform (RHOCOP) defines two groups of roles and bindings depending on the user's scope and responsibility: 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

Cluster administrators can use the `oc adm policy` command to both add and remove cluster roles and namespace roles.

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 access 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.

Default roles	Description
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.
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 constraints configured by a project administrator.

Project administrators can use the **oc policy** command to add and remove namespace roles.

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

```
[user@demo ~]$ oc 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 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 can 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 routers and registries to use, and various others. An anonymous system user 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/>

For more information about RBAC, refer to the *Using RBAC to define and apply permissions* chapter in the Red Hat OpenShift Container Platform 4.5 *Authentication and authorization* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#using-rbac

For more information about groups, refer to the *Understanding authentication* chapter in the Red Hat OpenShift Container Platform 4.5 *Authentication and authorization* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/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 project creation privileges from users who are not OpenShift cluster administrators.
- Create OpenShift groups and add members to these groups.
- Create a project and assign project administration privileges to the project.
- As a project administrator, assign read and write privileges to different groups of users.

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 some HTPasswd users for the exercise.

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

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

- 1.1. Log in to the cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>     https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. 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 who are not cluster administrators by deleting the **self-provisioner** cluster role from the **system:authenticated:oauth** virtual group.

- 2.1. Confirm that the **self-provisioners** cluster role binding that you found in the previous step assigns 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 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 with a password of **redhat**, and then try to create a project. Project creation should fail.

```
[student@workstation ~]$ oc login -u leader -p redhat
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 **auth-rbac** project.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation ~]$ oc new-project auth-rbac
Now using project "auth-rbac" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

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

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

```
[student@workstation ~]$ oc login -u leader -p redhat
Login successful.
...output omitted...
Using project "auth-rbac".
```

- 5.2. Add write privileges to **dev-group** on the **auth-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 **auth-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 **auth-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 the **developer** user 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 developer
Login successful.
...output omitted...
Using project "auth-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
  deployment.apps "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 "auth-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 redhat
Login successful.
...output omitted...
Using project "auth-rbac".
```

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

```
[student@workstation ~]$ oc scale deployment httpd --replicas 3
Error from server (Forbidden): deployments.apps "httpd" is forbidden: User "qa-
engineer" cannot patch resource "deployments/scale" in API group "apps" in the
namespace "auth-rbac"
```

- 8. Restore project creation privileges to all users.

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

```
[student@workstation ~]$ oc login -u admin -p redhat
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 auth-rbac finish
```

This concludes the section.

► Lab

Configuring Authentication and Authorization

Performance Checklist

In this lab, you will configure the HTPasswd identity provider, create groups, and assign roles to users and groups.

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.
- Remove the ability to create projects at the cluster level.
- Create groups and add users to groups.
- Manage user privileges in projects by granting privileges to groups.

Before You Begin

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

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

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

1. Update the existing **~/D0280/labs/auth-review/tmp_users** HTPasswd authentication file to remove the **analyst** user. Ensure that the **tester** and **leader** users in the file use a password of **L@bR3v!ew**. Add two new entries to the file for the users **admin** and **developer**. Use **L@bR3v!ew** as the password for each new user.
2. Log in to your OpenShift cluster as the **kubeadmin** user using the **RHT_OCP4_KUBEADM_PASSWD** variable defined in the **/usr/local/etc/ocp4.config** file as the password. Configure your cluster to use the HTPasswd identity provider using the user names and passwords defined in the **~/D0280/labs/auth-review/tmp_users** file.
3. Make the **admin** user a cluster administrator. Log in as both **admin** and as **developer** to verify HTPasswd user configuration and cluster privileges.
4. As the **admin** user, remove the ability to create projects cluster wide.
5. Create a group named **managers**, and add the **leader** user to the group. Grant project creation privileges to the **managers** group. As the **leader** user, create the **auth-review** project.

6. Create a group named **developers** and grant edit privileges on the **auth-review** project. Add the **developer** user to the group.
7. Create a group named **qa** and grant view privileges on the **auth-review** project. Add the **tester** user to the group.

Evaluation

On the **workstation** machine, run the **lab** command to grade your work. Correct any reported failures and rerun the script until successful.

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

This concludes the guided exercise.

► Solution

Configuring Authentication and Authorization

Performance Checklist

In this lab, you will configure the HTPasswd identity provider, create groups, and assign roles to users and groups.

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.
- Remove the ability to create projects at the cluster level.
- Create groups and add users to groups.
- Manage user privileges in projects by granting privileges to groups.

Before You Begin

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

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

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

1. Update the existing **~/D0280/labs/auth-review/tmp_users** HTPasswd authentication file to remove the **analyst** user. Ensure that the **tester** and **leader** users in the file use a password of **L@bR3v!ew**. Add two new entries to the file for the users **admin** and **developer**. Use **L@bR3v!ew** as the password for each new user.
 - 1.1. Remove the **analyst** user from the **~/D0280/labs/auth-review/tmp_users** HTPasswd authentication file.

```
[student@workstation ~]$ htpasswd -D ~/D0280/labs/auth-review/tmp_users analyst  
Deleting password for user analyst
```

- 1.2. Update the entries for the **tester** and **leader** users so that they use a password of **L@bR3v!ew**. Add entries for the **admin** and **developer** users using a password of **L@bR3v!ew**.

```
[student@workstation ~]$ for NAME in tester leader admin developer
>   do
>     htpasswd -b ~/D0280/labs/auth-review/tmp_users ${NAME} 'L@bR3v!ew'
>   done
Updating password for user tester
Updating password for user leader
Adding password for user admin
Adding password for user developer
```

13. Review the contents of the **~/D0280/labs/auth-review/tmp_users** file. It does not contain a line for the **analyst** user. It includes two new entries with hashed passwords for the **admin** and **developer** users.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-review/tmp_users
tester:$apr1$0eqhKgbU$Dwd0CB4IumhasaRuEr6hp0
leader:$apr1$.EB5IXlu$FDV.Av16njl0CMzgolScr
admin:$apr1$ItcCncDS$xQCUjQGTsXAup00KQfmw0
developer:$apr1$D8F1Hren$izDhAWq5DRjUHPv0i7FHn.
```

2. Log in to your OpenShift cluster as the **kubeadmin** user using the **RHT_OCP4_KUBEADM_PASSWD** variable defined in the **/usr/local/etc/ocp4.config** file as the password. Configure your cluster to use the HTPasswd identity provider using the user names and passwords defined in the **~/D0280/labs/auth-review/tmp_users** file.

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

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

- 2.2. Log in to the cluster as the **kubeadmin** user.

```
[student@workstation ~]$ oc login -u kubeadmin -p ${RHT_OCP4_KUBEADM_PASSWD} \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 2.3. Create a secret named **auth-review** using the **~/D0280/labs/auth-review/tmp_users** file.

```
[student@workstation ~]$ oc create secret generic auth-review \
>   --from-file htpasswd=/home/student/D0280/labs/auth-review/tmp_users \
>   -n openshift-config
secret/auth-review created
```

- 2.4. Export the existing OAuth resource to **~/D0280/labs/auth-review/oauth.yaml**.

```
[student@workstation ~]$ oc get oauth cluster \
>   -o yaml > ~/D0280/labs/auth-review/oauth.yaml
```

- 2.5. Edit the `~/DO280/labs/auth-review/oauth.yaml` file to replace the `spec: {}` line with the following bold lines. 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: auth-review
      mappingMethod: claim
      name: htpasswd
      type: HTPasswd
```



Note

For convenience, the `~/DO280/solutions/auth-review/oauth.yaml` file contains a minimal version of the OAuth configuration with the specified customizations.

- 2.6. Apply the customized resource defined in the previous step.

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

- 2.7. A successful update to the `oauth/cluster` resource recreates the `oauth-openshift` pods in the `openshift-authentication` namespace.

```
[student@workstation ~]$ watch oc get pods -n openshift-authentication
```

Wait until both new `oauth-openshift` pods are ready and running and the previous pods have terminated.

NAME	READY	STATUS	RESTARTS	AGE
oauth-openshift-6755d8795-h8bgv	1/1	Running	0	34s
oauth-openshift-6755d8795-rk4m6	1/1	Running	0	38s

Press **Ctrl+C** to exit the `watch` command.

3. Make the `admin` user a cluster administrator. Log in as both `admin` and as `developer` to verify HTPasswd user configuration and cluster privileges.

- 3.1. 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"
```

Chapter 10 | Configuring Authentication and Authorization

- 3.2. Log in to the cluster as the **admin** user to verify that HTPasswd authentication was configured correctly.

```
[student@workstation ~]$ oc login -u admin -p 'L@bR3v!ew'  
Login successful.  
...output omitted...
```

- 3.3. Use **oc get nodes** command to verify 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         AGE     VERSION  
master01   Ready     master,worker  46d    v1.18.3+012b3ec  
master02   Ready     master,worker  46d    v1.18.3+012b3ec  
master03   Ready     master,worker  46d    v1.18.3+012b3ec
```

- 3.4. Log in to the cluster as the **developer** user to verify the HTPasswd authentication is configured correctly.

```
[student@workstation ~]$ oc login -u developer -p 'L@bR3v!ew'  
Login successful.  
...output omitted...
```

- 3.5. Use the **oc get nodes** command to verify that the **developer** user does not have cluster administration privileges.

```
[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. As the **admin** user, remove the ability to create projects cluster wide.

- 4.1. Log in to the cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p 'L@bR3v!ew'  
Login successful.  
...output omitted...
```

- 4.2. 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.

5. Create a group named **managers**, and add the **leader** user to the group. Grant project creation privileges to the **managers** group. As the **leader** user, create the **auth-review** project.

- 5.1. Create a group named **managers**.

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

- 5.2. Add the **leader** user to the **managers** group.

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

- 5.3. Assign the **self-provisioner** cluster role to the **managers** group.

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

- 5.4. As the **leader** user, create the **auth-review** project.

```
[student@workstation ~]$ oc login -u leader -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

The user who creates a project is automatically assigned the **admin** role on the project.

```
[student@workstation ~]$ oc new-project auth-review
Now using project "auth-review" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

6. Create a group named **developers** and grant edit privileges on the **auth-review** project. Add the **developer** user to the group.

- 6.1. Log in to the cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p 'L@bR3v!ew'
Login successful.
...output omitted...
```

- 6.2. Create a group named **developers**.

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

6.3. Add the **developer** user to the **developers** group.

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

6.4. Grant edit privileges to the **developers** group on the **auth-review** project.

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

7. Create a group named **qa** and grant view privileges on the **auth-review** project. Add the **tester** user to the group.

7.1. Create a group named **qa**.

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

7.2. Add the **tester** user to the **qa** group.

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

7.3. Grant view privileges to the **qa** group on the **auth-review** project.

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

Evaluation

On the **workstation** machine, run the **lab** command to grade your work. Correct any reported failures and rerun the script until successful.

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

This concludes the guided exercise.

Summary

In this chapter, you learned:

- A newly-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.
- 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.
- 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, assign a user to one or more roles. A role binding contains all of the associations of a role to users and groups.
- To grant a user cluster administrator privileges, assign the **cluster-admin** role to that user.

Chapter 11

Configuring Application Security

Goal

Restrict permissions of applications using security context constraints and protect access credentials using secrets.

Objectives

- Create and apply secrets to manage sensitive information.
- Create service accounts and apply permissions.

Sections

- Managing Sensitive Information with Secrets (and Guided Exercise)
- Controlling Application Permissions with Security Context Constraints (and Guided Exercise)

Lab

Configuring Application Security

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 deploying an application to different environments without requiring a change to the application source code.

Applications often require access to sensitive information. For example, a back-end web application requires access to database credentials to perform a database query.

Kubernetes and OpenShift use 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, 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.



Note

You can encrypt the Etcd database, although this is not the default setting. When enabled, Etcd encrypts the following resources: secrets, configuration maps, routes, OAuth access tokens, and OAuth authorize tokens. Enabling Etcd encryption is outside the scope of this class.

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 that other team members can reference 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 in the pod does not update. 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, then those variables are destroyed when the pod is destroyed.

Secret data volumes are backed by temporary file storage. If a secret is mounted as a file in the pod, then the file is also destroyed when the pod is destroyed. A stopped pod does not contain secret data.

Use Cases for Secrets

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

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

Use a TLS certificate and key to secure communication to a pod. A TLS secret stores the certificate as **tls.crt** and the certificate key as **tls.key**. Developers can mount the secret as a volume and create a pass through route to the application.

Creating a Secret

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

- Create a generic secret containing key-value pairs from literal values typed on the command line:

```
[user@demo ~]$ oc create secret generic secret_name \
>   --from-literal key1=secret1 \
>   --from-literal key2=secret2
```

- Create a generic secret using key names specified on the command line and values from files:

```
[user@demo ~]$ oc create secret generic ssh-keys \
>   --from-file id_rsa=/path-to/id_rsa \
>   --from-file id_rsa.pub=/path-to/id_rsa.pub
```

- Create a TLS secret specifying a certificate and the associated key:

```
[user@demo ~]$ oc create secret tls secret-tls \
>   --cert /path-to-certificate --key /path-to-key
```

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: **user** with the **demo-user** value and **root_password** with the **zT1KTgk** value.

```
[user@demo ~]$ oc create secret generic demo-secret \
>   --from-literal user=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_PASSWORD ①
    valueFrom:
      secretKeyRef: ②
        name: demo-secret ③
        key: root_password ④
```

- ① The environment variable name in the pod that contains data from a secret.
- ② The **secretKeyRef** key expects a secret. Use the **configMapKeyRef** key for configuration maps.
- ③ The name of the secret that contains the desired sensitive information.
- ④ The name of the key that contains the sensitive information in the secret.

You can also use the **oc set env** command to set application environment variables from either secrets or configuration maps. In some cases, the names of the keys can be modified to match the names of environment variables by using the **--prefix** option. In the following example, the **user** key sets the **MYSQL_USER** environment variable, and the **root_password** key sets the **MYSQL_ROOT_PASSWORD** environment variable. If the key name is lowercase, then the corresponding environment variable is converted to uppercase.

```
[user@demo ~]$ oc set env deployment/demo --from secret/demo-secret \
>   --prefix MYSQL_
```

Secrets as Files in a Pod

A secret can be mounted to a directory within a pod. A file is created for each key in the secret using the name of the key. The content of each file is the decoded value of the secret.

```
[user@demo ~]$ oc set volume deployment/demo \
>   --add --type secret \ ②
>   --secret-name demo-secret \ ③
>   --mount-path /app-secrets ④
```

- ① Modify the volume configuration in the **demo** deployment.
- ② Add a new volume from a secret. Configuration maps can also be mounted as volumes.
- ③ Use the **demo-secret** secret.
- ④ Make the secret data available in the **/app-secrets** directory in the pod. The content of the **/app-secrets/user** file is **demo-user**. The content of the **/app-secrets/root_password** file is **zT1KTgk**.

The container image can dictate the location of the mount point and the expected file names. For example, a container image running **NGINX** can specify the SSL certificate location and the SSL certificate key location in the **/etc/nginx/nginx.conf** configuration file. If the expected files are not found, then the container might fail to run.



Important

If the mount point already exists in the pod, then any existing files at the mount point are obscured by the mounted secret. The existing files are not visible and are not accessible.

Configuration Map Overview

Similar to secrets, configuration maps decouple configuration information from container images. Unlike secrets, the information contained in configuration maps does not require protection. You can use the data in a configuration map to set environment variables in the container image, or mount the configuration map as a volume within the container image.

Container images do not need to be rebuilt when a secret or a configuration map changes. New pods use the updated secrets and configuration maps. You can delete pods using the older secrets and configuration maps.

The syntax for creating a configuration map closely matches the syntax for creating a secret. Key-value pairs can be entered on the command line or the content of a file can be used as the value of a specified key.

```
[user@demo ~]$ oc create configmap my-config \
>   --from-literal key1=config1 --from-literal key2=config2
```

Updating Secrets and Configuration Maps

Secrets and configuration maps occasionally require updates. Use the **oc extract** command to ensure you have the latest data. Save the data to a specific directory using the **--to** option. Each key in the secret or configuration map creates a file with the same name as the key. The content of each file is the value of the associated key. If you run the **oc extract** command more than once, then use the **--confirm** option to overwrite the existing files.

```
[user@demo ~]$ oc extract secret/htpasswd-ppklq -n openshift-config \
>   --to /tmp/ --confirm
```

After updating the locally saved files, use the **oc set data** command to update the secret or configuration map. For each key that requires an update, specify the name of a key and the associated value. If a file contains the value, use the **--from-file** option.

In the previous **oc extract** example, the **htpasswd-ppk1q** secret contained only one key named **htpasswd**. Using the **oc set data** command, you can explicitly specify the **htpasswd** key name using **--from-file htpasswd=/tmp/htpasswd**. If the key name is not specified, the file name is used as the key name.

```
[user@demo ~]$ oc set data secret/htpasswd-ppk1q -n openshift-config \
>   --from-file /tmp/htpasswd
```



References

For more information on secrets, refer to the *Understanding secrets* section in the *Working with pods* chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-pods-secrets-about_nodes-pods-secrets

For more information on Etcd encryption, refer to the *Encrypting Etcd data* chapter in the Red Hat OpenShift Container Platform 4.5 Security documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/security/index#encrypting-etcd

► Guided Exercise

Managing Sensitive Information With Secrets

In this exercise, you will 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

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

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

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

```
[student@workstation ~]$ oc new-project authorization-secrets
Now using project "authorization-secrets" on server
"https://api.ocp4.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 cannot be assigned from a secret using the **oc new-app** command.

```
[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
  deployment.apps "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. Press **Ctrl+C** to exit the command.

NAME	READY	STATUS	RESTARTS	AGE
mysql-786bb947f9-qz2fm	0/1	Error	3	71s
mysql-786bb947f9-qz2fm	0/1	CrashLoopBackOff	3	75s
mysql-786bb947f9-qz2fm	0/1	Error	4	103s
mysql-786bb947f9-qz2fm	0/1	CrashLoopBackOff	4	113s

**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. 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 as environment variables, adding the prefix **MYSQL_**.

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

- 3.4. To demonstrate how a secret can be mounted as a volume, mount the **mysql** secret to the **/run/kubernetes/mysql** directory within the pod.

```
[student@workstation ~]$ oc set volume deployment/mysql --add --type secret \
>   --mount-path /run/secrets/mysql --secret-name mysql
info: Generated volume name: volume-nrh7r
deployment.apps/mysql volume updated
```

- 3.5. Modifying the deployment using the **oc set env** command or the **oc set volume** command triggers a new application deployment. Verify that the **mysql** application deploys successfully after the modifications.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
mysql-7cd7499d66-gm2rh   1/1     Running   0          21s
```

Take note of the pod name in the **Running** state; you will need it in upcoming 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-7cd7499d66-gm2rh
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. List mount points in the pod containing the **mysql** pattern. Notice that the mount point is backed by a temporary file system (tmpfs). This is true for all secrets that are mounted as volumes.

```
sh-4.2$ df -h | grep mysql
tmpfs        7.9G   16K  7.9G  1% /run/secrets/mysql
```

- 4.4. Examine the files mounted at the **/run/secrets/mysql** mount point. Each file matches a key name in the secret, and the content of each file is the value of the associated key.

```
sh-4.2$ for FILE in $(ls /run/secrets/mysql)
> do
> echo "${FILE}: $(cat /run/secrets/mysql/${FILE})"
> done
database: test_secrets
hostname: mysql
password: redhat123
user: mysql
```

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

```
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:2.1
--> Found container image 7ff1a7b (2 days old) from quay.io for "quay.io/
redhattraining/famous-quotes:2.1"
...output omitted...
--> Creating resources ...
  imagestream.image.openshift.io "quotes" created
  deployment.apps "quotes" created
  service "quotes" created
--> Success
...output omitted...
```

5.2. Verify the status of the **quotes** application pod. The pod displays an error because it cannot connect to the database. This might take a while to display in the output. Press **Ctrl+C** to exit the command.

NAME	READY	STATUS	RESTARTS	AGE
quotes-6b658d57bc-vs28q	0/1	CrashLoopBackOff	3	86s
quotes-6b658d57bc-vs28q	0/1	Error	4	108s
quotes-6b658d57bc-vs28q	0/1	CrashLoopBackOff	4	2m1s

▶ 6. 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. The **mysql** secret can initialize environment variables for the **quotes** application by adding the **QUOTES_** prefix.

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

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

- 6.2. Wait until the **quotes** application pod redeploys. The older pods terminate automatically.

```
[student@workstation ~]$ oc get pods -l deployment=quotes
NAME                  READY   STATUS    RESTARTS   AGE
quotes-77df54758b-mqdtf   1/1     Running   3          7m17s
```



Note

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

- 7. Verify that the **quotes** pod successfully connects to the database and that the application displays a list of quotes.

- 7.1. Examine the pod logs using the **oc logs** command. The logs indicate a successful database connection.

```
[student@workstation ~]$ oc logs quotes-77df54758b-mqdtf | head -n2
... Connecting to the database: myuser:redhat123@tcp(mysql:3306)/test_secrets
... Database connection OK
```

- 7.2. Expose the **quotes** service so that it can be accessed from outside the cluster.

```
[student@workstation ~]$ oc expose service quotes \
>   --hostname quotes.apps.ocp4.example.com
route.route.openshift.io/quotes exposed
```

- 7.3. Verify the application host name.

```
[student@workstation ~]$ oc get route quotes
NAME      HOST/PORT           PATH  SERVICES  PORT  ...
quotes   quotes.apps.ocp4.example.com  quotes  8000-tcp ...
```

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

```
[student@workstation ~]$ curl -s \
>   http://quotes.apps.ocp4.example.com/env | grep QUOTES_
<li>QUOTES_USER: myuser </li>
<li>QUOTES_PASSWORD: redhat123 </li>
<li>QUOTES_DATABASE: test_secrets</li>
<li>QUOTES_HOST: mysql</li>
```

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

```
[student@workstation ~]$ curl -s http://quotes.apps.ocp4.example.com/status  
Database connection OK
```

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

```
[student@workstation ~]$ curl -s http://quotes.apps.ocp4.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 the **workstation** machine, 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

Objectives

After completing this section, you should be able to:

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

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

- Running privileged containers.
- Requesting extra capabilities for 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:
...output omitted...
```

Most pods created by OpenShift use the SCC named **restricted**, which provides limited access to resources external to OpenShift. Use the **oc describe** command to view the security context constraint that a pod uses.

```
[user@demo ~]$ oc describe pod console-5df4fcbb47-67c52 \
>   -n openshift-console | grep scc
          openshift.io/scc: restricted
```

Container images downloaded from public container registries, such as Docker Hub, might fail to run using the **restricted** SCC. For example, a container image that requires running as a specific user ID can fail because the **restricted** SCC runs the container using a random user ID. A container image that listens on port 80 or port 443 can fail for a related reason. The random user ID used by the **restricted** SCC cannot start a service that listens on a privileged network port (port numbers less than 1024). Use the **scc-subject-review** subcommand to list all the security context constraints that can overcome the limitations of a container.

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

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 strategy 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 must create a service account bound to a pod. Use the **oc create serviceaccount** command to create the service account, and use the **-n** option if the service account must be created in a namespace different than the current one.

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

To associate the service account with an SCC, use the **oc adm policy** command. Use the **-z** option to identify a service account, and use the **-n** option if the service account exists in a namespace different than the current one.

```
[user@demo ~]$ oc adm policy add-scc-to-user SCC -z service-account
```



Important

Assigning an SCC to a service account or removing an SCC from a service account must be performed by a cluster administrator. Allowing pods to run with a less restrictive SCC can make your cluster less secure. Use with caution.

Modify an existing deployment or deployment configuration to use the service account using the **oc set serviceaccount** command. If the command succeeds, then the pods associated with the deployment or deployment configuration redeploy.

```
[user@demo ~]$ oc set serviceaccount deployment/deployment-name \
>     service-account-name
```

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. Use SCCs 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 Red Hat OpenShift Container Platform 4.5 *Authentication and authorization* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/authentication_and_authorization/index#managing-pod-security-policies

► Guided Exercise

Controlling Application Permissions with Security Context Constraints

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

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

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

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

```
[student@workstation ~]$ oc new-project authorization-scc
Now using project "authorization-scc" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

- 2. Deploy an application named **gitlab** using the container image located at **quay.io/redhattraining/gitlab-ce:8.4.3-ce.0**. This image is a copy of the container image available at **docker.io/gitlab/gitlab-ce:8.4.3-ce.0**. Verify that the pod fails because the container image needs **root** privileges.

- 2.1. Deploy the **gitlab** application.

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

- 2.2. Determine if the application is 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-7d67db7875-gcsjl   0/1     Error      1          60s
```



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-7d67db7875-gcsjl
...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...
```

- 2.4. Log in as the **admin** user.

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

- 2.5. Check if using a different SCC can resolve the permissions problem.

```
[student@workstation ~]$ oc get pod/gitlab-7d67db7875-gcsjl -o yaml \
>   | oc adm policy scc-subject-review -f -
RESOURCE                      ALLOWED BY
Pod/gitlab-7d67db7875-gcsjl  anyuid
```

- 3. Create a new service account and assign 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. Assign the **anyuid** SCC to the **gitlab-sa** service account.

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z gitlab-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "gitlab-sa"
```

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

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

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

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

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

- 4.3. Verify that the **gitlab** redeployment succeeds. You might 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-86d6d65-zm2fd  1/1     Running   0          55s
```

- 5. Verify that the **gitlab** application works properly.

- 5.1. Expose the **gitlab** application. Because the **gitlab** service listens on ports 22, 80, and 443, you must use the **--port** option.

```
[student@workstation ~]$ oc expose service/gitlab --port 80 \
>   --hostname gitlab.apps.ocp4.example.com
route.route.openshift.io/gitlab exposed
```

- 5.2. Get the exposed route.

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT          PATH  SERVICES  PORT  ...
gitlab    gitlab.apps.ocp4.example.com  gitlab     80    ...
```

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

```
[student@workstation ~]$ curl -s \
>   http://gitlab.apps.ocp4.example.com/users/sign_in | grep '<title>'<br/>
<title>Sign in · GitLab</title>
```

▶ **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

Configuring Application Security

In this lab, you will create a secret to share sensitive configuration information and modify a deployment so that it runs with less restrictive settings.

Outcomes

You should be able to:

- Provide sensitive information to deployments using secrets.
- Allow applications to run in a less restrictive environment using security context constraints.

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 that you can log in to the cluster.

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

1. As the **developer** user, create the **authorization-review** project. All additional tasks in this exercise use the **authorization-review** project.
2. Create a secret named **review-secret** that you will use with the MySQL database and WordPress applications. The secret must include three key-value pairs: **user=wpuser**, **password=redhat123**, and **database=wordpress**.
3. Deploy a MySQL database application named **mysql** using the container image located at **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47**. Modify the **mysql** deployment to use the **review-secret** secret as environment variables. The environment variables must use the **MYSQL_** prefix.
4. Deploy a WordPress application named **wordpress** using the container image located at **quay.io/redhattraining/wordpress:5.3.0**. This image is a copy of the container image available at **docker.io/library/wordpress:5.3.0**. When creating the application, add the **WORDPRESS_DB_HOST=mysql** and **WORDPRESS_DB_NAME=wordpress** environment variables. Once deployed, modify the **wordpress** deployment to use the **review-secret** secret as additional environment variables. The additional environment variables must use the **WORDPRESS_DB_** prefix.



Note

The **wordpress** pod does not run successfully until you modify the deployment to use a less restrictive security context constraint.

5. As the **admin** user, identify a less restrictive SCC that allows the **wordpress** deployment to run successfully. Create a service account named **wordpress-sa** and grant the **anyuid** SCC to it. Modify the **wordpress** deployment to use the **wordpress-sa** service account.

6. As the **developer** user, make the **wordpress** service accessible to external requests using the **wordpress-review.apps.ocp4.example.com** host name. Access the route using a web browser and verify the WordPress application displays the setup wizard.

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

Configuring Application Security

In this lab, you will create a secret to share sensitive configuration information and modify a deployment so that it runs with less restrictive settings.

Outcomes

You should be able to:

- Provide sensitive information to deployments using secrets.
- Allow applications to run in a less restrictive environment using security context constraints.

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 that you can log in to the cluster.

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

1. As the **developer** user, create the **authorization-review** project. All additional tasks in this exercise use the **authorization-review** project.

- 1.1. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \  
> https://api.ocp4.example.com:6443  
Login successful.  
...output omitted...
```

- 1.2. Create the **authorization-review** project.

```
[student@workstation ~]$ oc new-project authorization-review  
Now using project "authorization-review" on server  
"https://api.ocp4.example.com:6443".  
...output omitted...
```

2. Create a secret named **review-secret** that you will use with the MySQL database and WordPress applications. The secret must include three key-value pairs: **user=wpuser**, **password=redhat123**, and **database=wordpress**.

- 2.1. 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
```

3. Deploy a MySQL database application named **mysql** using the container image located at **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47**. Modify the **mysql** deployment to use the **review-secret** secret as environment variables. The environment variables must use the **MYSQL_** prefix.

- 3.1. 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
  deployment.apps "mysql" created
  service "mysql" created
--> Success
...output omitted...
```

- 3.2. Use the **review-secret** secret to initialize the environment variables on the **mysql** deployment. The **--prefix** option ensures that all the variables injected from the secret into the pod start with **MYSQL_**.

```
[student@workstation ~]$ oc set env deployment/mysql --prefix MYSQL_ \
>   --from secret/review-secret
deployment.apps/mysql updated
```

- 3.3. Verify that the **mysql** pod redeploys successfully.

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

Press **Ctrl+C** to exit the **watch** command after the **mysql** pod displays both **1/1** and **Running**.

NAME	READY	STATUS	RESTARTS	AGE
mysql-f675b96f8-vspb9	1/1	Running	0	20s



Note

It may take a few minutes for the deployment to roll out successfully after setting the secret.

4. Deploy a WordPress application named **wordpress** using the container image located at **quay.io/redhattraining/wordpress:5.3.0**. This image is a copy of the container image available at **docker.io/library/wordpress:5.3.0**. When creating the application, add the **WORDPRESS_DB_HOST=mysql** and **WORDPRESS_DB_NAME=wordpress**

environment variables. Once deployed, modify the **wordpress** deployment to use the **review-secret** secret as additional environment variables. The additional environment variables must use the **WORDPRESS_DB_** prefix.

**Note**

The **wordpress** pod does not run successfully until you modify the deployment to use a less restrictive security context constraint.

4.1. Deploy a **wordpress** application.

```
[student@workstation ~]$ oc new-app --name wordpress \
>   --docker-image quay.io/redhattraining/wordpress:5.3.0 \
>   -e WORDPRESS_DB_HOST=mysql \
>   -e WORDPRESS_DB_NAME=wordpress
...output omitted...
-> Creating resources ...
  imagestream.image.openshift.io "wordpress" created
  deployment.apps "wordpress" created
  service "wordpress" created
--> Success
...output omitted...
```

4.2. Use the **review-secret** secret to initialize the environment variables on the **wordpress** deployment. The **--prefix** option ensures that the variables injected from the secret into the pod all start with **WORDPRESS_DB_**.

```
[student@workstation ~]$ oc set env deployment/wordpress \
>   --prefix WORDPRESS_DB_ --from secret/review-secret
deployment.apps/wordpress updated
```

4.3. Verify that the **wordpress** pod does not successfully redeploy, even after injecting variables from the **review-secret** secret.

```
[student@workstation ~]$ watch oc get pods -l deployment=wordpress
```

Wait up to one minute, and then press **Ctrl+C** to exit the **watch** command. The **wordpress** pod continually restarts. Each time the pod transitions to a status of **Error** and then **CrashLoopBackOff**.

```
Every 2.0s: oc get pods -l deployment=wordpress
```

```
...
```

NAME	READY	STATUS	RESTARTS	AGE
wordpress-68c49c9d4-wq46g	0/1	CrashLoopBackOff	5	4m30s

4.4. Check the pod logs for error messages.

```
[student@workstation ~]$ oc logs wordpress-68c49c9d4-wq46g
...output omitted...
(13)Permission denied: AH00072: make_sock: could not bind to address [::]:80
(13)Permission denied: AH00072: make_sock: could not bind to address 0.0.0.0:80
no listening sockets available, shutting down
AH00015: Unable to open logs
```

By default, OpenShift prevents pods from starting services that listen on ports lower than 1024.

5. As the **admin** user, identify a less restrictive SCC that allows the **wordpress** deployment to run successfully. Create a service account named **wordpress-sa** and grant the **anyuid** SCC to it. Modify the **wordpress** deployment to use the **wordpress-sa** service account.

- 5.1. Log in as the **admin** user.

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

- 5.2. Check whether using a different SCC resolves the permissions problem.

```
[student@workstation ~]$ oc get pod/wordpress-68c49c9d4-wq46g -o yaml \
>   | oc adm policy scc-subject-review -f -
RESOURCE                      ALLOWED BY
Pod/wordpress-68c49c9d4-wq46g  anyuid
```



Important

The **oc adm policy** command must be run as the **admin** user.

- 5.3. Create a service account named **wordpress-sa**.

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

- 5.4. Grant the **anyuid** SCC to the **wordpress-sa** service account. If the **wordpress** pod runs as the **root** user, then OpenShift allows the pod to start a service on port 80.

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z wordpress-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added:
"wordpress-sa"
```

- 5.5. Configure the **wordpress** deployment to use the **wordpress-sa** service account.

```
[student@workstation ~]$ oc set serviceaccount deployment/wordpress \
>   wordpress-sa
deployment.apps/wordpress serviceaccount updated
```

- 5.6. Verify that the **wordpress** pod successfully redeploys after configuring the service account.

```
[student@workstation ~]$ watch oc get pods -l deployment=wordpress
```

Press **Ctrl+C** to exit the **watch** command after the **wordpress** pod displays both **1/1** and **Running**.

```
Every 2.0s: oc get pods -l deployment=wordpress
```

```
...
```

NAME	READY	STATUS	RESTARTS	AGE
wordpress-bcb5d97f6-mwljs	1/1	Running	0	21s

6. As the **developer** user, make the **wordpress** service accessible to external requests using the **wordpress-review.apps.ocp4.example.com** host name. Access the route using a web browser and verify the WordPress application displays the setup wizard.

- 6.1. Use the **oc expose** command to create a route to the **wordpress** application.

```
[student@workstation ~]$ oc expose service/wordpress \
>   --hostname wordpress-review.apps.ocp4.example.com
route.route.openshift.io/wordpress exposed
```

- 6.2. Use a web browser to verify access to the URL **http://wordpress-review.apps.ocp4.example.com**. When you correctly deploy the application, a setup wizard displays in the browser.

Alternatively, use the **curl** command to access the installation URL directly.

```
[student@workstation ~]$ curl -s \
>   http://wordpress-review.apps.ocp4.example.com/wp-admin/install.php \
>   | grep Installation
<title>WordPress &rsaquo; Installation</title>
```

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.

Summary

In this chapter, you learned:

- 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 12

Configuring OpenShift Networking Components

Goal

Troubleshoot OpenShift software-defined networking (SDN) and configure network policies.

Objectives

- Troubleshoot OpenShift software-defined networking using the command-line interface.
- Allow and protect network connections to applications inside an OpenShift cluster.
- Restrict network traffic between projects and pods.

Sections

- Troubleshooting OpenShift Software-defined Networking (and Guided Exercise)
- Exposing Applications for External Access (and Guided Exercise)
- Configuring Network Policies (and Guided Exercise)

Lab

Configuring OpenShift Networking for Applications

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 satisfies 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 backward-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 *Container Network Interface (CNI)* is a common interface between the network provider and the container execution and is implemented as network plug-ins. The CNI provides the specification for plug-ins to configure network interfaces inside containers. Plug-ins written to the specification allow different network providers to control the OpenShift cluster network.

The SDN uses CNI plug-ins to create 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 routing tables. The SDN allocates a unique routable IP to each pod so that you can access the pod from any other service in the same network.

Some common CNI plug-ins used in OpenShift are OpenShift SDN, OVN-Kubernetes, and Kuryr. The OpenShift SDN network provider is currently the default network provider in OpenShift 4.5, but future OpenShift 4 releases will default to the OVN-Kubernetes network provider.

The OpenShift SDN network provider uses *Open vSwitch (OVS)* to connect pods on the same node and *Virtual Extensible LAN (VXLAN)* tunneling to connect nodes. OVN-Kubernetes uses *Open Virtual Network (OVN)* to manage the cluster network. OVN extends OVS with virtual network abstractions. Kuryr provides networking through the Neutron and Octavia Red Hat OpenStack Platform services.

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.

The following diagram shows how all pods are connected to a shared network.

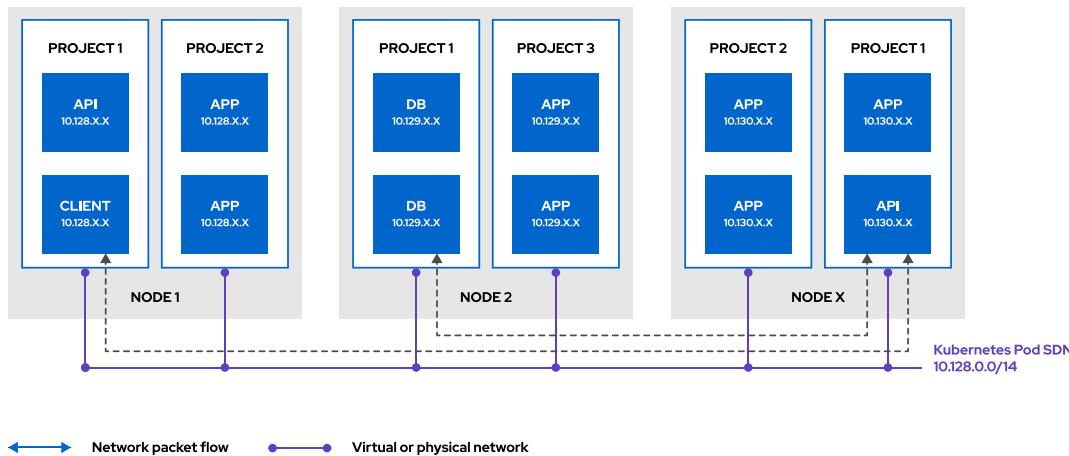
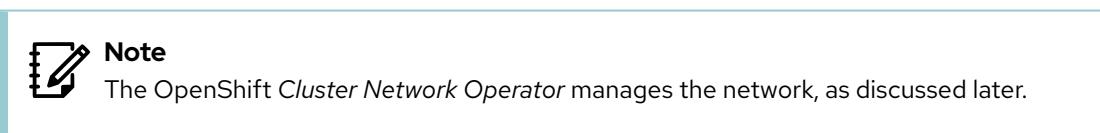


Figure 12.1: Kubernetes basic networking

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 the 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 to 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 balances the load between these three pods.

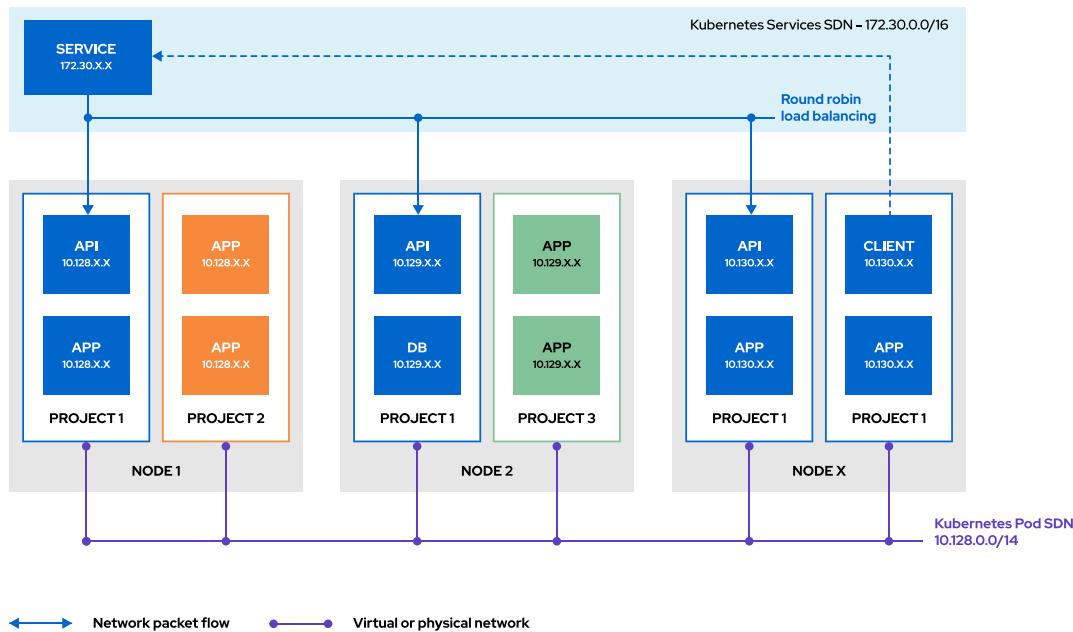


Figure 12.2: Using services for accessing applications

The following YAML definition shows you how to 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 8443.

```

kind: Service
apiVersion: v1
metadata:
  name: application-frontend ①
  labels:
    app: frontend-svc ②
spec:
  ports: ③
    - name: HTTP
      protocol: TCP
      port: 443 ④
      targetPort: 8443 ⑤
  selector: ⑥

```

```
app: shopping-cart
name: frontend
type: ClusterIP ⑦
```

- ① The name of the service. This identifier allows you to manage the service after its creation.
- ② A label that you can use as a selector. This allow you to logically group your services.
- ③ 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.
- ④ This is the port that the service exposes. You use this port to connect to the application that the service exposes.
- ⑤ This is the port on which the application listens. The service creates a forwarding rule from the service port to the service target port.
- ⑥ 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**.

- ⑤ This is how the service is exposed. **ClusterIP** exposes the service using an IP address internal to the cluster and is the default value. Other service types will be described in elsewhere this course.

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 DNS names to services that you define (for example, **db.backend.svc.cluster.local**)

For example, from a pod named **example-6c4984d949-7m26r** you can reach the **hello** pod through the **hello** service in the **test** project via the FQDN for the service.

```
[user@demo ~]$ oc rsh example-6c4984d949-7m26r curl \
>   hello.test.svc.cluster.local:8080
```

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.svc.cluster.local** host name from a container returns the IP address of the service. In this example, **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-name.namespace.svc.cluster-domain.cluster-domain
```

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._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 newly created **SRV** record 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 provider, and the IP address pools. Configuration of the Cluster Network Operator is done before installation, although it is possible to migrate from the OpenShift SDN default CNI network provider to the OVN-Kubernetes network provider.

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/cluster -o yaml
apiVersion: config.openshift.io/v1
kind: Network
...output omitted...
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14 ①
      hostPrefix: 23 ②
  externalIP:
    policy: {}
  networkType: OpenshiftSDN ③
```

```
serviceNetwork:  
- 172.30.0.0/16  
. . . output omitted . . .
```

- ➊ 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.
- ➋ Defines the host prefix. A value of **23** indicates a netmask of **255.255.254.0**, which translates to 512 allocatable IPs.
- ➌ Shows the current SDN provider. You can choose between **OpenShiftSDN**, **OVNKubernetes**, and **Kuryr**.



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 Multus CNI

Multus is an open source project to support multiple network cards in OpenShift. One of the challenges that Multus solves is the migration of network function virtualization to containers. Multus 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. Special edge cases often seen in telecommunication services, edge computing, and virtualization are handled by Multus, allowing multiple network interfaces to pods.

Be aware that all Kubernetes and OpenShift networking features, such as services, ingress, and routes, ignore the extra network devices provided by Multus.



References

For more information, refer to the *Cluster Network Operator in OpenShift Container Platform* chapter in the Red Hat OpenShift Container Platform 4.5 *Networking* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#cluster-network-operator

Cluster Networking

<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.
- Troubleshoot communication between pods in your application using **oc debug**.
- Update an OpenShift service.

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

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

1.2. Create the **network-sdn** project.

```
[student@workstation ~]$ oc new-project network-sdn
Now using project "network-sdn" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

► 2. Deploy the database and restore its data.

The `/home/student/D0280/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 ~/D0280/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` option against `todo-db.yaml` 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 the `oc status` command 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.ocp4.example.com:6443

svc/mysql - 172.30.223.41:3306
deployment/mysql deploys registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-47
  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 the `oc cp` command 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 the `oc rsh` command to connect to the pod and restore the database.

```
[student@workstation network-sdn]$ oc rsh mysql-94dc6645b-hjjqb bash
bash-4.2$ mysql -u root items < /tmp/db-data.sql
```

- 2.7. Ensure that the **Item** table is present in the database.

```
bash-4.2$ mysql -u root items -e "show tables;"
```

Tables_in_items
Item

- 2.8. Exit the container.

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

- 3. Deploy the front end application. The **/home/student/D0280/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 the **oc create** command 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 the **oc get pods** command.

```
[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 create this route, 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.apps.ocp4.example.com
route.route.openshift.io/frontend exposed
```

- 4.2. List the routes in the project.

```
[student@workstation network-sdn]$ oc get routes
NAME      HOST/PORT          PATH  SERVICES  PORT  ...
frontend  todo.apps.ocp4.example.com  frontend  8080  ...
```

As you can see in the example, 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.ocp4.example.com/todo/`. Make sure to add the trailing slash at the end of the URL.

The application is not reachable, as shown in the following screen capture.

Application is not available

The application is currently not serving requests at this endpoint. It may not have been started or is still starting.

i Possible reasons you are seeing this page:

- **The host doesn't exist.** Make sure the hostname was typed correctly and that a route matching this hostname exists.
- **The host exists, but doesn't have a matching path.** Check if the URL path was typed correctly and that the route was created using the desired path.
- **Route and path matches, but all pods are down.** Make sure that the resources exposed by this route (pods, services, deployment configs, etc) have at least one pod running.

- 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 the **curl** command 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** deployment and the database is using **curl**, which supports a variety of protocols.

Use **curl** to connect to the database over port 3306, which is the MySQL default port. Make sure to replace the IP address with the one that you obtained previously for the **mysql** service. When finished, type **Ctrl+C** to exit the session, and then 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...
* Connected to 172.30.103.29 (172.30.103.29) port 3306 (#0)
J
8.0.21
* RCVD IAC 2
* RCVD IAC 199
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 then 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 the **curl** command to connect to the **frontend** application over port 8080. Make sure to replace the IP address with the one that you obtained previously for the **frontend** service.
- The following output indicates that the **curl** command times out. This could indicate either 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
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 the **curl** command in verbose mode against the **frontend** pod on port 8080. Replace the IP address with the one that you obtained previously for the **frontend** pod.

```
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
...output omitted...
```

The **curl** command 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 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 in the previous step uses **api** as the value.

```
[student@workstation network-sdn]$ oc describe deployment/frontend | \
> grep Labels -A1
Labels:          app=todonodejs
                 name=frontend
--
Labels:  app=todonodejs
         name=frontend
```

► 9. Update the **frontend** service and access the application.

- 9.1. Run the **oc edit** command 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 the **workstation** machine, open Firefox and access the To Do application at <http://todo.apps.ocp4.example.com/todo/>.

You should see the To Do application.

To Do List

Id	Description	Done	
1	Pick up newsp...	false	X
2	Buy groceries	true	X

Add Task

Description:

Completed:

Clear Save

First Previous **1** Next Last

- ▶ 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  
project.project.openshift.io "network-sdn" 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 network-sdn finish
```

This concludes the guided exercise.

Exposing Applications for External Access

Objectives

After completing this section, you should be able to allow and protect network connections to applications inside an OpenShift cluster.

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**.

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.

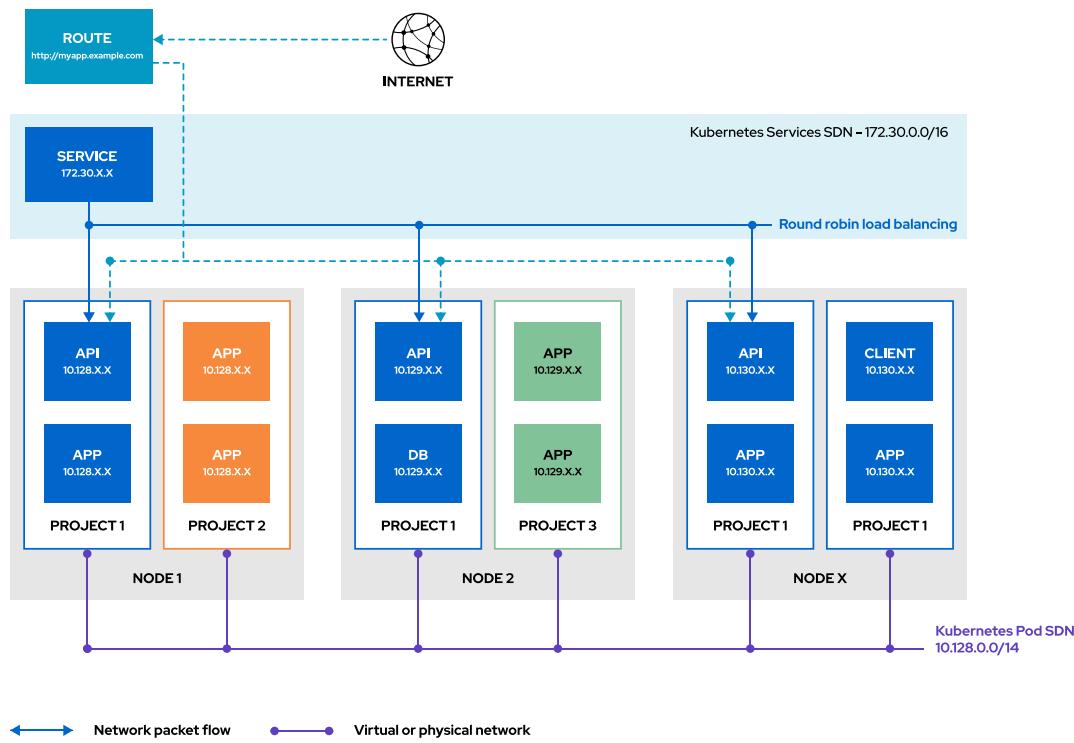


Figure 12.5: Using routes to expose applications

**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.

Describing Methods for Managing Ingress Traffic

The most common way to manage ingress traffic is with the Ingress Controller. OpenShift implements the Ingress Controller with 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.

Routes and ingress are the main resources for handling ingress traffic.

Route

Routes provide ingress traffic to services in the cluster. Routes were created before Kubernetes ingress objects and provide more features. Routes provide advanced features that may not be supported by Kubernetes ingress controllers through a standard interface, such as TLS re-encryption, TLS passthrough, and split traffic for blue-green deployments.

Ingress

An ingress is a Kubernetes resource that provides some of the same features as routes (which are an OpenShift resource). Ingresses accept external requests and proxy them based on the route. You can only allow certain types of traffic: HTTP, HTTPS and *server name identification (SNI)*, and TLS with SNI. In OpenShift, routes are generated to meet the conditions specified by the ingress object.

There are alternatives to ingress and routes, but they are for special use cases. The following service types provide external access to services.

External load balancer

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.

Service external IP

This method instructs OpenShift to set NAT rules to redirect traffic from one of the cluster IPs to the container.

NodePort

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.

Creating Routes

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 you omit 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 an **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.

Consider the following settings 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 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 ①
  labels: ②
    app: API
    name: api-frontend
spec:
  host: api.apps.acme.com ③
  to:
    kind: Service
```

```
name: api-frontend ④
port: ⑤
targetPort: 8443
```

- ①** The name of the route. This name must be unique.
- ②** A set of labels that you can use as selectors.
- ③** The host name of the route. This host name must be a subdomain of your wildcard domain because OpenShift routes the wildcard domain to the routers.
- ④** 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.
- ⑤** The application port. Because routes bypass services, this must match the application port and not the service port.

Securing Routes

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.

Passthrough

With passthrough 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. Passthrough 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.

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 --hostname 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.

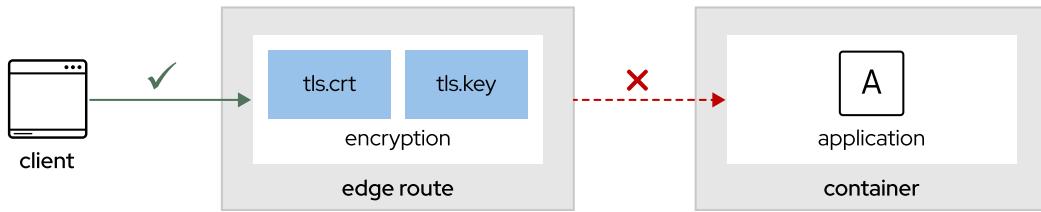


Figure 12.6: Securing applications with edge routes



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 Passthrough Routes

The previous example demonstrates how to create an edge route, that is, an OpenShift route that presents a certificate at the edge. Passthrough 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 passthrough 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.

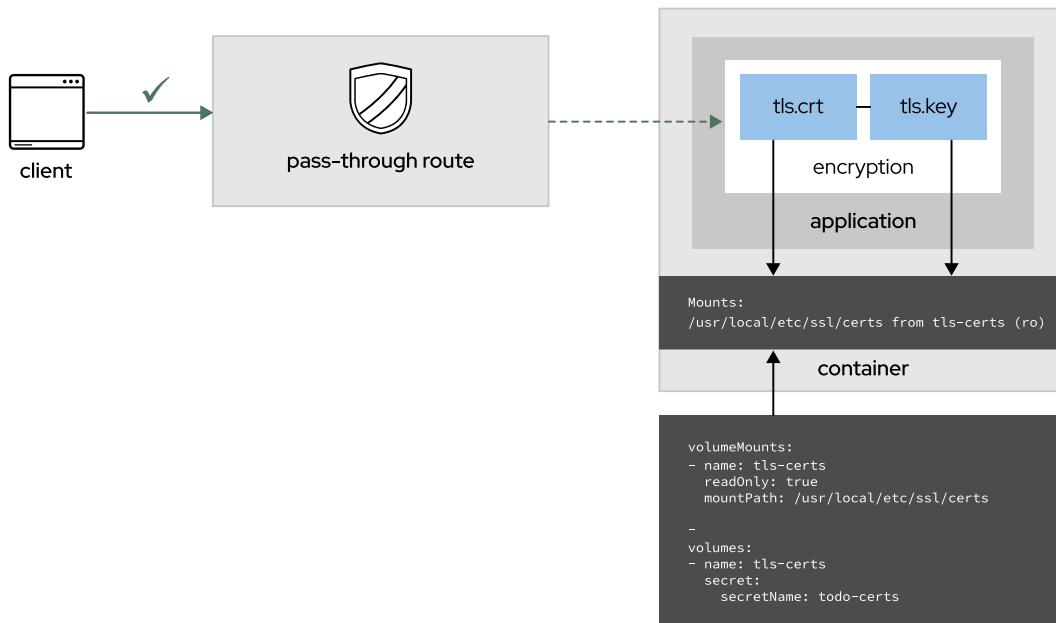


Figure 12.7: Securing applications with passthrough routes



References

For more information on how to manage routes, refer to *Configuring Routes* chapter in the Red Hat OpenShift Container Platform 4.5 *Networking* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#configuring-routes

For more information on how to configure ingress cluster traffic, refer to *Configuring ingress cluster traffic* chapter in the Red Hat OpenShift Container Platform 4.5 *Networking* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#configuring-ingress-cluster-traffic

Routes

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

Exposing Applications for External Access

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

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 **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.ocp4.example.com** and ***.ocp4.example.com** domains.

The CA certificate is accessible at **~/D0280/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. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

1.2. Create the **network-ingress** project.

```
[student@workstation ~]$ oc new-project network-ingress
Now using project "network-ingress" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 2. The OpenShift deployment file for the application is accessible at **~/D0280/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-http** service that points to the application pod. Create the application and expose the service.
- 2.1. Use the **oc create** command to deploy the application in the **network-ingress** OpenShift project.

```
[student@workstation ~]$ oc create -f \
> ~/D0280/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-ingress on server https://api.ocp4.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 the **oc expose** command to create a route for accessing the application. Give the route a host name of **todo-http.apps.ocp4.example.com**.

```
[student@workstation ~]$ oc expose svc todo-http \
> --hostname todo-http.apps.ocp4.example.com
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	SERVICES	PORT	...
todo-http	todo-http.apps.ocp4.example.com		todo-http	8080	...

- 2.5. On the **workstation** machine, open Firefox and access **http://todo-http.apps.ocp4.example.com**. Confirm that you can see the application.
- 2.6. Open a new terminal tab and run the **tcpdump** command with the following options to intercept the traffic on port 80:
 - **-i eth0** 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 eth0
[student@workstation ~]$ sudo tcpdump -i eth0 -A \
> -n port 80 | grep js
```



Note

The full command is available at [~/D0280/labs/network-ingress/tcpdump-command.txt](#).

- 2.7. On Firefox, refresh the page and notice the activity in the terminal. Press **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 route certificate is signed.

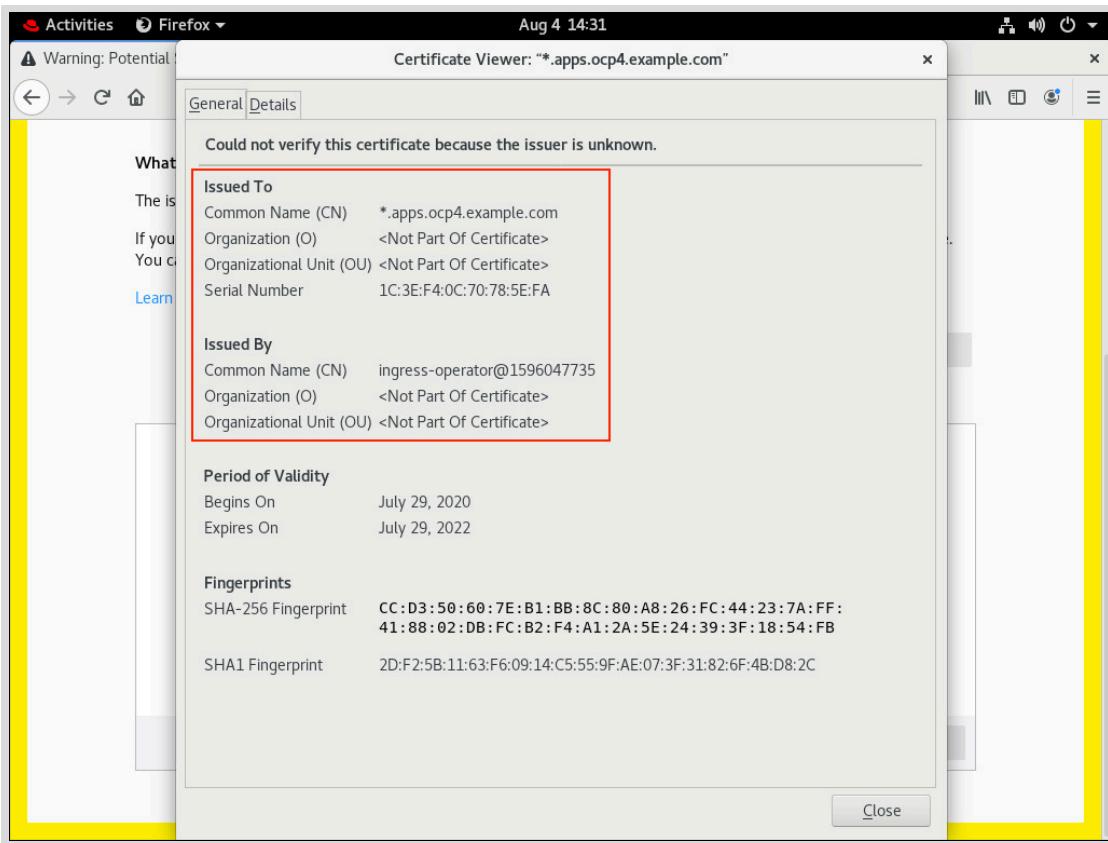
- 3.1. Go to [~/D0280/labs/network-ingress](#) and run the **oc create route** command to define the new route.

Give the route a host name of **todo-https.apps.ocp4.example.com**.

```
[student@workstation ~]$ cd ~/D0280/labs/network-ingress
[student@workstation network-ingress]$ oc create route edge todo-https \
> --service todo-http \
> --hostname todo-https.apps.ocp4.example.com
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.ocp4.example.com>. Upon first access, Firefox warns you about the certificate. Click **Advanced**, scroll down, and then click **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 the `curl` command to further confirm rejection of the certificate.

```
[student@workstation network-ingress]$ curl \
> https://todo-https.apps.ocp4.example.com
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 the **workstation** machine, log in to the cluster as the **admin** user. The **developer** user is not allowed to retrieve the CA.



Important

In a real world scenario, administrators with access to the namespace retrieve the CA and give it to the developers.

```
[student@workstation network-ingress]$ oc login -u admin -p redhat
```

3.5. Run the `oc extract` command 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 the **curl** command to retrieve the connection headers. Use the **--cacert** option to pass the CA to the **curl** command.

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.apps.ocp4.example.com
...output omitted...
* Server certificate:
*   subject: CN=*.apps.ocp4.example.com
*   start date: Jul 29 18:35:37 2020 GMT
*   expire date: Jul 29 18:35:38 2022 GMT
*   subjectAltName: host "todo-https.apps.ocp4.example.com" matched cert's
  "*.apps.ocp4.example.com"
*   issuer: CN=ingress-operator@1596047735
*   SSL certificate verify ok.
...output omitted...
```

The output indicates that the remote certificate is trusted because it matches the CA.

- 3.7. Log back as the **developer** user.

```
[student@workstation network-ingress]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 3.8. Although the traffic is encrypted at the edge with a certificate, you can still access the insecure traffic at the service level 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 the **curl** command 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
...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 passthrough 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 passthrough 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 -l
total 16
-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-**

https.apps.ocp4.example.com. Make sure to type the subject of the request on one line. Alternatively, remove the **-subj** option and its content. Without the **-subj** option, the **openssl** command prompts you for the values; make sure to indicate a common name (**CN**) of **todo-https.apps.ocp4.example.com**.

```
[student@workstation certs]$ openssl req -new \
> -subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\
> CN=todo-https.apps.ocp4.example.com" \
> -key training.key -out training.csr
```

- 4.4. Finally, generate the signed certificate. Notice the use 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.ocp4.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 -l
total 36
-rw-r--r--. 1 student student 599 Jul 31 09:35 openssl-commands.txt
-rw-r--r--. 1 student student 33 Aug 3 12:38 passphrase.txt
-rw-----. 1 student student 1743 Aug 3 12:38 training-CA.key
-rw-r--r--. 1 student student 1334 Aug 3 12:38 training-CA.pem
-rw-rw-r--. 1 student student 41 Aug 3 13:40 training-CA.srl
-rw-rw-r--. 1 student student 1391 Aug 3 13:40 training.crt
-rw-rw-r--. 1 student student 1017 Aug 3 13:39 training.csr
-rw-r--r--. 1 student student 352 Aug 3 12:38 training.ext
-rw-----. 1 student student 1675 Aug 3 13:38 training.key
```

- 4.6. Return 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 the **workstation** machine. 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**. Use the **--cert** and **--key** options to embed the TLS certificates. Use **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 **~/D0280/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 route.

- 6.1. Run the **oc create route** command to define the new route.

Give the route a host name of **todo-https.apps.ocp4.example.com**.

```
[student@workstation network-ingress]$ oc create route passthrough todo-https \
>   --service todo-https --port 8443 \
>   --hostname todo-https.apps.ocp4.example.com
route.route.openshift.io/todo-https created
```

- 6.2. Use the **curl** command in verbose mode to test the route and read the certificate. Use the **--cacert** option to pass the CA certificate to the **curl** command.

The output indicates a match between the certificate chain and the application certificate. This match 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.apps.ocp4.example.com
...output omitted...
* Server certificate:
*   subject: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; CN=todo-
https.apps.ocp4.example.com
*   start date: Aug  3 17:40:30 2020 GMT
*   expire date: Aug  2 17:40:30 2025 GMT
*   subjectAltName: host "todo-https.apps.ocp4.example.com" matched cert's
"*.apps.ocp4.example.com"
*   issuer: C=US; ST=North Carolina; L=Raleigh; O=Red Hat; CN=ocp4.example.com
*   SSL certificate verify ok.
...output omitted...
```

- 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 the **curl** command 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://172.30.121.154
HTTP/1.1 301 Moved Permanently
Server: nginx/1.14.1
Date: Sat, Thu, 20 Aug 2020 00:01:35 GMT
Content-Type: text/html
Connection: keep-alive
Location: https://172.30.121.154:8443/
```

- 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 -s -k https://172.30.121.154:8443 | head -n5
<!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. Navigate to the home directory and delete the **network-ingress** project.

```
[student@workstation network-ingress]$ cd
[student@workstation ~]$ oc delete project network-ingress
project.project.openshift.io "network-ingress" 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 network-ingress finish
```

This concludes the guided exercise.

Configuring Network Policies

Objectives

After completing this section, you should be able to restrict network traffic between projects and pods.

Managing Network Policies in OpenShift

Network policies allow you to configure isolation policies for individual pods. Network policies do not require administrative privileges, giving developers more control over the applications in their projects.

You can use network policies to create logical zones in the SDN that map to your organization network zones. The benefit of this approach is that the location of running pods becomes irrelevant because network policies allow you to segregate traffic regardless of where it originates.

To manage network communication between two namespaces, assign a label to the namespace that needs access to another namespace. The following command assigns the **name=network-1** label to the **network-1** namespace.

```
[user@demo ~]$ oc label namespace network-1 name=network-1
```

The following examples describe network policies that allow communication between the **network-1** and **network-2** namespaces.

- The following network policy applies to all pods with the label **deployment="product-catalog"** in the **network-1** namespace. The policy allows TCP traffic over port 8080 from pods whose label is **role="qa"** in the **network-2** namespace.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: network-1-policy
spec:
  podSelector: ①
  matchLabels:
    deployment: product-catalog

  ingress: ②
  - from: ③
    - namespaceSelector:
        matchLabels:
          name: network-2
    podSelector:
      matchLabels:
        role: qa
```

```
ports:④
- port: 8080
  protocol: TCP
```

- ➊ The top-level **podSelector** field is required and defines which pods use the network policy. If the **podSelector** is empty, all pods in the namespace are matched.
- ➋ The **ingress** field defines a list of ingress traffic rules to apply to the matched pods from the top-level **podSelector**.
- ➌ The **from** field defines a list of rules to match traffic from all sources. The selectors are not limited to the project in which the network policy is defined.
- ➍ The **ports** field is a list of destination ports that allow traffic to reach the selected pods.
- The following network policy allows traffic from all the pods and ports in the **network-1** namespace to all pods and ports in the **network-2** namespace. This policy is less restrictive than the **network-1** policy, because it does not restrict traffic from any pods in the **network-1** namespace.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: network-2-policy
spec:
  podSelector: {}

  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: network-1
```



Note

Network policies are Kubernetes resources. As such, you can manage them using **oc** commands.

One benefit of using network policies is the management of security between projects (tenants) that you cannot do with layer 2 technologies, such as VLANs. This approach allows you to create tailored policies between projects to make sure users can only access what they should (which conforms to the least privilege approach).

The fields in the network policy that take a list of objects can either be combined in the same object or listed as multiple objects. If combined, the conditions are combined with a logical *AND*. If separated in a list, the conditions are combined with a logical *OR*. The logic options allow you to create very specific policy rules. The following examples highlight the differences the syntax can make.

- This example combines the selectors into one rule, thereby only allowing access from pods in the **dev** namespace with the **app=mobile** label. This is an example of a logical *AND*.

```
...output omitted...
ingress:
- from:
  - namespaceSelector:
```

```

matchLabels:
  name: dev
podSelector:
  matchLabels:
    app: mobile

```

- By changing the **podSelector** field in the previous example to be an item in the **from** list, all pods in the **dev** namespace and all pods from any namespace labeled **app=mobile** can reach the pods that match the top-level **podSelector** field. This is an example of a logical OR.

```

...output omitted...
ingress:
- from:
  - namespaceSelector:
    matchLabels:
      name: dev
  - podSelector:
    matchLabels:
      app: mobile

```

If a pod is matched by selectors in one or more network policies, then the pod will only accept connections that are allowed by at least one of those network policies. A strict example is a policy to deny all ingress traffic to pods in your project, including from other pods inside your project. An empty pod selector means that this policy applies to all pods in this project. The following policy blocks all traffic because no ingress rules are defined. Traffic is blocked unless you also define an explicit policy that overrides this default behavior.

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector: {}

```

If you have Cluster Monitoring or exposed routes, then you need to allow ingress from them as well. The following policies allow ingress from OpenShift monitoring and Ingress Controllers.

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
      matchLabels:
        network.openshift.io/policy-group: ingress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:

```

```
name: allow-from-openshift-monitoring
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: monitoring
```



Important

If the **default** Ingress Controller uses the **HostNetwork** endpoint publishing strategy, then the **default** namespace requires the **network.openshift.io/policy-group=ingress** label.

Check the endpoint publishing strategy using the **oc describe** command to describe the **ingresscontroller/default** resource in the **openshift-ingress-controller** namespace.

For more information, refer to the documentation linked below in the references.



References

For more information about network policy, refer to the *Network policy* chapter in the Red Hat OpenShift Container Platform 4.5 *Networking* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/networking/index#network-policy

► Guided Exercise

Configuring Network Policies

In this exercise, you will create network policies and review pod isolation created by these network policies.

Outcomes

You should be able to:

- Create network policies to control communication between pods.
- Verify ingress traffic is limited to pods.

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 environment is ready and downloads the resource files necessary for the exercise.

```
[student@workstation ~]$ lab network-policy start
```

► 1. Log in to the OpenShift cluster and create the **network-policy** project.

- 1.1. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Create the **network-policy** project.

```
[student@workstation ~]$ oc new-project network-policy
Now using project "network-policy" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

► 2. Create two deployments and create a route for one of them.

- 2.1. Create two deployments using the **oc new-app** command with the **quay.io/redhattraining/hello-world-nginx:v1.0** image. Name the first deployment **hello** and the second deployment **test**.

```
[student@workstation ~]$ oc new-app --name hello --docker-image \
> quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
```

```

imagestream.image.openshift.io "hello" created
deployment.apps "hello" created
service "hello" created
--> Success
...output omitted...
[student@workstation ~]$ oc new-app --name test --docker-image \
>   quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "test" created
deployment.apps "test" created
service "test" created
--> Success
...output omitted...

```

- 2.2. Use the **oc expose** command to create a route to the **hello** service.

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

- 3. Verify access to the **hello** pod with the **oc rsh** and **curl** commands.

- 3.1. Open a second terminal and run the script located at **~/D0280/labs/network-policy/display-project-info.sh**. This script provides information about the pods, service, and route used in the rest of this exercise.

```
[student@workstation ~]$ ~/D0280/labs/network-policy/display-project-info.sh
=====
PROJECT: network-policy

POD NAME          IP ADDRESS
hello-6c4984d949-g28c4  10.8.0.13
test-c4d74c9d5-5pq9s  10.8.0.14

SERVICE NAME    CLUSTER-IP
hello           172.30.137.226
test            172.30.159.119

ROUTE NAME      HOSTNAME                      PORT
hello           hello-network-policy.apps.ocp4.example.com  8080-tcp
=====
```

- 3.2. Use the **oc rsh** and **curl** commands to confirm that the **test** pod can access the IP address of the **hello** pod.

```
[student@workstation ~]$ oc rsh test-c4d74c9d5-5pq9s curl 10.8.0.13:8080 | \
>   grep Hello
<h1>Hello, world from nginx!</h1>
```

- 3.3. Use the **oc rsh** and **curl** commands to confirm that the **test** pod can access the IP address of the **hello** service.

```
[student@workstation ~]$ oc rsh test-c4d74c9d5-5pq9s curl 172.30.137.226:8080 | \
>   grep Hello
<h1>Hello, world from nginx!</h1>
```

- 3.4. Verify access to the **hello** pod using the **curl** command against the URL of the **hello** route.

```
[student@workstation ~]$ curl -s hello-network-policy.apps.ocp4.example.com | \
>   grep Hello
<h1>Hello, world from nginx!</h1>
```

▶ 4. Create the **network-test** project and a deployment named **sample-app**.

- 4.1. Create the **network-test** project.

```
[student@workstation ~]$ oc new-project network-test
Now using project "network-test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 4.2. Create the **sample-app** deployment with the **quay.io/redhattraining/hello-world-nginx:v1.0** image. The web app listens on port 8080.

```
[student@workstation ~]$ oc new-app --name sample-app --docker-image \
>   quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "sample-app" created
deployment.apps "sample-app" created
service "sample-app" created
--> Success
...output omitted...
```

▶ 5. Verify that pods in a different namespace can access the **hello** and **test** pods in the **network-policy** namespace.

- 5.1. In the second terminal, run the **display-project-info.sh** script again to view the full name of the **sample-app** pod.

```
[student@workstation ~]$ ~/DO280/labs/network-policy/display-project-info.sh
...output omitted...
PROJECT: network-test

POD NAME
sample-app-d5f945-spx9q
=====
```

- 5.2. Returning to the first terminal, use the **oc rsh** and **curl** commands to confirm that the **sample-app** pod can access the IP address of the **hello** pod.

```
[student@workstation ~]$ oc rsh sample-app-d5f945-spx9q curl 10.8.0.13:8080 | \
>   grep Hello
<h1>Hello, world from nginx!</h1>
```

- 5.3. Use the **oc rsh** and **curl** commands to confirm access to the **test** pod from the **sample-app** pod. Target the IP address previously retrieved for the **test** pod.

```
[student@workstation ~]$ oc rsh sample-app-d5f945-spx9q curl 10.8.0.14:8080 | \
>   grep Hello
<h1>Hello, world from nginx!</h1>
```

- ▶ 6. From the **network-policy** project, create the **deny-all** network policy using the resource file available at **~/D0280/labs/network-policy/deny-all.yaml**.

- 6.1. Switch to the **network-policy** project.

```
[student@workstation ~]$ oc project network-policy
Now using project "network-policy" on server "https://api.ocp4.example.com:6443".
```

- 6.2. Go to the **~/D0280/labs/network-policy/** directory.

```
[student@workstation ~]$ cd ~/D0280/labs/network-policy/
```

- 6.3. Use a text editor to update the **deny-all.yaml** file with an empty **podSelector** to target all pods in the namespace. A solution is provided at **~/D0280/solutions/network-policy/deny-all.yaml**.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
spec:
  podSelector: {}
```

- 6.4. Create the network policy with the **oc create** command.

```
[student@workstation network-policy]$ oc create -f deny-all.yaml
networkpolicy.networking.k8s.io/deny-all created
```

- ▶ 7. Verify there is no longer access to the pods in the **network-policy** namespace.

- 7.1. Verify there is no longer access to the **hello** pod via the exposed route. Wait a few seconds, and then press **Ctrl+C** to exit the **curl** command that does not reply.

```
[student@workstation network-policy]$ curl -s \
>   hello-network-policy.apps.ocp4.example.com | grep Hello
^C
```

**Important**

If the **hello** pod lands on the same node as a **router-default** pod, then the **curl** command works when traffic goes through that router pod. This is only the case with three-node clusters. In a traditional OpenShift cluster, where the control plane or infrastructure nodes are separated from the worker nodes, the network policy would apply to all the router pods in the cluster.

If the **curl** command succeeds, then run the command again to validate the network policy works as expected. This second attempt should go through the other router pod in the cluster.

- 7.2. Verify that the **test** pod can no longer access the IP address of the **hello** pod. Wait a few seconds, and then press **Ctrl+C** to exit the **curl** command that does not reply.

```
[student@workstation network-policy]$ oc rsh test-c4d74c9d5-5pq9s curl \
>   10.8.0.13:8080 | grep Hello
^CCommand terminated with exit code 130
```

- 7.3. From the **network-test** project, confirm that the **sample-app** pod can no longer access the IP address of the **test** pod. Wait a few seconds, and then press **Ctrl+C** to exit the **curl** command that does not reply.

```
[student@workstation network-policy]$ oc project network-test
Now using project "network-test" on server "https://api.ocp4.example.com:6443".
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
>   10.8.0.14:8080 | grep Hello
^CCommand terminated with exit code 130
```

- 8. Create a network policy to allow traffic to the **hello** pod in the **network-policy** namespace from the **sample-app** pod in the **network-test** namespace over TCP on port 8080. Use the resource file available at **~/DO280/labs/network-policy/allow-specific.yaml**.

- 8.1. Use a text editor to replace the **CHANGE_ME** sections in the **allow-specific.yaml** file as follows. A solution has been provided at **~/DO280/solutions/network-policy/allow-specific.yaml**.

```
...output omitted...
spec:
  podSelector:
    matchLabels:
      deployment: hello
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            name: network-test
  podSelector:
    matchLabels:
      deployment: sample-app
```

```
ports:
- port: 8080
  protocol: TCP
```

- 8.2. Create the network policy with the **oc create** command.

```
[student@workstation network-policy]$ oc create -n network-policy -f \
>   allow-specific.yaml
networkpolicy.networking.k8s.io/allow-specific created
```

- 8.3. View the network policies in the **network-policy** namespace.

```
[student@workstation network-policy]$ oc get networkpolicies -n network-policy
NAME          POD-SELECTOR      AGE
allow-specific deployment=hello  11s
deny-all       <none>           5m6s
```

- 9. As the **admin** user, label the **network-test** namespace with the **name=network-test** label.

- 9.1. Log in as the **admin** user.

```
[student@workstation network-policy]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 9.2. Use the **oc label** command to apply the **name=network-test** label.

```
[student@workstation network-policy]$ oc label namespace network-test \
>   name=network-test
namespace/network-test labeled
```



Important

The **allow-specific** network policy uses labels to match the name of a namespace. By default, namespaces and projects do not get any labels automatically.

- 9.3. Confirm the label was applied and log in as the **developer** user.

```
[student@workstation network-policy]$ oc describe namespace network-test
Name:        network-test
Labels:      name=network-test
...output omitted...
[student@workstation network-policy]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 10. Verify that the **sample-app** pod can access the IP address of the **hello** pod, but cannot access the IP address of the **test** pod.

- 10.1. Verify access to the **hello** pod in the **network-policy** namespace.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
>   10.8.0.13:8080 | grep Hello
<h1>Hello, world from nginx!</h1>
```

- 10.2. Verify there is no response from the **hello** pod on another port. Because the network policy only allows access to port 8080 on the **hello** pod, requests made to any other port are ignored and eventually time out. Wait a few seconds, and then press **Ctrl+C** to exit the **curl** command that does not reply.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
>   10.8.0.13:8181 | grep Hello
^CCommand terminated with exit code 130
```

- 10.3. Verify there is no access to the **test** pod. Wait a few seconds, and then press **Ctrl+C** to exit the **curl** command that does not reply.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q curl \
>   10.8.0.14:8080 | grep Hello
^CCommand terminated with exit code 130
```

- 11. Create a network policy to allow traffic to the **hello** pod from the exposed route. Use the resource file available at [~/DO280/labs/network-policy/allow-from-openshift-ingress.yaml](#).

- 11.1. Use a text editor to replace the **CHANGE_ME** values in the **allow-from-openshift-ingress.yaml** file as follows. A solution is provided at [~/DO280/solutions/network-policy/allow-from-openshift-ingress.yaml](#).

```
...output omitted...
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
```

- 11.2. Create the network policy with the **oc create** command.

```
[student@workstation network-policy]$ oc create -n network-policy -f \
>   allow-from-openshift-ingress.yaml
networkpolicy.networking.k8s.io/allow-from-openshift-ingress created
```

- 11.3. View the network policies in the **network-policy** namespace.

```
[student@workstation network-policy]$ oc get networkpolicies -n network-policy
NAME                POD-SELECTOR   AGE
allow-from-openshift-ingress <none>    10s
allow-specific        deployment=hello  8m16s
deny-all             <none>       13m
```

- 11.4. As the **admin** user, label the **default** namespace with the **network.openshift.io/policy-group=ingress** label.

```
[student@workstation network-policy]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation network-policy]$ oc label namespace default \
>   network.openshift.io/policy-group=ingress
namespace/default labeled
```



Note

Applying this label to the **default** namespace is only necessary because the classroom's **default** Ingress Controller uses the **HostNetwork** endpoint publishing strategy.

- 11.5. Verify access to the **hello** pod via the exposed route.

```
[student@workstation network-policy]$ curl -s \
>   hello-network-policy.apps.ocp4.example.com | grep Hello
<h1>Hello, world from nginx!</h1>
```

- ▶ 12. Close the terminal window that has the output of the **display-project-info.sh** script. Navigate to the home directory.

```
[student@workstation network-policy]$ cd
```

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-policy finish
```

This concludes the guided exercise.

► Lab

Configuring OpenShift Networking for Applications

In this lab, you will configure a TLS passthrough route for your application.

Outcomes

You should be able to:

- Deploy an application and configure an insecure route.
- Restrict traffic to the applications.
- Generate a TLS certificate for an application.
- Configure a passthrough route for an application with a TLS certificate.

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 self-signed certificate authority (CA) that you use in this lab.

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

In this review, you deploy a PHP application that prints some information about the system. The application is available with two different configurations: 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.
2. As the **developer** user, deploy the insecure version of the PHP application to the **network-review** project. Use the resource file available at `~/DO280/labs/network-review/php-http.yaml`.

Before deploying the application, make the necessary changes to the file, specifically, 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 hostname of `php-http.apps.ocp4.example.com`, to access the application.

From the **workstation** machine, use Firefox to access `http://php-http.apps.ocp4.example.com`. Confirm the availability of the application before proceeding to the next step.

4. Create a network policy in the **network-review** namespace to deny all ingress traffic by default. When configured correctly, the network policy also prevents pods within the **network-review** namespace from communicating with each other.
Use the resource file available at `~/DO280/labs/network-review/deny-all.yaml`. Make the necessary changes to target all pods in the namespace.
5. Create a network policy to allow ingress traffic to routes in the **network-review** namespace.
Use the resource file available at `~/DO280/labs/network-review/allow-from-openshift-ingress.yaml`. Make the necessary changes to target all pods in the namespace and allow traffic from the default ingress controller.
Because the classroom environment uses the **HostNetwork** endpoint strategy, label the **default** namespace with the **network.openshift.io/policy-group=ingress** label. This action must be performed as the **admin** user.
6. As the **developer** user, generate and sign a TLS certificate for the encrypted version of the application.
Create a certificate signing request (CSR) for the `php-https.apps.ocp4.example.com` hostname. 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 certificate 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.
7. Create an OpenShift TLS secret named **php-certs** in the **network-review** project. Use the `~/DO280/labs/network-review/certs/training.crt` file for the certificate and the `~/DO280/labs/network-review/certs/training.key` file for the key.
8. Use the resource file, available at `~/DO280/labs/network-review/php-https.yaml`, to deploy the secure version of the PHP application. Deploy the application to the **network-review** project.
Before deploying the application, make the necessary changes to the resources file, specifically:
 - The location of the container.
 - The port the application listens on.
 - The name of the secret to mount as a volume.
9. Create a secure passthrough route named **php-https**, with a hostname of `php-https.apps.ocp4.example.com`, to access the secure version of the application.
From the **workstation** machine, use Firefox to access `https://php-https.apps.ocp4.example.com`. Accept the signed certificate and confirm the availability of the application.
10. **Optional step:** from the **workstation** machine, use the **curl** command to access the HTTPS version of the application.
Pass the CA certificate to the **curl** command to validate the secure connection.

11. Return to the home directory because the **lab network-review finish** command will delete the **network-review** directory.

```
[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 for Applications

In this lab, you will configure a TLS passthrough route for your application.

Outcomes

You should be able to:

- Deploy an application and configure an insecure route.
- Restrict traffic to the applications.
- Generate a TLS certificate for an application.
- Configure a passthrough route for an application with a TLS certificate.

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 self-signed certificate authority (CA) that you use in this lab.

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

In this review, you deploy a PHP application that prints some information about the system. The application is available with two different configurations: 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.

- 1.1. Log in to the cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Create the **network-review** project.

```
[student@workstation ~]$ oc new-project network-review
Now using project "network-review" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

2. As the **developer** user, deploy the insecure version of the PHP application to the **network-review** project. Use the resource file available at **~/D0280/labs/network-review/php-http.yaml**.

Before deploying the application, make the necessary changes to the file, specifically, 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. Go to the **~/D0280/labs/network-review/** directory.

```
[student@workstation ~]$ cd ~/D0280/labs/network-review/
```

- 2.2. Use a 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 the **oc create** command 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 the **oc get pods** command to ensure that there is a pod 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 hostname of **php-http.apps.ocp4.example.com**, to access the application.

From the **workstation** machine, use Firefox to access **http://php-http.apps.ocp4.example.com**. Confirm the availability of the application before proceeding to the next step.

- 3.1. Run the **oc expose** command to create a route for accessing the application. Give the route a hostname of **php-http.apps.ocp4.example.com**.

```
[student@workstation network-review]$ oc expose svc php-http \
>   --hostname php-http.apps.ocp4.example.com
route.route.openshift.io/php-http exposed
```

- 3.2. Retrieve the name of the route and copy it to the clipboard.

```
[student@workstation network-review]$ oc get routes
NAME      HOST/PORT          PATH  SERVICES  PORT  ...
php-http  php-http.apps.ocp4.example.com  php-http  8080  ...
```

- 3.3. From the **workstation** machine, open Firefox and access <http://php-http.apps.ocp4.example.com>.

Confirm that you can see the application.

About this application

A The application is currently served over HTTP

- **Current system load:** 2.5
- **Number of connections:** 1
- **Memory usage:** 8 Mb

4. Create a network policy in the **network-review** namespace to deny all ingress traffic by default. When configured correctly, the network policy also prevents pods within the **network-review** namespace from communicating with each other.

Use the resource file available at [~/D0280/labs/network-review/deny-all.yaml](#). Make the necessary changes to target all pods in the namespace.

- 4.1. Use a text editor to update the **deny-all.yaml** file with an empty pod selector to target all pods in the namespace.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
spec:
  podSelector: {}
```

- 4.2. Use the **oc create** command to create the network policy.

```
[student@workstation network-review]$ oc create -f deny-all.yaml
networkpolicy.networking.k8s.io/deny-all created
```

- 4.3. Use the **curl** command to verify that there is no access to the **php-http** pod from the route. Wait a few seconds, and then press **Ctrl+C** to exit the **curl** command.

```
[student@workstation network-review]$ curl http://php-http.apps.ocp4.example.com
^C
```

Important

If the **php-http** pod landed on the same node as a **router-default** pod, then the **curl** command works when traffic goes through that router pod. This is only the case with three-node clusters. In a traditional OpenShift cluster, where the control plane or infrastructure nodes are separated from the worker nodes, the network policy would apply to all the router pods in the cluster.

If the **curl** command succeeds, then run the command again to validate the network policy works as expected. This second attempt should go through the other router pod in the cluster.

5. Create a network policy to allow ingress traffic to routes in the **network-review** namespace.

Use the resource file available at **~/DO280/labs/network-review/allow-from-openshift-ingress.yaml**. Make the necessary changes to target all pods in the namespace and allow traffic from the default ingress controller.

Because the classroom environment uses the **HostNetwork** endpoint strategy, label the **default** namespace with the **network.openshift.io/policy-group=ingress** label. This action must be performed as the **admin** user.

- 5.1. Use a text editor to update the **allow-from-openshift-ingress.yaml** file with an empty pod selector to target all pods in the namespace. Include a namespace selector to match the **network.openshift.io/policy-group=ingress** label.

```
...output omitted...
spec:
  podSelector: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              network.openshift.io/policy-group: ingress
```

- 5.2. Use the **oc create** command to create the network policy.

```
[student@workstation network-review]$ oc create -f \
>   allow-from-openshift-ingress.yaml
networkpolicy.networking.k8s.io/allow-from-openshift-ingress created
```

- 5.3. As the **admin** user, label the **default** namespace with the **network.openshift.io/policy-group=ingress** label.

```
[student@workstation network-review]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation network-policy]$ oc label namespace default \
>   network.openshift.io/policy-group=ingress
namespace/default labeled
```

- 5.4. Use the **curl** command to verify there is access to the **php-http** pod from the route. Because the classroom is running a three-node cluster, run the **curl** command multiple times to validate access through all router pods.

```
[student@workstation network-review]$ for X in {1..4}
>   do
>     curl -s http://php-http.apps.ocp4.example.com | grep "PHP"
>   done
<title>PHP Application</title>
<title>PHP Application</title>
<title>PHP Application</title>
<title>PHP Application</title>
```

6. As the **developer** user, generate and sign a TLS certificate for the encrypted version of the application.

Create a certificate signing request (CSR) for the **php-https.apps.ocp4.example.com** hostname. Save the CSR to **~/D0280/labs/network-review/certs/training.csr**.

Use the CSR to generate a certificate and save it to **~/D0280/labs/network-review/certs/training.crt**. To generate the certificate, pass as arguments the CA certificate accessible at **~/D0280/labs/network-review/certs/training-CA.pem** and the CSR.

You can use the **~/D0280/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.

- 6.1. Log in as the **developer** user to complete the rest of this lab.

```
[student@workstation network-review]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 6.2. Go to the **~/D0280/labs/network-review/certs** directory.

```
[student@workstation network-review]$ cd certs
```

- 6.3. Generate the certificate signing request (CSR) for **php-https.apps.ocp4.example.com**. Make sure to type the subject of the request on one line. Alternatively, remove the **-subj** option and its content. This command prompts you for the values; make sure to indicate a common name (**CN**) of **php-https.apps.ocp4.example.com**.

**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 -key training.key \
> -subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\
> CN=php-https.apps.ocp4.example.com" \
> -out training.csr
```

Alternatively, open the **openssl-commands.txt** text file. Copy and paste the first **openssl** command to your terminal. Replace the wildcard domain with **apps.ocp4.example.com** and the output file with **training.csr**.

**Note**

The command does not generate any output.

- 6.4. 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 \
> -CA training-CA.pem -CAkey training-CA.key -CAcreateserial \
> -passin file:passphrase.txt \
> -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.ocp4.example.com
Getting CA Private Key
```

Alternatively, copy and paste the second **openssl** command in the **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**.

- 6.5. Ensure that the newly created certificate and the key are present in the current directory.

```
[student@workstation certs]$ ls -l
total 36
-rw-r--r--. 1 student student 564 Jul 31 09:35 openssl-commands.txt
-rw-r--r--. 1 student student 33 Aug 3 13:59 passphrase.txt
-rw-----. 1 student student 1751 Aug 3 13:59 training-CA.key
-rw-r--r--. 1 student student 1334 Aug 3 13:59 training-CA.pem
-rw-rw-r--. 1 student student 41 Aug 3 14:20 training-CA.srl
-rw-rw-r--. 1 student student 1395 Aug 3 14:20 training.crt
-rw-rw-r--. 1 student student 1021 Aug 3 14:16 training.csr
-rw-r--r--. 1 student student 352 Aug 3 13:59 training.ext
-rw-----. 1 student student 1675 Aug 3 13:59 training.key
```

- 6.6. 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 ~/D0280/labs/network-review
```

7. Create an OpenShift TLS secret named **php-certs** in the **network-review** project. Use the **~/D0280/labs/network-review/certs/training.crt** file for the certificate and the **~/D0280/labs/network-review/certs/training.key** file for the key.
 - 7.1. 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-review]$ oc create secret tls php-certs \
>   --cert certs/training.crt \
>   --key certs/training.key
secret/php-certs created
```

- 7.2. Retrieve the list of secrets to make sure that it is present.

```
[student@workstation network-review]$ oc get secrets
NAME          TYPE           DATA   AGE
...output omitted...
php-certs     kubernetes.io/tls   2      93s
```

8. Use the resource file, available at **~/D0280/labs/network-review/php-https.yaml**, to deploy the secure version of the PHP application. Deploy the application to the **network-review** project.

Before deploying the application, make the necessary changes to the resources file, specifically:

- The location of the container.
- The port the application listens on.
- The name of the secret to mount as a volume.

- 8.1. Use a 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.

- Use the **oc create** command 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
```

- Wait a few moments, and then run the **oc get pods** command to ensure that the **php-https** pod is running.

```
[student@workstation network-review]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
php-https-6cb58c847b-7qsbd   1/1     Running   0          8m11s
php-https-84498cd794-hvf7w   1/1     Running   0          26s
```

- Create a secure passthrough route named **php-https**, with a hostname of **php-https.apps.ocp4.example.com**, to access the secure version of the application.

From the **workstation** machine, use Firefox to access **https://php-https.apps.ocp4.example.com**. Accept the signed certificate and confirm the availability of the application.

- Run the **oc create route** command to create a passthrough route for accessing the application. Give the route a hostname of **php-https.apps.ocp4.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.apps.ocp4.example.com
route.route.openshift.io/php-https created
```

- Retrieve the name of the route and copy it to the clipboard.

```
[student@workstation network-review]$ oc get routes
NAME      HOST/PORT          ... SERVICES   PORT  TERMINATION
php-http  php-http.apps.ocp4.example.com  ...  php-http  8080
php-https php-https.apps.ocp4.example.com ...  php-https 8443  passthrough
```

- 9.3. From the **workstation** machine, open Firefox and access `https://php-https.apps.ocp4.example.com`.

Accept the signed certificate and confirm that you can see secure version of the application.

About this application

The application is currently served over TLS

- Current system load: 1
- Number of connections: 0
- Memory usage: 8 Mb

10. **Optional step:** from the **workstation** machine, use the `curl` command to access the HTTPS version of the application.

Pass the CA certificate to the `curl` command to validate the secure connection.

Use the `--cacert` option to pass the CA certificate to the `curl` command.

```
[student@workstation network-review]$ curl -v --cacert certs/training-CA.pem \
>   https://php-https.apps.ocp4.example.com
...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.ocp4.example.com
...output omitted...
The application is currently served over TLS      </span></strong>
...output omitted...
```

11. Return to the home directory because the `lab network-review finish` command will delete the `network-review` directory.

```
[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.

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 available to 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, passthrough, and re-encryption.
- Network policies control network traffic to pods. Logical zones can be created in the SDN to segregate traffic among pods in any namespace.

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 containers, pods, and projects.
- Control the number of replicas of a pod.

Sections

- Controlling Pod Scheduling Behavior (and Guided Exercise)
- Limiting Resource Usage by an Application (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, the methods used to influence scheduling, and apply these methods.

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 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, control plane 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 control plane node.



Note

The classroom environment uses a three-node cluster, which does not include additional compute nodes. The three-node cluster is available for bare-metal installation in OpenShift Container Platform 4.5. This type of cluster is applicable for resource constrained environments, such as far-edge deployments.

The control plane nodes in a three-node cluster do not have the **node-role.kubernetes.io/master:NoSchedule** taint. Regular application pods can be scheduled to the control plane nodes.

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.

- 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.

- 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 on nodes that have the same value for the **region** label.
- Replica pods are scheduled to run on nodes that have 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.

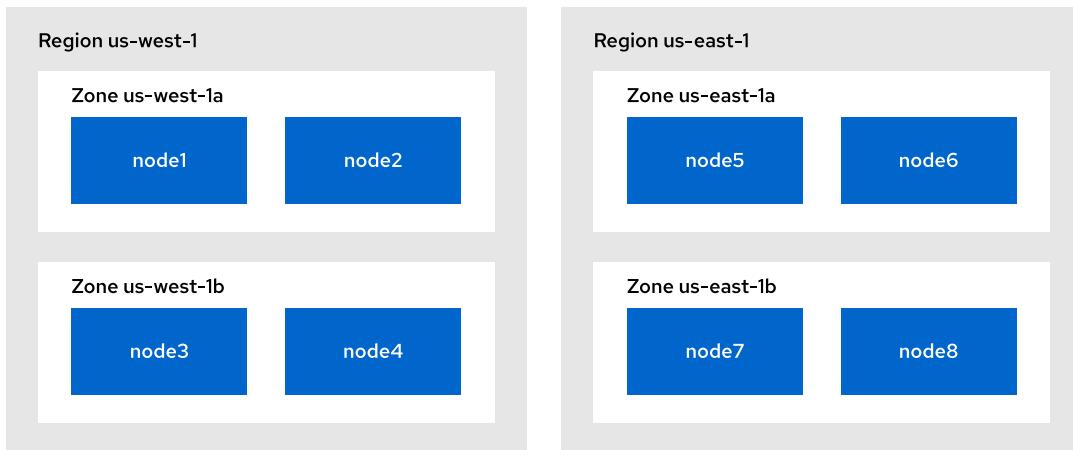


Figure 13.1: Sample cluster topology using regions and zones

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. The nodes displayed in these sample commands use the AWS full-stack automation setup.

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, 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. Machine sets are found in clusters using full-stack automation and in some clusters using pre-existing infrastructure that enable cloud provider integration. Bare-metal clusters do not use machine sets.

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 control plane 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:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: null
  labels:
    deployment: 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 4.5 Nodes documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/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 and creates a project that you will be using in the activity.

```
[student@workstation ~]$ lab schedule-pods start
```

- 1. As the **developer** user, create a new project named **schedule-pods**.

- 1.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Create a new project named **schedule-pods**.

```
[student@workstation ~]$ oc new-project schedule-pods
Now using project "schedule-pods" on server "https://api.ocp4.example.com".
...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 new-app --name hello \
>   --docker-image quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Creating resources ...
  imagestream.image.openshift.io "hello" created
  deployment.apps "hello" created
  service "hello" created
--> Success
...output omitted...
```

2.2. Create a route to the application.

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

2.3. Manually scale the application so there are four running pods.

```
[student@workstation ~]$ oc scale --replicas 4 deployment/hello
deployment.apps/hello scaled
```

2.4. Verify that the four running pods are distributed between the nodes.

```
[student@workstation ~]$ oc get pods -o wide
NAME           READY   STATUS    ...   IP          NODE   ...
hello-6c4984d949-78qsp  1/1     Running   ...  10.9.0.30  master02  ...
hello-6c4984d949-cf6tb  1/1     Running   ...  10.10.0.20  master01  ...
hello-6c4984d949-kwgbg  1/1     Running   ...  10.8.0.38  master03  ...
hello-6c4984d949-mb8z7  1/1     Running   ...  10.10.0.19  master01  ...
```



Note

Depending on the existing load on each node, your output may be different. Although the scheduler will attempt to distribute the pods, the distribution may not be even.

- 3. Prepare the 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 **master01** node and the **env=prod** label to the **master02** 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 and cannot label nodes.

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

3.2. Verify that none of the nodes use the **env** label.

```
[student@workstation ~]$ oc get nodes -L env
NAME      STATUS    ROLES      AGE      VERSION      ENV
master01   Ready     master,worker  5d18h   v1.18.3+012b3ec
master02   Ready     master,worker  5d18h   v1.18.3+012b3ec
master03   Ready     master,worker  5d18h   v1.18.3+012b3ec
```

- 3.3. Add the **env=dev** label to the **master01** node to indicate that it is a development node.

```
[student@workstation ~]$ oc label node master01 env=dev
node/master01 labeled
```

- 3.4. Add the **env=prod** label to the **master02** node to indicate that it is a production node.

```
[student@workstation ~]$ oc label node master02 env=prod
node/master02 labeled
```

- 3.5. Verify that the 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
NAME      STATUS    ROLES      AGE      VERSION      ENV
master01   Ready     master,worker  5d18h   v1.18.3+012b3ec   dev
master02   Ready     master,worker  5d18h   v1.18.3+012b3ec   prod
master03   Ready     master,worker  5d18h   v1.18.3+012b3ec
```

- ▶ 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 developer
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.apps/hello edited
```

- 4.3. Verify that the application pods are deployed to the node with the **env=dev** label. Although it may take a little time to redeploy, the application pods must be deployed to the **master01** node.

```
[student@workstation ~]$ oc get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP           NODE      ...
hello-b556ccf8b-8scxd  1/1    Running   0          80s    10.10.0.14  master01 ...
hello-b556ccf8b-hb24w  1/1    Running   0          77s    10.10.0.16  master01 ...
hello-b556ccf8b-qxlj8  1/1    Running   0          80s    10.10.0.15  master01 ...
hello-b556ccf8b-sdxpj  1/1    Running   0          76s    10.10.0.17  master01 ...
```

- 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 all nodes.

- 5.1. Log in to your OpenShift cluster as the **admin** user.

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

- 5.2. Remove the **env** label from all nodes that have it.

```
[student@workstation ~]$ oc label node -l env env-
node/master01 labeled
node/master02 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 developer
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
"https://api.ocp4.example.com:6443".
```

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          6m19s
```

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  ...  0/3 nodes are available: 3 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 three nodes have 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.apps/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 by an Application

Objectives

After completing this section, you should be able to limit the resources consumed by containers, pods, and projects.

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 cannot 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 pod's resource limits.

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, then 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,mem=20Mi --limits cpu=80m,mem=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 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 preceding output, 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 also 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. Instead, to track resource limits, 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      AGE   REQUEST
compute-quota  51s   cpu: 500m/10, memory: 300Mi/1Gi ...
count-quota   28s   pods: 1/3, replicationcontrollers: 1/5, services: 1/2 ...
```

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 content creation. When creating a new resource, the quota usage immediately increments. When deleting a resource, the quota use decrements 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 OpenShift denies the action and returns an appropriate error message 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 the limits for a single pod or 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. Similarly, 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 listing shows a limit range defined using YAML syntax:

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "dev-limits"
spec:
  limits:
    - type: "Pod"
      max: ①
        cpu: "500m"
        memory: "750Mi"
      min: ②
        cpu: "10m"
        memory: "5Mi"
    - type: "Container"
      max: ③
        cpu: "500m"
        memory: "750Mi"
      min: ④
        cpu: "10m"
        memory: "5Mi"
    default: ⑤
      cpu: "100m"
      memory: "100Mi"
    defaultRequest: ⑥
      cpu: "20m"
      memory: "20Mi"
    - type: openshift.io/Image ⑦
      max:
        storage: 1Gi
    - type: openshift.io/ImageStream ⑧
      max:
        openshift.io/image-tags: 10
        openshift.io/images: 20
    - type: "PersistentVolumeClaim" ⑨

```

```

min:
  storage: "1Gi"
max:
  storage: "50Gi"

```

- ➊ The maximum amount of CPU and memory that all containers within a pod can consume. A new pod that exceeds the maximum limits is not created. An existing pod that exceeds the maximum limits is restarted.
- ➋ The minimum amount of CPU and memory consumed across all containers within a pod. A pod that does not satisfy the minimum requirements is not created. Because many pods only have one container, you might set the minimum pod values to the same values as the minimum container values.
- ➌ The maximum amount of CPU and memory that an individual container within a pod can consume. A new container that exceeds the maximum limits does not create the associated pod. An existing container that exceeds the maximum limits restarts the entire pod.
- ➍ The minimum amount of CPU and memory that an individual container within a pod can consume. A container that does not satisfy the minimum requirements prevents the associated pod from being created.
- ➎ The default maximum amount of CPU and memory that an individual container can consume. This is used when a CPU resource limit or a memory limit is not defined for the container.
- ➏ The default CPU and memory an individual container requests. This default is used when a CPU resource request or a memory request is not defined for the container. If CPU and memory quotas are enabled for a namespace, then configuring the **defaultRequest** section allows pods to start, even if the containers do not specify resource requests.
- ➐ The maximum image size that can be pushed to the internal registry.
- ➑ The maximum number of image tags and versions that an image stream resource can reference.
- ➒ The minimum and maximum sizes allowed for a persistent volume claim.

Users can create a limit range resource in 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 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 ...
Pod        cpu        10m   500m  -       ...
Pod        memory     5Mi   750Mi -       ...
Container  memory     5Mi   750Mi 20Mi  ...
Container  cpu        10m   500m  20m  ...
openshift.io/Image  storage   -     1Gi   -       ...
openshift.io/ImageStream openshift.io/image-tags -     10    -       ...
openshift.io/ImageStream openshift.io/images    -     20    -       ...
PersistentVolumeClaim  storage   1Gi   50Gi  -       ...
```

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, resulting in error messages. 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, similar 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**

To avoid large locking overheads, it is not recommended to have a single cluster resource quota that matches over a hundred projects. When updating or creating project resources, the project is locked while searching for all applicable resource quotas.

Customizing the Default Project Template

Cluster administrators can customize the default project template. Additional resources, such as quotas, limit ranges, and network policies, are created when a user creates a new project.

As a cluster administrator, create a new project template using the **oc adm create-bootstrap-project-template** command, and redirect the output to a file.

```
[user@demo ~]$ oc adm create-bootstrap-project-template \
> -o yaml > /tmp/project-template.yaml
```

Customize the template resource file to add additional resources, such as quotas, limit ranges, and network policies. Recall that quotas are configured by cluster administrators and cannot be added, modified, or deleted by project administrators. Project administrators can modify and delete limit ranges and network policies, even if those resources are created by the project template.

New projects create resources specified in the **objects** section. Additional objects should follow the same indentation as the **Project** and **RoleBinding** resources.

The following example creates a quota using the name of the project; the quota imposes a limit of 3 CPUs, 10 GB of memory, and 10 pods on the project.

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: project.openshift.io/v1
  kind: Project
  ...output omitted...
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  ...output omitted...
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: ${PROJECT_NAME}-quota
  spec:
    hard:
      cpu: "3"
      memory: 10Gi
      pods: "10"
parameters:
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
```

```
- name: PROJECT_DESCRIPTION  
- name: PROJECT_ADMIN_USER  
- name: PROJECT_REQUESTING_USER
```

Use the **oc create** command to create a new template resource in the **openshift-config** namespace.

```
[user@demo ~]$ oc create -f /tmp/project-template.yaml -n openshift-config  
template.template.openshift.io/project-request created
```

Update the **projects.config.openshift.io/cluster** resource to use the new project template. Modify the **spec** section. By default, the name of the project template is **project-request**.

```
apiVersion: config.openshift.io/v1  
kind: Project  
metadata:  
...output omitted...  
  name: cluster  
...output omitted...  
spec:  
  projectRequestTemplate:  
    name: project-request
```

A successful update to the the **projects.config.openshift.io/cluster** resource creates new **apiserver** pods in the **openshift-apiserver** namespace. After the new **apiserver** pods are running, new projects create the resources specified in the customized project template.

To revert to the original project template, modify the **projects.config.openshift.io/cluster** resource to clear the **spec** resource so that it matches: **spec: {}**



References

For more information, refer to the Quotas chapter in the Red Hat OpenShift Container Platform 4.5 Applications documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/applications/index#quotas

For more information about limit ranges, refer to the *Setting limit ranges* section in the Working with clusters chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/nodes/index#nodes-cluster-limit-ranges

Customizing OpenShift project creation

<https://developers.redhat.com/blog/2020/02/05/customizing-openshift-project-creation/>

► Guided Exercise

Limiting Resource Usage by an Application

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. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. 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.ocp4.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=client -o yaml > ~/D0280/labs/schedule-limit/hello-limit.yaml
```

- 2.2. Edit **~/D0280/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 ~/D0280/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 ~/D0280/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  0/3
nodes are available: 3 Insufficient cpu.
```

- 3. Redeploy your application so that it requests fewer CPU resources.

- 3.1. Edit **~/D0280/labs/schedule-limit/hello-limit.yaml** to request 1.2 CPUs for the container. Change the **cpu: "3"** line to match the highlighted line below.

```
[student@workstation ~]$ vim ~/D0280/labs/schedule-limit/hello-limit.yaml

...output omitted...
resources:
  requests:
    cpu: "1200m"
    memory: 20Mi
```

- 3.2. Apply the changes to your application.

```
[student@workstation ~]$ oc apply -f ~/D0280/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.

NAME	READY	STATUS	RESTARTS	AGE
hello-limit-d86874d86b-fpmrt	0/1	Terminating	0	2m19s
hello-limit-7c7998ff6b-ctsjp	1/1	Running	0	6s



Note

If your application pod does not get scheduled, modify the **~/D0280/labs/schedule-limit/hello-limit.yaml** file to reduce the CPU request to **1100m**. Apply the changes again and verify the pod status is **Running**.

4. Attempt to scale your application from one pod to four 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 four pods.

```
[student@workstation ~]$ oc scale --replicas 4 deployment/hello-limit
deployment.apps/hello-limit scaled
```

- 4.2. Check to see if all four pods are running. You might need to run **oc get pods** multiple times until you see that at least one pod is pending. Depending on the current cluster load, multiple pods might be in a pending state.



Note

If your application pod still does not deploy, scale the number of application pods to zero and then modify **~/D0280/labs/schedule-limit/hello-limit.yaml** to reduce the CPU request to **1000m**. Run **oc apply -f ~/D0280/labs/schedule-limit/hello-limit.yaml** to apply the changes then rerun the **oc scale** command to scale out to four pods.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
hello-limit-d55cd65c-765s9  1/1     Running   0          12s
hello-limit-d55cd65c-hmlbw  0/1     Pending    0          12s
hello-limit-d55cd65c-r8lvw  1/1     Running   0          12s
hello-limit-d55cd65c-vkrhk  0/1     Pending    0          12s
```

- 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
...output omitted...
76s           Warning   FailedScheduling   pod/hello-limit-d55cd65c-vkrhk      0/3
nodes are available: 3 Insufficient cpu.
```

- 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-d55cd65c-765s9" deleted
pod "hello-limit-d55cd65c-hmlbw" deleted
pod "hello-limit-d55cd65c-r8lvw" deleted
pod "hello-limit-d55cd65c-vkrhk" deleted
deployment.apps "hello-limit" deleted
replicaset.apps "hello-limit-5cc86ff6b8" deleted
replicaset.apps "hello-limit-7d6bdcc99b" 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/D0280/labs/schedule-limit/loadtest.yaml** 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 ~/D0280/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.apps.ocp4.example.com ...
```

- 6. Generate additional memory load that can be handled by the container.

- 6.1. Open two additional terminal windows to continuously monitor the load of your application. Access the application API from the first terminal to simulate additional memory pressure on the container.

From the second terminal window, run **watch oc get pods** to continuously monitor the status of each pod.

Finally, from the third terminal, run **watch oc adm top pod** to continuously monitor the CPU and memory usage of each pod.

- 6.2. 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.apps.ocp4.example.com/api/loadtest/v1/mem/150/60
curl: (52) Empty reply from server
```

- 6.3. 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				
	READY	STATUS	RESTARTS	AGE
loadtest-f7495948-tlxgm	1/1	Running	0	7m34s

- 6.4. In the third terminal window, observe the output of **watch oc adm top pod**. The starting memory usage for the pod is about 20-30Mi.

Every 2.0s: oc adm top pod		
	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 170-180Mi.

Every 2.0s: oc adm top pod		
	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 20-30Mi.

Every 2.0s: oc adm top pod		
	CPU(cores)	MEMORY(bytes)
loadtest-f7495948-tlxgm	0m	20Mi

- 7. Generate additional memory load that cannot be handled by the container.

71. 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.apps.ocp4.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>
```

72. 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
...
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
...
NAME          READY   STATUS    RESTARTS   AGE
loadtest-f7495948-tlxgm   1/1     Running   1          9m33s
```

73. In the third terminal window, observe the output of **watch oc adm top pod**. After the pod is killed, metrics will show that the pod is using little to no resources for a brief period of time.

```
Every 2.0s: oc adm top pod
...
NAME          CPU(cores)   MEMORY(bytes)
loadtest-f7495948-tlxgm   8m           0Mi
```

74. 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 redhat
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 developer
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. As a cluster administrator, configure all new projects with default quota and limit range resources.

- 10.1. Log in to your OpenShift cluster as the **admin** user.

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

- 10.2. Redirect the output of the **oc adm create-bootstrap-project-template** command so that you can customize project creation.

```
[student@workstation ~]$ oc adm create-bootstrap-project-template \
> -o yaml > /tmp/project-template.yaml
```

- 10.3. Edit the **/tmp/project-template.yaml** file to add the quota and limit range resources defined in the **/home/student/D0280/labs/schedule-limit/quota-limits.yaml** file. Add the lines before the **parameters** section.

```
...output omitted...
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: ${PROJECT_NAME}-quota
  spec:
    hard:
      cpu: 3
      memory: 10G
- apiVersion: v1
  kind: LimitRange
  metadata:
    name: ${PROJECT_NAME}-limits
  spec:
    limits:
      - type: Container
        defaultRequest:
          cpu: 30m
          memory: 30M
  parameters:
    - name: PROJECT_NAME
    - name: PROJECT_DISPLAYNAME
    - name: PROJECT_DESCRIPTION
    - name: PROJECT_ADMIN_USER
    - name: PROJECT_REQUESTING_USER
```



Note

The **/home/student/D0280/solutions/schedule-limit/project-template.yaml** file contains the correct configuration and can be used for comparison.

- 10.4. Use the **/tmp/project-template.yaml** file to create a new template resource in the **openshift-config** namespace.

```
[student@workstation ~]$ oc create -f /tmp/project-template.yaml \
> -n openshift-config
template.template.openshift.io/project-request created
```

- 10.5. Update the cluster to use the custom project template.

```
[student@workstation ~]$ oc edit projects.config.openshift.io/cluster
```

Modify the **spec** section to use the following lines in bold.

```
...output omitted...
spec:
  projectRequestTemplate:
    name: project-request
```

Ensure proper indentation, and then save your changes.

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

- 10.6. After a successful change, the **apiserver** pods in the **openshift-apiserver** namespace are recreated.

```
[student@workstation ~]$ watch oc get pods -n openshift-apiserver
```

Wait until all three new **apiserver** pods are ready and running.

```
Every 2.0s: oc get pods -n openshift-apiserver ...
NAME           READY   STATUS    RESTARTS   AGE
apiserver-868dccf5fb-885dz   1/1     Running   0          63s
apiserver-868dccf5fb-8j4vh   1/1     Running   0          39s
apiserver-868dccf5fb-r4j9b   1/1     Running   0          24s
```

Press **Ctrl+C** to end the **watch** command.

- 10.7. Create a test project to confirm that the custom project template works as expected.

```
[student@workstation ~]$ oc new-project template-test
Now using project "template-test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 10.8. List the resource quota and limit range resources in the test project.

```
[student@workstation ~]$ oc get resourcequotas,limitranges
NAME                           AGE   REQUEST           LIMIT
resourcequota/template-test-quota   87s   cpu: 0/3, memory: 0/10G

NAME                           CREATED AT
limitrange/template-test-limits   2020-09-16T21:13:25Z
```

- 11. Clean up the lab environment by deleting the projects created in this exercise.

- 11.1. Delete the **schedule-limit** project.

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

- 11.2. Delete the **template-test** project.

```
[student@workstation ~]$ oc delete project template-test  
project.project.openshift.io "template-test" 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 ①
  selector:
    matchLabels:
      deployment: scale ②
  strategy: {}
  template: ③
    metadata:
      labels:
        deployment: scale ④
  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 web console or the **oc new-app** command with the **--as-deployment-config** option, defines the same information, but in a slightly different way. Deployment configurations use a **deploymentconfig** selector with a matching **deploymentconfig** label in the pod metadata.

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
...output omitted...
spec:
  replicas: 1
  selector:
    deploymentconfig: hello
  strategy:
    resources: {}
  template:
    metadata:
      annotations:
        openshift.io/generated-by: OpenShiftNewApp
      labels:
        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, rather than requiring a separate install, as in OpenShift 3. 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 4.5.

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

The horizontal pod autoscaler initially has a value of <unknown> in the TARGETS column. It might take up to five minutes before <unknown> changes to display a percentage for current usage.

A persistent value of <unknown> in the TARGETS column might indicate 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 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 with the Horizontal Pod Autoscaler* section in the *Working with pods* chapter in the Red Hat OpenShift Container Platform 4.5 Nodes documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/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. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p developer \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. 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.ocp4.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 **~/D0280/labs/schedule-scale/loadtest.yaml** to set both requests and limits for CPU and memory usage.

```
[student@workstation ~]$ vim ~/D0280/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
```

- 2.5. Verify that your application pod specifies resource limits and requests.

```
[student@workstation ~]$ oc describe pod/loadtest-5f9565dbfb-jm9md \
>   | grep -A2 -E "Limits|Requests"
Limits:
  cpu: 100m
  memory: 100Mi
Requests:
  cpu: 25m
  memory: 25Mi
```

- 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.apps/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.apps/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. Wait until the **loadtest** horizontal pod autoscaler reports usage in the **TARGETS** column.

```
[student@workstation ~]$ watch oc get hpa/loadtest
```

Press **Ctrl+C** to exit the **watch** command after <unknown> changes to a percentage.

```
Every 2.0s: oc get hpa/loadtest
...
NAME      REFERENCE      TARGETS      MINPODS   MAXPODS   REPLICAS   ...
loadtest  Deployment/loadtest  0%/50%       2          10         2          ...
```



Note

It can take up to five minutes before the <unknown> entry in the **TARGETS** column changes to **0%**. If the <unknown> entry does not change, then use the **oc describe** command to verify that the containers for the **loadtest** application request CPU resources.

Chapter 13 | Controlling Pod Scheduling

- 4.3. 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.ocp4.example.com  ...
```

- 4.4. 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.ocp4.example.com/api/loadtest/v1/cpu/1
curl: (52) Empty reply from server
```

- 4.5. 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 ...
NAME      REFERENCE          TARGETS      MINPODS   MAXPODS   REPLICAS   ...
loadtest  Deployment/loadtest  172%/50%    2          10        9          ...
```

**Note**

Although the horizontal pod **autoscaler** 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**. Use the **--as-deployment-config** option to create a **deploymentconfig** (dc) resource instead of the default **deployment** resource.

```
[student@workstation ~]$ oc new-app --name scaling --as-deployment-config \
>   --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. 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 deploymentconfig=scaling
NAME          READY   STATUS    RESTARTS   AGE     IP           NODE   ...
scaling-1-bm4m2 1/1     Running   0          45s    10.10.0.29  master01 ...
scaling-1-w7whl 1/1     Running   0          45s    10.8.0.45   master03 ...
scaling-1-xqvs2 1/1     Running   0          6m1s   10.9.0.58   master02 ...
```

- 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.ocp4.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. Run the script located at **~/D0280/labs/schedule-scale/curl-route.sh**.

```
[student@workstation ~]$ ~/D0280/labs/schedule-scale/curl-route.sh
34 Server IP: 10.10.0.29
34 Server IP: 10.8.0.45
32 Server IP: 10.9.0.58
```

- 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 and 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 ...  
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 nodes, and to scale with load.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Add a new label to 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. As the **admin** user, label two nodes with the **tier** label. Give the **master01** node the label of **tier=gold** and the **master02** 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.ocp4.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.ocp4.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.

- 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 nodes, and to scale with load.

Outcomes

You should be able to use the OpenShift command-line interface to:

- Add a new label to 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. As the **admin** user, label two nodes with the **tier** label. Give the **master01** node the label of **tier=gold** and the **master02** node the label of **tier=silver**.

- 1.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
> https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Identify if any nodes already use the **tier** label.

```
[student@workstation ~]$ oc get nodes -L tier
NAME      STATUS    ROLES      AGE      VERSION      TIER
master01   Ready     master,worker  5d20h   v1.18.3+012b3ec
master02   Ready     master,worker  5d20h   v1.18.3+012b3ec
master03   Ready     master,worker  5d20h   v1.18.3+012b3ec
```

- 1.3. Label the **master01** node with the label **tier=gold**.

```
[student@workstation ~]$ oc label node master01 tier=gold
node/master01 labeled
```

- 1.4. Label the **master02** node with the label **tier=silver**.

```
[student@workstation ~]$ oc label node master02 tier=silver
node/master02 labeled
```

- 1.5. Confirm that the labels have been added correctly.

```
[student@workstation ~]$ oc get nodes -L tier
NAME      STATUS    ROLES     AGE      VERSION   TIER
master01  Ready     master,worker  5d20h   v1.18.3+012b3ec  gold
master02  Ready     master,worker  5d20h   v1.18.3+012b3ec  silver
master03  Ready     master,worker  5d20h   v1.18.3+012b3ec
```

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 developer
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.ocp4.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=client \
>   --image quay.io/redhattraining/loadtest:v1.0 \
>   -o yaml > ~/D0280/labs/schedule-review/loadtest.yaml
```

- 3.2. Edit **~/D0280/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 ~/D0280/labs/schedule-review/loadtest.yaml
...output omitted...
spec:
  nodeSelector:
```

```
tier: silver
containers:
- image: quay.io/redhattraining/loadtest:v1.0
  name: loadtest
  resources: {}
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:
  nodeSelector:
    tier: silver
  containers:
    - image: quay.io/redhattraining/loadtest:v1.0
      name: loadtest
      resources:
        requests:
          cpu: "100m"
          memory: 20Mi
  status: {}
```

- 3.4. Create the **loadtest** application.

```
[student@workstation ~]$ oc create --save-config \
>   -f ~/DO280/labs/schedule-review/loadtest.yaml
deployment.apps/loadtest created
```

- 3.5. Verify that your application pod is running. You might need to run the `oc get pods` command multiple times.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
loadtest-85f7669897-z4mq7   1/1     Running   0          53s
```

- 3.6. Verify that your application pod specifies resource requests.

```
[student@workstation ~]$ oc describe pod/loadtest-85f7669897-z4mq7 \
>   | grep -A2 Requests
Requests:
  cpu:        100m
  memory:    20Mi
```

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.ocp4.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.ocp4.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.ocp4.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.ocp4.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. Wait until the **loadtest** horizontal pod autoscaler reports default usage in the **TARGETS** column.

```
[student@workstation ~]$ watch oc get hpa/loadtest
```

Press **Ctrl+C** to exit the **watch** command after **<unknown>** changes to **0%**.

Every 2.0s: <code>oc get hpa/loadtest</code> ...						
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	...
loadtest	Deployment/loadtest	0%/70%	2	40	2	...

**Note**

It can take up to five minutes before the <unknown> entry in the **TARGETS** column changes to **0%**. If the <unknown> entry does not change, then use the **oc describe** command to verify that the containers for the **loadtest** application request CPU resources.

- 5.3. 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.\n> apps.ocp4.example.com/api/loadtest/v1/cpu/3
```

- 5.4. 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.

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	...
loadtest	Deployment/loadtest	1043%/70%	2	40	21	...

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 redhat\nLogin successful.\n...output omitted...
```

- 6.2. Create the resource quota.

```
[student@workstation ~]$ oc create quota review-quota \\n> --hard cpu="1",memory="2G",pods="20"\nresourcequota/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.

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.

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.
- A customized project template can automatically create quotas and limit ranges for new projects.
- The **oc scale** command manually scales the number of replicas of a pod.
- Horizontal pod autoscalers dynamically scale pod replicas based on load.

Describing 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.

Red Hat hosts and manages this service at <https://cloud.redhat.com>, and hosts cluster images at <https://quay.io>.



Important

As of OpenShift 4.5, the OTA system requires a persistent connection to the Internet. It is not possible to deploy this feature on-premise.

For more information on how to update disconnected clusters, consult the *Update* guide and the *Installation configuration* chapter listed in the references section.

OTA enables faster updates by allowing the skipping of intermediary versions. For example, you can update from 4.5.1 to 4.5.3, thus bypassing 4.5.2.

You use a single interface (<https://cloud.redhat.com>) to manage the life cycle of all your OpenShift clusters.

The screenshot shows the Red Hat OpenShift Clusters management interface. On the left is a sidebar with navigation links: Clusters, Subscriptions, Overview, Documentation, Support Cases, Cluster Manager Feedback, and Red Hat Marketplace. The main area is titled 'Clusters' and contains a table with three rows. The columns are: Name, Status, Type, Created, Version, and Provider (Lo...). The first cluster is named 'rhoc-001-0001-0001-0001-0001-0001', status Ready, type OCP, created 60-day trial, version 4.2.2, provider AWS (US East, Ohio). The second cluster is named 'rhoc-001-0001-0001-0001-0001-0001', status Ready, type OCP, created 60-day trial, version 4.5.3, provider OPENSTACK (N/A). The third cluster is named 'rhoc-001-0001-0001-0001-0001-0001', status Ready, type OCP, created 60-day trial, version 4.5.3, provider OPENSTACK (N/A). There are filter and create cluster buttons at the top of the table.

Figure 14.1: 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: **stable-4.5**, **fast-4.5**, and **candidate-4.5**. Each channel delivers patches for a given cluster version.

Describing the Candidate Channel

The *candidate* channel delivers updates for testing feature acceptance in the next version of OpenShift Container Platform. The release candidate versions are subject to further checks and are promoted to the fast or stable channels when they meet the quality standards.



Note

The updates listed in the *candidate* channel are not supported by Red Hat.

Describing the Fast Channel

The *fast* channel delivers updates as soon as they are available. This channel is best suited for production and QA environments. You can use the **fast-4.5** channel to upgrade from a previous minor version of OpenShift Container Platform.



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, which means that it delivers only minor updates for a given cluster version and is better suited for production environments.

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.

Describing Upgrade Paths

The following describes how these upgrade paths would apply to Red Hat OpenShift Container Platform version 4.5.

- When using the **stable-4.5** channel, you can upgrade your cluster from 4.5.0 to 4.5.1 or 4.5.2. If an issue is discovered in the 4.5.3 release, then you cannot upgrade to that version. When a patch becomes available in the 4.5.4 release, you can update your cluster to that version.

This channel is suited for production environments, as the releases in that channel are tested by Red Hat SREs and support services.

- The **fast-4.5** channel can deliver 4.5.1 and 4.5.2 updates but not 4.6.1. This channel is also supported by Red Hat and can be applied to production environments.

Administrators must specifically choose a different minor version channel, such as **fast-4.6**, in order to upgrade to a new release in a new minor version.

- The **candidate-4.5** channel allows you to install the latest features of OpenShift. With this channel, you can upgrade to all z-stream releases, such as 4.5.1, 4.5.2, 4.5.3, and so on.

You use this channel to have access to the latest features of the product as they get released. This channel is suited for development and pre-production environments.

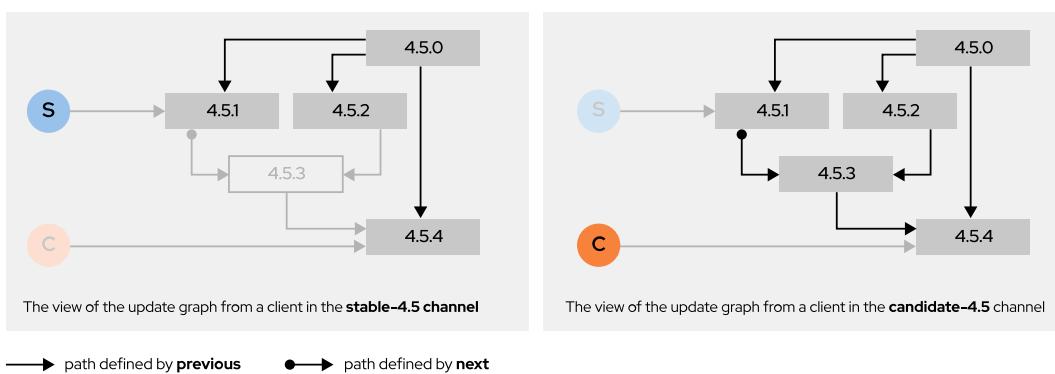


Figure 14.2: Update graphs for stable and candidate channels

The *stable* and *fast* channels are classified as General Availability (GA), whereas the *candidate* channel (release candidate channel) is not supported by Red Hat.

To ensure the stability of the cluster and the proper level of support, you should only switch from a stable channel to a fast channel, and vice versa. Although it is possible to switch from a stable channel or fast channel to a candidate channel, it is not recommended. The candidate channel is best suited for testing feature acceptance and assisting in qualifying the next version of OpenShift Container Platform.

**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.

Changing the Update Channel

You can change the update channel to **stable-4.5**, **fast-4.5**, or **candidate-4.5** using the web console or the OpenShift CLI client:

- In the web console, navigate to the **Administration** → **Cluster Settings** page on the details tab, and then click the **pencil** icon.

Figure 14.3: Current update channel in the web console

A window displays options to select an update channel.

Figure 14.4: Changing the update channel in the web console

- Execute the following command to switch to another update channel using the **oc** client. You can also switch to another update channel, such as **stable-4.6**, to update to the next minor version of OpenShift Container Platform.

```
[user@demo ~]$ oc patch clusterversion version --type="merge" --patch \
>   '{"spec":{"channel":"fast-4.5"}}'
clusterversion.config.openshift.io/version patched
```

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. The web console sends a notification 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.

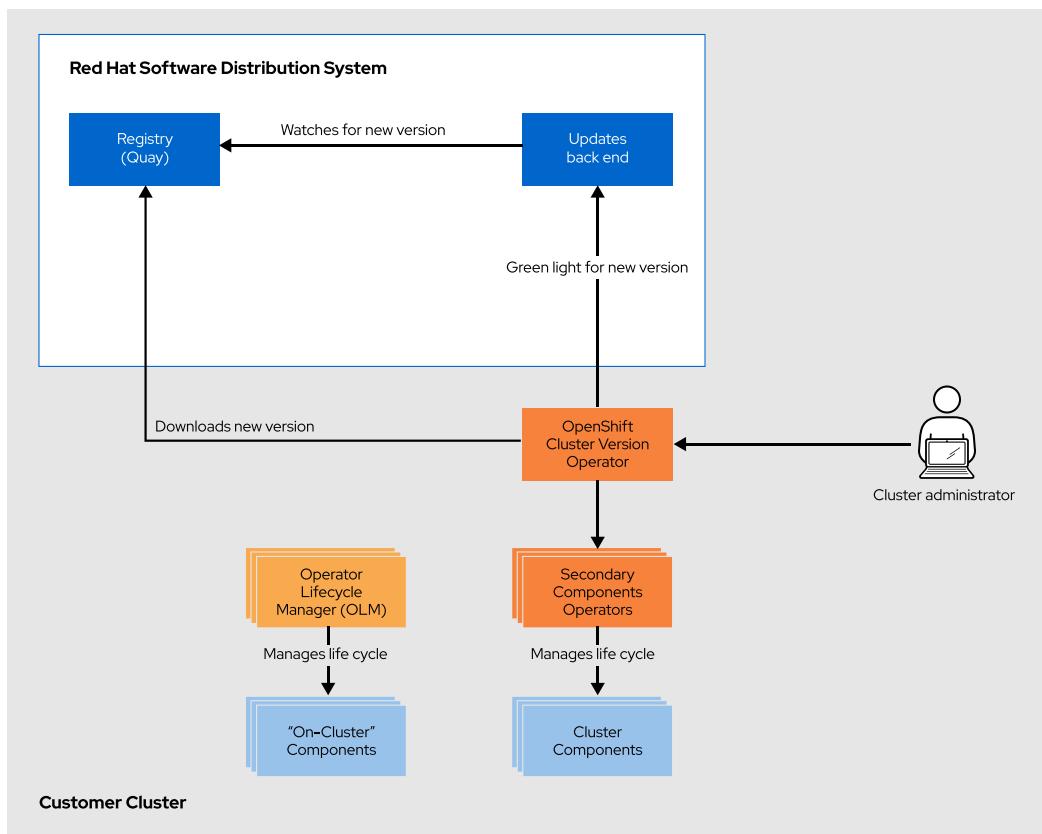


Figure 14.5: OpenShift Container Platform Updates Architecture

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 *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.

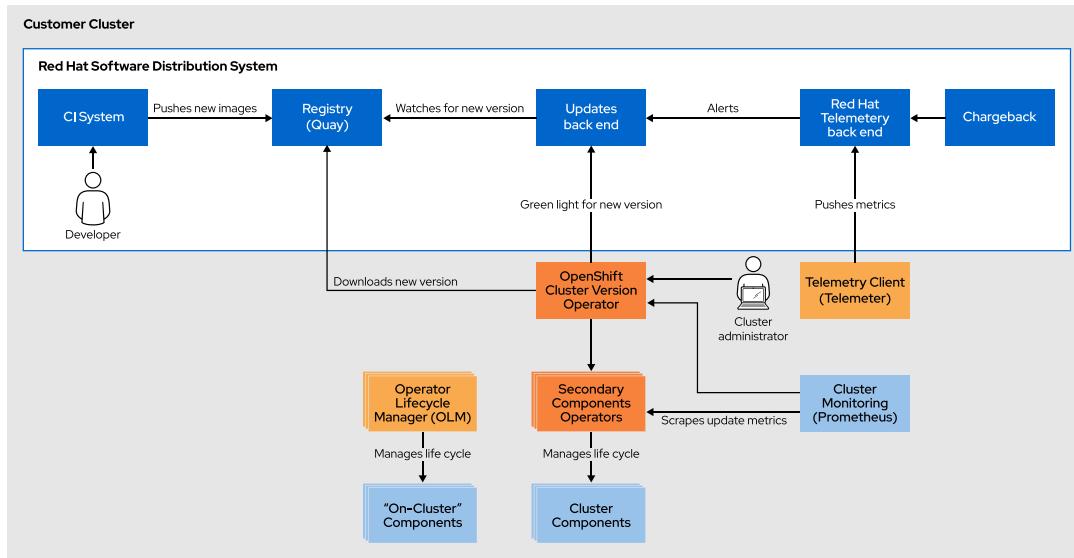


Figure 14.6: Managing cluster updates using telemetry

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 **Administration** → **Cluster Settings** page displays an **Update Status** of **Update available** when a new update is available. From this page, click **Update now** to begin the process.

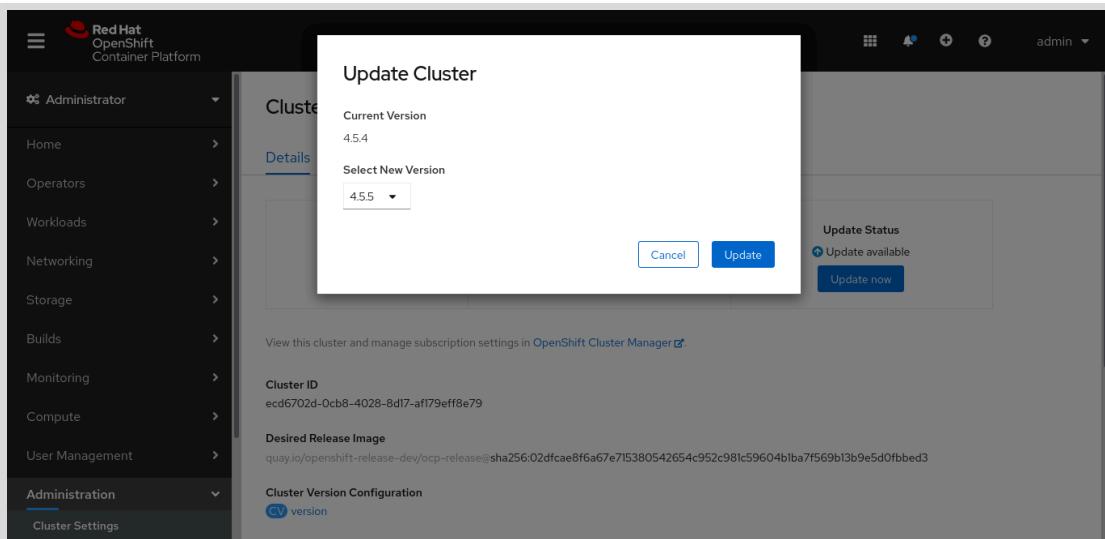


Figure 14.7: Updating the cluster using the web console

**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.

**Important**

Be sure to update all operators installed through the Operator Lifecycle Manager (OLM) to the latest version before updating the OpenShift cluster.

1. Retrieve the cluster version and 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
NAME      VERSION      AVAILABLE      PROGRESSING      SINCE      STATUS
version   4.5.4        True          False           5d        Cluster version is 4.5.4

[user@demo ~]$ oc get clusterversion -o json | jq ".items[0].spec.channel"
"stable-4.5"
```

2. 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 4.5.4

Updates:

VERSION IMAGE
4.5.5 quay.io/openshift-release-dev/ocp-release@sha256:...
...output omitted...
```

3. Apply the latest update to your cluster or update to a specific version:

- Run the following command to install the latest available update for your cluster.

```
[user@demo ~]$ oc adm upgrade --to-latest=true
```

- 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
```

4. The previous command initializes the update process. Run the following command to review the status of the Cluster Version Operator (CVO) and the installed cluster operators.

```
[user@demo ~]$ oc get clusterversion
NAME      VERSION      AVAILABLE      PROGRESSING      SINCE      STATUS
version   4.5.4        True          True            32m        Working towards 4.5.5 ...

[user@demo ~]$ oc get clusteroperators
NAME              VERSION      AVAILABLE      PROGRESSING      DEGRADED
authentication    4.5.4        True          False          False
cloud-credential  4.5.5        False         True           False
openshift-apiserver  4.5.5        True          False          True
...output omitted...
```

5. 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.

If the rollout completed successfully, updates in the history have a state of **Completed**. Otherwise, the update has a state of **Partial** if the update failed or did not complete.

```
[user@demo ~]$ oc describe clusterversion
...output omitted...
History:
Completion Time: 2020-09-28T16:02:18Z
Image: quay.io/openshift-release-dev/ocp-release@sha256:...
Started Time: 2020-09-28T15:31:13Z
State: Completed
Verified: true
```

```
Version:          4.5.5
Completion Time: 2020-08-05T18:35:08Z
Image:           quay.io/openshift-release-dev/ocp-release@sha256:...
Started Time:    2020-08-05T18:22:42Z
State:           Completed
Verified:        true
Version:         4.5.4
Observed Generation: 5
Version Hash:    AF5-oeav9wI=
Events:          none
```



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.

- 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   STATUS
version   4.5.5    True       False        11m     Cluster version is 4.5.5
```



References

For more information about installing Red Hat OpenShift Container Platform in a disconnected environment, refer to the *Installation configuration* chapter in the Red Hat OpenShift Container Platform 4.5 *Installing* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/installing/index#installation-configuration

For more information about update channels, update prerequisites, and updating clusters in disconnected environments, refer to the *Updating a restricted network cluster*, and *Updating a cluster between minor versions* chapters in the the Red Hat OpenShift Container Platform 4.5 *Updating clusters* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/updating_clusters/index#updating-restricted-network-cluster

For more information about the update of operators installed through the Operator Lifecycle Manager (OLM), refer to the *Upgrading installed Operators* section in the *Administrator tasks* chapter in the Red Hat OpenShift Container Platform 4.5 *Working with Operators* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/operators/index#olm-upgrading-operators

For more information about the OpenShift Container Platform upgrade paths, visit the following page in the customer portal:

<https://access.redhat.com/solutions/4583231/>

Sample Metrics

<https://github.com/openshift/cluster-monitoring-operator/blob/master/Documentation/sample-metrics.md>

Cincinnati

<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 - 4.5 update channel? (Choose two.)
 - a. 4.4.2
 - b. 4.5.1
 - c. 4.6.1
 - d. 4.5.5

- ▶ 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 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 4.3.27 to 4.3.29, OpenShift must install the 4.3.28 version first.
 - c. It is not recommended to switch from a stable channel or a fast channel to a candidate channel. However, you can switch from a fast channel 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 4.5.2 to 4.5.3, but not from 4.5.3 to 4.6).

► **6. Which two of the following channels are classified as general availability? (Choose two.)**

- a. candidate-4.5.stable
- b. stable-4.5
- c. candidate-stable-4.5
- d. fast-4.5
- e. fast-4.5.1

► 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 - 4.5 update channel? (Choose two.)
 - a. 4.4.2
 - b. 4.5.1
 - c. 4.6.1
 - d. 4.5.5

- ▶ 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 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 4.3.27 to 4.3.29, OpenShift must install the 4.3.28 version first.
 - c. It is not recommended to switch from a stable channel or a fast channel to a candidate channel. However, you can switch from a fast channel 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 4.5.2 to 4.5.3, but not from 4.5.3 to 4.6).

► **6. Which two of the following channels are classified as general availability? (Choose two.)**

- a. candidate-4.5.stable
- b. stable-4.5
- c. candidate-stable-4.5
- d. fast-4.5
- e. fast-4.5.1

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 three distribution channels:
 - The *stable* channel delivers delayed updates.
 - The *fast* channel delivers updates as soon as they are available.
 - The *candidate* channel delivers updates for testing feature acceptance in the next version of OpenShift Container Platform.
- Red Hat does not support reverting your cluster to a previous version. Red Hat only supports upgrading to a newer version.

Chapter 15

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 applications and Kubernetes Operators with the web console.
- Examine performance and health metrics for cluster nodes and applications.

Sections

- Performing Cluster Administration (and Guided Exercise)
- Managing Workloads and Operators (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. The ability to view individual items depends upon the roles and role bindings associated with a user. Users with the **cluster-admin** cluster role can view and modify everything. Users with the **view** cluster role can view most items, but cannot make changes. Additional roles can provide access to specific items.

Home

The **Home** → **Overview** page 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.

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 manage 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.

User Management

View and manage users, groups, service accounts, roles, and role bindings.

Administration

View and manage a wide variety of settings that are of particular interest to cluster administrators, such as cluster updates, cluster operators, CRDs, 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.

Name	Display Name	Status	Requester	Memory	CPU
PR openshift-apiserver	No display name	Active	No requester	361.0 MiB	0.033 cores
PR openshift-apiserver-operator	No display name	Active	No requester	68.5 MiB	0.015 cores

It may be useful to filter pods by state to identify potential issues or problematic deployments.

Status	Namespace	Status	Ready	Owner
Running	openshift-apiserver	Running	1/1	RS apiserver-84db66db8
Running	openshift-apiserver	Running	1/1	RS apiserver-84db66db8
Running	openshift-apiserver	Running	1/1	RS apiserver-84db66db8

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 **Users** page accessible from **User Management** → **Users** displays users who have previously logged in to OpenShift. As discussed in *Chapter 10, Configuring Authentication and Authorization*, OpenShift supports a variety of identity providers (IdPs), including HTPasswd, LDAP, and OpenID Connect.

When using the HTPasswd identity provider, the **secrets** editor can simplify adding, updating, or removing entries in the HTPasswd secret. After using a terminal to generate a new or updated HTPasswd entry, switch to the web console to modify 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 encoding for you. Add a line to allow a new user to log in to OpenShift. Update a line to change the password for a user. Delete a line so that a user cannot log in to OpenShift.

The **Groups** page accessible from **User Management** → **Groups** displays existing groups, and provides the ability to create new groups.

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 4.5 *Web console* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/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. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Identify the URL for the web console.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 1.3. Open a web browser and navigate to <https://console-openshift-console.apps.ocp4.example.com>.

- 1.4. 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.

- 1.5. Click **localusers** and log in as the **admin** user with the **redhat** password.
- 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 **Search by name** field, and then click the **openshift-console-operator** link.

Name	Display Name	Status	Requester
PR openshift-console	No display name	Active	No requester
PR openshift-console-operator	No display name	Active	No requester

- 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.

Name	Namespace	Status	Ready	Node	Memory	CPU
P console-operator-5877c747c4-hmm52	NS openshift-console-operator	Running	1/1	N master02	43.4 MiB	0.004 cores

- 2.4. Review the **Pod Details** page and notice the pod metrics, running status, and volumes.
- 2.5. Click **YAML** to navigate to the pod resource editor.
- 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.

The screenshot shows the OpenShift Web Console interface. At the top, it says "Project: openshift-console-operator". Below that is a search bar with "openshift-console" typed in. A sidebar on the left lists "Create Project", "+ openshift-console", and "+ openshift-console-operator". The main area shows a table with one row: "17c4-hmm52" status "Running". Below the table are tabs for "Events" and "Terminal". Under "Terminal", there's a "Log streaming..." button, a dropdown for "console-operator", and download/expand buttons. The log output shows 514 lines of text, including log entries from "I0804 14:07:40.829844" to "I0804 14:07:40.830464".

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 Details** 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 **User Management** → **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.

The screenshot shows the "Roles" page. At the top right is a "Create Role" button. Below it are filters for "Name" and "Search by name...". The main table has columns for "Name" (sorted by ascending), "Namespace", and three dots for more options. There are three rows: 1) "CR admin" (highlighted with a red border), "All Namespaces", "..."; 2) "CR aggregate-olm-edit", "All Namespaces", "..."; 3) "CR aggregate-olm-view", "All Namespaces", "...".

- 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 **Search 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 **redhat** as the password.

```
[student@workstation ~]$ htpasswd -n -b tester redhat
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$RjqTY4cv$xqlz.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. Use the following information for the new project, and then click **Create**.

Create Project Form

Field	Value
Name	console-apps
Display Name	Console chapter applications
Description	Example project

- 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 a name for the limit range. 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
```

Click **Create**.

- 6.5. Click **User Management** → **Groups**, and then 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 **User Management** → **Role Bindings**, and then click **Create Binding**. 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.

- 6.7. 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.

- 6.8. Click **admin** → **Log out**, and then log back in as the **developer** user with the **developer** password.

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.

- 6.9. 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 and Operators

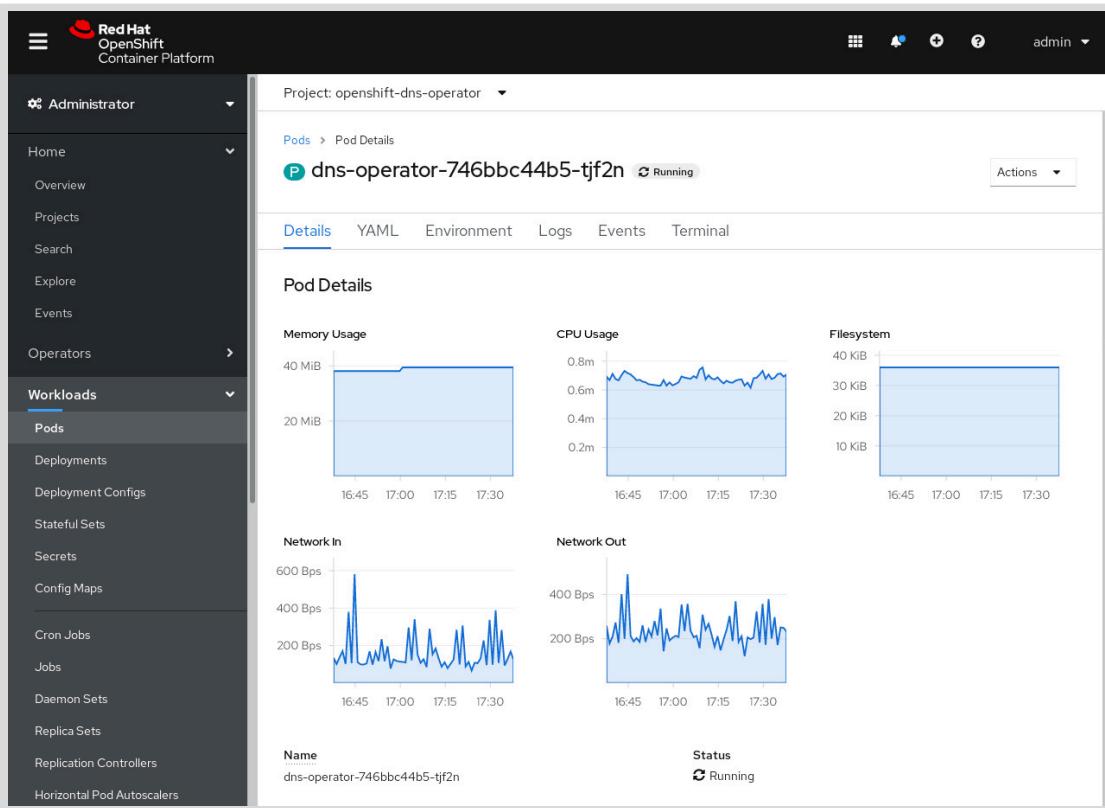
Objectives

After completing this section, you should be able to manage applications and Kubernetes Operators 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 **openshift-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.

The screenshot shows the Deployment Details page for the deployment named 'openshift-apiserver-operator'. The 'Actions' dropdown menu is open, displaying various management options. The deployment currently has 1 pod.

Name	openshift-apiserver-operator	Update Strategy	Recreate
Namespace	NS openshift-apiserver-operator	Progress Deadline Seconds	600 seconds

Figure 15.8: 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 Pod Count**.
- To modify the update strategy for a deployment, such as changing rolling update parameters, navigate to the **Deployment Details** page and click **Actions → Edit Update Strategy**. To modify the update strategy for a deployment configuration, navigate to the **Deployment Config Details** page and click **Actions → Edit Deployment Config**.
- Navigate to the **Secret Details** page and click **Actions → Edit Secret** 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.

```

1 kind: Deployment
2 apiVersion: apps/v1
3 metadata:
4   annotations:
5     deployment.kubernetes.io/revision: '1'
6     exclude.release.openshift.io/internal-openshift-hosted: 'true'
7   selfLink: >-
8     /apis/apps/v1/namespaces/openshift-apiserver-operator/deployments/openshift-apiserver-operator
9   resourceVersion: '35597'
10  name: openshift-apiserver-operator
11  uid: 93b3f536-65c3-4b47-baa1-67511a26aad4
12  creationTimestamp: '2020-07-29T18:30:38Z'
13  generation: 1
14  managedFields:
15    - manager: cluster-version-operator
16      operation: Update
17      apiVersion: apps/v1
18      time: '2020-07-29T18:30:38Z'
19      fieldsType: FieldsV1
20      fieldsV1:
21        'f:metadata':
22          'f:annotations':
23            .: {}
24          'f:exclude.release.openshift.io/internal-openshift-hosted': {}
25        'f:labels':
26          .: {}
27        'f:app': {}

```

Save Reload Cancel Download

Figure 15.9: 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 Pod 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** → **Deployment Configs** 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 360 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.

When you are ready to install an operator, click **Install** to start the installation. Complete the **Operator Installation** form to select the target namespace and operator approval strategy. You can install operators to target all namespaces 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 screenshot shows the Red Hat OpenShift Container Platform web console. The left sidebar is collapsed. The main navigation bar shows 'Project: example-project'. The 'Operators' menu is open, with 'Installed Operators' selected. The main content area shows the 'etcd' operator details. The 'etcd' operator is version 0.9.4, provided by CNCF. It lists two provided APIs: 'etcd Cluster' and 'etcd Backup'. Each API has a 'Create Instance' button. The 'Actions' dropdown is visible on the right.

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.



References

For more information, refer to the Red Hat OpenShift Container Platform 4.5 *Web console* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/web_console/index

For more information, refer to the Red Hat OpenShift Container Platform 4.5 *Operators* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/operators/index

► Guided Exercise

Managing Workloads and Operators

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. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Identify the URL for the web console.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 1.3. Open a web browser and navigate to **https://console-openshift-console.apps.ocp4.example.com**.

- 1.4. 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.

- 1.5. Click **localusers** and log in as the **admin** user with the password of **redhat**.
- 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 **Search 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.

Name	Namespace	Status	Labels	Pod Selector
console	openshift-console	2 of 2 pods	app=console component=ui	app=console, component=ui
console-operator	openshift-console-operator	1 of 1 pods	No labels	name=console-operator

- 2.2. Click **Workloads → Replica Sets**, and type **console** in the **Search 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 Dev4Devs.com**. Click **Continue** to view the community operator page, and then click **Install**.

OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through [Red Hat Marketplace](#). You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a self-service experience.

The screenshot shows the OperatorHub interface. On the left, there's a sidebar with categories like All Items, Database, AI/Machine Learning, Application Runtime, Big Data, Cloud Provider, Database (which is selected), Developer Tools, Integration & Delivery, Logging & Tracing, Monitoring, Networking, OpenShift Optional, Security, Storage, Streaming & Messaging, Install State (with options for Installed (0) and Not Installed (4)), and Provider Type (with Red Hat (0)). The main area has a search bar with 'postgres'. It shows four items under 'Database': two from 'Marketplace' (Crunchy PostgreSQL for OpenShift) and two from 'Community' (PostgreSQL Operator by Dev4Ddevs.com). The PostgreSQL Operator item is highlighted with a red border.

- 3.3. Select the **console-apps** namespace, and then click **Install** 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 user **developer** with a password of **developer**.
- 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 Dev4Ddevs.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.

Name	Managed Namespaces	Status	Provided APIs
 PostgreSQL Operator by Dev4Devs.com	NS console-apps	✓ Succeeded	Database Backup Database Database

- 5.3. Click the **Database Database** tab and then click **Create Database**.
- 5.4. Switch from **Form View** to **YAML View**, and then 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:
  ...
  databaseUserKeyEnvVar: POSTGRES_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 `~/D0280/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.

Events

Resources All ▾ All Types ▾ Filter Events by name or message...

Resource A All ×

Streaming events... Showing 39 events

books-5f7fbffdb7-ngbk5	Generated from kubelet on master01	NS console-apps	a few seconds ago 4 times in the last 2 minutes
Pulling image "quay.io/redhattraining/books:v0.9"			
books-5f7fbffdb7-ngbk5	Generated from kubelet on master01	NS console-apps	a few seconds ago 4 times in the last 2 minutes
Failed to pull image "quay.io/redhattraining/books:v0.9": rpc error: code = Unknown desc = Error reading manifest v0.9 in quay.io/redhattraining/books: manifest unknown: manifest unknown			
books-5f7fbffdb7-ngbk5	Generated from kubelet on master01	NS console-apps	a few seconds ago 4 times in the last 2 minutes
Error: ErrImagePull			
books-5f7fbffdb7-ngbk5	Generated from kubelet on master01	NS console-apps	a few seconds ago 6 times in the last 2 minutes
Back-off pulling image "quay.io/redhattraining/books:v0.9"			
books-5f7fbffdb7-ngbk5	Generated from kubelet on master01	NS console-apps	a few seconds ago 6 times in the last 2 minutes
Error: ImagePullBackOff			

- 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	MinimumReplicasUnavailable	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 **Details** tab at the top of the **Deployment Details** page, and monitor the pod deployment. Unfortunately, the pod still fails to start.
- 8.6. Click **Home** → **Events**, and search for evidence of additional problems. A new event message indicates a quota problem.

```
Error creating: pods "books-5c65dc95-z9bss" is forbidden: exceeded quota: quota, requested: limits.memory=5Gi, used: limit.memory=10752Mi, limited: limits.memory=12Gi
```

Updating the **books** deployment created a new replica set, but scheduling a pod from the new replica set would exceed the project quota for memory limits.

- 8.7. To solve this problem, identify the **books** replica set with an existing pod and delete it. Deleting the replica set with the failing pod reduces quota usage and allows scheduling the pod from the new replica set. Click **Workloads** → **Replica Sets**. It is expected that there are two replica sets for the **books** deployment. The **books** replica set with a status of **1 of 1 pods** specifies the wrong container image version. Delete that replica set using the vertical ellipsis menu for the row and selecting **Delete Replica Set**. Confirm the deletion by clicking **Delete**.

Name	Namespace	Status	Labels	Owner	Created
books-5c65dc95	NS console-apps	0 of 1 pods	app=books pod-template...=5c65d...	books	2 minutes ago
books-5f7fbffdb7	NS console-apps	1 of 1 pods	app=books pod-template...=5f7fb...	books	8 minutes ago
database-599f5f4f8b	NS console-apps	1 of 1 pods	cr=database own...=postgresqloperat... pod-template...=599f5f...	database	8 minutes
postgresql-operator-67df97f444	NS console-apps	1 of 1 pods	na...=postgresql-operat... pod-template...=67df97...	postgresql-operator	9 minutes

- 8.8. Click **Workloads** → **Deployments**, and then click the link for the **books** deployment. Wait until the donut indicates that one pod is running.
- 8.9. 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.
- 8.10. 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

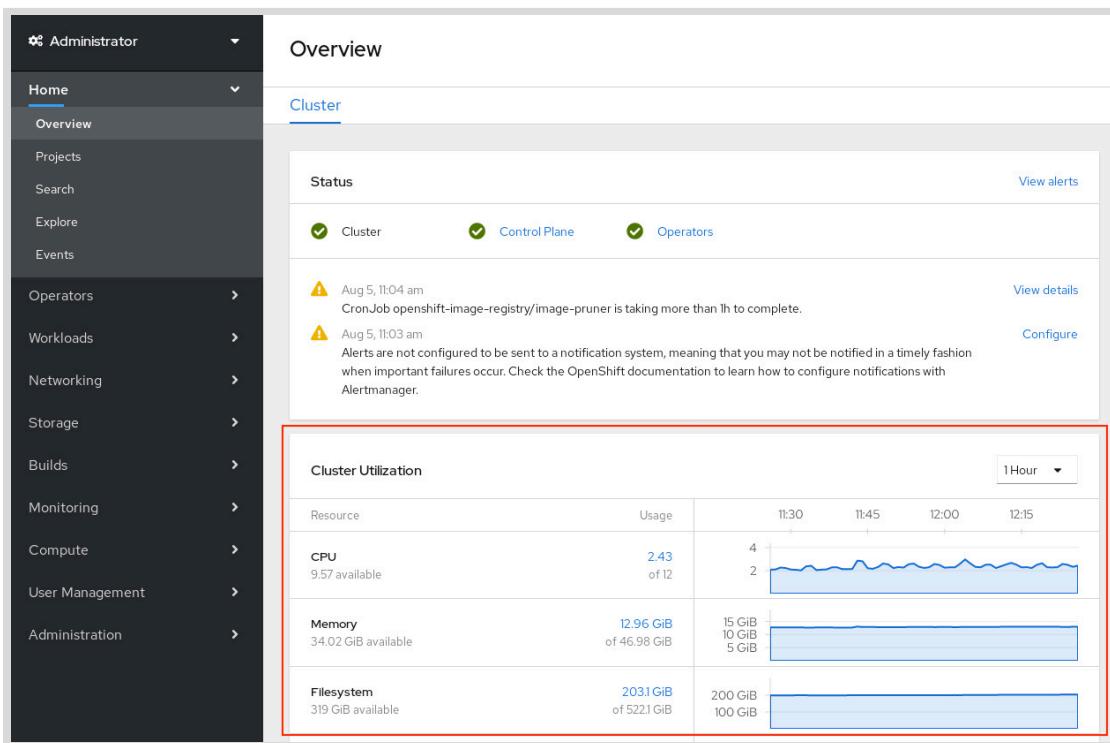
After completing this section, you should be able to examine performance and health metrics for cluster nodes and applications.

Viewing Cluster Metrics

The OpenShift web console incorporates useful graphs to visualize cluster and resource analytics. Cluster administrators and users with either the **view** cluster role or the **cluster-monitoring-view** cluster role can access the **Home → Overview** page. The **Overview** page displays a collection of cluster-wide metrics, provides a high-level view of the overall health of the cluster.

The overview includes:

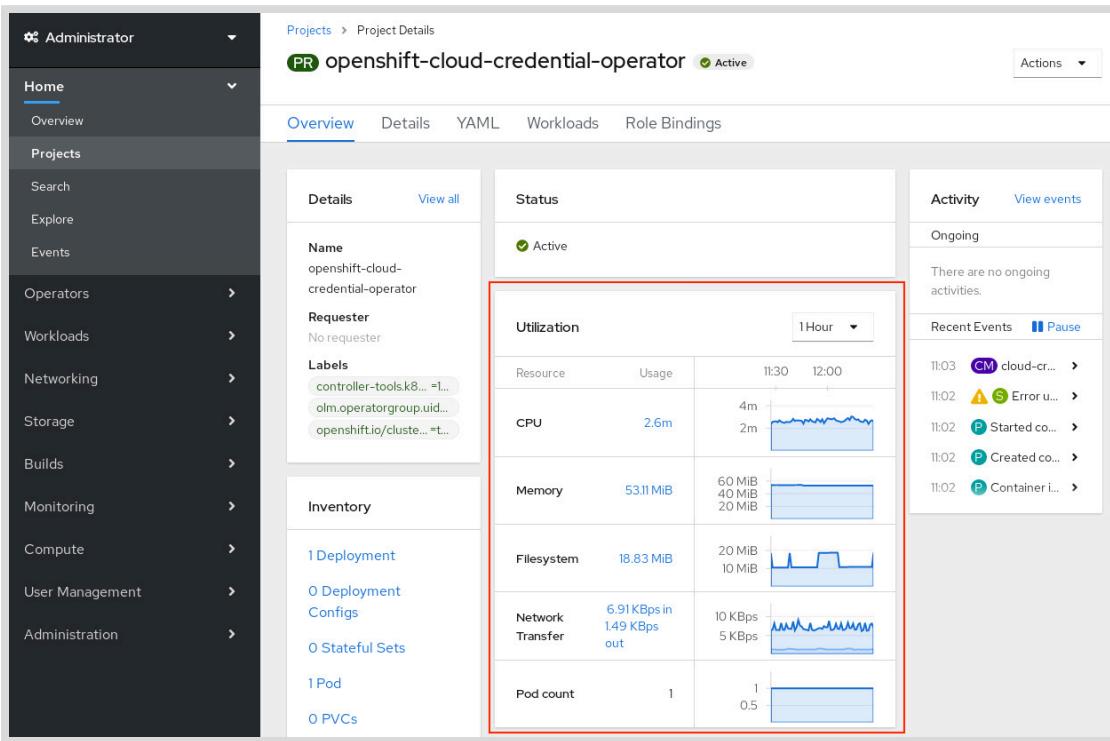
- Current cluster capacity based on CPU, memory, storage, and network usage.
- A time-series graph of total CPU, memory, and disk utilization.
- The ability to display the top consumers of CPU, memory, and storage.



For any of the resources listed in the **Cluster Utilization** section, administrators can click the link for current resource usage. The link displays a window with a breakdown of top consumers for that resource. Top consumers can be sorted by project, by pod, or by node. The list of top consumers 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 **Utilization** section displays usage information about resources such as CPU and memory along with the ability to display the top consumers for each resource.



All metrics are pulled from Prometheus. Click any graph to navigate to the **Metrics** page. 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.

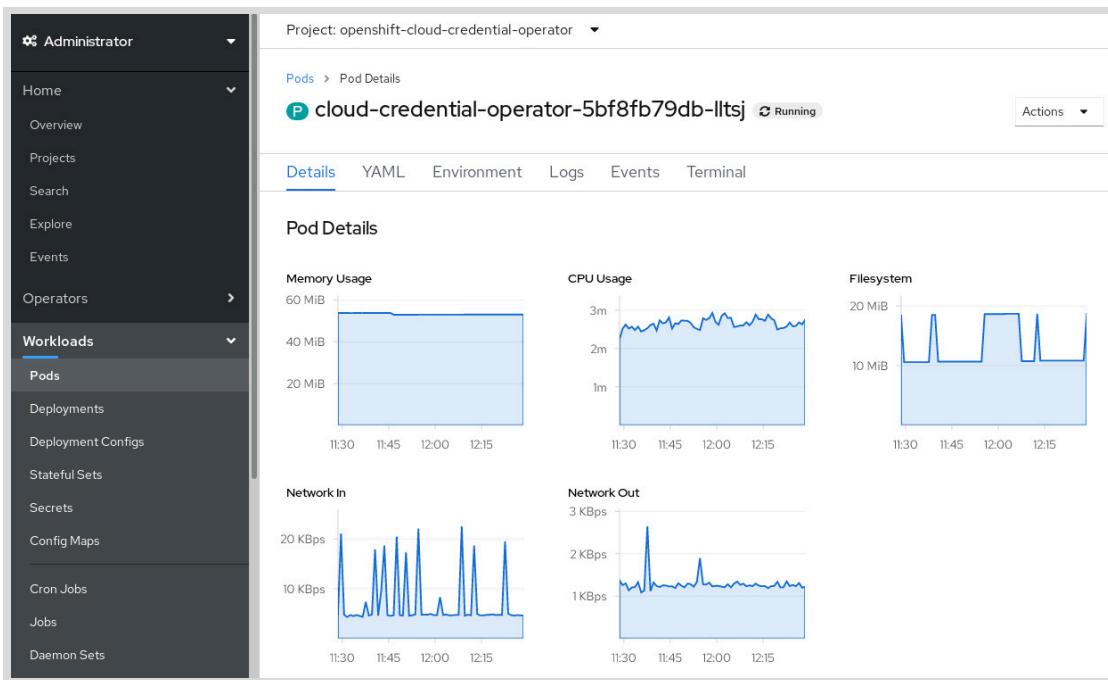


Figure 15.19: Time-series graphs showing various metrics for a pod.

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 **Monitoring** → **Metrics**, 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.

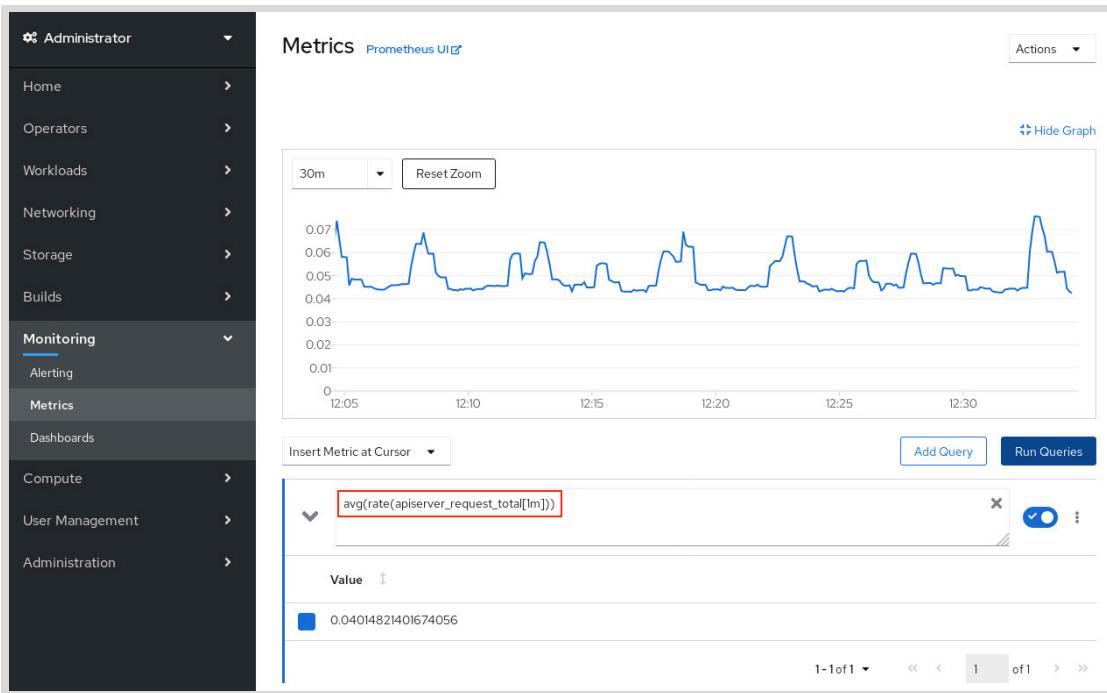


Figure 15.20: Using a Prometheus query to display a time-series graph.



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 4.5 *Monitoring* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/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

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-metrics start
```

- 1. As the **admin** user, locate and navigate to the OpenShift web console.

- 1.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Identify the URL for the web console.

```
[student@workstation ~]$ oc whoami --show-console
https://console-openshift-console.apps.ocp4.example.com
```

- 1.3. Open a web browser and navigate to **https://console-openshift-console.apps.ocp4.example.com**.
- 1.4. 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.

- 1.5. Click **localusers** and log in as the **admin** user with the password of **redhat**.

- 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 Overview, Pod Details, and Project Details pages.
- 2.1. Click **Home** → **Overview** to display the **Overview** page. 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. For each resource in the table, such as **CPU**, **Memory**, or **Filesystem**, click the link in the **Usage** column to view the **Top Consumers** of that resource. By default, the window filters top consumers by project, but you can filter by pod or by node instead.
 - 2.3. Click the usage link for **Memory**, filter the top consumers by pod, and then click the name of the pod that consumes the most memory resources.

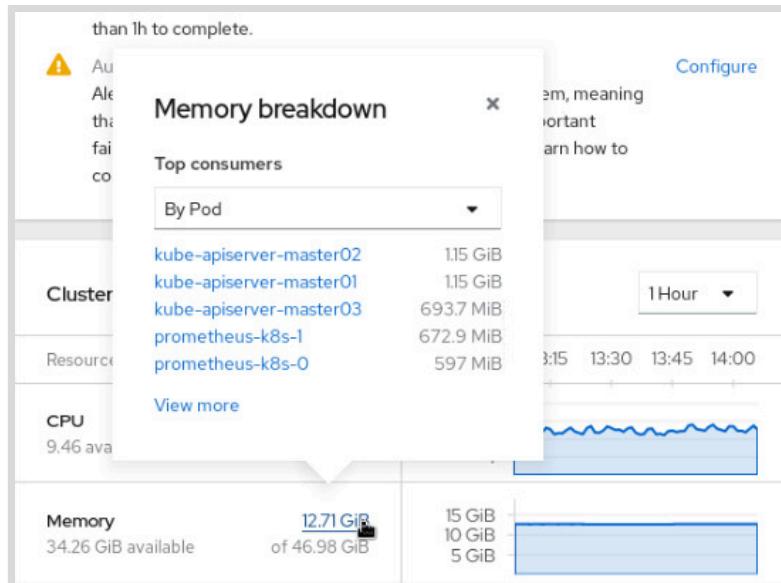


Figure 15.21: Memory breakdown: Top consumers by pod

- 2.4. The **Pod Details** page displays **Memory Usage**, **CPU Usage**, and **Filesystem** time-series historical graphs at the top of the page.
 - 2.5. Click **Home** → **Projects**, and then click **console-apps** to display the **console-apps Project Details** page. Notice the **Utilization** section, which displays the metrics for the workloads running in the **console-apps** project. The links in the **Usage** column open windows displaying the pods that consume the most resources. The workloads are running safely within their limits.
 - 2.6. Scroll down to the **Resource Quotas** section, which displays the current CPU and memory usage compared to the allotted quota.
- 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 Details** page, notice the time-series graphs that display the metrics for the individual node that you selected.

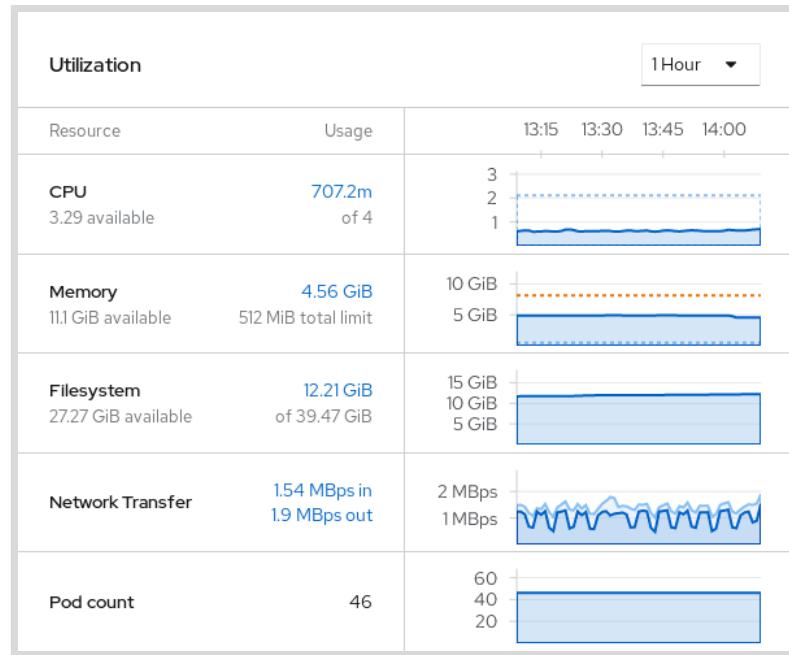


Figure 15.22: Time-series graphs showing various metrics for a node.

- ▶ 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 to monitor for changes.

The memory leak may take a minute or two before it is significant enough to be visible. Although both CPU and memory increase, the total CPU usage remains low.

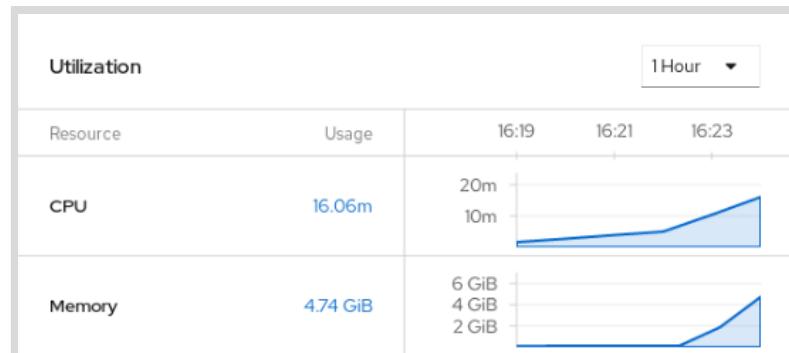


Figure 15.23: Utilization graphs indicating a possible memory leak.

- 5.2. Click **Home** → **Overview** to display the **Overview** page. The memory consumed by the load test may be too small to notice across a large cluster, but the **Memory breakdown** window (sorted by pod) provides a convenient list of pods using the most memory. View the **Memory breakdown** window by clicking the usage link for **Memory**. Sort the top consumers by pod.

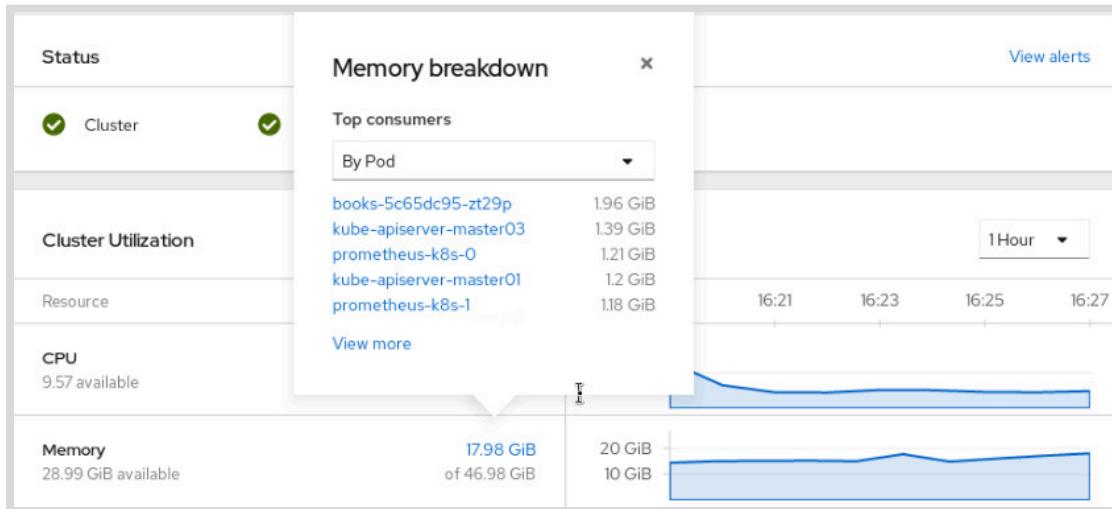
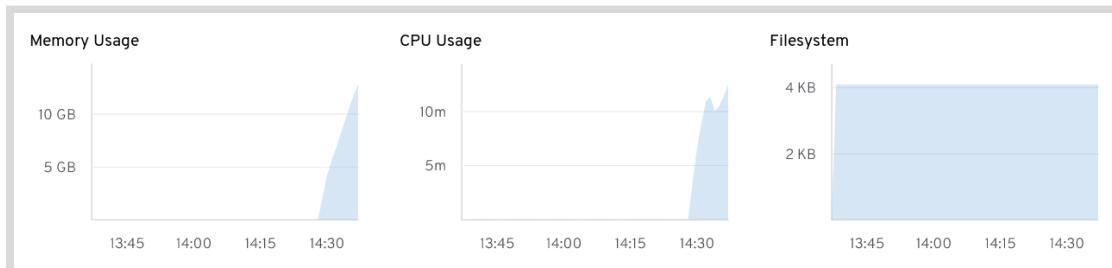


Figure 15.24: The books pod is a top memory consumer.

The **books** pod appears at or near the top of the list. If it's not on the list, you may need to wait a minute longer for the load script to complete.

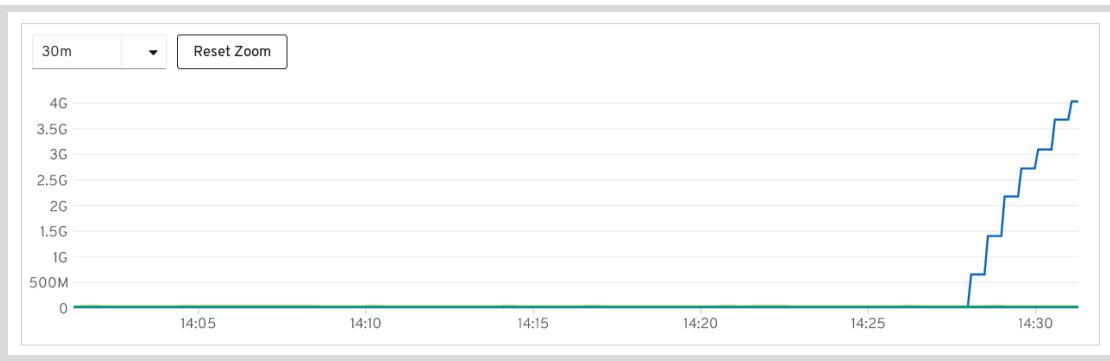
- 5.3. Click the **books** pod link in the **Memory breakdown** window to navigate to the **Pod Details** page. Notice the climbing memory leak visible in the **Memory Usage** time-series graph.



- 5.4. 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.

Project	Status	Owner	Labels	Actions
PR console-apps	Active	admin	No labels	⋮
PR default	Active	No requester	No labels	⋮
PR kube-node-lease	Active	No requester	No labels	⋮
PR kube-public	Active	No requester	No labels	⋮

- 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 a 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 **redhat** as the 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 four pods.
5. Install the certified version of the Cockroach Operator for use in the **console-review** namespace.
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 Cluster database instance into the **console-review** project using the OpenShift web console. You must modify the **CrdbCluster** resource to disable TLS. From the **YAML View**, ensure the **tlsEnabled** setting has a value of false.
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/redhat-training/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 the **workstation** machine.

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 a 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. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Identify the URL for the web console.

```
[student@workstation ~]$ oc whoami --show-console
https://console.openshift-console.apps.ocp4.example.com
```

- 1.3. Open a web browser and navigate to **https://console.openshift-console.apps.ocp4.example.com**.



Note

If prompted with an untrusted certificate message, click **Add Exception** and then click **Confirm Security Exception**.

- 1.4. Click **localusers** and log in as the **admin** user with **redhat** as the password.
2. Add **htpasswd** entries to the **localusers** secret for users named **dba** and **tester** using **redhat** as the password.
- 2.1. In the Red Hat OpenShift Container Platform web UI, click **Workloads → Secrets**, and then select **openshift-config** from the **Project** search 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 redhat
dba:$apr1$YF4aCK.9$qho0THlWTC.cLByNEHDaV
[student@workstation ~]$ htpasswd -n -b tester redhat
tester:$apr1$XdTSqET7$i0hKC5bIs7PhYUm2KhiiI.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$/abo4Hybn7a.tG5Zo0Bn.QwefXckiy1
developer:$apr1$RjqTY4cv$xql3.BQfg42moSxwnTNkh.
dba:$apr1$YF4aCK.9$qho0THlWTC.cLByNEHDaV
tester:$apr1$XdTSqET7$i0hKC5bIs7PhYUm2KhiiI.0
```

3. Create a new **app-team** group that contains the **developer** and **dba** users.
- 3.1. Click **User Management → Groups**, and then click **Create Group**. 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 four 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
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 four pods as follows:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: console-review
spec:
  hard:
    pods: '4'
```

Remove the CPU and memory requests and limits, and then click **Create**.

5. Install the certified version of the Cockroach Operator for use in the **console-review** namespace.
- 5.1. Click **Operators** → **OperatorHub**, and then click **Database** to display the list of database operators available from OperatorHub.
 - 5.2. Filter the **Provider Type** to display only **Certified** operators, and then click **Cockroach Operator**.
 - 5.3. Click **Install** on the **Cockroach Operator** page. On the **Install Operator** page, select **console-review** under **Installed Namespace**, and then click **Install**.
6. Create a RoleBinding that allows the **dba** user to view resources in the **openshift-operators** project.
- 6.1. Click **User Management** → **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 Cluster database instance into the **console-review** project using the OpenShift web console. You must modify the **CrdbCluster** resource to disable TLS. From the **YAML View**, ensure the **tlsEnabled** setting has a value of false.
- 7.1. Click **admin** → **Log out**, and then log in as the **dba** user with the password **redhat**.
 - 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 **Cockroach Operator** name.
 - 7.4. From the **Operator Details** page, click **Create Instance**. Use the **YAML View** to configure the **CrdbCluster** resource. Use the name **crdb-example**, and ensure the **tlsEnabled** setting has a value of **false**. Click **Create** to create the **CrdbCluster** resource.

```
apiVersion: crdb.cockroachlabs.com/v1alpha1
kind: CrdbCluster
metadata:
```

```

name: crdb-example
namespace: console-review
spec:
  tlsEnabled: false
  nodes: 3
  dataStore:
    pvc:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
        volumeMode: Filesystem
  
```

- 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 the **workstation** machine.

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.

- Click **dba** → **Log out**, and then log in as the **developer** user with the password of **developer**.
- Click **Home** → **Projects**, and then click the **console-review** project to switch to the **console-review** project.
- 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 **redhat**.
 - 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=4, limited:
pods=4
```

 - 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 six pods as follows, and then click **Save**.

```
kind: ResourceQuota
apiVersion: v1
metadata:
  name: quota
  namespace: console-review
...output omitted...
spec:
  hard:
    pods: '6'
...output omitted...
```



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 display a **CrashLoopBackOff** status.
- 9.6. Click the pod name, and then click the **Logs** tab. Notice the connection error.

```
2020/09/25 13:33:25 Connecting to database with: host=localhost port=26257
user=root password= dbname=postgres sslmode=disable
2020/09/25 13:33:25 dial tcp [::1]:26257: connect: connection refused
```

- 9.7. Click **Networking** → **Services** to view the services in the **console-review** namespace. There are two services related to the Cockroach database. The service name with the **-public** suffix is the only Cockroach service with an IP address, and the service uses port 26257.
- The **exoplanets** pod attempts to connect to port 26257 on the localhost instead of port 26257 on the database server. Copy the name of the service with the **-public** suffix to the clipboard.
- 9.8. Click **Workloads** → **Deployments**, and then click the **exoplanets** Deployment link. Click the **Environment** tab and change the **DB_HOST** value from **localhost** to the name of the Cockroach service that you previously copied. Click **Save**.
- 9.9. 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.

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.

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 16

Comprehensive Review

Goal

Review tasks from *Containers, Kubernetes, and Red Hat OpenShift Administration II*

Objectives

- Review tasks from *Containers, Kubernetes, and Red Hat OpenShift Administration II*

Sections

- Comprehensive Review

Lab

- Troubleshoot an OpenShift Cluster and Applications
- Configure a Project Template with Resource and Network Restrictions

Comprehensive Review

Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills learned in *Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster*.

Reviewing Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster

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.

Describing the Red Hat OpenShift Container Platform

Describe the architecture of OpenShift Container Platform.

- Describe the typical usage of the product and its features.
- Describe the architecture of Red Hat OpenShift Container Platform.
- Describe what a cluster operator is, how it works, and name the major cluster operators.

Verifying the Health of a Cluster

Describe OpenShift installation methods and verify the health of a newly installed cluster.

- Describe the OpenShift installation process, full-stack automation, and pre-existing infrastructure installation methods.
- Execute commands that assist in troubleshooting common problems.
- Identify the components and resources of persistent storage and deploy an application that uses a persistent volume claim.

Configuring Authentication and Authorization

Configure authentication with the HTPasswd identity provider and assign roles to users and groups.

- Configure the HTPasswd identity provider for OpenShift authentication.
- Define role-based access controls and apply permissions to users and groups.

Configuring Application Security

Restrict permissions of applications using security context constraints and protect access credentials using secrets.

- Create and apply secrets to manage sensitive information.
- Create service accounts and apply permissions.

Configuring OpenShift Networking for Applications

Troubleshoot OpenShift software-defined networking (SDN) and configure network policies.

- Troubleshoot OpenShift software-defined networking using the command-line interface.
- Allow and protect network connections to applications inside an OpenShift cluster.
- Restrict network traffic between projects and pods.

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 containers, pods, and projects.
- Control the number of replicas of a pod.

Describing Cluster Updates

Describe how to perform a cluster update.

Describe the cluster update process.

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 applications and Kubernetes Operators with the web console.
- Examine performance and health metrics for cluster nodes and applications.

► Lab

Troubleshoot an OpenShift Cluster and Applications

In this review, you will allow developers to access a cluster and troubleshoot application deployments.

Outcomes

You should be able to:

- Create a new project.
- Perform a smoke test of the OpenShift cluster by creating an application using the source-to-image process.
- Create applications using either **deployment** or **deploymentconfig** resources.
- Use the HTPasswd identity provider for managing users.
- Create and manage groups.
- Manage RBAC and SCC for users and groups.
- Manage secrets for databases and applications.
- 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.
- Removes existing users and groups.
- Removes existing identity providers.
- Removes the **cluster-admin** cluster role binding from the **admin** user.
- Ensures that authenticated users can create new projects.

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

Instructions

Complete the following tasks:

1. As the **kubeadmin** user, create the **review-troubleshoot** project. The password for the **kubeadmin** user is located in the **/usr/local/etc/ocp4.config** file on

- the **RHT_OCP4_KUBEADM_PASSWD** line. Perform all subsequent tasks in the **review-troubleshoot** project.
2. Perform a smoke test of the cluster to verify basic cluster functionality. Use a **deployment configuration** to create an application named **hello-world-nginx**. The application source code is located in the **hello-world-nginx** subdirectory of the <https://github.com/RedHatTraining/D0280-apps> repository.
- Create a route for the application using any available hostname in the **apps.ocp4.example.com** subdomain, and then verify that the application responds to external requests.
3. 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 **comprevew-users** secret.
- Ensure that four user accounts exist: **admin**, **leader**, **developer**, and **qa-engineer**. All user accounts must use **review** as the password.
- Add the **cluster-admin** role to the **admin** user.
4. As the **admin** user, create three user groups: **leaders**, **developers**, and **qa**.
- Assign the **leader** user to the **leaders** group, the **developer** user to the **developers** group, 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 **review-troubleshoot** project only, which allows members to create and delete project resources.
 - Assign the **view** role to the **qa** group for the **review-troubleshoot** project only, which provides members with read access to project resources.
5. As the **developer** user, use a **deployment** to create an application named **mysql** in the **review-troubleshoot** project. Use the container image available at registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7. This application provides a shared database service for other project applications.
- Create a generic secret named **mysql** using **password** as the key and **r3dh4t123** as the value.
- Set the **MYSQL_ROOT_PASSWORD** environment variable to the value of the **password** key in the **mysql** secret.
- Configure the **mysql** database application to mount a persistent volume claim (PVC) to the **/var/lib/mysql/data** directory within the pod. The PVC must be **2 GB** in size and must only request the **ReadWriteOnce** access mode.
6. As the **developer** user, use a **deployment** to create an application named **wordpress**. Create the application in the **review-troubleshoot** project. Use the image available at quay.io/redhattraining/wordpress:5.3.0. This image is a copy of the container image available at docker.io/library/wordpress:5.3.0.
- 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** environment variable to have a value of **root**. Set the **WORDPRESS_DB_PASSWORD** environment variable to the value of the **password** key in the **mysql** secret. The value of the **WORDPRESS_DB_PASSWORD** environment variable must be the same as the **mysql** root user password.

The **wordpress** application also requires the **anyuid** security context constraint. Create a service account named **wordpress-sa**, and then assign the **anyuid** security context constraint to it. Configure the **wordpress** deployment to use the **wordpress-sa** service account.

Create a route for the application using any available hostname in the **apps.ocp4.example.com** subdomain. If you correctly deploy the application, then an installation wizard displays when you access the application from a browser.

7. As the **developer** user, deploy the **famous-quotes** application in the **review-troubleshoot** project using the **~/DO280/labs/review-troubleshoot/deploy_famous-quotes.sh** script. This script creates the **defaultdb** database and the resources defined in the **~/DO280/labs/review-troubleshoot/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 using any available hostname in the **apps.ocp4.example.com** subdomain, and then verify that the application responds to external requests.

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 review-troubleshoot grade
```

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise.

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

This concludes the comprehensive review.

► Solution

Troubleshoot an OpenShift Cluster and Applications

In this review, you will allow developers to access a cluster and troubleshoot application deployments.

Outcomes

You should be able to:

- Create a new project.
- Perform a smoke test of the OpenShift cluster by creating an application using the source-to-image process.
- Create applications using either **deployment** or **deploymentconfig** resources.
- Use the HTPasswd identity provider for managing users.
- Create and manage groups.
- Manage RBAC and SCC for users and groups.
- Manage secrets for databases and applications.
- 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.
- Removes existing users and groups.
- Removes existing identity providers.
- Removes the **cluster-admin** cluster role binding from the **admin** user.
- Ensures that authenticated users can create new projects.

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

Instructions

Complete the following tasks:

- As the **kubeadmin** user, create the **review-troubleshoot** project. The password for the **kubeadmin** user is located in the **/usr/local/etc/ocp4.config** file on the **RHT_OCP4_KUBEADM_PASSWD** line. Perform all subsequent tasks in the **review-troubleshoot** project.

- 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} \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- Create the **review-troubleshoot** project.

```
[student@workstation ~]$ oc new-project review-troubleshoot
Now using project "review-troubleshoot" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

- Perform a smoke test of the cluster to verify basic cluster functionality. Use a **deployment configuration** to create an application named **hello-world-nginx**. The application source code is located in the **hello-world-nginx** subdirectory of the <https://github.com/RedHatTraining/D0280-apps> repository.

Create a route for the application using any available hostname in the **apps.ocp4.example.com** subdomain, and then verify that the application responds to external requests.

- Use the **oc new-app** command to create the **hello-world-nginx** deployment configuration. Make sure to use the **--as-deployment-config** option to create the application using a deployment configuration.

```
[student@workstation ~]$ oc new-app --name hello-world-nginx \
>   --as-deployment-config \
>   https://github.com/RedHatTraining/D0280-apps \
>   --context-dir hello-world-nginx
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "ubi8" created
imagestream.image.openshift.io "hello-world-nginx" created
buildconfig.build.openshift.io "hello-world-nginx" created
deploymentconfig.apps.openshift.io "hello-world-nginx" created
service "hello-world-nginx" created
--> Success
...output omitted...
```

- Create a route to the application by exposing the **hello-world-nginx** service.

```
[student@workstation ~]$ oc expose service hello-world-nginx \
>   --hostname hello-world.apps.ocp4.example.com
route.route.openshift.io/hello-world-nginx exposed
```

2.3. Wait until the application pod is running.

```
[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-m2lzs   1/1     Running    0          100s
```

2.4. Verify access to the application.

```
[student@workstation ~]$ curl -s http://hello-world.apps.ocp4.example.com \
>   | grep Hello
<h1>Hello, world from nginx!</h1>
```

3. 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 **comprevew-users** secret.

Ensure that four user accounts exist: **admin**, **leader**, **developer**, and **qa-engineer**. All user accounts must use **review** as the password.

Add the **cluster-admin** role to the **admin** user.

- 3.1. Create a temporary **htpasswd** authentication file at **/tmp/cluster-users**.

```
[student@workstation ~]$ touch /tmp/cluster-users
```

- 3.2. Populate the **/tmp/cluster-users** file with the required user and password values.

```
[student@workstation ~]$ for user in admin leader developer qa-engineer
>   do
>     htpasswd -B -b /tmp/cluster-users ${user} review
>   done
Adding password for user admin
Adding password for user leader
Adding password for user developer
Adding password for user qa-engineer
```

- 3.3. Create a **comprevew-users** secret from the **/tmp/cluster-users** file.

```
[student@workstation ~]$ oc create secret generic comprevew-users \
>   --from-file htpasswd=/tmp/cluster-users -n openshift-config
secret/comprevew-users created
```

- 3.4. Export the existing OAuth resource to a YAML file.

```
[student@workstation ~]$ oc get oauth cluster -o yaml > /tmp/oauth.yaml
```

- 3.5. Edit the **/tmp/oauth.yaml** file to add the **HTPasswd** identity provider definition to the **identityProviders** list. Set the identity provider name to **cluster-users**, and set the **fileData** name to **comprevew-users**.

After making these modifications, the file reads as follows:

```

apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  ...output omitted...
  name: cluster
  ...output omitted...
spec:
  identityProviders:
    - name: cluster-users
      mappingMethod: claim
      type: HTPasswd
      htpasswd:
        fileData:
          name: compreview-users

```

**Note**

The **name**, **mappingMethod**, **type**, and **htpasswd** keys all use the same indentation.

- Replace the existing OAuth resource with the resource definition in the modified file:

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

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

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
>   cluster-admin admin
Warning: User 'admin' not found
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "admin"
```

**Important**

You can safely ignore the warning about the **admin** user not being found. The **user/admin** resource does not exist in your OpenShift cluster until after the **admin** user logs in for the first time.

- As the **admin** user, create three user groups: **leaders**, **developers**, and **qa**.

Assign the **leader** user to the **leaders** group, the **developer** user to the **developers** group, 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 **review-troubleshoot** project only, which allows members to create and delete project resources.

- Assign the **view** role to the **qa** group for the **review-troubleshoot** project only, which provides members with read access to project resources.

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

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

4.2. Create the three user groups.

```
[student@workstation ~]$ for group in leaders developers qa
> do
> oc adm groups new ${group}
> done
group.user.openshift.io/leaders created
group.user.openshift.io/developers created
group.user.openshift.io/qa created
```

4.3. Add each user to the appropriate group.

```
[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"
```

4.4. Allow members of the **leaders** group to create new projects:

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

4.5. Remove the **self-provisioner** cluster role from the **system:authenticated:oauth** 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-provisioners
'rbac.authorization.kubernetes.io/autoupdate=false' --overwrite
clusterrole.rbac.authorization.k8s.io/self-provisioner removed:
"system:authenticated:oauth"
```

4.6. Allow members of the **developers** group to create and delete resources in the **review-troubleshoot** project:

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

- 4.7. Allow members of the **qa** group to view project resources:

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

5. As the **developer** user, use a **deployment** to create an application named **mysql** in the **review-troubleshoot** project. Use the container image available at **registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7**. This application provides a shared database service for other project applications.

Create a generic secret named **mysql** using **password** as the key and **r3dh4t123** as the value.

Set the **MYSQL_ROOT_PASSWORD** environment variable to the value of the **password** key in the **mysql** secret.

Configure the **mysql** database application to mount a persistent volume claim (PVC) to the **/var/lib/mysql/data** directory within the pod. The PVC must be **2 GB** in size and must only request the **ReadWriteOnce** access mode.

- 5.1. Log in to the cluster as the **developer** user.

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

- 5.2. Create a new application to deploy a **mysql** database server. Use the **oc new-app** command to create a deployment.

```
[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
  deployment.apps "mysql" created
  service "mysql" created
--> Success
...output omitted...
```

- 5.3. Create a generic secret for the MySQL database named **mysql** using a key of **password** and a value of **r3dh4t123**.

```
[student@workstation ~]$ oc create secret generic mysql \
>   --from-literal password=r3dh4t123
secret/mysql created
```

- 5.4. Use the **mysql** secret to initialize environment variables for the **mysql** deployment.

```
[student@workstation ~]$ oc set env deployment mysql \
>   --prefix MYSQL_ROOT_ --from secret/mysql
deployment.apps/mysql updated
```

- 5.5. Use the **oc set volumes** command to configure persistent storage for the **mysql** deployment. The command automatically creates a persistent volume claim with the

specified size and access mode. Mounting the volume to the `/var/lib/mysql/data` directory provides access to stored data, even if one database pod is deleted and another database pod is created.

```
[student@workstation ~]$ oc set volumes deployment/mysql --name mysql-storage \
>   --add --type pvc --claim-size 2Gi --claim-mode rwo \
>   --mount-path /var/lib/mysql/data
deployment.apps/mysql volume updated
```

- 5.6. Verify that the `mysql` pod successfully redeploys after configuring the deployment to use a secret and a volume. You may need to run the `oc get pods` command multiple times until the `mysql` pod displays both **1/1** and **Running**.

```
[student@workstation ~]$ oc get pods -l deployment=mysql
NAME          READY   STATUS    RESTARTS   AGE
mysql-bbb6b5fbb-dmq9x   1/1     Running   0          63s
```

- 5.7. Verify that a persistent volume claim exists with the correct size and access mode.

```
[student@workstation ~]$ oc get pvc
NAME      STATUS  ...  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
pvc-ks52v  Bound  ...  2G        RWO          nfs-storage  2m33s
```

- 5.8. Verify that the running `mysql` pod mounts a volume to the `/var/lib/mysql/data` directory.

```
[student@workstation ~]$ oc exec mysql-bbb6b5fbb-dmq9x -- df -h \
>   | grep /var/lib/mysql/data
192.168.50.254:/exports/review-troubleshoot-pvc-ks52v-pvc-0d6a63bd-286e-44ec-b29c-
ffbb34928b86  40G  859M  40G   3% /var/lib/mysql/data
```

6. As the `developer` user, use a `deployment` to create an application named `wordpress`. Create the application in the `review-troubleshoot` project. Use the image available at `quay.io/redhattraining/wordpress:5.3.0`. This image is a copy of the container image available at `docker.io/library/wordpress:5.3.0`.

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` environment variable to have a value of `root`. Set the `WORDPRESS_DB_PASSWORD` environment variable to the value of the `password` key in the `mysql` secret. The value of the `WORDPRESS_DB_PASSWORD` environment variable must be the same as the `mysql` root user password.

The `wordpress` application also requires the `anyuid` security context constraint. Create a service account named `wordpress-sa`, and then assign the `anyuid` security context constraint to it. Configure the `wordpress` deployment to use the `wordpress-sa` service account.

Create a route for the application using any available hostname in the **apps.ocp4.example.com** subdomain. If you correctly deploy the application, then an installation wizard displays when you access the application from a browser.

6.1. Deploy a **wordpress** application as a **deployment**.

```
[student@workstation ~]$ oc new-app --name wordpress \
>   --docker-image quay.io/redhattraining/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
deployment.apps "wordpress" created
service "wordpress" created
--> Success
...output omitted...
```

6.2. Add the **WORDPRESS_DB_PASSWORD** environment variable to the **wordpress** deployment. Use the value of the **password** key in the **mysql** secret as the value for the variable.

```
[student@workstation ~]$ oc set env deployment/wordpress \
>   --prefix WORDPRESS_DB_ --from secret/mysql
deployment.apps/wordpress updated
```

6.3. Create the **wordpress-sa** service account.

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

6.4. 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 review
Login successful.
...output omitted...
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z wordpress-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added:
"wordpress-sa"
```

6.5. Switch back to the **developer** user to perform the remaining steps. Log in to the cluster as the **developer** user.

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

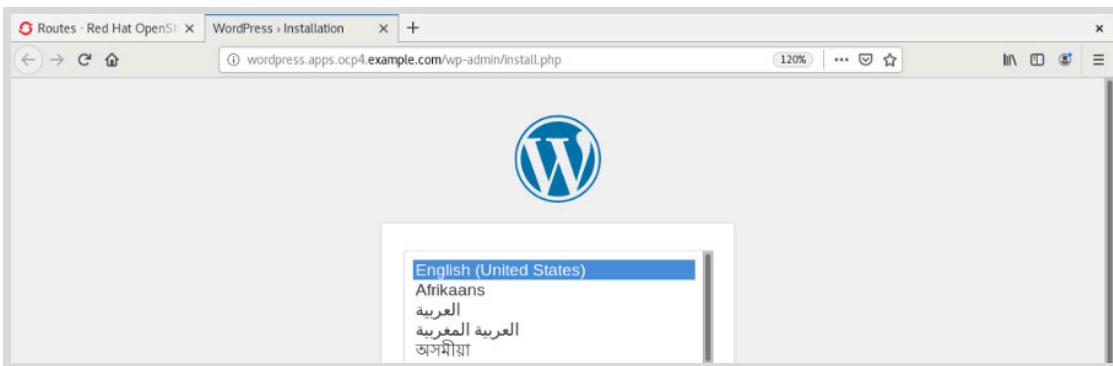
6.6. Configure the **wordpress** deployment to use the **wordpress-sa** service account.

```
[student@workstation ~]$ oc set serviceaccount deployment/wordpress \
>   wordpress-sa
deployment.apps/wordpress serviceaccount updated
```

- 6.7. Create a route for the **wordpress** application.

```
[student@workstation ~]$ oc expose service wordpress \
>   --hostname wordpress.apps.ocp4.example.com
route.route.openshift.io/wordpress exposed
```

- 6.8. Use a web browser to verify access to the URL `http://wordpress.apps.ocp4.example.com`. When you correctly deploy the application, a setup wizard displays in the browser.



7. As the **developer** user, deploy the **famous-quotes** application in the **review-troubleshoot** project using the `~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh` script. This script creates the **defaultdb** database and the resources defined in the `~/D0280/labs/review-troubleshoot/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 using any available hostname in the `apps.ocp4.example.com` subdomain, and then verify that the application responds to external requests.

- 7.1. Run the `~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh` script.

```
[student@workstation ~]$ ~/D0280/labs/review-troubleshoot/deploy_famous-quotes.sh
Creating famous-quotes database
Deploying famous-quotes application
deploymentconfig.apps.openshift.io/famous-quotes created
service/famous-quotes created
```

- 7.2. 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...
```

7.3. See if any project events provide information about the problem.

```
[student@workstation ~]$ oc get events --sort-by='{.lastTimestamp}'
...output omitted...
34s  Warning  FailedScheduling      pod/famous-quotes-1-deploy  0/3 nodes are
available: 3 node(s) didn't match node selector.
...output omitted...
```

7.4. Save the **famous-quotes** deployment configuration resource to a file.

```
[student@workstation ~]$ oc get dc/famous-quotes -o yaml > /tmp/famous-dc.yaml
```

7.5. Use an editor to remove the node selector from the **/tmp/famous-dc.yaml** file. Search for the **nodeSelector** line in the file. Delete the following two lines from the **/tmp/famous-dc.yaml** file.

```
nodeSelector:
  env: quotes
```

7.6. Replace the **famous-quotes** deployment configuration with the modified file.

```
[student@workstation ~]$ oc replace -f /tmp/famous-dc.yaml
deploymentconfig.apps.openshift.io/famous-quotes replaced
```

7.7. Wait a few moments and then run the **oc get pods** command to ensure that the **famous-quotes** application is now running.

```
[student@workstation ~]$ oc get pods -l app=famous-quotes
NAME          READY   STATUS    RESTARTS   AGE
famous-quotes-2-gmz2j  1/1     Running   0          53s
```

7.8. Create a route for the **famous-quotes** application.

```
[student@workstation ~]$ oc expose service famous-quotes \
>   --hostname quotes.apps.ocp4.example.com
route.route.openshift.io/famous-quotes exposed
```

7.9. Verify that the **famous-quotes** application responds to requests sent to the **http://quotes.apps.ocp4.example.com** URL.

```
[student@workstation ~]$ curl -s http://quotes.apps.ocp4.example.com \
>   | grep Quote
<title>Quotes</title>
<h1>Quote List</h1>
```

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 review-troubleshoot grade
```

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise.

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

This concludes the comprehensive review.

▶ Lab

Configure a Project Template with Resource and Network Restrictions

In this review, you will configure the cluster's default project template to ensure that new projects enforce default quotas, resource limits, and network policies.

Outcomes

You should be able to:

- Modify the default project template to automatically create limit ranges, resource quotas, and network policies.
- Create a TLS secret using the provided files.
- Mount a secret as a volume within an application.
- Create a passthrough route to an application.
- Configure an application to automatically scale.

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.
- Configures the HTPasswd identity provider and provides access to the **admin**, **leader**, and **developer** users.
- Downloads sample YAML files to **~/DO280/labs/review-template/sample-files/**.
- Creates the **review-template-test** project and deploys an application to the project that you can use to test your network policies.
- Adds the **network.openshift.io/policy-group=ingress** label to the **default** namespace.
- Generates certificate files needed for the secure application.

```
[student@workstation ~]$ lab review-template start
```



Note

If you need help getting started, consider using the *Post-installation network configuration* chapter of the Red Hat OpenShift Container Platform Post-installation configuration guide as a reference.

Instructions

Complete the following tasks:

- As the **admin** user, update the OpenShift cluster to use a new project template. The project template must automatically create network policy, limit range, and quota resources for new projects. New projects must automatically have a label matching the name of the project. For example, a project named **test** has the **name=test** label.

The following table provides guidance on the needed resources.

Resource	Requirements
Project	<ul style="list-style-type: none"> Includes a label with the name of the project.
NetworkPolicy	<p>Policy 1:</p> <ul style="list-style-type: none"> Routes are accessible to external traffic; this means that traffic is allowed from pods in namespaces with the network.openshift.io/policy-group=ingress label. <p>Policy 2:</p> <ul style="list-style-type: none"> Pods in the same namespace can communicate with each other. Pods do not respond to pods that exist in a different namespace, except namespaces with the network.openshift.io/policy-group=ingress label.
LimitRange	<ul style="list-style-type: none"> Each container requests 30 millicores of CPU. Each container requests 30 MiB of memory. Each container is limited to 100 millicores of CPU. Each container is limited to 100 MiB of memory.
ResourceQuota	<ul style="list-style-type: none"> Projects are limited to 10 pods. Projects can request a maximum of 1 GiB of memory. Projects can request a maximum of 2 CPUs. Projects can use a maximum of 4 GiB of memory. Projects can use a maximum of 4 CPUs.

- As the **developer** user, create a project named **review-template**. Ensure that the **review-template** project inherits the settings specified in the new project template. In

the **review-template** project, create a deployment named **hello-secure** using the container image located at **quay.io/redhattraining/hello-world-secure:v1.0**.

**Note**

The **hello-secure** pod does not run successfully until after you provide access to the TLS certificate and key required by the NGINX server.

3. As the **developer** user, create a TLS secret using the **hello-secure-combined.pem** certificate and the **hello-secure-key.pem** key located in the **~/D0280/labs/review-template/** directory. Use the logs from the failed **hello-secure** pod to determine the expected mount point for the certificate. Mount the TLS secret as a volume in the pod using the identified directory. Verify that the **hello-secure** pod successfully redeploys.
4. The **hello-secure-combined.pem** certificate is valid for a single host name. Use the **openssl x509** command with the **-noout** and **-ext 'subjectAltName'** options to read the **hello-secure-combined.pem** certificate and identify the host name. As the **developer** user, create a passthrough route to the **hello-secure** service using the identified host name. Verify that the route responds to external requests.

**Note**

The **x509(1)** man page provides information on the **openssl x509** command.

5. As the **developer** user, configure the **hello-secure** deployment to scale automatically. The deployment must have at least one pod running. If the average CPU utilization exceeds 80%, then the deployment scales to a maximum of five pods.

**Note**

You can use the script located at **~/D0280/solutions/review-template/test-hpa.sh** to test that your deployment scales as expected.

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 review-template grade
```

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise.

```
[student@workstation ~]$ lab review-template finish
```

This concludes the comprehensive review.

► Solution

Configure a Project Template with Resource and Network Restrictions

In this review, you will configure the cluster's default project template to ensure that new projects enforce default quotas, resource limits, and network policies.

Outcomes

You should be able to:

- Modify the default project template to automatically create limit ranges, resource quotas, and network policies.
- Create a TLS secret using the provided files.
- Mount a secret as a volume within an application.
- Create a passthrough route to an application.
- Configure an application to automatically scale.

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.
- Configures the HTPasswd identity provider and provides access to the **admin**, **leader**, and **developer** users.
- Downloads sample YAML files to **~/DO280/labs/review-template/sample-files/**.
- Creates the **review-template-test** project and deploys an application to the project that you can use to test your network policies.
- Adds the **network.openshift.io/policy-group=ingress** label to the **default** namespace.
- Generates certificate files needed for the secure application.

```
[student@workstation ~]$ lab review-template start
```



Note

If you need help getting started, consider using the *Post-installation network configuration* chapter of the Red Hat OpenShift Container Platform Post-installation configuration guide as a reference.

Instructions

Complete the following tasks:

- As the **admin** user, update the OpenShift cluster to use a new project template. The project template must automatically create network policy, limit range, and quota resources for new projects. New projects must automatically have a label matching the name of the project. For example, a project named **test** has the **name=test** label.

The following table provides guidance on the needed resources.

Resource	Requirements
Project	<ul style="list-style-type: none"> Includes a label with the name of the project.
NetworkPolicy	<p>Policy 1:</p> <ul style="list-style-type: none"> Routes are accessible to external traffic; this means that traffic is allowed from pods in namespaces with the network.openshift.io/policy-group=ingress label. <p>Policy 2:</p> <ul style="list-style-type: none"> Pods in the same namespace can communicate with each other. Pods do not respond to pods that exist in a different namespace, except namespaces with the network.openshift.io/policy-group=ingress label.
LimitRange	<ul style="list-style-type: none"> Each container requests 30 millicores of CPU. Each container requests 30 MiB of memory. Each container is limited to 100 millicores of CPU. Each container is limited to 100 MiB of memory.
ResourceQuota	<ul style="list-style-type: none"> Projects are limited to 10 pods. Projects can request a maximum of 1 GiB of memory. Projects can request a maximum of 2 CPUs. Projects can use a maximum of 4 GiB of memory. Projects can use a maximum of 4 CPUs.

- Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
>   https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 1.2. Use the **oc adm create-bootstrap-project-template** command to create a new YAML file that you will customize for this exercise. Save the file to **~/D0280/labs/review-template/project-template.yaml**.

```
[student@workstation ~]$ oc adm create-bootstrap-project-template \
> -o yaml > ~/D0280/labs/review-template/project-template.yaml
```

- 1.3. Change to the **~/D0280/labs/review-template/** directory.

```
[student@workstation ~]$ cd ~/D0280/labs/review-template/
```

- 1.4. Edit the **project-template.yaml** file to add a label for new projects. Add the bold lines, ensuring proper indentation, and then save the file. The **PROJECT_NAME** variable takes the value of the project name.

```
...output omitted...
- apiVersion: project.openshift.io/v1
  kind: Project
  metadata:
    labels:
      name: ${PROJECT_NAME}
  annotations:
...output omitted...
```

- 1.5. List the files in the **~/D0280/labs/review-template/sample-files/** directory. The directory provides two sample network policy files, a sample limit range file, and a sample resource quota file.

```
[student@workstation review-template]$ ls sample-files/
allow-from-openshift-ingress.yaml  allow-same-namespace.yaml  limitrange.yaml
resourcequota.yaml
```

- 1.6. Add the content of the **sample-files/allow-*.yaml** files to the **project-template.yaml** file. Because the **project-template.yaml** file expects a list of resources, the first line of each resource must begin with a **-**, and you must indent the remainder of the content accordingly.

Edit the **project-template.yaml** file to add the network policy resources. Add the bold lines, ensuring proper indentation, and then save the file.

```
...output omitted...
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: ${PROJECT_ADMIN_USER}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    podSelector: {}
    ingress:
```

```

- from:
  - namespaceSelector:
    matchLabels:
      network.openshift.io/policy-group: ingress
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-same-namespace
  spec:
    podSelector: {}
    ingress:
      - from:
        - podSelector: {}
parameters:
- name: PROJECT_NAME
...output omitted...

```

- 1.7. Add the content of the **sample-files/limitrange.yaml** and **sample-files/resourcequota.yaml** files to the **project-template.yaml** file. Because the **project-template.yaml** file expects a list of resources, the first line of each resource must begin with a **-**, and you must indent the remainder of the content accordingly.
- Edit the **project-template.yaml** file to add the limit range and resource quota resources. Add the bold lines, ensuring proper indentation, and then save the file.

```

...output omitted...
  ingress:
    - from:
      - podSelector: {}
- apiVersion: v1
  kind: LimitRange
  metadata:
    name: project-limitrange
  spec:
    limits:
      - default:
          memory: 100Mi
          cpu: 100m
        defaultRequest:
          memory: 30Mi
          cpu: 30m
        type: Container
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: project-quota
  spec:
    hard:
      pods: '10'
      requests.cpu: '2'
      requests.memory: 1Gi
      limits.cpu: '4'
      limits.memory: 4Gi

```

```
parameters:
- name: PROJECT_NAME
...output omitted...
```

- 1.8. Use the **oc create** command to create a new template resource in the **openshift-config** namespace using the **project-template.yaml** file.

**Note**

The `~/DO280/solutions/review-template/project-template.yaml` file contains the correct configuration and can be used for comparison.

```
[student@workstation review-template]$ oc create -f project-template.yaml \
> -n openshift-config
template.template.openshift.io/project-request created
```

- 1.9. List the templates in the **openshift-config** namespace. You use the template name in the next step.

```
[student@workstation review-template]$ oc get templates -n openshift-config
NAME          DESCRIPTION      PARAMETERS     OBJECTS
project-request           5 (5 blank)    6
```

- 1.10. Update the **projects.config.openshift.io/cluster** resource to use the new template.

```
[student@workstation review-template]$ oc edit \
> projects.config.openshift.io/cluster
```

Modify the **spec: {}** line to match the following bold lines. Use the template name identified in the **openshift-config** namespace. Ensure proper indentation, and then save your changes.

```
...output omitted...
spec:
  projectRequestTemplate:
    name: project-request
```

- 1.11. A successful change redeploys the **apiserver** pods in the **openshift-apiserver** namespace. Monitor the redeployment.

```
[student@workstation review-template]$ watch oc get pods -n openshift-apiserver
```

Press **Ctrl+C** to end the **watch** command after all three new pods are running.

```
Every 2.0s: oc get pods -n openshift-apiserver
...
NAME          READY   STATUS    RESTARTS   AGE
apiserver-75cfdfc877-257vs  1/1     Running   0          61s
apiserver-75cfdfc877-l2xnv  1/1     Running   0          29s
apiserver-75cfdfc877-rn9fs  1/1     Running   0          47s
```

2. As the **developer** user, create a project named **review-template**. Ensure that the **review-template** project inherits the settings specified in the new project template. In the **review-template** project, create a deployment named **hello-secure** using the container image located at **quay.io/redhattraining/hello-world-secure:v1.0**.

**Note**

The **hello-secure** pod does not run successfully until after you provide access to the TLS certificate and key required by the NGINX server.

- 2.1. Log in to your OpenShift cluster as the **developer** user.

```
[student@workstation review-template]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 2.2. Create the **review-template** project.

```
[student@workstation review-template]$ oc new-project review-template
Now using project "review-template" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 2.3. List the network policy, limit range, and quota resources in the **review-template** project.

```
[student@workstation review-template]$ oc get \
>   networkpolicy,limitrange,resourcequota
NAME
networkpolicy.networking.k8s.io/allow-from-openshift-ingress   ...
networkpolicy.networking.k8s.io/allow-same-namespace           ...

NAME          CREATED AT
limitrange/project-limitrange  2020-10-19T16:17:19Z

NAME          AGE    REQUEST
resourcequota/project-quota  37s    pods: 0/10, requests.cpu: 0/2,
                                requests.memory: 0/1Gi

LIMIT
limits.cpu: 0/4, limits.memory: 0/4Gi
```

- 2.4. Verify that the **review-template** project has the **name=review-template** label.

```
[student@workstation review-template]$ oc get project/review-template \
>   --show-labels
NAME          DISPLAY NAME    STATUS    LABELS
review-template          Active    name=review-template
```

- 2.5. Use the **oc new-app** command to create the **hello-secure** deployment using the **quay.io/redhattraining/hello-world-secure:v1.0** container image.

```
[student@workstation review-template]$ oc new-app --name hello-secure \
>   --docker-image quay.io/redhattraining/hello-world-secure:v1.0
...output omitted...
--> Creating resources ...
  imagestream.image.openshift.io "hello-secure" created
  deployment.apps "hello-secure" created
  service "hello-secure" created
--> Success
...output omitted...
```

- 2.6. Verify that the **hello-secure** pod does not start successfully.

```
[student@workstation review-template]$ watch oc get pods
```

Press **Ctrl+C** to end the **watch** command after the pod has a status of either **CrashLoopBackOff** or **Error**.

```
Every 2.0s: oc get pods

NAME          READY    STATUS        RESTARTS   AGE
hello-secure-6475f657c9-rmgsr  0/1     CrashLoopBackOff  1          14s
```

3. As the **developer** user, create a TLS secret using the **hello-secure-combined.pem** certificate and the **hello-secure-key.pem** key located in the **~/DO280/labs/review-template/** directory. Use the logs from the failed **hello-secure** pod to determine the expected mount point for the certificate. Mount the TLS secret as a volume in the pod using the identified directory. Verify that the **hello-secure** pod successfully redeploys.

- 3.1. Create a TLS secret using the **hello-secure-combined.pem** certificate and the **hello-secure-key.pem** key.

```
[student@workstation review-template]$ oc create secret tls hello-tls \
>   --cert hello-secure-combined.pem --key hello-secure-key.pem
secret/Hello-TLS created
```

- 3.2. Identify the name of the failed pod.

```
[student@workstation review-template]$ oc get pods
NAME          READY    STATUS        RESTARTS   AGE
hello-secure-6475f657c9-rmgsr  0/1     CrashLoopBackOff  4          2m45s
```

- 3.3. Examine the logs of the failed pod. The logs indicate that the pod attempts to use the **/run/secrets/nginx/tls.crt** file, but that the file does not exist.

```
[student@workstation review-template]$ oc logs hello-secure-6475f657c9-rmgsr
...output omitted...
nginx: [emerg] BIO_new_file("/run/secrets/nginx/tls.crt") failed (SSL:
error:02001002:system library:fopen:No such file or directory:fopen('/run/
secrets/nginx/tls.crt','r') error:2006D080:BIO routines:BIO_new_file:no such file)
```

- 3.4. Use the **oc set volumes** command to mount the secret into the **/run/secrets/nginx** directory.

```
[student@workstation review-template]$ oc set volumes deployment/hello-secure \
>   --add --type secret --secret-name hello-tls --mount-path /run/secrets/nginx
info: Generated volume name: volume-hlrltf
deployment.apps/hello-secure volume updated
```

- 3.5. Verify that the **hello-secure** pod successfully redeploys.

```
[student@workstation review-template]$ watch oc get pods
```

Press **Ctrl+C** to end the **watch** command after the pod displays **1/1** and **Running**.

```
Every 2.0s: oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hello-secure-6bd8fcccb4-nhwr2	1/1	Running	0	25s

4. The **hello-secure-combined.pem** certificate is valid for a single host name. Use the **openssl x509** command with the **-noout** and **-ext 'subjectAltName'** options to read the **hello-secure-combined.pem** certificate and identify the host name. As the **developer** user, create a passthrough route to the **hello-secure** service using the identified host name. Verify that the route responds to external requests.



Note

The **x509(1)** man page provides information on the **openssl x509** command.

- 4.1. Examine the **hello-secure-combined.pem** certificate using the **openssl x509** command.

```
[student@workstation review-template]$ openssl x509 \
>   -in hello-secure-combined.pem -noout -ext 'subjectAltName'
X509v3 Subject Alternative Name:
DNS:hello-secure.apps.ocp4.example.com
```

- 4.2. Create a passthrough route to the **hello-secure** service pointing to **hello-secure.apps.ocp4.example.com**.

```
[student@workstation review-template]$ oc create route passthrough \
>   --service hello-secure --hostname hello-secure.apps.ocp4.example.com
route.route.openshift.io/hello-secure created
```

- 4.3. Return to the `/home/student` directory.

```
[student@workstation review-template]$ cd
```

Use the `curl` command to verify that the route responds to external requests.

```
[student@workstation ~]$ curl -s https://hello-secure.apps.ocp4.example.com \
>     | grep Hello
<h1>Hello, world from nginx!</h1>
```

5. As the **developer** user, configure the **hello-secure** deployment to scale automatically. The deployment must have at least one pod running. If the average CPU utilization exceeds 80%, then the deployment scales to a maximum of five pods.



Note

You can use the script located at `~/D0280/solutions/review-template/test-hpa.sh` to test that your deployment scales as expected.

- 5.1. Use the `oc autoscale` command to create the horizontal pod autoscaler resource.

```
[student@workstation ~]$ oc autoscale deployment/hello-secure \
>     --min 1 --max 5 --cpu-percent 80
horizontalpodautoscaler.autoscaling/hello-secure autoscaled
```

- 5.2. Run the provided `test-hpa.sh` script. The script uses the `ab` command to apply load using the Apache benchmarking tool.

```
[student@workstation ~]$ ~/D0280/solutions/review-template/test-hpa.sh
...output omitted...
Benchmarking hello-secure.apps.ocp4.example.com (be patient)
Completed 10000 requests
Completed 20000 requests
...output omitted...
Finished 100000 requests
...output omitted...
```

- 5.3. Verify that at least two, but not more than five, **hello-secure** pods are running.

```
[student@workstation ~]$ oc get pods
NAME                      READY   STATUS    RESTARTS   AGE
hello-secure-6bd8fcccb4-7qjc2  1/1    Running   0          96s
hello-secure-6bd8fcccb4-d67xd  1/1    Running   0          66s
hello-secure-6bd8fcccb4-m8vxp  1/1    Running   0          96s
hello-secure-6bd8fcccb4-nhwr2  1/1    Running   0          9m36s
```

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 review-template grade
```

Finish

As the **student** user on the **workstation** machine, use the **lab** command to complete this exercise.

```
[student@workstation ~]$ lab review-template finish
```

This concludes the comprehensive review.

Appendix A

Creating a GitHub Account

Goal

Describe how to create a GitHub account for labs in the course.

Creating a GitHub Account

Objectives

After completing this section, you should be able to create a GitHub account and create public Git repositories for the labs in the course.

Creating a GitHub Account

You need a GitHub account to create one or more *public* Git repositories for the labs in this course. If you already have a GitHub account, you can skip the steps listed in this appendix.



Important

If you already have a GitHub account, ensure that you only create *public* Git repositories for the labs in this course. The lab grading scripts and instructions require unauthenticated access to clone the repository. The repositories must be accessible without providing passwords, SSH keys, or GPG keys.

To create a new GitHub account, perform the following steps:

1. Navigate to <https://github.com> using a web browser.
2. Enter the required details and then click **Sign up for GitHub**.

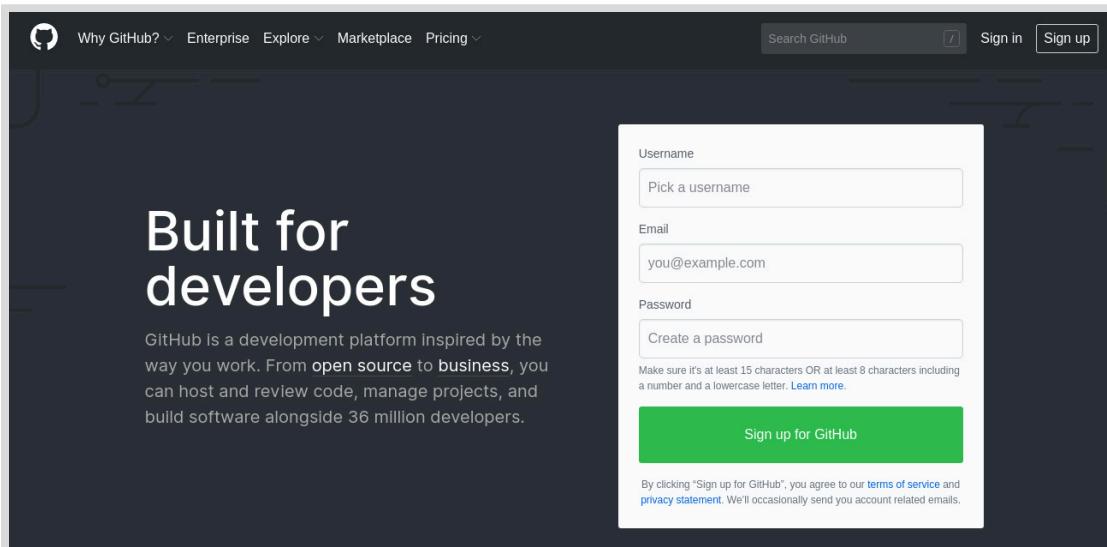


Figure A.1: Creating a GitHub account

3. You will receive an email with instructions on how to activate your GitHub account. Verify your email address and then sign in to the GitHub website using the username and password you provided during account creation.
4. After you have logged in to GitHub, you can create new Git repositories by clicking **New** in the **Repositories** pane on the left of the GitHub home page.

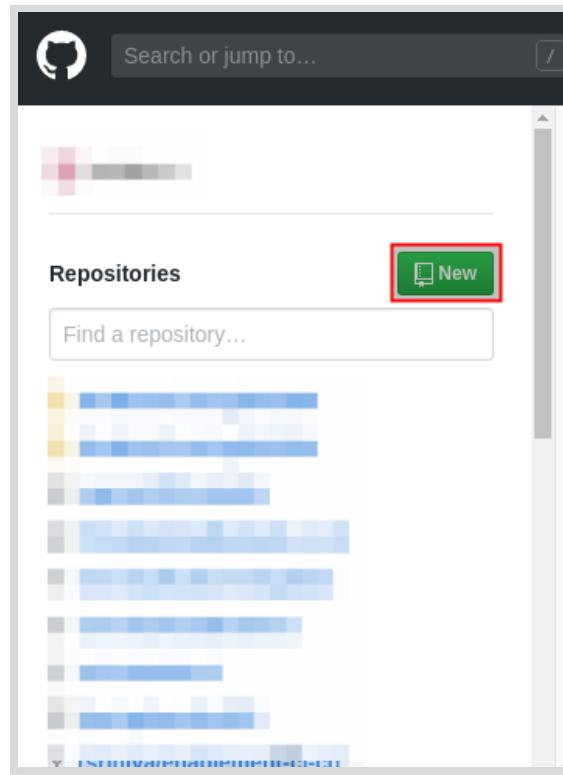


Figure A.2: Creating a new Git repository

Alternatively, click the plus icon (+) in the upper-right corner (to the right of the bell icon) and then click **New repository**.

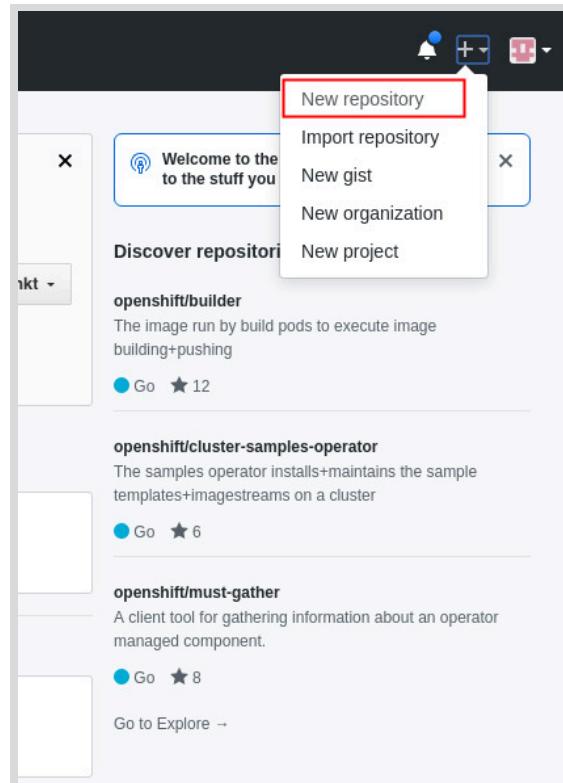


Figure A.3: Creating new Git repository



References

Signing up for a new GitHub account

<https://help.github.com/en/articles/signing-up-for-a-new-github-account>

Appendix B

Creating a Quay Account

Goal

Describe how to create a Quay account for labs in the course.

Creating a Quay Account

Objectives

After completing this section, you should be able to create a Quay account and create public container image repositories for the labs in the course.

Creating a Quay Account

You need a Quay account to create one or more *public* container image repositories for the labs in this course. If you already have a Quay account, you can skip the steps to create a new account listed in this appendix.



Important

If you already have a Quay account, ensure that you only create *public* container image repositories for the labs in this course. The lab grading scripts and instructions require unauthenticated access to pull container images from the repository.

To create a new Quay account, perform the following steps:

1. Navigate to <https://quay.io> using a web browser.
2. Click **Sign in** in the upper-right corner (next to the search bar).
3. On the **Sign in** page, you can log in using your Google or GitHub credentials (created in Appendix A).

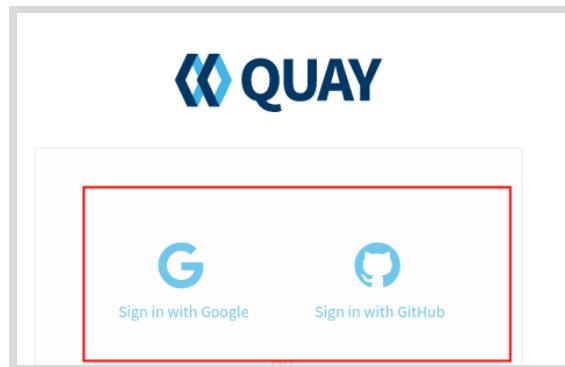


Figure B.1: Sign in using Google or GitHub credentials.

Alternatively, click **Create Account** to create a new account.

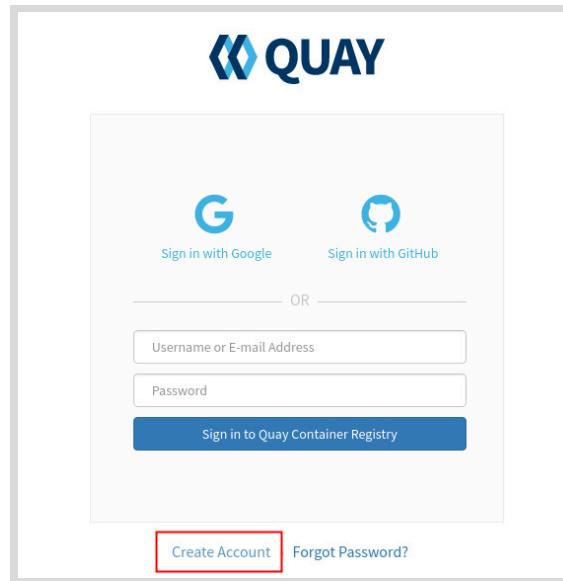


Figure B.2: Creating a new account

4. If you chose to skip the Google or GitHub log-in method and instead opted to create a new account, you will receive an email with instructions on how to activate your Quay account. Verify your email address and then sign in to the Quay website with the username and password you provided during account creation.
5. After you have logged in to Quay you can create new image repositories by clicking **Create New Repository** on the **Repositories** page.

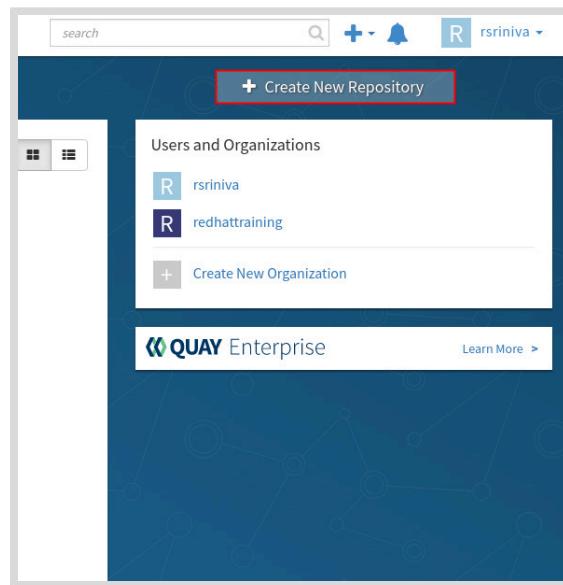


Figure B.3: Creating a new image repository

Alternatively, click the plus icon (+) in the upper-right corner (to the left of the bell icon), and then click **New Repository**.

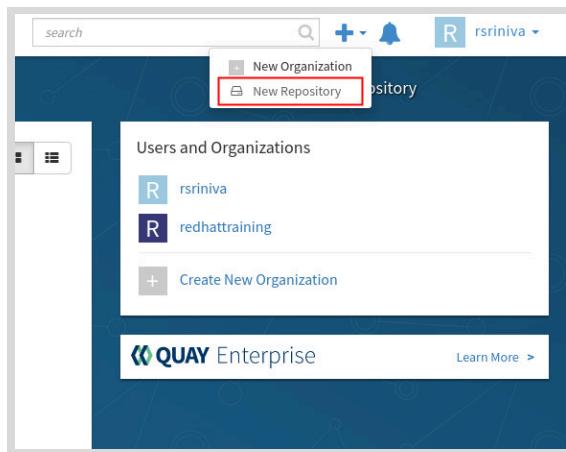


Figure B.4: Creating a new image repository



References

Getting Started with Quay.io

<https://docs.quay.io/solution/getting-started.html>

Repositories Visibility

Objectives

After completing this section, you should be able to control repository visibility on Quay.io.

Quay.io Repositories Visibility

Quay.io offers the possibility of creating public and private repositories. Public repositories can be read by anyone without restrictions, despite write permissions must be explicitly granted. Private repositories have both read and write permissions restricted. Nevertheless, the number of private repositories in quay . io is limited depending on the namespace's plan.

Default Repository Visibility

Repositories created by pushing images to quay . io are private by default. In order OpenShift (or any other tool) to fetch those images you can either configure a private key in both OpenShift and Quay, or make the repository public, so no authentication is required. Setting up private keys is out of scope of this document.

The screenshot shows the Red Hat Quay.io interface. At the top, there are navigation icons (back, forward, search) and a URL bar showing <https://quay.io/repository/>. The header includes the Red Hat Quay.io logo, EXPLORE, APPLICATIONS, REPOSITORIES (which is highlighted in red), and TUTORIAL. Below the header is a dark banner with the word "Repositories". The main content area is titled "Starred" with a yellow star icon. It displays the message "You haven't starred any repositories yet." and a sub-instruction "Stars allow you to easily access your favorite repositories." Below this, there is a section titled "RHT_OCP4_QUAY_USER" with a small brown square icon. It lists four repositories in a grid:

Repository	Action
do180-mysql-57-rhel7	star
do180-todonodejs	star
do180-quote-php	star
nexus	star

Updating Repository Visibility

In order to set repository visibility to public select the appropriate repository in <https://quay.io/repository/> (log in to your account if needed) and open the **Settings** page by clicking on the gear on the left-bottom edge. Scroll down to the **Repository Visibility** section and click on the **Make Public** button.

Appendix B | Creating a Quay Account

The screenshot shows the Quay.io repository settings interface. At the top, it says "Trust Disabled" with a gear icon and a button to "Enable Trust". Below that is the "Events and Notifications" section, which indicates "No notifications have been setup for this repository." and a link to "Create Notification". The "Repository Visibility" section shows the repository is "private" with a lock icon, and a button to "Make Public". The "Delete Repository" section contains a warning message: "Deleting a Repository cannot be undone. Here be dragons!" and a red "Delete Repository" button.

Get back the the list of repositories. The lock icon besides the repository name have disappeared, indicating the repository became public.

Appendix C

Useful Git Commands

Goal

Describe useful Git commands that are used for the labs in this course.

Git Commands

Objectives

After completing this section, you should be able to restart and redo exercises in this course. You should also be able to switch from one incomplete exercise to perform another, and later continue the previous exercise where you left off.

Working with Git Branches

This course uses a Git repository hosted on GitHub to store the application course code source code. At the beginning of the course, you create your own fork of this repository, which is also hosted on GitHub.

During this course, you work with a local copy of your fork, which you clone to the **workstation** VM. The term *origin* refers to the remote repository from which a local repository is cloned.

As you work through the exercises in the course, you use separate Git branches for each exercise. All changes you make to the source code happen in a new branch that you create only for that exercise. Never make any changes on the **master** branch.

A list of scenarios and the corresponding Git commands that you can use to work with branches, and to recover to a known good state are listed below.

Redoing an Exercise from Scratch

To redo an exercise from scratch after you have completed it, perform the following steps:

1. You commit and push all the changes in your local branch as part of performing the exercise. You finished the exercise by running its **finish** subcommand to clean up all resources:

```
[student@workstation ~]$ lab your-exercise finish
```

2. Change to your local clone of the **D0180-apps** repository and switch to the **master** branch:

```
[student@workstation ~]$ cd ~/D0180-apps  
[student@workstation D0180-apps]$ git checkout master
```

3. Delete your local branch:

```
[student@workstation D0180-apps]$ git branch -d your-branch
```

4. Delete the remote branch on your personal GitHub account:

```
[student@workstation D0180-apps]$ git push origin --delete your-branch
```

5. Use the **start** subcommand to restart the exercise:

```
[student@workstation D0180-apps]$ cd ~  
[student@workstation ~]$ lab your-exercise start
```

Abandoning a Partially Completed Exercise and Restarting it from Scratch

You may run into a scenario where you have partially completed a few steps in the exercise, and you want to abandon the current attempt, and restart it from scratch. Perform the following steps:

1. Run the exercise's **finish** subcommand to clean up all resources.

```
[student@workstation ~]$ lab your-exercise finish
```

2. Enter your local clone of the **D0180-apps** repository and discard any pending changes on the current branch using **git stash**:

```
[student@workstation ~]$ cd ~/D0180-apps  
[student@workstation D0180-apps]$ git stash
```

3. Switch to the **master** branch of your local repository:

```
[student@workstation D0180-apps]$ git checkout master
```

4. Delete your local branch:

```
[student@workstation D0180-apps]$ git branch -d your-branch
```

5. Delete the remote branch on your personal GitHub account:

```
[student@workstation D0180-apps]$ git push origin --delete your-branch
```

6. You can now restart the exercise by running its **start** subcommand:

```
[student@workstation D0180-apps]$ cd ~  
[student@workstation ~]$ lab your-exercise start
```

Switching to a Different Exercise from an Incomplete Exercise

You may run into a scenario where you have partially completed a few steps in an exercise, but you want to switch to a different exercise, and revisit the current exercise at a later time.

Avoid leaving too many exercises uncompleted to revisit later. These exercises tie up cloud resources and you may use up your allotted quota on the cloud provider and on the OpenShift cluster you share with other students. If you think it may be a while until you can go back to the current exercise, consider abandoning it and later restarting from scratch.

If you prefer to pause the current exercise and work on the next one, perform the following steps:

1. Commit any pending changes in your local repository and push them to your personal GitHub account. You may want to record the step where you stopped the exercise:

```
[student@workstation ~]$ cd ~/D0180-apps  
[student@workstation D0180-apps]$ git commit -a -m 'Paused at step X.Y'  
[student@workstation D0180-apps]$ git push
```

2. Do not run the **finish** command of the original exercise. This is important to leave your existing OpenShift projects unchanged, so you can resume later.
3. Start the next exercise by running its **start** subcommand:

```
[student@workstation ~]$ lab your-exercise start
```

4. The next exercise switches to the **master** branch and optionally creates a new branch for its changes. This means the changes made to the original exercise in the original branch are left untouched.
5. Later, after you have completed the next exercise, and you want to go back to the original exercise, switch back to its branch:

```
[student@workstation ~]$ git checkout original-branch
```

Then you can continue with the original exercise at the step where you left off.



References

Git branch man page

<https://git-scm.com/docs/git-branch>

What is a Git branch?

<https://git-scm.com/book/en/v2/Git-Branching-Banches-in-a-Nutshell>

Git Tools - Stashing

<https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning>

Appendix D

Scaling an OpenShift Cluster

Goal

Control the size of an OpenShift cluster.

Objectives

- Describe resources used for scaling an OpenShift cluster, and identify methods to manually scale compute nodes.
- Define parameters to automatically control the size of a cluster.

Sections

- Manually Scaling an OpenShift Cluster
- Automatically Scaling an OpenShift Cluster

Manually Scaling an OpenShift Cluster

Objectives

After completing this section, you should be able to describe resources used for scaling an OpenShift cluster, and identify methods to manually scale compute nodes.

Introducing the Machine API

OpenShift Container Platform can scale the cluster by adding or removing compute nodes through the Machine API. These scaling capabilities allow the cluster to provide enough computing power for your applications. The scaling 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 compute nodes.



Important

The machine API operator must be operational in order to perform either manual or automatic scaling. The machine API operator requires cloud provider integration and is available to clusters installed with the full-stack automation method.

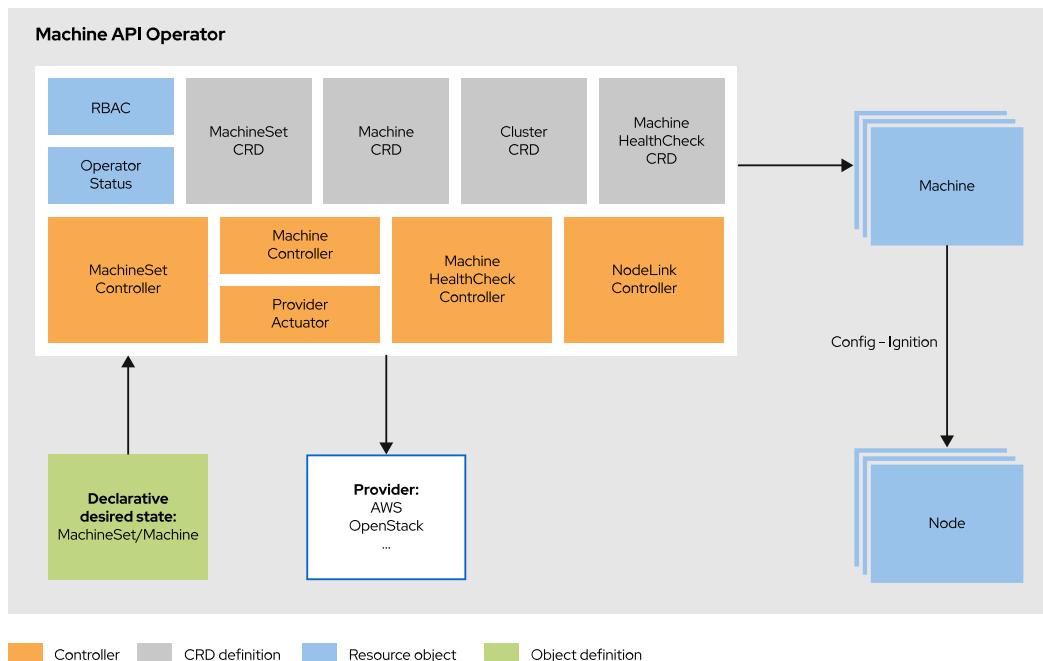


Figure D.1: The Machine API operator

**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.

The Machine API provides the following custom resources:

Machines

Machines are the fundamental compute unit in a cluster. Each machine resource correlates to a physical or virtual node. A machine is an instance from a machine set that inherits information defined in the machine set.

MachineSets

A **MachineSet** describes a group of machines, but does not include control plane nodes. Machine sets are to machines what replica sets are to pods. A machine set defines settings that apply to all machines in the machine set. These settings include labels, such as for regions and zones, and an instance type, such as the AWS **m4.xlarge** instance type. You can scale the number of machines in a machine set either by modifying the **replicas** setting in the machine set resource or by running the **oc scale** command.

Like replica sets, machine sets use labels to determine their members. The machine API uses this information to determine if the machine set has the correct number of machine replicas.

Use machine sets to customize your cluster topology. For example, you might define a unique machine set per region. An OpenShift cluster deployed to AWS creates one machine set per availability zone.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: ocp-xxxy-worker-us-west-2b
  ...output omitted...
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: ocp-rnh2q
      machine.openshift.io/cluster-api-machineset: ocp-xxxy-worker-us-west-2b
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: ocp-xxxy
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: ocp-xxxy-worker-us-west-2b
    spec:
      providerSpec:
  ...output omitted...
      placement:
        availabilityZone: us-west-2b
        region: us-west-2
        credentialsSecret:
```

```
name: aws-cloud-credentials
instanceType: m4.xlarge
...output omitted...
```

**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.

MachineAutoscalers

MachineAutoscalers define the machine sets that have the ability to scale automatically. Machine autoscalers are not required for manual scaling.

ClusterAutoscaler

A **ClusterAutoscaler** resource enables the automatic scaling of machines in a cloud deployment. Automatic scaling ensures that if a compute node becomes unresponsive, then pods are quickly evacuated to another compute node for improved availability. Automatic scaling is useful when the cluster is under stress, or for workloads whose computing requirements change.

The cluster autoscaler resource allows you to specify limits for a variety of components, such as nodes, cores, and memory. A cluster autoscaler is not required for manual scaling.

**Note**

The following section discusses both the **MachineAutoscalers** and **ClusterAutoscaler** resources.

MachineHealthChecks

A **MachineHealthCheck** verifies the health of a machine (such as a compute node) and takes action if the resource is unhealthy.

Manually Scaling Compute Nodes

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 method. Reasons to scale a cluster include minimizing resource congestion and better management of resources, such as dedicating a set of compute nodes to a particular workload.

Like deployments and deployment configurations, you change the number of instances by adjusting the replicas specified by the machine set resource. Modify the **replicas** line in the machine set resource or use the **oc scale** command.

```
[user@demo ~]$ oc scale --replicas 2 \
>   machineset MACHINE-SET -n openshift-machine-api
```

If the number of machines in the machine set does not match the replica count, then OpenShift either adds or removes machines until the desired replica count is reached.

Scaling down the machine set instructs the scheduler to evacuate pods running on the machine targeted for removal. Reducing the machine replica count only works if the remaining compute nodes have enough capacity to run the pods. If there are not enough resources, then the controller refuses to reduce the machine replica count.



References

For more information on manually scaling compute nodes, refer to the *Manually scaling a MachineSet* chapter in the Red Hat OpenShift Container Platform 4.5 *Machine management* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/machine_management/index#manually-scaling-machineset

Automatically Scaling an OpenShift Cluster

Objectives

After completing this section, you should be able to define parameters to automatically control the size of a cluster.

Automatically Scaling a 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**.



Important

The machine API operator must be operational in order to perform either manual or automatic scaling. The machine API operator requires cloud provider integration and is available to clusters installed with the full-stack automation method.

A machine autoscaler resource automatically scales the number of replicas in a machine set, depending on the load. This API resource interacts with machine sets and instructs them to add or remove compute nodes. The machine autoscaler resource supports the definition of lower and upper boundaries. The cluster autoscaler resource enforces limits for the whole cluster, such as the total number of nodes. For example, **MaxNodesTotal** sets the maximum number of nodes in the whole cluster, and **MaxMemoryTotal** sets the maximum memory in the whole cluster. If desired, then configure the cluster autoscaler to scale down the number of nodes as load decreases.

Each OpenShift cluster can only have one cluster autoscaler resource. The cluster autoscaler 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). These limits prevent machine autoscaler 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 cluster autoscaler 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 machine autoscaler resource to scale the number of machines defined by a machine set resource. Scaling only works if adding a new machine does not exceed any of the values defined by the cluster autoscaler 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 machine autoscaler does not increase or decrease the number of machines unless cluster autoscaling is allowed.

The following diagram shows how the machine autoscaler resource interacts with machine sets to scale the number of compute nodes.

- If the machine autoscaler resource must scale, then it checks to see if a cluster autoscaler resource exists. If it does not, then no scaling occurs.

If a cluster autoscaler resource does exist, then the machine autoscaler resource evaluates whether adding the new machine violates any of the limits defined in the cluster autoscaler resource.

- Provided the request does not exceed the limits, a new machine is created.
- When the new machine is ready, OpenShift schedules pods to it as a new node.

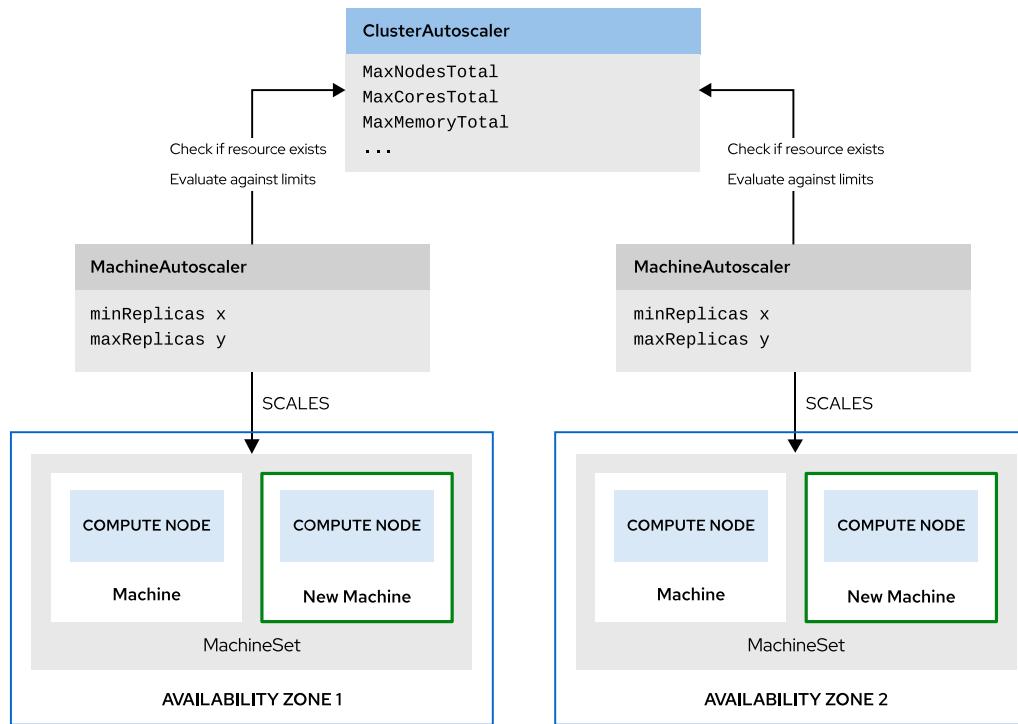


Figure D.2: Dynamic scaling with full-stack automation

There is a one-to-one mapping between a machine autoscaler and a machine set resource. This mapping allows each machine autoscaler resource to manage its own machine set. If you want a machine set to have the ability to scale, then create a machine autoscaler resource for it. Alternatively, do not create machine autoscaler resources for machine sets that should not scale.

In a default full-stack automation deployment, each machine set resource maps to an availability zone, but this design is arbitrary. For example, you can use a machine set resource to separate compute nodes based on their disk speeds or functional roles.

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 compute nodes.
- A cluster autoscaler resource, if the infrastructure supports it. Additionally, the cluster autoscaler resource might limit the maximum number of nodes, and define minimum and maximum values for cores, memory, and GPUs. The **enabled: true** entry in the **scaleDown** section of the cluster autoscaler resource authorizes this resource to automatically scale down the number of machines when they are not used.
- At least one machine autoscaler resource. Each machine autoscaler resource defines the minimum and maximum number of replicas for a specific machine set.

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
  name: "MACHINE-AUTOSCALER-NAME"
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1
  maxReplicas: 4
  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 compute nodes is triggered.

The cluster autoscaler resource scales out the number of compute nodes 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 automatically scaling a cluster, refer to the *Applying autoscaling to an OpenShift Container Platform cluster* chapter in the Red Hat OpenShift Container Platform 4.5 Machine management documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html-single/machine_management/index#applying-autoscaling

Scaling an OpenShift 4 Cluster

<https://github.com/openshift/training/blob/master/docs/04-scaling-cluster.md>

Summary

In this appendix, you learned:

- OpenShift Container Platform provides mechanisms for automatically increasing and decreasing the number of compute nodes in your cluster. 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 nodes.
- When scaling a cluster automatically, you interact with the **MachineAutoscaler** resource that 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.