



Red Hat Training and Certification

Student Workbook (ROLE)

OCP 4.14 DO280

Red Hat OpenShift Administration II: Configuring a Production Cluster

Edition 2



Join a community dedicated to learning open source

The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.



Network with tens of thousands of community members



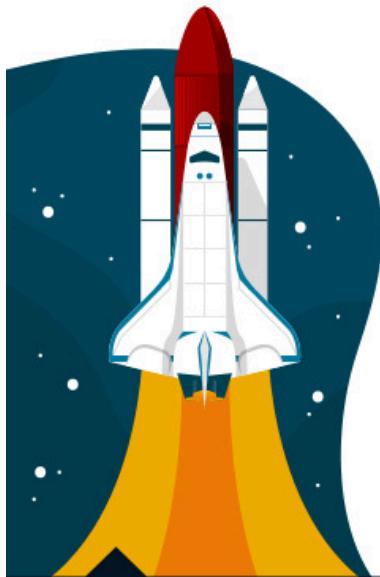
Engage in thousands of active conversations and posts



Join and interact with hundreds of certified training instructors



Unlock badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

Access free Red Hat training videos

Discover the latest Red Hat Training and Certification news

Connect with your instructor - and your classmates - before, after, and during your training course.

Join peers as you explore Red Hat products

Join the conversation learn.redhat.com



Copyright © 2020 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Red Hat logo, and Ansible are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Red Hat OpenShift Administration II: Configuring a Production Cluster



OCP 4.14 DO280
Red Hat OpenShift Administration II: Configuring a Production Cluster
Edition 2 20240725
Publication date 20240725

Authors: Alejandro Serna-Borja, Alex Corcoles, Andrés Hernández, Austin Garrigus, Bernardo Gargallo Jaquotot, Tayler Geiger, Manna Kong, Michael Jarrett, Maria Ordóñez, Harpal Singh, Randy Thomas
Course Architect: Fernando Lozano
DevOps Engineer: Benjamin Chardi Marco
Editor: Julian Cable

© 2024 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are © 2024 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 [mailto: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, OpenShift, Fedora, Hibernate, Ansible, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries 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 American, Inc. and/or its affiliates.

XFS® is a registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is a trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

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: Jordi Sola, Nikki Lucas, Natalie Watkins

Document Conventions	ix
Admonitions	ix
Inclusive Language	x
Introduction	xi
Red Hat OpenShift Administration II: Configuring a Production Cluster	xi
Orientation to the Classroom Environment	xii
Performing Lab Exercises	xx
1. Declarative Resource Management	1
Resource Manifests	2
Guided Exercise: Resource Manifests	13
Kustomize Overlays	21
Guided Exercise: Kustomize Overlays	33
Lab: Declarative Resource Management	44
Summary	54
2. Deploy Packaged Applications	55
OpenShift Templates	56
Guided Exercise: OpenShift Templates	63
Helm Charts	68
Guided Exercise: Helm Charts	74
Lab: Deploy Packaged Applications	80
Summary	91
3. Authentication and Authorization	93
Configure Identity Providers	94
Guided Exercise: Configure Identity Providers	101
Define and Apply Permissions with RBAC	113
Guided Exercise: Define and Apply Permissions with RBAC	117
Lab: Authentication and Authorization	123
Summary	132
4. Network Security	133
Protect External Traffic with TLS	134
Guided Exercise: Protect External Traffic with TLS	139
Configure Network Policies	151
Guided Exercise: Configure Network Policies	156
Protect Internal Traffic with TLS	165
Guided Exercise: Protect Internal Traffic with TLS	169
Lab: Network Security	175
Summary	187
5. Expose non-HTTP/SNI Applications	189
Load Balancer Services	190
Guided Exercise: Load Balancer Services	193
Multus Secondary Networks	200
Guided Exercise: Multus Secondary Networks	205
Lab: Expose non-HTTP/SNI Applications	214
Summary	223
6. Enable Developer Self-Service	225
Project and Cluster Quotas	226
Guided Exercise: Project and Cluster Quotas	235
Per-Project Resource Constraints: Limit Ranges	242
Guided Exercise: Per-Project Resource Constraints: Limit Ranges	247
The Project Template and the Self-Provisioner Role	258
Guided Exercise: The Project Template and the Self-Provisioner Role	265

Lab: Enable Developer Self-Service	274
Summary	283
7. Manage Kubernetes Operators	285
Kubernetes Operators and the Operator Lifecycle Manager	286
Quiz: Kubernetes Operators and the Operator Lifecycle Manager	290
Install Operators with the Web Console	292
Guided Exercise: Install Operators with the Web Console	297
Install Operators with the CLI	308
Guided Exercise: Install Operators with the CLI	315
Lab: Manage Kubernetes Operators	324
Summary	332
8. Application Security	333
Control Application Permissions with Security Context Constraints	334
Guided Exercise: Control Application Permissions with Security Context Constraints	337
Allow Application Access to Kubernetes APIs	341
Guided Exercise: Allow Application Access to Kubernetes APIs	346
Cluster and Node Maintenance with Kubernetes Cron Jobs	351
Guided Exercise: Cluster and Node Maintenance with Kubernetes Cron Jobs	358
Lab: Application Security	366
Summary	374
9. OpenShift Updates	375
The Cluster Update Process	376
Quiz: The Cluster Update Process	387
Detect Deprecated Kubernetes API Usage	389
Quiz: Detect Deprecated Kubernetes API Usage	396
Update Operators with the OLM	398
Quiz: Update Operators with the OLM	403
Quiz: OpenShift Updates	405
Summary	409
10. Comprehensive Review	411
Comprehensive Review	412
Lab: Cluster Self-service Setup	414
Lab: Secure Applications	431
Lab: Deploy Packaged Applications	446

Document Conventions

This section describes various conventions and practices that are used throughout all Red Hat Training courses.

Admonitions

Red Hat Training courses use the following admonitions:



References

These describe where to find external documentation that is 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 something that makes your life easier.



Important

Important sections provide details of information that is easily missed: configuration changes that apply only to the current session, or services that need restarting before an update applies. Ignoring these admonitions will not cause data loss, but might cause irritation and frustration.



Warning

Do not ignore warnings. Ignoring these admonitions will most likely cause data loss.

Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services that are covered in Red Hat Training courses. Red Hat appreciates your patience during this process.

Introduction

Red Hat OpenShift Administration II: Configuring a Production Cluster

This course prepares a senior OpenShift Cluster Administrator to perform daily administration tasks on clusters that host applications that internal teams and external vendors provide; enable self-service for cluster users with different roles; and deploy applications that require special permissions, such as CI/CD tooling, performance monitoring, and security scanners.

DO280 focuses on configuring multi-tenancy and security features of OpenShift. DO280 also teaches how to manage OpenShift add-ons based on operators. This course is based on Red Hat® OpenShift® Container Platform 4.14.

Course Objectives

- Configure and manage OpenShift clusters to maintain security and reliability across multiple applications and development teams.
- Configure authentication, authorization, and resource quotas.
- Protect network traffic with network policies and TLS security (HTTPS).
- Expose applications by using protocols other than HTTP and TLS, and attach applications to multi-homed networks.
- Manage OpenShift cluster updates and Kubernetes operator updates.
- This course, together with the *Red Hat OpenShift I: Containers & Kubernetes* (DO180) course, prepares the student to take the *Red Hat Certified Specialist in OpenShift Administration* exam (EX280).

Audience

- System Administrators interested in the ongoing management of OpenShift clusters, applications, users, and add-ons.
- Site Reliability Engineers interested in the ongoing maintenance and troubleshooting of Kubernetes clusters.
- System and Software Architects interested in understanding the security of an OpenShift cluster.

Prerequisites

- Red Hat System Administration I (RH124)*, or equivalent skills in managing Linux systems and servers from the Bash shell.
- Red Hat OpenShift I: Containers & Kubernetes* (DO180 v4.14), or equivalent skills in deploying and managing Kubernetes applications by using the OpenShift web console and command-line interfaces.

Orientation to the Classroom Environment

In this course, the main computer system that is used for hands-on learning activities is **workstation**. The systems called **bastion** and **classroom** must always be running for proper use of the lab environment.

These three systems are in the `lab.example.com` DNS domain.

A Red Hat OpenShift Container Platform (RHOCP) 4.14 single-node (SNO) bare metal User Provisioned Infrastructure (UPI) installation is used in this classroom. Infrastructure systems for the RHOCP cluster are in the `ocp4.example.com` DNS domain.

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

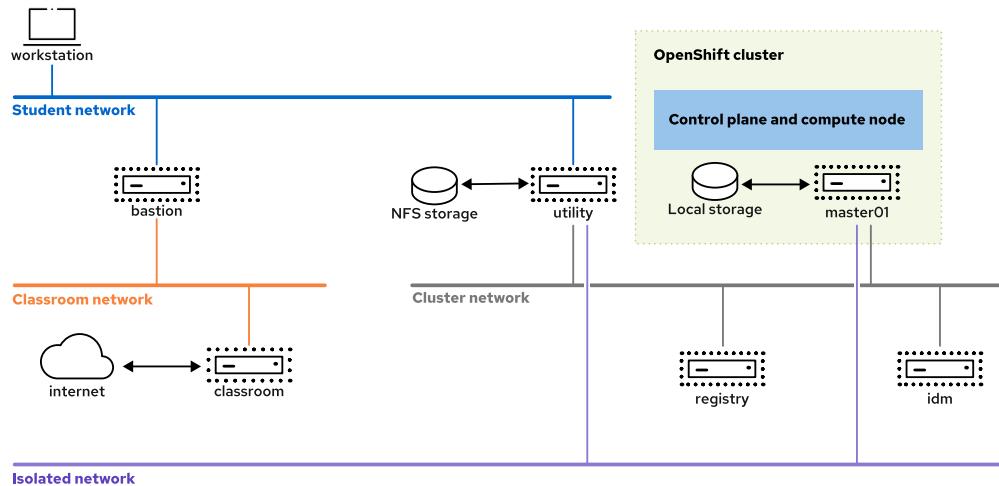


Figure 0.1: Classroom environment

Classroom Machines

Machine name	IP addresses	Role
<code>bastion.lab.example.com</code>	172.25.250.254	Router that links VMs to central servers
<code>classroom.lab.example.com</code>	172.25.252.254	Server that hosts the required classroom materials
<code>idm.ocp4.example.com</code>	192.168.50.40	Identity management server for cluster authentication and authorization support
<code>master01.ocp4.example.com</code>	192.168.50.10	An RHOCP single-node (SNO) cluster

Machine name	IP addresses	Role
registry.ocp4.example.com	192.168.50.50	Registry server to provide a private registry and GitLab services to the cluster
utility.lab.example.com	192.168.50.254	Server that provides supporting services that the RHOCP cluster requires, including DHCP, NFS, and routing to the cluster network
workstation.lab.example.com	172.25.250.9	Graphical workstation that students use

The primary function of **bastion** is to act as a router between the network that connects the student machines and the classroom network. If **bastion** is down, then other student machines do not function properly, or might even hang during boot.

The **utility** system acts as a router between the network that connects the RHOCP cluster machines and the student network. If **utility** is down, then the RHOCP cluster does not function properly, or might even hang during boot.

For some exercises, the classroom contains an isolated network. Only the **utility** system and the cluster are connected to this network.

Several systems in the classroom provide supporting services. The **classroom** server hosts software and lab materials for the hands-on activities. The **registry** server is a private Red Hat Quay container registry that hosts the container images for the hands-on activities. Information about how to use these servers is provided in the instructions for those activities.

The **master01** system serves as the control plane and compute node for the RHOCP cluster. The cluster uses the **registry** system as its own private container image registry and GitLab server. The **idm** system provides LDAP services to the RHOCP cluster for authentication and authorization support.

Students use the **workstation** machine to access a dedicated RHOCP cluster, for which they have cluster administrator privileges.

RHOCP Access Methods

Access method	Endpoint
Web console	https://console-openshift-console.apps.ocp4.example.com
API	https://api.ocp4.example.com:6443

The RHOCP cluster has a standard user account, **developer**, which has the **developer** password. The administrative account, **admin**, has the **redhatocp** password.

Classroom Registry

The DO280 course uses a private Red Hat Quay container image registry that is accessible only within the classroom environment. The container image registry hosts the container images that students use in the hands-on activities. By using a private container image registry, the classroom environment is self-contained to not require internet access.

The registry server provides the `https://registry.ocp4.example.com:8443/` container image registry to the classroom environment. The registry is configured with a user account, developer, which has the developer password.

The following table provides the container image repositories that are used in this course and their public repositories.

Classroom Container Image Repositories and Public Sources

Public Source Repository	Classroom Registry Repository
<code>quay.io/jkube/jkube-java-binary-s2i:0.0.9</code>	<code>registry.ocp4.example.com:8443/jkube/jkube-java-binary-s2i:0.0.9</code>
<code>quay.io/openshift/origin-cli:4.12</code>	<code>registry.ocp4.example.com:8443/openshift/origin-cli:4.12</code>
<code>quay.io/redhattraining/books:v1.4</code>	<code>registry.ocp4.example.com:8443/redhattraining/books:v1.4</code>
<code>quay.io/redhattraining/builds-for-managers</code>	<code>registry.ocp4.example.com:8443/redhattraining/builds-for-managers</code>
<code>quay.io/redhattraining/do280-beeper-api:1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-beeper-api:1.0</code>
<code>quay.io/redhattraining/do280-payroll-api:1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-payroll-api:1.0</code>
<code>quay.io/redhattraining/do280-product:1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-product:1.0</code>
<code>quay.io/redhattraining/do280-product-stock:1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-product-stock:1.0</code>
<code>quay.io/redhattraining/do280-project-cleaner:v1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-project-cleaner:v1.0</code>
<code>quay.io/redhattraining/do280-project-cleaner:v1.1</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-project-cleaner:v1.1</code>
<code>quay.io/redhattraining/do280-show-config-app:1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-show-config-app:1.0</code>
<code>quay.io/redhattraining/do280-stakater-reloader:v0.0.125</code>	<code>registry.ocp4.example.com:8443/redhattraining/do280-stakater-reloader:v0.0.125</code>
<code>quay.io/redhattraining/exoplanets:v1.0</code>	<code>registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.0</code>

Public Source Repository	Classroom Registry Repository
quay.io/redhattraining/famous-quotes:2.1	registry.ocp4.example.com:8443/redhattraining/famous-quotes:2.1
quay.io/redhattraining/famous-quotes:latest	registry.ocp4.example.com:8443/redhattraining/famous-quotes:latest
quay.io/redhattraining/gitlab-ce:8.4.3-ce.0	registry.ocp4.example.com:8443/redhattraining/gitlab-ce:8.4.3-ce.0
quay.io/redhattraining/hello-world-nginx:latest	registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:latest
quay.io/redhattraining/hello-world-nginx:v1.0	registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0
quay.io/redhattraining/loadtest:v1.0	registry.ocp4.example.com:8443/redhattraining/loadtest:v1.0
quay.io/redhattraining/php-hello-dockerfile	registry.ocp4.example.com:8443/redhattraining/php-hello-dockerfile
quay.io/redhattraining/php-ssl:v1.0	registry.ocp4.example.com:8443/redhattraining/php-ssl:v1.0
quay.io/redhattraining/php-ssl:v1.1	registry.ocp4.example.com:8443/redhattraining/php-ssl:v1.1
quay.io/redhattraining/scaling:v1.0	registry.ocp4.example.com:8443/redhattraining/scaling:v1.0
quay.io/redhattraining/todo-angular:v1.1	registry.ocp4.example.com:8443/redhattraining/todo-angular:v1.1
quay.io/redhattraining/todo-angular:v1.2	registry.ocp4.example.com:8443/redhattraining/todo-angular:v1.2
quay.io/redhattraining/todo-backend:release-46	registry.ocp4.example.com:8443/redhattraining/todo-backend:release-46
quay.io/redhattraining/do280-roster:v1	registry.ocp4.example.com:8443/redhattraining/do280-roster:v1
quay.io/redhattraining/do280-roster:v2	registry.ocp4.example.com:8443/redhattraining/do280-roster:v2
quay.io/redhattraining/wordpress:5.7-php7.4-apache	registry.ocp4.example.com:8443/redhattraining/wordpress:5.7-php7.4-apache
registry.access.redhat.com/rhscl/httpd-24-rhel7:latest	registry.ocp4.example.com:8443/rhscl/httpd-24-rhel7:latest

Public Source Repository	Classroom Registry Repository
registry.access.redhat.com/rhscl/mysql-57-rhel7:latest	registry.ocp4.example.com:8443/rhscl/mysql-57-rhel7:latest
registry.access.redhat.com/rhscl/nginx-18-rhel7:latest	registry.ocp4.example.com:8443/rhscl/nginx-18-rhel7:latest
registry.access.redhat.com/rhscl/nodejs-6-rhel7:latest	registry.ocp4.example.com:8443/rhscl/nodejs-6-rhel7:latest
registry.access.redhat.com/rhscl/php-72-rhel7:latest	registry.ocp4.example.com:8443/rhscl/php-72-rhel7:latest
registry.access.redhat.com/ubi7/nginx-118:latest	registry.ocp4.example.com:8443/ubi7/nginx-118:latest
registry.access.redhat.com/ubi8/httpd-24:latest	registry.ocp4.example.com:8443/ubi8/httpd-24:latest
registry.access.redhat.com/ubi8:latest/	registry.ocp4.example.com:8443/ubi8:latest/
registry.access.redhat.com/ubi8/nginx-118:latest	registry.ocp4.example.com:8443/ubi8/nginx-118:latest
registry.access.redhat.com/ubi8/nodejs-10:latest	registry.ocp4.example.com:8443/ubi8/nodejs-10:latest
registry.access.redhat.com/ubi8/nodejs-16:latest	registry.ocp4.example.com:8443/ubi8/nodejs-16:latest
registry.access.redhat.com/ubi8/php-72:latest	registry.ocp4.example.com:8443/ubi8/php-72:latest
registry.access.redhat.com/ubi8/php-73:latest	registry.ocp4.example.com:8443/ubi8/php-73:latest
registry.access.redhat.com/ubi8/ubi:8.0	registry.ocp4.example.com:8443/ubi8/ubi:8.0
registry.access.redhat.com/ubi8/ubi:8.4	registry.ocp4.example.com:8443/ubi8/ubi:8.4
registry.access.redhat.com/ubi8/ubi:latest	registry.ocp4.example.com:8443/ubi8/ubi:latest
registry.access.redhat.com/ubi9/httpd-24:latest	registry.ocp4.example.com:8443/ubi9/httpd-24:latest
registry.access.redhat.com/ubi9/nginx-120:latest	registry.ocp4.example.com:8443/ubi9/nginx-120:latest
registry.access.redhat.com/ubi9/ubi:latest	registry.ocp4.example.com:8443/ubi9/ubi:latest

Public Source Repository	Classroom Registry Repository
registry.redhat.io/redhat-openjdk-18/openjdk18-openshift:1.8	registry.ocp4.example.com:8443/redhat-openjdk-18/openjdk18-openshift:1.8
registry.redhat.io/redhat-openjdk-18/openjdk18-openshift:latest	registry.ocp4.example.com:8443/redhat-openjdk-18/openjdk18-openshift:latest
registry.redhat.io/rhel8/mysql-80:1-211.1664898586	registry.ocp4.example.com:8443/rhel8/mysql-80:1-211.1664898586
registry.redhat.io/rhel8/mysql-80:latest	registry.ocp4.example.com:8443/rhel8/mysql-80:latest
registry.redhat.io/rhel8/postgresql-13:1-7	registry.ocp4.example.com:8443/rhel8/postgresql-13:1-7
registry.redhat.io/rhel8/postgresql-13:latest	registry.ocp4.example.com:8443/rhel8/postgresql-13:latest
registry.redhat.io/ubi8/ubi:8.6-943	registry.ocp4.example.com:8443/ubi8/ubi:8.6-943

Controlling Your Systems

You are assigned remote computers in a Red Hat Online Learning (ROLE) classroom. Self-paced courses are accessed through a web application that is hosted at rol.redhat.com [<http://rol.redhat.com>]. Log in to this site with your Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

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

The screenshot shows a web-based interface for managing lab environments. At the top, there's a navigation bar with 'Table of Contents', 'Course', 'Lab Environment', and other icons. Below the navigation is a section titled 'Lab Controls' with instructions: 'Click CREATE to build all of the virtual machines needed for the classroom lab environment. This may take several minutes to complete. Once created the environment can then be stopped and restarted to pause your experience.' It also notes that deleting the lab will remove all virtual machines and lose progress. Below this is a table listing five virtual machines:

Name	Status	Action Buttons
bastion	active	ACTION - OPEN CONSOLE
classroom	active	ACTION - OPEN CONSOLE
servera	building	ACTION - OPEN CONSOLE
serverb	building	ACTION - OPEN CONSOLE
workstation	active	ACTION - OPEN CONSOLE

Figure 0.2: An example course lab environment management page

Machine States

Virtual machine state	Description
building	The virtual machine is being created.
active	The virtual machine is running and available. If the virtual machine just started, then it might still be starting services.
stopped	The virtual machine is shut down. On starting, the virtual machine boots into the same state that it was in before shutdown. The disk state is preserved.

Classroom Actions

Button or action	Description
CREATE	Create the ROLE classroom. Creates and starts all the necessary virtual machines for this classroom.
CREATING	The ROLE classroom virtual machines are being created. Creation can take several minutes to complete.
DELETE	Delete the ROLE classroom. Deletes all virtual machines in the classroom. All saved work on those systems' disks is lost.
START	Start all virtual machines in the classroom.
STARTING	All virtual machines in the classroom are starting.
STOP	Stop all virtual machines in the classroom.

Machine Actions

Button or action	Description
OPEN CONSOLE	Connect to the system console of the virtual machine in a new browser tab. You can log in directly to the virtual machine and run commands, when required. Normally, log in to the workstation virtual machine only, and from there, use ssh to connect to the other virtual machines.
ACTION > Start	Start (power on) the virtual machine.
ACTION > Shutdown	Gracefully shut down the virtual machine, and preserve disk contents.
ACTION > Power Off	Forcefully shut down the virtual machine, and still preserve disk contents. This action is equivalent to removing the power from a physical machine.

Button or action	Description
ACTION > Reset	Forcefully shut down the virtual machine and reset the associated storage to its initial state. All saved work on that system's disks is lost.

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

At the start of an exercise, if you are instructed to reset all virtual machines, then click ACTION > **Reset** on every virtual machine in the list.

To return the classroom environment to its original state at the start of the course, you can click **DELETE** to remove the entire classroom environment. After the lab is deleted, click **CREATE** to provision a new set of classroom systems.



Warning

The **DELETE** operation cannot be undone. All completed work in the classroom environment is lost.

The Auto-stop and Auto-destroy Timers

The Red Hat Online Learning enrollment entitles you to a set allotment of computer time. To help to conserve your allotted time, the ROLE classroom uses timers, which shut down or delete the classroom environment when the appropriate timer expires.

To adjust the timers, locate the two + buttons at the bottom of the course management page. Click the auto-stop + button to add another hour to the auto-stop timer. Click the auto-destroy + button to add another day to the auto-destroy timer. Auto-stop has a maximum of 11 hours, and auto-destroy has a maximum of 14 days. Be careful to keep the timers set while you are working, so that your environment is not unexpectedly shut down. Be careful not to set the timers unnecessarily high, which could waste your subscription time allotment.

Performing Lab Exercises

You might see the following lab activity types in this course:

- A *guided exercise* is a hands-on practice exercise that follows a presentation section. It walks you through a procedure to perform, step by step.
- A *quiz* is typically used when checking knowledge-based learning, or when a hands-on activity is impractical for some other reason.
- An *end-of-chapter lab* is a gradable hands-on activity to help you to check your learning. You work through a set of high-level steps, based on the guided exercises in that chapter, but the steps do not walk you through every command. A solution is provided with a step-by-step walk-through.
- A *comprehensive review lab* is used at the end of the course. It is also a gradable hands-on activity, and might cover content from the entire course. You work through a specification of what to do in the activity, without receiving the specific steps to do so. Again, a solution is provided with a step-by-step walk-through that meets the specification.

To prepare your lab environment at the start of each hands-on activity, run the `lab start` command with a specified activity name from the activity's instructions. Likewise, at the end of each hands-on activity, run the `lab finish` command with that same activity name to clean up after the activity. Each hands-on activity has a unique name within a course.

The syntax for running an exercise script is as follows:

```
[student@workstation ~]$ lab action exercise
```

The *action* is a choice of `start`, `grade`, or `finish`. All exercises support `start` and `finish`. Only end-of-chapter labs and comprehensive review labs support `grade`.

start

The `start` action verifies the required resources to begin an exercise. It might include configuring settings, creating resources, confirming prerequisite services, and verifying necessary outcomes from previous exercises. You can perform an exercise at any time, even without performing preceding exercises.

grade

For gradable activities, the `grade` action directs the `lab` command to evaluate your work, and shows a list of grading criteria with a `PASS` or `FAIL` status for each. To achieve a `PASS` status for all criteria, fix the failures and rerun the `grade` action.

finish

The `finish` action cleans up resources that were configured during the exercise. You can perform an exercise as many times as you want.

The `lab` command supports tab completion. For example, to list all exercises that you can start, enter `lab start` and then press the Tab key twice.

Lab Directory Considerations

The DO280 course uses a Python-based `lab` script that configures the directory structure for each guided exercise and lab activity. The **workspace** directory for this course is `/home/student/D0280`.

The `lab` script copies the necessary files for each course activity to the workspace directory.

For example, the `lab start updates-rollout` command does the following tasks:

- Creates an `updates-rollout` directory in the workspace: `/home/student/D0280/labs/updates-rollout` workspace.
- Copies the files for the activity to the `/home/student/D0280/labs/updates-rollout` directory.

Troubleshooting Lab Scripts

If an error occurs while running the `lab` command, then you might want to review the following files:

- `/tmp/log/labs`: This directory contains log files. The `lab` script creates a unique log file for each activity. For example, the log file for the `lab start updates-rollout` command is `/tmp/log/labs/updates-rollout`.
- `/home/student/.grading/config.yaml`: This file contains the course-specific configuration. Do not modify this file.

The `lab start` commands usually verify whether the Red Hat OpenShift Container Platform (RHOCP) cluster is ready and reachable. If you run the `lab start` command right after creating the classroom environment, then you might get errors when the command verifies the cluster API or the credentials. These errors occur because the RHOCP cluster might take up to 15 minutes to become available. A convenient solution is to run the `lab finish` command to clean up the scenario, wait a few minutes, and then rerun the `lab start` command.



Important

In this course, the `lab start` scripts normally create a specific RHOCP project for each exercise. The `lab finish` scripts remove the exercise-specific RHOCP project.

If you are retrying an exercise, then you might need to wait before running the `lab start` command again. The project removal process might take up to 10 minutes to be fully effective.

Chapter 1

Declarative Resource Management

Goal

Deploy and update applications from resource manifests that are parameterized for different target environments.

Objectives

- Deploy and update applications from resource manifests that are stored as YAML files.
- Deploy and update applications from resource manifests that are augmented by Kustomize.

Sections

- Resource Manifests (and Guided Exercise)
- Kustomize Overlays (and Guided Exercise)

Lab

- Declarative Resource Management

Resource Manifests

Objectives

- Deploy and update applications from resource manifests that are stored as YAML files.

An application in a Kubernetes cluster often consists of multiple resources that work together. Each resource has a definition and a configuration. Many of the resource configurations share common attributes that must match to operate correctly. Imperative commands configure each resource, one at time. However, using imperative commands has some issues:

- Impaired reproducibility
- Lacking version control
- Lacking support for GitOps

Rather than imperative commands, declarative commands are instead the preferred way to manage resources, by using resource manifests. A resource manifest is a file, in JSON or YAML format, with resource definition and configuration information. Resource manifests simplify the management of Kubernetes resources, by encapsulating all the attributes of an application in a file or a set of related files. Kubernetes uses declarative commands to read the resource manifests and to apply changes to the cluster to meet the state that the resource manifest defines.

The resource manifests are in YAML or JSON format, and thus can be version-controlled. Version control of resource manifests enables tracing of configuration changes. As such, adverse changes can be rolled back to an earlier version to support recoverability.

Resource manifests ensure that applications can be precisely reproduced, typically with a single command to deploy many resources. The reproducibility from resource manifests supports the automation of the GitOps practices of continuous integration and continuous delivery (CI/CD).

Imperative Versus Declarative Workflows

The Kubernetes CLI uses both imperative and declarative commands. Imperative commands perform an action that is based on a command, and use command names that closely reflect the action. In contrast, declarative commands use a resource manifest file to declare the intended state of a resource.

A Kubernetes manifest is a YAML- or JSON-formatted file with declaration statements for Kubernetes resources such as deployments, pods, or services. Instead of using imperative commands to create Kubernetes resources, manifest files provide all the details for the resource in a single file. Working with manifest files enables the use of more reproducible processes. Instead of reproducing sequences of imperative commands, manifest files contain the entire definition of resources and can be applied in a single step. Using manifest files is also useful for tracking system configuration changes in a source code management system.

Given a new or updated resource manifest, Kubernetes provides commands that compare the intended state that is specified in the resource manifest to the current state of the resource. These commands then apply transformations to the current state to match the intended state.

Imperative Workflow

An imperative workflow is useful for developing and testing. The following example uses the `kubectl create deployment` imperative command, to create a deployment for a MySQL database.

```
[user@host ~]$ kubectl create deployment db-pod --port 3306 \
--image registry.ocp4.example.com:8443/rhel8/mysql-80
deployment.apps/db-pod created
```

In addition to using verbs that reflect the action of the command, imperative commands use options to provide the details. The example command uses the `--port` and the `--image` options to provide the required details to create the deployment.

The use of imperative commands affects applying changes to live resources. For example, the pod from the previous deployment would fail to start due to missing environment variables. The following `kubectl set env deployment` imperative command resolves the problem by adding the required environment variables to the deployment:

```
[user@host ~]$ kubectl set env deployment/db-pod \
MYSQL_USER='user1' \
MYSQL_PASSWORD='mypa55w0rd' \
MYSQL_DATABASE='items'
deployment.apps/db-pod updated
```

Executing this `kubectl set env deployment` command changes the deployment resource named `db-pod`, and provides the extra needed variables to start the container. A developer can continue building out the application, by using imperative commands to add components, such as services, routes, volume mounts, and persistent volume claims. With the addition of each component, the developer can run tests to ensure that the component correctly executes the intended function.

Imperative commands are useful for developing and experimenting. With imperative commands, a developer can build up an application one component at a time. When a component is added, the Kubernetes cluster provides error messages that are specific to the component. The process is analogous to using a debugger to step through code execution one line at a time. Using imperative commands usually provides clearer error messages, because an error occurs after adding a specific component.

However, long command lines and a fragmented application deployment are not ideal for deploying an application in production. With imperative commands, changes are a sequence of commands that must be maintained to reflect the intended state of the resources. The sequence of commands must be tracked and kept up to date.

Using Declarative Commands

Instead of tracking a sequence of commands, a manifest file captures the intended state of the sequence. In contrast to using imperative commands, declarative commands use a manifest file, or a set of manifest files, to combine all the details for creating those components into YAML files that can be applied in a single command. Future changes to the manifest files require only reapplying the manifests. Instead of tracking a sequence of complex commands, version control systems can track changes to the manifest file.

Although manifest files can also use the JSON syntax, YAML is generally preferred and is more popular. To continue the debugging analogy, debugging an application that is deployed from

manifests is similar to trying to debug a full, completed running application. It can take more effort to find the source of the error, especially when the error is not a result of manifest errors.

Creating Kubernetes Manifests

Creating manifest files from scratch can take time. You can use the following techniques to provide a starting point for your manifest files:

- Use the YAML view of a resource from the web console.
- Use imperative commands with the `--dry-run=client` option to generate manifests that correspond to the imperative command.

The `kubectl explain` command provides the details for any field in the manifest. For example, use the `kubectl explain deployment.spec.template.spec` command to view field descriptions that specify a pod object within a deployment manifest.

To create a starter deployment manifest, use the `kubectl create deployment` command to generate a manifest by using the `--dry-run=client` option:

```
[user@host ~]$ kubectl create deployment hello-openshift -o yaml \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0 \
--save-config \ ①
--dry-run=client \ ②
> ~/my-app/example-deployment.yaml
```

- ① The `--save-config` option adds configuration attributes that declarative commands use. For deployments resources, this option saves the resource configuration in an `kubectl.kubernetes.io/last-applied-configuration` annotation.
- ② The `--dry-run=client` option prevents the command from creating resources in the cluster.

The following example shows a minimal deployment manifest file, not production-ready, for the `hello-openshift` deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    ...output omitted...
  creationTimestamp: null
  labels:
    app: hello-openshift
    name: hello-openshift
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-openshift
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
```

```

    app: hello.openshift
spec:
  containers:
    - image: quay.io/redhattraining/hello-world-nginx:v1.0
      name: hello-world-nginx
      resources: {}
status: {}

```

When using imperative commands to create manifests, the resulting manifests might contain fields that are not necessary for creating a resource. For example, the following example changes the manifest by removing the empty and null fields. Removing unnecessary fields can significantly reduce the length of the manifests, and in turn reduce the overhead to work with them.

Additionally, you might need to further customize the manifests. For example, in a deployment, you might customize the number of replicas, or declare the ports that the deployment provides. The following notes explain the additional changes:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: resource-manifests ①
  labels:
    app: hello.openshift
    name: hello.openshift
spec:
  replicas: 2 ②
  selector:
    matchLabels:
      app: hello.openshift
  template:
    metadata:
      labels:
        app: hello.openshift
    spec:
      containers:
        - image: quay.io/redhattraining/hello-world-nginx:v1.0
          name: hello-world-nginx
          ports:
            - containerPort: 8080 ③
              protocol: TCP

```

- ① Add a namespace attribute to prevent deployment to the wrong project.
- ② Requires two replicas instead of one.
- ③ Specifies the container port for the service to use.

You can create a manifest file for each resource that you manage. Alternatively, add each of the manifests to a single multi-part YAML file, and use a `---` line to separate the manifests.

```

---
apiVersion: apps/v1
kind: Deployment
metadata:

```

```

namespace: resource-manifests
annotations:
  ...output omitted...
---
apiVersion: v1
kind: Service
metadata:
  namespace: resource-manifests
  labels:
    app: hello-openshift
    name: hello-openshift
spec:
  ...output omitted...

```

Using a single file with multiple manifests versus using manifests that are defined in multiple manifest files is a matter of organizational preference. The single file approach has the advantage of keeping together related manifests. With the single file approach, it can be more convenient to change a resource that must be reflected across multiple manifests. In contrast, keeping manifests in multiple files can be more convenient for sharing resource definitions with others.

After creating manifests, you can test them in a non-production environment, or proceed to deploy the manifests. Validate the resource manifests before deploying applications in the production environment.

Declarative Workflows

Declarative commands use a resource manifest instead of adding the details to many options on the command line. To create a resource, use the `kubectl create -f resource.yaml` command. Instead of a file name, you can pass a directory to the command to process all the resource files in a directory. Add the `--recursive=true` or `-R` option to recursively process resource files that are provided in multiple subdirectories.

The following example creates the resources from the manifests in the `my-app` directory. In this example, the `my-app` directory contains the `example-deployment.yaml` and `service/example-service.yaml` files from previously.

```

[user@host ~]$ tree my-app
my-app
├── example_deployment.yaml
└── service
    └── example_service.yaml

[user@host ~]$ kubectl create -R -f ~/my-app
deployment.apps/hello-openshift created
service/hello-openshift created

```

The command also accepts a URL:

```

[user@host ~]$ kubectl create -f \
  https://example.com/example-apps/deployment.yaml
deployment.apps/hello-openshift created

```

Updating Resources

The `kubectl apply` command can also create resources with the same `-f` option that is illustrated with the `kubectl create` command. However, the `kubectl apply` command can also update a resource.

Updating resources is more complex than creating resources. The `kubectl apply` command implements several techniques to apply the updates without causing issues.

The `kubectl apply` command writes the contents of the configuration file to the `kubectl.kubernetes.io/last-applied-configuration` annotation. The `kubectl create` command can also generate this annotation by using the `--save-config` option. The `kubectl apply` command uses the `last-applied-configuration` annotation to identify fields that are removed from the configuration file and that must be cleared from the live configuration.

Although the `kubectl create -f` command can create resources from a manifest, the command is imperative and thus does not account for the current state of a live resource. Executing `kubectl create -f` against a manifest for a live resource gives an error. In contrast, the `kubectl apply -f` command is declarative, and considers the difference between the current resource state in the cluster and the intended resource state that is expressed in the manifest.

For example, to update the container's image from version `v1.0` to `latest`, first update the YAML resource manifest to specify the new tag on the image. Then, use the `kubectl apply` command to instruct Kubernetes to create a version of the deployment resource by using the updated image version that is specified in the manifest.

YAML Validation

Before applying the changes to the resource, use the `--dry-run=server` and the `--validate=true` flags to inspect the file for errors.

- The `--dry-run=server` option submits a server-side request without persisting the resource.
- The `--validate=true` option uses a schema to validate the input and fails the request if it is invalid.

Any syntax errors in the YAML are included in the output. Most importantly, the `--dry-run=server` option prevents applying any changes to the Kubernetes runtime.

```
[user@host ~]$ kubectl apply -f ~/my-app/example-deployment.yaml \
--dry-run=server --validate=true
deployment.apps/hello-openshift created (server dry-run) ①
```

- ① The output line that ends in `(server dry-run)` provides the action that the resource file would perform if applied.



Note

The `--dry-run=client` option prints only the object that would be sent to the server. The cluster resource controllers can refuse a manifest even if the syntax is valid YAML. In contrast, the `--dry-run=server` option sends the request to the server to confirm that the manifest conforms to current server policies, without creating resources on the server.

Comparing Resources

Use the `kubectl diff` command to review differences between live objects and manifests. When updating resource manifests, you can track differences in the changed files. However, many manifest changes, when applied, do not change the state of the cluster resources. A text-based diff tool would show all such differences, and result in a noisy output.

In contrast, using the `kubectl diff` command might be more convenient to preview changes. The `kubectl diff` command emphasizes the significant changes for the Kubernetes cluster. Review the differences to validate that manifest changes have the intended effect.

```
[user@host ~]$ kubectl diff -f example-deployment.yaml
...output omitted...
diff -u -N /tmp/LIVE-2647853521/apps.v1.Deployment.resource...
--- /tmp/LIVE-2647853521/apps.v1.Deployment.resource-manife...
+++ /tmp/MERGED-2640652736/apps.v1.Deployment.resource-mani...
@@ -6,7 +6,7 @@
    kubectl.kubernetes.io/last-applied-configuration: |
        ...output omitted...
    creationTimestamp: "2023-04-27T16:07:47Z"
-   generation: 1 ①
+   generation: 2
    labels:
        app: hello-openshift
        name: hello-openshift
@@ -32,7 +32,7 @@
        app: hello-openshift
    spec:
        containers:
-           - image: registry.ocp4.example.com:8443/.../hello-world-nginx:v1.0 ②
+           - image: registry.ocp4.example.com:8443/.../hello-world-nginx:latest
            imagePullPolicy: IfNotPresent
            name: hello-openshift
        ports:
```

- ① The line that starts with the - character shows that the current deployment is on generation 1. The following line, which starts with the + character, shows that the generation changes to 2 when the manifest file is applied.
- ② The image line, which starts with the - character, shows that the current image uses the v1.0 version. The following line, which starts with the + character, shows a version change to latest when the manifest file is applied.

Kubernetes resource controllers automatically add annotations and attributes to the live resource that make the output of other text-based diff tools misleading, by reporting many differences that have no impact on the resource configuration. Extracting manifests from live resources and making comparisons with tools such as the `diff` command reports many differences of no value. Using the `kubectl diff` command confirms that a live resource matches a resource configuration that a manifest provides. GitOps tools depend on the `kubectl diff` command to determine whether anyone changed resources outside the GitOps workflow. Because the tools themselves cannot know all details about how any controllers might change a resource, the tools defer to the cluster to determine whether a change is meaningful.

Update Considerations

When using the `oc diff` command, recognize when applying a manifest change does not generate new pods. For example, if an updated manifest changes only values in secret or a configuration map, then applying the updated manifest does not generate new pods that use those values. Because pods read secret and configuration maps at startup, in this case applying the updated manifest leaves the pods in a vulnerable state, with stale values that are not synchronized with the updated secret or with the configuration map.

As a solution, use the `oc rollout restart deployment deployment-name` command to force a restart of the pods that are associated with the deployment. The forced restart generates pods that use the new values from the updated secret or configuration map.

In deployments with a single replica, you can also resolve the problem by deleting the pod. Kubernetes responds by automatically creating a pod to replace the deleted pod. However, for multiple replicas, using the `oc rollout` command to restart the pods is preferred, because the pods are stopped and replaced in a smart manner that minimizes downtime.

This course covers other resource management mechanisms that can automate or eliminate some of these challenges.

Applying Changes

The `kubectl create` command attempts to create the specified resources in the manifest file. Using the `kubectl create` command generates an error if the targeted resources are already live in the cluster. In contrast, the `kubectl apply` command compares three sources to determine how to process the request and to apply changes.

1. The manifest file
2. The live configuration of the resource in the cluster
3. The configuration that is stored in the `last-applied-configuration` annotation

If the specified resource in the manifest file does not exist, then the `kubectl apply` command creates the resource. If any fields in the `last-applied-configuration` annotation of the live resource are not present in the manifest, then the command removes those fields from the live configuration. After applying changes to the live resource, the `kubectl apply` command updates the `last-applied-configuration` annotation of the live resource to account for the change.

When creating a resource, the `--save-config` option of the `kubectl create` command produces the required annotations for future `kubectl apply` commands to operate.

Patching Kubernetes Resources

You can modify objects in OpenShift in a repeatable way with the `oc patch` command. The `oc patch` command updates or adds fields in an existing object from a provided JSON or YAML snippet or file. A software developer might distribute a patch file or snippet to fix problems before a full update is available.

To patch an object from a snippet, use the `oc patch` command with the `-p` option and the snippet. The following example updates the `hello` deployment to have a CPU resource request of `100m` with a JSON snippet:

```
[user@host ~]$ oc patch deployment hello -p \
  '{"spec":{"template":{"spec":{"resources":{"requests":{"cpu": "100m"}}}}}' \
deployment/hello patched
```

To patch an object from a patch file, use the `oc patch` command with the `--patch-file` option and the location of the patch file. The following example updates the `hello` deployment to include the content of the `~/volume-mount.yaml` patch file:

```
[user@host ~]$ oc patch deployment hello --patch-file ~/volume-mount.yaml
deployment.apps/hello patched
```

The contents of the patch file describe mounting a persistent volume claim as a volume:

```
spec:
  template:
    spec:
      containers:
        - name: hello
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html/
      volumes:
        - name: www
          persistentVolumeClaim:
            claimName: nginx-www
```

This patch results in the following manifest for the `hello` deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
  ...output omitted...
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      containers:
        ...output omitted...
        name: server
        ...output omitted...
        volumeMounts:
          - mountPath: /usr/share/nginx/html/
            name: www
          - mountPath: /etc/nginx/conf.d/
            name: tls-conf
        ...output omitted...
      volumes:
        - configMap:
            defaultMode: 420
            name: tls-conf
```

```
name: tls-conf
- persistentVolumeClaim:
  claimName: nginx-www
  name: www
...output omitted...
```

The patch applies to the `hello` deployment regardless of whether the `www` volume mount exists. The `oc patch` command modifies existing fields in the object that are specified in the patch:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
  ...output omitted...
spec:
  ...output omitted...
  template:
    spec:
      containers:
        ...output omitted...
        name: server
        ...output omitted...
      volumeMounts:
        - mountPath: /usr/share/nginx/www/ 1
          name: www
        - mountPath: /etc/nginx/conf.d/
          name: tls-conf
        ...output omitted...
      volumes:
        - configMap:
            defaultMode: 420
            name: tls-conf
            name: tls-conf
        - persistentVolumeClaim:
            claimName: deprecated-www 2
            name: www
...output omitted...
```

1 **2** The `www` volume already exists. The patch replaces the existing data with the new data.



References

For more information, refer to the *OpenShift CLI Developer Command Reference* section in the *OpenShift CLI (oc)* chapter in the Red Hat OpenShift Container Platform 4.14 *CLI Tools* documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/cli_tools/index#cli-developer-commands

For more information, refer to the *Using Deployment Strategies* section in the *Deployments* chapter in the Red Hat OpenShift Container Platform 4.14 *Building Applications* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/building_applications/index#deployment-strategies

For more information about the `oc patch` command, refer to the `oc patch` section in the *OpenShift CLI Developer Command Reference* chapter in the Red Hat OpenShift Container Platform 4.14 *CLI Tools* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#oc-patch

Kubernetes Documentation – Replicaset

<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>

Kubernetes Documentation – Deployment Strategy

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#strategy>

Kubernetes Documentation – Deployment

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

► Guided Exercise

Resource Manifests

Deploy and update an application from resource manifests from YAML files that are stored in a Git server.

Outcomes

- Deploy applications from resource manifests from YAML files that are stored in a GitLab repository.
- Inspect new manifests for potential update issues.
- Update application deployments from new YAML manifests.
- Force the redeployment of pods when necessary.

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 start declarative-manifests
```

Instructions

- 1. Log in to the OpenShift cluster and create the `declarative-manifests` 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 `declarative-manifests` project.

```
[student@workstation ~]$ oc new-project declarative-manifests
Now using project "declarative-manifests" on server ...
...output omitted...
```

- 2. Clone the `declarative-manifest` project from the Git repository.

- 2.1. Change your directory to the project `labs` directory.

```
[student@workstation ~]$ cd ~/D0280/labs
```

- 2.2. Clone the Git repository from <https://git.ocp4.example.com/developer/declarative-manifests.git>. Use developer for both the username and for the password.

```
[student@workstation lab]$ git clone \
  https://git.ocp4.example.com/developer/declarative-manifests.git
Cloning into 'declarative-manifests'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 24 (delta 8), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (24/24), done.
Resolving deltas: 100% (8/8), done.
```

- 2.3. Go to the declarative-manifest directory.

```
[student@workstation lab]$ cd declarative-manifests
[student@workstation declarative-manifests]$
```

► 3. Inspect the contents of the Git repository.

- 3.1. List the contents of the declarative-manifests directory.

```
[student@workstation declarative-manifests]$ ls -lA
total 12
-rw-rw-r--. 1 student student 3443 Jun 21 16:39 database.yaml
-rw-rw-r--. 1 student student 2278 Jun 21 16:39 exoplanets.yaml
drwxrwxr-x. 8 student student 163 Jun 21 16:39 .git
-rw-rw-r--. 1 student student 0 Jun 21 16:39 .gitkeep
-rw-rw-r--. 1 student student 11 Jun 21 16:39 README.md
```

- 3.2. List the commits, branches, and tags on the Git repository.

```
[student@workstation declarative-manifests]$ git log --oneline
4045336 (HEAD -> main, tag: third, origin/v1.1.1, origin/main, origin/HEAD)
  Exoplanets v1.1.1 1
ad455b2 Database v1.1.1
821420c (tag: second, origin/v1.1.0) Exoplanets v1.1.0 2
d9abeb0 (tag: first, origin/v1.0) Exoplanets v1.0 3
a11396e Database v1.0
e868a90 README
18ddf3c Initial commit
```

- 1** The v1.1.1 branch points to the *third* version of the application.
- 2** The v1.1.0 branch points to the *second* version of the application.
- 3** The v1.0 branch points to the *first* version of the application.

► 4. Deploy the resource manifests of the first application version.

- 4.1. Switch to the v1.0 branch, which contains the YAML manifests for the first version of the application.

```
[student@workstation declarative-manifests]$ git checkout v1.0
branch 'v1.0' set up to track 'origin/v1.0'.
Switched to a new branch 'v1.0'
```

4.2. Validate the YAML resource manifest for the application.

```
[student@workstation declarative-manifests]$ oc apply -f . \
--validate=true --dry-run=server
configmap/database created (server dry run)
secret/database created (server dry run)
deployment.apps/database created (server dry run)
service/database created (server dry run)
configmap/exoplanets created (server dry run)
secret/exoplanets created (server dry run)
deployment.apps/exoplanets created (server dry run)
service/exoplanets created (server dry run)
route.route.openshift.io/exoplanets created (server dry run)
```

4.3. Deploy the exoplanets application.

```
[student@workstation declarative-manifests]$ oc apply -f .
configmap/database created
secret/database created
deployment.apps/database created
service/database created
configmap/exoplanets created
secret/exoplanets created
deployment.apps/exoplanets created
service/exoplanets created
route.route.openshift.io/exoplanets created
```

4.4. List the deployments and pods. The exoplanets pod can go into a temporary crash loop backoff state if it attempts to access the database before it becomes available. Wait for the pods to be ready. Press **Ctrl+C** to exit the **watch** command.

```
[student@workstation declarative-manifests]$ watch oc get deployments,pods

Every 2.0s: oc get deployments,pods ...

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database  1/1     1           1          32s
deployment.apps/exoplanets 1/1     1           1          32s

NAME           READY   STATUS    RESTARTS   AGE
pod/database-6fddbfb94f-2pghj  1/1     Running   0          32s
pod/exoplanets-64c87f5796-bw8tm 1/1     Running   0          32s
```

4.5. List the route for the exoplanets application.

```
[student@workstation declarative-manifests]$ oc get routes -l app=exoplanets
NAME          HOST/PORT
exoplanets    exoplanets-declarative-manifests.apps.ocp4.example.com  ...
...
```

- 4.6. Open the route URL in the web browser. The application version is v1.0.

<http://exoplanets-declarative-manifests.apps.ocp4.example.com/>

Exoplanets

The planets listed here are a small subset of the known planets found outside of our solar system. Mass and radius are listed in "Jupiter mass" and "Jupiter radius" units. The orbital period is measured in Earth days. The full dataset is available from the [Open Exoplanet Catalogue](#).

2M 0746+20 b

Mass	Radius	Period
30	0.97	4640

2M 2140+16 b

Mass	Radius	Period
20	0.92	7340

2M 2206-20 b

Mass	Radius	Period
30	1.3	8686

- 5. Deploy the second version of the exoplanets application.

- 5.1. Switch to the v1.1.0 branch of the Git repository.

```
[student@workstation declarative-manifests]$ git checkout v1.1.0
branch 'v1.1.0' set up to track 'origin/v1.1.0'.
Switched to a new branch 'v1.1.0'
```

- 5.2. Inspect the changes from the new manifests.

```
[student@workstation declarative-manifests]$ oc diff -f .
...output omitted...
- secretRef:
  name: exoplanets
- image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.0
+ image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.0
  imagePullPolicy: Always
  livenessProbe:
    failureThreshold: 3
```

The new version changes the image that is deployed to the cluster. Because the change is in the deployment, the new manifest produces new pods for the application.

- 5.3. Apply the changes from the new manifests.

```
[student@workstation declarative-manifests]$ oc apply -f .
configmap/database unchanged
secret/database configured
deployment.apps/database configured
service/database configured
configmap/exoplanets unchanged
secret/exoplanets configured
deployment.apps/exoplanets configured
service/exoplanets unchanged
route.route.openshift.io/exoplanets configured
```

- 5.4. List the deployments and pods. Wait for the application pod to be ready. Press **Ctrl+C** to exit the `watch` command.

```
[student@workstation declarative-manifests]$ watch oc get deployments,pods
Every 2.0s: oc get deployments,pods ...

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database  1/1      1          1          6m32s
deployment.apps/exoplanets 1/1      1          1          6m33s

NAME           READY   STATUS    RESTARTS   AGE
pod/database-6fdbbf94f-2phgj  1/1      Running   0          6m33s
pod/exoplanets-74c85f5796-tw8tf 1/1      Running   0          32s
```

- 5.5. List the route for the exoplanets application.

```
[student@workstation declarative-manifests]$ oc get routes -l app=exoplanets
NAME          HOST/PORT
exoplanets   exoplanets-declarative-manifests.apps.ocp4.example.com ...
```

- 5.6. Open the route URL in the web browser. The application version is v1.1.0.

<http://exoplanets-declarative-manifests.apps.ocp4.example.com/>

Exoplanets - v1.1.0

The planets listed here are a small subset of the known planets found outside of our solar system.

- Mass and radius are listed in "Jupiter mass" and "Jupiter radius" units.
- The orbital period is measured in "Earth days".

The full dataset is available from the [Open Exoplanet Catalogue](#).

2M 0746+20 b			2M 2140+16 b			2M 2206-20 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
30	4640	0.97	20	7340	0.92	30	8686	1.3

51 Eri b			55 Cancri e			BD+20 594 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
2	15000	1	0.0251	0.7365474	0.167280	0.05129	41.6855	0.1989

beta Pic b			CoRoT-10 b			CoRoT-11 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
0.0251	0.7365474	0.167280	0.05129	41.6855	0.1989	0.0251	0.7365474	0.167280

- 6. Deploy the third version of the exoplanets application.

Chapter 1 | Declarative Resource Management

- 6.1. Switch to the v1.1.1 branch of the Git repository.

```
[student@workstation declarative-manifests]$ git checkout v1.1.1
branch 'v1.1.1' set up to track 'origin/v1.1.1'.
Switched to a new branch 'v1.1.1'
```

- 6.2. View the differences between the currently deployed version of the application and the updated resource manifests.

```
[student@workstation declarative-manifests]$ oc diff -f .
...output omitted...
kind: Secret ①
metadata:
  annotations:
...output omitted...
- DB_USER: '*** (before)' ②
+ DB_USER: '*** (after)'
kind: Secret
metadata:
  annotations:
```

- ① The secret resource is changed.
- ② The DB_USER field of the secret resource is changed.

- 6.3. Inspect the current application pods.

```
[student@workstation declarative-manifests]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
database-6fddbbf94f-brlj6   1/1     Running   0          44m
exoplanets-674cc57b5d-mv8kd 1/1     Running   0          18m
```

- 6.4. Deploy the new version of the application.

```
[student@workstation declarative-manifests]$ oc apply -f .
configmap/database unchanged
secret/database configured
deployment.apps/database configured
service/database configured
configmap/exoplanets unchanged
secret/exoplanets configured
deployment.apps/exoplanets unchanged
service/exoplanets unchanged
route.route.openshift.io/exoplanets configured
```

- 6.5. Inspect the current application pods again

```
[student@workstation declarative-manifests]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
database-6fddbbf94f-brlj6   1/1     Running   0          10m
exoplanets-674cc57b5d-mv8kd 0/1     CrashLoopBackOff 4 (14s ago) 2m
```

Chapter1 | Declarative Resource Management

Although the secret is updated, the deployed application pods are not changed. These non-updated pods are a problem, because the pods load secrets and configuration maps at startup. Currently, the pods have stale values from the previous configuration, and therefore could crash.

- 7. Force the exoplanets application to restart, to flush out any stale configuration data.

- 7.1. Use the `oc get deployments` command to confirm the deployments.

```
[student@workstation declarative-manifests]$ oc get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
database   1/1     1           1           32m
exoplanets 0/1     1           0           32m
```

- 7.2. Use the `oc rollout` command to restart the database deployment.

```
[student@workstation declarative-manifests]$ oc rollout restart \
deployment/database
deployment.apps/database restarted
```

- 7.3. Use the `oc rollout` command to restart the exoplanets deployment.

```
[student@workstation declarative-manifests]$ oc rollout restart \
deployment/exoplanets
deployment.apps/exoplanets restarted
```

- 7.4. List the pods. The exoplanets pod can go into a temporary crash loop backoff state if it attempts to access the database before it becomes available. Wait for the application pod to be ready. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation declarative-manifests]$ watch oc get pods
Every 2.0s: oc get deployments,pods ...

NAME                  READY   STATUS    RESTARTS   AGE
database-7c767c4bd7-m72nk   1/1     Running   0          32s
exoplanets-64c87f5796-bw8tm   1/1     Running   0          32s
```

- 7.5. Use the `oc get deployment` command with the `-o yaml` option to view the `last-applied-configuration` annotation.

```
[student@workstation declarative-manifests]$ oc get deployment \
exoplanets -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "3"
    description: Defines how to deploy the application server
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"Deployment","metadata": {"annotations":...}}
      template.alpha.openshift.io/wait-for-ready: "true"
    ...output omitted...
```

- 7.6. Open the route URL in the web browser. The application version is v1.1.1.

<http://exoplanets-declarative-manifests.apps.ocp4.example.com/>

The screenshot shows a web application titled "Exoplanets - v1.1.1". It displays a table of exoplanet data with columns for Mass, Radius, and Period. The data is organized into three main sections: "2M 0746+20 b", "2M 2140+16 b", and "2M 2206-20 b", each containing one or more planets. Below these are three additional entries: "51 Eri b", "55 Cancri e", and "BD+20 594 b". Each entry has its own table with the same three columns. At the bottom, there are three more entries: "beta Pic b", "CoRoT-10 b", and "CoRoT-11 b".

Mass	Radius	Period
30	4640	0.97

Mass	Radius	Period
20	7340	0.92

Mass	Radius	Period
30	8686	1.3

Mass	Radius	Period
2	15000	1

Mass	Radius	Period
0.0251	0.7365474	0.16728

Mass	Radius	Period
0.05129	41.6855	0.1989

Mass	Radius	Period

Mass	Radius	Period

Mass	Radius	Period

► **8.** Clean up the resources.

- 8.1. Delete the application resources.

```
[student@workstation declarative-manifests]$ oc delete -f .  
configmap "database" deleted  
secret "database" deleted  
deployment.apps "database" deleted  
service "database" deleted  
configmap "exoplanets" deleted  
secret "exoplanets" deleted  
deployment.apps "exoplanets" deleted  
service "exoplanets" deleted  
route.route.openshift.io "exoplanets" deleted
```

- 8.2. Change to the student HOME directory.

```
[student@workstation declarative-manifests]$ cd  
[student@workstation ~]
```

Finish

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

```
[student@workstation ~]$ lab finish declarative-manifests
```

Kustomize Overlays

Objectives

- Deploy and update applications from resource manifests that are augmented by Kustomize.

Kustomize

When using Kubernetes, multiple teams use multiple environments, such as development, staging, testing, and production, to deploy applications. These environments use applications with minor configuration changes.

Many organizations deploy a single application to multiple data centers for multiple teams and regions. Depending on the load, the organization needs a different number of replicas for every region. The organization might need various configurations that are specific to a data center or team.

All these use cases require a single set of manifests with multiple customizations at multiple levels. Kustomize can support such use cases.

Kustomize is a configuration management tool to make declarative changes to application configurations and components and preserve the original base YAML files. You group in a directory the Kubernetes resources that constitute your application, and then use Kustomize to copy and adapt these resource files to your environments and clusters. Both the `kubectl` and `oc` commands integrate the Kustomization tool.

Kustomize File Structure

Kustomize works on directories that contain a `kustomization.yaml` file at the root. Kustomize supports compositions and customization of different resources such as deployment, service, and secret. You can use patches to apply customization to different resources. Kustomize has a concept of *base* and *overlays*.

Base

A base directory contains a `kustomization.yaml` file. The `kustomization.yaml` file has a list resource field to include all resource files. As the name implies, all resources in the base directory are a common resource set. You can create a base application by composing all common resources from the base directory.

The following diagram shows the structure of a base directory:

```
base
└── configmap.yaml
└── deployment.yaml
└── secret.yaml
└── service.yaml
└── route.yaml
└── kustomization.yaml
```

The base directory has YAML files to create configuration map, deployment, service, secret, and route resources. The base directory also has a `kustomization.yaml` file, such as the following example:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- configmap.yaml
- deployment.yaml
- secret.yaml
- service.yaml
- route.yaml
```

The `kustomization.yaml` file lists all resource files.

Overlays

Kustomize overlays declarative YAML artifacts, or patches, that override the general settings without modifying the original files. The overlay directory contains a `kustomization.yaml` file. The `kustomization.yaml` file can refer to one or more directories as bases. Multiple overlays can use a common base kustomization directory.

The following diagram shows the structure of all Kustomize directories:

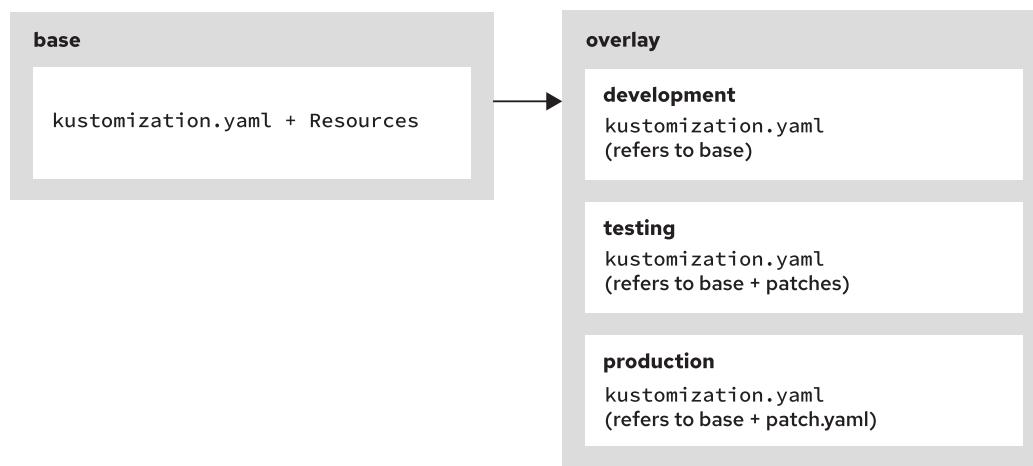


Figure 1.4: Kustomize file structure

The following example shows the directory structure of the `frontend-app` directory containing the `base` and `overlay` directories:

```
[user@host frontend-app]$ tree
base
├── configmap.yaml
├── deployment.yaml
├── secret.yaml
├── service.yaml
└── route.yaml
└── kustomization.yaml
overlay
```

```

└── development
    └── kustomization.yaml
└── testing
    └── kustomization.yaml
└── production
    ├── kustomization.yaml
    └── patch.yaml

```

The following example shows a `kustomization.yaml` file in the `overlays/development` directory:

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: dev-env
resources:
- ../../base

```

The `frontend-app/overlay/development/kustomization.yaml` file uses the base `kustomization` file at `../../base` to create all the application resources in the `dev-env` namespace.

Kustomize provides fields to set values for all resources in the `kustomization` file:

Field	Description
<code>namespace</code>	Set a specific namespace for all resources.
<code>namePrefix</code>	Add a prefix to the name of all resources.
<code>nameSuffix</code>	Add a suffix to the name of all resources.
<code>commonLabels</code>	Add labels to all resources and selectors.
<code>commonAnnotations</code>	Add annotations to all resources and selectors.

You can customize for multiple environments by using overlays and patching. The `patches` mechanism has two elements: `patch` and `target`.

Previously, Kustomize used the `PatchesJson6902` and `PatchesStrategicMerge` keys to add resource patches. These keys are deprecated in Kustomize version 5 and are replaced with a single key. However, the content of the `patches` key continues to use the same patch formats.

You can use JSON Patch and strategic merge patches. See the references section for further information about both patch formats.

The following is an example of a `kustomization.yaml` file in the `overlays/testing` directory:

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: test-env
patches: ①
- patch: |
  - op: replace ②
    path: /metadata/name

```

```

    value: frontend-test
  target: ③
    kind: Deployment
    name: frontend
- patch: |- ④
  - op: replace
    path: /spec/replicas
    value: 15
  target:
    kind: Deployment
    name: frontend
resources: ⑤
- ../../base
commonLabels: ⑥
  env: test

```

- ① The `patches` field contains a list of patches.
- ② The `patch` field defines operation, path, and value keys. In this example, the name changes to `frontend-test`.
- ③ The `target` field specifies the kind and name of the resource to apply the patch. In this example, you are changing the `frontend` deployment name to `frontend-test`.
- ④ This patch updates the number of replicas of the `frontend` deployment.
- ⑤ The `frontend-app/overlay/testing/kustomization.yaml` file uses the base kustomization file at `../../base` to create an application.
- ⑥ The `commonLabels` field adds the `env: test` label to all resources.

The `patches` mechanism also provides an option to include patches from a separate YAML file by using the `path` key.

The following example shows a `kustomization.yaml` file that uses a `patch.yaml` file:

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: prod-env
patches: ①
- path: patch.yaml ②
  target: ③
    kind: Deployment
    name: frontend
  options:
    allowNameChange: true ④
resources: ⑤
- ../../base
commonLabels: ⑥
  env: prod

```

- ① The `patches` field lists the patches that are applied by using a production kustomization file.
- ② The `path` field specifies the name of the patching YAML file.

- ③ The `target` field specifies the kind and name of the resource to apply the patch. In this example, you are targeting the `frontend` deployment.
- ④ The `allowNameChange` field enables kustomization to update the name by using a patch YAML file.
- ⑤ The `frontend-app/overlay/production/kustomization.yaml` file uses the base kustomization file at `..../base` to create an application.
- ⑥ The `commonLabels` field adds an `env: prod` label to all resources.

The `patch.yaml` file has the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-prod ①
spec:
  replicas: 5 ②
```

- ① The `metadata.name` field in the patch file updates the `frontend` deployment name to `frontend-prod` if the `allowNameChange` field is set to `true` in the kustomization YAML file.
- ② The `spec/relicas` field in the patch file updates the number of replicas of the `frontend-prod` deployment.

View and Deploy Resources by Using Kustomize

Run the `kubectl kustomize kustomization-directory` command to render the manifests without applying them to the cluster.

```
[user@host frontend-app]$ kubectl kustomize overlay/production
...output omitted...
kind: Deployment
metadata:
  labels:
    app: frontend
    env: prod
  name: frontend-prod
...output omitted...
spec:
  replicas: 5
  selector:
    matchLabels:
      app: frontend
      env: prod
...output omitted...
```

The `kubectl apply` command applies configurations to the resources in the cluster. If resources are not available, then the `kubectl apply` command creates resources. The `kubectl apply` command applies a kustomization with the `-k` flag.

```
[user@host frontend-app]$ kubectl apply -k overlay/production
deployment.apps/frontend-prod created
...output omitted...
```

Delete Resources by Using Kustomize

Run the `oc delete -k kustomization-directory` command to delete the resources that were deployed by using Kustomize.

```
[user@host frontend-app]$ oc delete -k overlay/production
configmap "database" deleted
secret "database" deleted
service "database" deleted
deployment.apps "database" deleted
```

Kustomize Generators

Configuration maps hold non-confidential data by using a key-value pair. Secrets are similar to configuration maps, but secrets hold confidential information such as usernames and passwords. Kustomize has `configMapGenerator` and `secretGenerator` fields that generate configuration map and secret resources.

The configuration map and secret generators can include content from external files in the generated resources. By keeping the content of the generated resources outside the resource definitions, you can use files that other tools generated, or that are stored in different systems. Generators help to manage the content of configuration maps and secrets, by taking care of encoding and including content from other sources.

Configuration Map Generator

Kustomize provides a `configMapGenerator` field to create a configuration map. The configuration map that a `configMapGenerator` field creates behaves differently from configuration maps that are created without a Kustomize file. With generated configuration maps, Kustomize appends a hash to the name, and any change in the configuration map triggers a rolling update.

The following example adds a configuration map by using the `configMapGenerator` field in the staging kustomization file. The `hello` application deployment has two environment variables to refer to the `hello-app-configmap` configuration map.

The `kustomization.yaml` file has the following content:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: hello-stage
resources:
- ../../base
configMapGenerator:
- name: hello-app-configmap
  literals:
    - msg="Welcome!"
    - enable="true"
```

The deployment.yaml file has the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
  labels:
    app: hello
    name: hello
spec:
  ...output omitted...
  spec:
    containers:
      - name: hello
        image: quay.io/hello-app:v1.0
        env:
          - name: MY_MESSAGE
            valueFrom:
              configMapKeyRef:
                name: hello-app-configmap
                key: msg
          - name: MSG_ENABLE
            valueFrom:
              configMapKeyRef:
                name: hello-app-configmap
                key: enable
```

You can view and deploy all resources and customizations that the kustomization YAML file defines, in the development directory.

```
[user@host hello-app]$ kubectl kustomize overlays/staging
apiVersion: v1
data:
  enable: "true"
  msg: Welcome!
kind: ConfigMap
metadata:
  name: hello-app-configmap-9tcmf95d77
  namespace: hello-stage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: hello
    name: hello
    name: hello
    namespace: hello-stage
spec:
  ...output omitted...
  spec:
    containers:
      - env:
          - name: MY_MESSAGE
```

```

valueFrom:
  configMapKeyRef:
    key: msg
    name: hello-app-configmap-9tcmf95d77
  - name: MSG_ENABLE
    valueFrom:
      configMapKeyRef:
        key: enable
        name: hello-app-configmap-9tcmf95d77
...output omitted...

[user@host hello-app]$ kubectl apply -k overlays/staging
configmap/hello-app-configmap-9tcmf95d77 created
deployment.apps/hello created

[user@host hello-app]$ oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/hello-75dc9cfc87-jh62k 1/1     Running   0          97s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello 1/1       1           1          97s

NAME                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hello 1         1         1         97s

```

The `kubectl apply -k` command creates a `hello-app-configmap-9tcmf95d77` configuration map and a `hello` deployment. Update the `kustomization.yaml` file with the configuration map values.

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: hello-stage
resources:
- ../../base
configMapGenerator:
- name: hello-app-configmap
  literals:
  - msg="Welcome Back!"
  - enable="true"

```

Then, apply the overlay with the `kubectl apply` command.

```

[user@host hello-app]$ kubectl apply -k overlays/staging
configmap/hello-app-configmap-696dm8h728 created
deployment.apps/hello configured

[user@host hello-app]$ oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/hello-55bc55ff9-hrszh 1/1     Running   0          3s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello 1/1       1           1          5m5s

```

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/hello-55bc55ff9	1	1	1	3s
replicaset.apps/hello-75dc9cf87	0	0	0	5m5s

The `kubectl apply -k` command applies kustomization. Kustomize appends a new hash to the configuration map name, which creates a `hello-app-configmap-696dm8h728` configuration map. The new configuration map triggers the generation of a new `hello-55bc55ff9-hrszh` pod.

You can generate a configuration map by using the `files` key from the `.properties` file or from the `.env` file by using the `envs` key with the file name as the value. You can also create a configuration map from a literal key-value pair by using the `literals` key.

The following example shows a `kustomization.yaml` file with the `configMapGenerator` field.

```
...output omitted...
configMapGenerator:
- name: configmap-1 ①
  files:
    - application.properties
- name: configmap-2 ②
  envs:
    - configmap-2.env
- name: configmap-3 ③
  literals:
    - name="configmap-3"
    - description="literal key-value pair"
```

- ① The `configmap-1` key is using the `application.properties` file.
- ② The `configmap-2` key is using the `configmap-2.env` file.
- ③ The `configmap-3` key is using a literal key-value pair.

The following example shows the `application.properties` file that is referenced in the `configmap-1` key.

```
Day=Monday
Enable=True
```

The following example shows the `configmap-2.env` file that is referenced in the `configmap-2` key.

```
Greet=Welcome
Enable=True
```

Run the `kubectl kustomize` command to view details of resources and customizations that the kustomization YAML file defines:

```
[user@host base]$ kubectl kustomize .
apiVersion: v1
data:
  application.properties: |
```

```

Day=Monday
Enable=True
kind: ConfigMap
metadata:
  name: configmap-1-5g2mh569b5 ①
---
apiVersion: v1
data:
  Enable: "True"
  Greet: Welcome
kind: ConfigMap
metadata:
  name: configmap-2-92m84tg9kt ②
---
apiVersion: v1
data:
  description: literal key-value pair
  name: configmap-3
kind: ConfigMap
metadata:
  name: configmap-3-k7g7d5bffd ③
---
...output omitted...

```

- ① The `configMapGenerator` field appends a hash to all `ConfigMap` resources. The `configmap-1-5g2mh569b5` configuration map is generated from the `application.properties` file, and the `data` field has a single key with the `application.properties` value.
- ② The `configmap-2-92m84tg9kt` configuration map is generated from the `configmap-2.env` file, and the `data` field has separate keys for each listed variable in the `configmap-2.env` file.
- ③ The `configmap-3-k7g7d5bffd` configuration map is generated from a literal key-value pair.

Secret Generator

A secret resource has sensitive data such as a username and a password. You can generate the secret by using the `secretGenerator` field. The `secretGenerator` field works similarly to the `configMapGenerator` field. However, the `secretGenerator` field also performs the base64 encoding that secret resources require.

The following example shows a `kustomization.yaml` file with the `secretGenerator` field:

```

...output omitted...
secretGenerator:
- name: secret-1 ①
  files:
    - password.txt
- name: secret-2 ②
  envs:
    - secret-mysql.env
- name: secret-3 ③

```

```
literals:
- MYSQL_DB=mysql
- MYSQL_PASS=root
```

- ① The secret-1 key is using the `password.txt` file.
- ② The secret-2 key is using the `secret-mysql.env` file.
- ③ The secret-3 key is using literal key-value pairs.

Generator Options

Kustomize provides a `generatorOptions` field to alter the default behavior of Kustomize generators. The `configMapGenerator` and `secretGenerator` fields append a hash suffix to the name of the generated resources.

Workload resources such as deployments do not detect any content changes to configuration maps and secrets. Any changes to a configuration map or secret do not apply automatically.

Because the generators append a hash, when you update the configuration map or secret, the resource name changes. This change triggers a rollout.

In some cases, the hash is not needed. Some operators observe the contents of the configuration maps and secrets that they use, and apply changes immediately. For example, the OpenShift OAuth operator applies changes to `htpasswd` secrets automatically. You can disable this feature with the `generatorOptions` field.

You can also add labels and annotations to the generated resources by using the `generatorOptions` field.

The following example shows the use of the `generatorOptions` field.

```
...output omitted...
configMapGenerator:
- name: my-configmap
  literals:
  - name="configmap-3"
    - description="literal key-value pair"
generatorOptions:
  disableNameSuffixHash: true
  labels:
    type: generated-disabled-suffix
  annotations:
    note: generated-disabled-suffix
```

You can use the `kubectl kustomize` command to render the changes to verify their effect.

```
[user@host base]$ kubectl kustomize .
apiVersion: v1
data:
  description: literal key-value pair
  name: configmap-3
kind: ConfigMap
metadata:
  annotations:
```

```
note: generated-disabled-suffix
labels:
  type: generated-disabled-suffix
name: my-configmap
```

The `my-configmap` configuration map is without a hash suffix, and has a label and annotations that are defined in the kustomization file.



References

Declarative Management of Kubernetes Objects Using Kustomize

<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/>

Your Guide to Continuous Delivery with OpenShift GitOps and Kustomize

<https://cloud.redhat.com/blog/your-guide-to-continuous-delivery-with-openshift-gitops-and-kustomize>

Customization of Kubernetes YAML Configurations

<https://github.com/kubernetes-sigs/kustomize/blob/master/api/types/kustomization.go>

JavaScript Object Notation (JSON) Patch

<https://www.rfc-editor.org/rfc/rfc6902>

Notes on the Strategic Merge Patch

<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/update-api-object-kubectl-patch/#notes-on-the-strategic-merge-patch>

► Guided Exercise

Kustomize Overlays

Deploy and update an application by applying different Kustomize overlays that are stored in a Git server.

Outcomes

- Deploy an application by using Kustomize from provided files.
- Apply an application update that changes a deployment.
- Deploy an overlay of the application that increases the number of replicas.

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.

```
[student@workstation ~]$ lab start declarative-kustomize
```

Instructions

- 1. Clone the v1.1.0 version of the application. Because this repository uses Git branches to represent application versions, you must use the v1.1.0 branch.

Clone the repository from the following URL:

<https://git.ocp4.example.com/developer/declarative-kustomize.git>

- 1.1. Change to the ~/D0280/labs/declarative-kustomize directory.

```
[student@workstation ~]$ cd D0280/labs/declarative-kustomize  
[student@workstation declarative-kustomize]$
```

- 1.2. Clone the initial version of the application.

```
[student@workstation declarative-kustomize]$ git clone \  
https://git.ocp4.example.com/developer/declarative-kustomize.git --branch v1.1.0  
Cloning into 'declarative-kustomize'...  
...output omitted...
```

- 1.3. Change to the repository directory.

```
[student@workstation declarative-kustomize]$ cd declarative-kustomize
```

- 2. Examine the first version of the application.

- 2.1. Use the `tree` command to review the structure of the repository.

```
[student@workstation declarative-kustomize]$ tree
.
├── base
│   ├── database ①
│   │   ├── configmap.yaml
│   │   ├── deployment.yaml
│   │   ├── kustomization.yaml
│   │   └── service.yaml
│   ├── exoplanets ②
│   │   ├── deployment.yaml
│   │   ├── kustomization.yaml
│   │   ├── route.yaml
│   │   └── service.yaml
│   └── kustomization.yaml ③
└── README.md

3 directories, 10 files
```

- ① The `database` base defines a configuration map, a deployment manifest, and a service definition to deploy a database.
- ② The `exoplanets` base defines a deployment manifest, a route definition, and a service definition to deploy an application that uses the database.
- ③ The repository has a `kustomization.yaml` file at the root, which uses two other bases.

2.2. Examine the `base/kustomization.yaml` file.

```
[student@workstation declarative-kustomize]$ cat base/kustomization.yaml
kind: Kustomization
resources: ①
- database
- exoplanets
secretGenerator: ②
- name: db-secrets
  literals: ③
  - DB_ADMIN_PASSWORD=postgres
  - DB_NAME=database
  - DB_PASSWORD=password
  - DB_USER=user
configMapGenerator: ④
- name: db-config
  literals: ⑤
  - DB_HOST=database
  - DB_PORT=5432
```

- ① The `base/kustomization.yaml` file uses the other two bases.
- ② The `secretGenerator` generator creates a secret from a file or from literal values. The database deployment manifest uses this secret.
- ③ The name and values of the credentials that are stored in the secret.

- ④ The configMapGenerator generator creates a configuration map from a file or from literal values. The exoplanets deployment manifest uses this configuration map.
 - ⑤ Key-value pairs of configuration data that are stored in the configuration map.
- 3. Deploy the base directory of the repository to a new declarative-kustomize project. Verify that the v1.1.0 version of the application is available at <http://exoplanets-declarative-kustomize.apps.ocp4.example.com>.
- 3.1. Log in to the OpenShift cluster as the developer user with the developer password.

```
[student@workstation declarative-kustomize]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

3.2. Create the declarative-kustomize project.

```
[student@workstation declarative-kustomize]$ oc new-project declarative-kustomize
...output omitted...
```

3.3. Use the oc apply -k command to deploy the application with Kustomize.

```
[student@workstation declarative-kustomize]$ oc apply -k base
configmap/database created
configmap/db-config-2d7thbcgk created
secret/db-secrets-55cbgc8c6m created
service/database created
service/exoplanets created
deployment.apps/database created
deployment.apps/exoplanets created
route.route.openshift.io/exoplanets created
```

3.4. Use the watch command to wait until the workloads are running.

```
[student@workstation declarative-kustomize]$ watch oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/database-55d6c77787-47649   1/1     Running   0          57s
pod/exoplanets-d6f57869d-jhkhc   1/1     Running   2 (54s ago) 57s

NAME            TYPE      CLUSTER-IP        EXTERNAL-IP      PORT(S)      AGE
service/database   ClusterIP   172.30.236.123   <none>           5432/TCP   57s
service/exoplanets   ClusterIP   172.30.248.130   <none>           8080/TCP   57s

NAME              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database   1/1       1           1           57s
deployment.apps/exoplanets   1/1       1           1           57s

NAME          DESIRED   CURRENT   READY   AGE

```

Chapter 1 | Declarative Resource Management

```
replicaset.apps/database-55d6c77787    1        1        1      57s
replicaset.apps/exoplanets-d6f57869d    1        1        1      57s
```

NAME	HOST/PORT	...
route.route.openshift.io/exoplanets		
exoplanets-declarative-kustomize.apps.ocp4.example.com		...

Press **Ctrl+C** to exit the `watch` command.

- 3.5. Open a web browser and navigate to `http://exoplanets-declarative-kustomize.apps.ocp4.example.com`.

2M 0746+20 b	2M 2140+16 b	2M 2206-20 b						
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
30	4640	0.97	20	7340	0.92	30	8686	1.3

51 Eri b	55 Cancri e	BD+20 594 b						
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
2	15000	1	0.0251	0.7365474	0.16728	0.05129	41.6855	0.1989

beta Pic b	CoRoT-10 b	CoRoT-11 b						
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
...

The browser displays the v1.1.0 version of the application.

- 4. Change to the v1.1.1 version of the application and examine the changes.

- 4.1. Change to the v1.1.1 branch.

```
[student@workstation declarative-kustomize]$ git checkout v1.1.1
branch 'v1.1.1' set up to track 'origin/v1.1.1'.
Switched to a new branch 'v1.1.1'
```

- 4.2. Use the `git show` command to display the last commit.

```
[student@workstation declarative-kustomize]$ git show
...output omitted...
diff --git a/base/exoplanets/deployment.yaml b/base/exoplanets/deployment.yaml
index 8bc4cf9..8389b69 100644
--- a/base/exoplanets/deployment.yaml
+++ b/base/exoplanets/deployment.yaml
@@ -23,7 +23,7 @@ spec:
      name: exoplanets
      - secretRef:
          name: exoplanets
-      image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.0
+      image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.1
```

```
imagePullPolicy: Always
livenessProbe:
  httpGet:
```

The v1.1.1 version updates the application to the v1.1.1 image in the base/exoplanets/deployment.yaml file.

```
[student@workstation declarative-kustomize]$ cat base/exoplanets/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: exoplanets
  name: exoplanets
...output omitted...
spec:
  containers:
    - envFrom:
        - configMapRef:
          name: db-config
    - secretRef:
      name: db-secrets
  image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.1
...output omitted...
```

- 5. Deploy the updated application and verify that the URL now displays the v1.1.1 version.

- 5.1. Use the oc apply -k command to execute the changes.

```
[student@workstation declarative-kustomize]$ oc apply -k base
...output omitted...
```

- 5.2. Use the watch command to wait until the application redeploys.

```
[student@workstation declarative-kustomize]$ watch oc get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/database-55d6c77787-47649  1/1     Running   0          57s
pod/exoplanets-d6f57869d-jhkhc 1/1     Running   2 (54s ago) 57s

NAME            TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/database ClusterIP  172.30.236.123  <none>           5432/TCP   57s
service/exoplanets ClusterIP 172.30.248.130  <none>           8080/TCP   57s

NAME              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database  1/1     1           1           57s
deployment.apps/exoplanets 1/1     1           1           57s

NAME              DESIRED  CURRENT   READY   AGE
replicaset.apps/database-55d6c77787  1        1        1      57s
replicaset.apps/exoplanets-d6f57869d  1        1        1      57s

NAME
```

```
HOST/PORT
route.route.openshift.io/exoplanets
exoplanets-declarative-kustomize.apps.ocp4.example.com ...
```

Press **Ctrl+C** to exit the `watch` command.

- 5.3. Open a web browser and navigate to `http://exoplanets-declarative-kustomize.apps.ocp4.example.com`.

2M 0746+20 b			2M 2140+16 b			2M 2206-20 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
30	4640	0.97	20	7340	0.92	30	8686	1.3
51 Eri b			55 Cancri e			BD+20 594 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
2	15000	1	0.0251	0.7365474	0.16728	0.05129	41.6855	0.1989
beta Pic b			CoRoT-10 b			CoRoT-11 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period

The browser displays the v1.1.1 version of the application.

- 6. Change to the v1.1.2 version of the application and examine the changes.

- 6.1. Change to the v1.1.2 branch.

```
[student@workstation declarative-kustomize]$ git checkout v1.1.2
branch 'v1.1.2' set up to track 'origin/v1.1.2'.
Switched to a new branch 'v1.1.2'
```

- 6.2. Use the `git show` command to display the last commit.

```
[student@workstation declarative-kustomize]$ git show
...output omitted...
diff --git a/base/kustomization.yaml b/base/kustomization.yaml
index fdf129a..8de16e8 100644
--- a/base/kustomization.yaml
+++ b/base/kustomization.yaml
@@ -7,7 +7,7 @@ secretGenerator:
    literals:
      - DB_ADMIN_PASSWORD=postgres
      - DB_NAME=database
-     - DB_PASSWORD=password
+     - DB_PASSWORD=newpassword
      - DB_USER=user
 configMapGenerator:
   - name: db-config
```

The v1.1.2 version updates the base kustomization. This update changes the password that the database uses. This change is possible because the sample application re-creates the database on startup.

```
[student@workstation declarative-kustomize]$ cat base/kustomization.yaml
kind: Kustomization
resources:
- database
- exoplanets
secretGenerator:
- name: db-secrets
  literals:
    - DB_ADMIN_PASSWORD=postgres
    - DB_NAME=database
    - DB_PASSWORD=newpassword
    - DB_USER=user
```

6.3. List the secrets in the namespace.

```
[student@workstation declarative-kustomize]$ oc get secret
NAME          TYPE        DATA   AGE
builder-dockercfg-qwn4v  kubernetes.io/dockercfg      1      4m31s
builder-token-z754n     kubernetes.io/service-account-token 4      4m31s
db-secrets-55cbgc8c6m  Opaque           4      4m28s
default-dockercfg-w4v89  kubernetes.io/dockercfg      1      4m31s
default-token-zw89c      kubernetes.io/service-account-token 4      4m31s
deployer-dockercfg-l8sct kubernetes.io/dockercfg      1      4m31s
deployer-token-knvhb    kubernetes.io/service-account-token 4      4m31s
```

When creating a secret, Kustomize appends a hash to the secret name.

6.4. Extract the contents of the secret. The name of the secret can change in your environment. Use the output from a previous step to learn the name of the secret.

```
[student@workstation declarative-kustomize]$ oc extract \
secret/db-secrets-55cbgc8c6m --to=-
# DB_PASSWORD
password
# DB_USER
user
# DB_ADMIN_PASSWORD
postgres
# DB_NAME
database
```

► 7. Deploy the updated application.

7.1. Use the `oc apply -k` command to execute the changes.

```
[student@workstation declarative-kustomize]$ oc apply -k base
configmap/database unchanged
configmap/db-config-2d7thbcgk unchanged
secret/db-secrets-6h668tk789 created
```

Chapter 1 | Declarative Resource Management

```
service/database unchanged
service/exoplanets unchanged
deployment.apps/database configured
deployment.apps/exoplanets configured
route.route.openshift.io/exoplanets configured
```

Because the password is different, Kustomize creates another secret. Kustomize also updates the two deployments that use the secret to use the new secret.

- 7.2. Use the `watch` command to wait until the application redeploys.

```
[student@workstation declarative-kustomize]$ watch oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/database-55d6c77787-47649   1/1     Running   0          57s
pod/exoplanets-d6f57869d-jhkhc   1/1     Running   2 (54s ago) 57s

NAME            TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/database   ClusterIP   172.30.236.123  <none>           5432/TCP   57s
service/exoplanets   ClusterIP   172.30.248.130  <none>           8080/TCP   57s

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database   1/1       1           1           57s
deployment.apps/exoplanets   1/1       1           1           57s

NAME          DESIRED   CURRENT   READY   AGE
replicaset.apps/database-55d6c77787   1         1         1         57s
replicaset.apps/exoplanets-d6f57869d   1         1         1         57s

NAME
HOST/PORT
route.route.openshift.io/exoplanets
exoplanets-declarative-kustomize.apps.ocp4.example.com ...
```

Press `Ctrl+C` to exit the `watch` command.

- 7.3. Open a web browser and navigate to `http://exoplanets-declarative-kustomize.apps.ocp4.example.com`.

2M 0746+20 b			2M 2140+16 b			2M 2206-20 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
30	4640	0.97	20	7340	0.92	30	8686	1.3

51 Eri b			55 Cancri e			BD+20 594 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
2	15000	1	0.0251	0.7365474	0.16728	0.05129	41.6855	0.1989

beta Pic b			CoRoT-10 b			CoRoT-11 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period

The browser continues showing the v1.1.1 version of the application.

7.4. Examine the deployment.

```
[student@workstation declarative-kustomize]$ oc get deployment exoplanets \
-o jsonpath='{.spec.template.spec.containers[0].envFrom}{"\n"}'
[{"configMapRef": {"name": "db-config-2d7thbcgkc"}}, {"secretRef": {"name": "db-secrets-6h668tk789"}}]
```

The deployment uses the new secret.

7.5. Examine the secret. Use the name of the secret from a previous step.

```
[student@workstation declarative-kustomize]$ oc extract \
secret/db-secrets-6h668tk789 --to=-
# DB_ADMIN_PASSWORD
postgres
# DB_NAME
database
# DB_PASSWORD
newpassword
# DB_USER
user
```

The deployment uses the changed password.

► 8. Change to the v1.1.3 version of the application and examine the changes.

8.1. Change to the v1.1.3 branch.

```
[student@workstation declarative-kustomize]$ git checkout v1.1.3
branch 'v1.1.3' set up to track 'origin/v1.1.3'.
Switched to a new branch 'v1.1.3'
```

8.2. Use the `git show` command to display the last commit.

```
[student@workstation declarative-kustomize]$ git show
...output omitted...
diff --git a/overlays/production/kustomization.yaml b/overlays/production/
kustomization.yaml
new file mode 100644
index 0000000..73bb7fe
--- /dev/null
+++ b/overlays/production/kustomization.yaml
@@ -0,0 +1,8 @@
+kind: Kustomization
+resources:
+- ../../base/
+patches:
+- path: patch-replicas.yaml
+ target:
+   kind: Deployment
+   name: exoplanets
diff --git a/overlays/production/patch-replicas.yaml b/overlays/production/patch-
replicas.yaml
new file mode 100644
```

Chapter 1 | Declarative Resource Management

```
index 0000000..a025aa0
--- /dev/null
+++ b/overlays/production/patch-replicas.yaml
@@ -0,0 +1,6 @@
+apiVersion: apps/v1
+kind: Deployment
+metadata:
+  name: exoplanets
+spec:
+  replicas: 2
```

The v1.1.3 version creates the `overlays` directory, and the `kustomization.yaml` and `patch-replicas.yaml` files, which add a production overlay that increases the number of replicas.

```
[student@workstation declarative-kustomize]$ cat
  overlays/production/kustomization.yaml
kind: Kustomization
resources:
- ../../base/
patches:
- path: patch-replicas.yaml
  target:
    kind: Deployment
    name: exoplanets

[student@workstation declarative-kustomize]$ cat
  overlays/production/patch-replicas.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: exoplanets
spec:
  replicas: 2
```

► 9. Deploy the updated application and verify the number of replicas.

- 9.1. Use the `oc apply -k` command to execute the changes.

```
[student@workstation declarative-kustomize]$ oc apply -k overlays/production
...output omitted...
```

- 9.2. Use the `watch` command to wait until the application redeploys.

```
[student@workstation declarative-kustomize]$ watch oc get all
NAME                  READY   STATUS    RESTARTS   AGE
pod/database-7dfb559cf7-rvxhx  1/1     Running   0          11m
pod/exoplanets-957bb5b48-5xl2d  1/1     Running   2 (11m ago)  11m
pod/exoplanets-957bb5b48-mgbrx  1/1     Running   0          19s

NAME            TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/database  ClusterIP   172.30.87.214  <none>           5432/TCP    19m
service/exoplanets ClusterIP  172.30.25.65   <none>           8080/TCP    19m
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/database	1/1	1	1	19m
deployment.apps/exoplanets	2/2	2	2	19m
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/database-7dfb559cf7	1	1	1	11m
replicaset.apps/database-d4cd8dcc	0	0	0	19m
replicaset.apps/exoplanets-6c7b4bb44c	0	0	0	19m
replicaset.apps/exoplanets-7ccb754c8b	0	0	0	18m
replicaset.apps/exoplanets-957bb5b48	2	2	2	11m
NAME	HOST/PORT
route.route.openshift.io/exoplanets	exoplanets-declarative-kustomize.apps.ocp4.example.com

Press **Ctrl+C** to exit the `watch` command. After you run the command, the application has two replicas.



Note

Unchanged resources are not restarted.

▶ 10. Delete the application.

10.1. Use the `oc delete -k` command to delete the resources that Kustomize manages.

```
[student@workstation declarative-kustomize]$ oc delete -k base
# Warning: 'bases' is deprecated. Please use 'resources' instead. Run 'kustomize
# edit fix' to update your Kustomization automatically.
configmap "database" deleted
configmap "db-config-2d7thbcgkc" deleted
secret "db-secrets-h9hdmt2g79" deleted
service "database" deleted
service "exoplanets" deleted
deployment.apps "database" deleted
deployment.apps "exoplanets" deleted
route.route.openshift.io "exoplanets" deleted
```

10.2. Change to the home directory.

```
[student@workstation declarative-kustomize]$ cd
[student@workstation ~]$
```

Finish

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

```
[student@workstation ~]$ lab finish declarative-kustomize
```

▶ Lab

Declarative Resource Management

Deploy and update applications from resource manifests that are parameterized for different target environments.

Outcomes

- Deploy an application by using Kustomize from provided files.
- Apply an application update that changes a deployment.
- Deploy an overlay of the application that increases the number of replicas.

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.

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

Instructions

1. Clone the v1.1.0 version of the application from the `https://git.ocp4.example.com/developer/declarative-review.git` URL. Because this repository uses Git branches to represent application versions, you must use the `v1.1.0` branch.
2. Examine the first version of the application.
3. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Deploy the base directory of the repository to a new `declarative-review` project. Verify that the v1.1.0 version of the application is available at `http://exoplanets-declarative-review.apps.ocp4.example.com`.
4. Change to the v1.1.1 version of the application and examine the changes.
5. Deploy the updated application and verify that the URL now displays the v1.1.1 version.
6. Examine the overlay in the `overlays/production` path.
7. Deploy the production overlay to a new `declarative-review-production` project. Verify that the v1.1.1 version of the application is available at `http://exoplanets-declarative-review-production.apps.ocp4.example.com` with two replicas.

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 grade declarative-review
```

Finish

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

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

► Solution

Declarative Resource Management

Deploy and update applications from resource manifests that are parameterized for different target environments.

Outcomes

- Deploy an application by using Kustomize from provided files.
- Apply an application update that changes a deployment.
- Deploy an overlay of the application that increases the number of replicas.

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.

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

Instructions

1. Clone the v1.1.0 version of the application from the `https://git.ocp4.example.com/developer/declarative-review.git` URL. Because this repository uses Git branches to represent application versions, you must use the v1.1.0 branch.
 - 1.1. Change to the `~/D0280/labs/declarative-review` directory.

```
[student@workstation ~]$ cd D0280/labs/declarative-review  
[student@workstation declarative-review]$
```

- 1.2. Clone the initial version of the application.

```
[student@workstation declarative-review]$ git clone \  
https://git.ocp4.example.com/developer/declarative-review.git --branch v1.1.0  
Cloning into 'declarative-review'...  
...output omitted...
```

- 1.3. Change to the repository directory.

```
[student@workstation declarative-review]$ cd declarative-review
```

2. Examine the first version of the application.

- 2.1. Use the `tree` command to review the structure of the repository.

```
[student@workstation declarative-review]$ tree
.
├── base
│   ├── database ❶
│   │   ├── configmap.yaml
│   │   ├── deployment.yaml
│   │   ├── kustomization.yaml
│   │   ├── secret.yaml
│   │   └── service.yaml
│   ├── exoplanets ❷
│   │   ├── configmap.yaml
│   │   ├── deployment.yaml
│   │   ├── kustomization.yaml
│   │   ├── route.yaml
│   │   ├── secret.yaml
│   │   └── service.yaml
│   └── kustomization.yaml ❸
└── overlays ❹
    └── production
        ├── kustomization.yaml
        └── patch-replicas.yaml

```

5 directories, 15 files

- ❶ The database base defines resources to deploy a database.
- ❷ The exoplanets base defines resources to deploy an application that uses the database.
- ❸ The repository has a `kustomization.yaml` file at the root, which uses two other bases.
- ❹ The repository also has a `production` overlay.

2.2. Examine the `base/kustomization.yaml` file.

```
[student@workstation declarative-review]$ cat base/kustomization.yaml
kind: Kustomization
resources:
- database
- exoplanets
```

The `base/kustomization.yaml` file uses the other two bases.

3. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Deploy the `base` directory of the repository to a new `declarative-review` project. Verify that the `v1.1.0` version of the application is available at `http://exoplanets-declarative-review.apps.ocp4.example.com`.
- 3.1. Log in to the OpenShift cluster as the `developer` user with the `developer` password.

Chapter 1 | Declarative Resource Management

```
[student@workstation declarative-review]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

3.2. Create the declarative-review project.

```
[student@workstation declarative-review]$ oc new-project declarative-review
...output omitted...
```

3.3. Use the `oc apply -k` command to deploy the application with Kustomize.

```
[student@workstation declarative-review]$ oc apply -k base
configmap/database created
configmap/exoplanets created
secret/database created
secret/exoplanets created
service/database created
service/exoplanets created
deployment.apps/database created
deployment.apps/exoplanets created
route.route.openshift.io/exoplanets created
```

3.4. Use the `watch` command to wait until the workloads are running.

```
[student@workstation declarative-review]$ watch oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/database-55d6c77787-47649   1/1     Running   0          57s
pod/exoplanets-d6f57869d-jhkhc   1/1     Running   2 (54s ago) 57s

NAME            TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/database   ClusterIP   172.30.236.123  <none>           5432/TCP   57s
service/exoplanets   ClusterIP   172.30.248.130  <none>           8080/TCP   57s

NAME            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database   1/1      1           1           57s
deployment.apps/exoplanets   1/1      1           1           57s

NAME            DESIRED   CURRENT   READY   AGE
replicaset.apps/database-55d6c77787   1         1         1         57s
replicaset.apps/exoplanets-d6f57869d   1         1         1         57s

NAME            HOST/PORT
route.../exoplanets   exoplanets-declarative-review.apps.ocp4.example.com ...
```

Press `Ctrl+C` to exit the `watch` command.

3.5. Open a web browser and navigate to `http://exoplanets-declarative-review.apps.ocp4.example.com`.

Exoplanets - v1.1.0

The planets listed here are a small subset of the known planets found outside of our solar system.

- Mass and radius are listed in "Jupiter mass" and "Jupiter radius" units.
- The orbital period is measured in "Earth days".

The full dataset is available from the [Open Exoplanet Catalogue](#).

2M 0746+20 b	2M 2140+16 b	2M 2206-20 b																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Mass</th> <th>Radius</th> <th>Period</th> </tr> </thead> <tbody> <tr> <td>30</td> <td>4640</td> <td>0.97</td> </tr> </tbody> </table>	Mass	Radius	Period	30	4640	0.97	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Mass</th> <th>Radius</th> <th>Period</th> </tr> </thead> <tbody> <tr> <td>20</td> <td>7340</td> <td>0.92</td> </tr> </tbody> </table>	Mass	Radius	Period	20	7340	0.92	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Mass</th> <th>Radius</th> <th>Period</th> </tr> </thead> <tbody> <tr> <td>30</td> <td>8686</td> <td>1.3</td> </tr> </tbody> </table>	Mass	Radius	Period	30	8686	1.3
Mass	Radius	Period																		
30	4640	0.97																		
Mass	Radius	Period																		
20	7340	0.92																		
Mass	Radius	Period																		
30	8686	1.3																		
51 Eri b	55 Cancri e	BD+20 594 b																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Mass</th> <th>Radius</th> <th>Period</th> </tr> </thead> <tbody> <tr> <td>2</td> <td>15000</td> <td>1</td> </tr> </tbody> </table>	Mass	Radius	Period	2	15000	1	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Mass</th> <th>Radius</th> <th>Period</th> </tr> </thead> <tbody> <tr> <td>0.0251</td> <td>0.7365474</td> <td>0.167280</td> </tr> <tr> <td>0.05129</td> <td>41.6855</td> <td>0.1989</td> </tr> </tbody> </table>	Mass	Radius	Period	0.0251	0.7365474	0.167280	0.05129	41.6855	0.1989	CoRoT-10 b			
Mass	Radius	Period																		
2	15000	1																		
Mass	Radius	Period																		
0.0251	0.7365474	0.167280																		
0.05129	41.6855	0.1989																		
beta Pic b	CoRoT-11 b																			

The browser displays version v1.1.0 of the application.

4. Change to the v1.1.1 version of the application and examine the changes.

4.1. Change to the v1.1.1 branch.

```
[student@workstation declarative-review]$ git checkout v1.1.1
branch 'v1.1.1' set up to track 'origin/v1.1.1'.
Switched to a new branch 'v1.1.1'
```

4.2. Use the git show command to display the last commit.

```
[student@workstation declarative-review]$ git show
...output omitted...
diff --git a/base/exoplanets/deployment.yaml b/base/exoplanets/deployment.yaml
index 8bc4cf9..8389b69 100644
--- a/base/exoplanets/deployment.yaml
+++ b/base/exoplanets/deployment.yaml
@@ -23,7 +23,7 @@ spec:
      name: exoplanets
      - secretRef:
          name: exoplanets
-     image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.0
+     image: registry.ocp4.example.com:8443/redhattraining/exoplanets:v1.1.1
      imagePullPolicy: Always
      livenessProbe:
        httpGet:
```

The v1.1.1 version updates the application to the v1.1.1 image.

5. Deploy the updated application and verify that the URL now displays the v1.1.1 version.

5.1. Use the oc apply -k command to execute the changes.

```
[student@workstation declarative-review]$ oc apply -k base
...output omitted...
```

- 5.2. Use the `watch` command to wait until the application redeploys.

```
[student@workstation declarative-review]$ watch oc get all
NAME                   READY   STATUS    RESTARTS   AGE
pod/database-55d6c77787-47649   1/1     Running   0          57s
pod/exoplanets-d6f57869d-jhkhc  1/1     Running   2 (54s ago) 57s

NAME            TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/database   ClusterIP   172.30.236.123 <none>           5432/TCP    57s
service/exoplanets ClusterIP   172.30.248.130  <none>           8080/TCP    57s

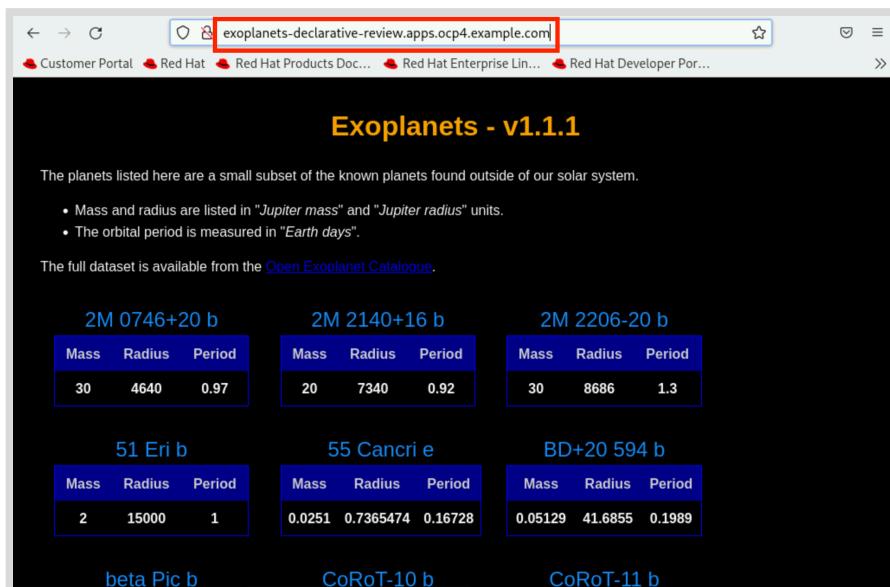
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database   1/1       1           1          57s
deployment.apps/exoplanets 1/1       1           1          57s

NAME             DESIRED  CURRENT  READY   AGE
replicaset.apps/database-55d6c77787  1         1         1      57s
replicaset.apps/exoplanets-d6f57869d  1         1         1      57s

NAME           HOST/PORT
route.../exoplanets  exoplanets-declarative-review.apps.ocp4.example.com  ...
...
```

Press `Ctrl+C` to exit the `watch` command.

- 5.3. Open a web browser and navigate to `http://exoplanets-declarative-review.apps.ocp4.example.com`.



The browser displays version v1.1.0 of the application.

6. Examine the overlay in the `overlays/production` path.

- 6.1. Examine the `overlays/production/kustomization.yaml` file.

```
[student@workstation declarative-review]$ cat \
  overlays/production/kustomization.yaml
kind: Kustomization
resources:
```

```
- ../../base/
patches:
- path: patch-replicas.yaml
  target:
    kind: Deployment
    name: exoplanets
```

This overlay applies a patch over the base.

6.2. Examine the overlays/production/patch-replicas.yaml file.

```
[student@workstation declarative-review]$ cat \
overlays/production/patch-replicas.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: exoplanets
spec:
  replicas: 2
```

This patch increases the number of replicas of the deployment, so that the production deployment can handle more users.

7. Deploy the production overlay to a new declarative-review-production project. Verify that the v1.1.1 version of the application is available at <http://exoplanets-declarative-review-production.apps.ocp4.example.com> with two replicas.

7.1. Create the declarative-review-production project.

```
[student@workstation declarative-review]$ oc new-project
declarative-review-production
Now using project "declarative-review-production" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

7.2. Use the oc apply -k command to deploy the overlay.

```
[student@workstation declarative-review]$ oc apply -k overlays/production
configmap/database created
configmap/exoplanets created
secret/database created
secret/exoplanets created
service/database created
service/exoplanets created
deployment.apps/database created
deployment.apps/exoplanets created
route.route.openshift.io/exoplanets created
```

7.3. Use the watch command to wait until the workloads are running.

```
[student@workstation declarative-review]$ watch oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/database-55d6c77787-b5x4n   1/1     Running   0          5m11s
pod/exoplanets-55666f556f-ndwkz  1/1     Running   2 (5m8s ago)  5m11s
```

pod/exoplanets-55666f556f-q7s7j	1/1	Running	2 (5m7s ago)	5m11s
<hr/>				
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/database	ClusterIP	172.30.24.165	<none>	5432/TCP
service/exoplanets	ClusterIP	172.30.90.176	<none>	8080/TCP
<hr/>				
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/database	1/1	1	1	5m11s
deployment.apps/exoplanets	2/2	2	2	5m11s
<hr/>				
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/database-55d6c77787	1	1	1	5m11s
replicaset.apps/exoplanets-55666f556f	2	2	2	5m11s
<hr/>				
NAME	HOST/PORT			
route.../exoplanets	exoplanets-declarative-review-production.apps.ocp4.example.com			

The exoplanets deployment has two replicas.

- 7.4. Open a web browser and navigate to <http://exoplanets-declarative-review-production.apps.ocp4.example.com>.

The planets listed here are a small subset of the known planets found outside of our solar system.

- Mass and radius are listed in "Jupiter mass" and "Jupiter radius" units.
- The orbital period is measured in "Earth days".

The full dataset is available from the [Open Exoplanet Catalogue](#).

2M 0746+20 b			2M 2140+16 b			2M 2206-20 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
30	4640	0.97	20	7340	0.92	30	8686	1.3

51 Eri b			55 Cancri e			BD+20 594 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period
2	15000	1	0.0251	0.7365474	0.16728	0.05129	41.6855	0.1989

beta Pic b			CoRoT-10 b			CoRoT-11 b		
Mass	Radius	Period	Mass	Radius	Period	Mass	Radius	Period

The browser displays version v1.1.1 of the application.

- 7.5. Change to the home directory.

```
[student@workstation declarative-review]$ cd
[student@workstation ~]$
```

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 grade declarative-review
```

Finish

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

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

Summary

- Imperative commands perform actions, such as creating a deployment, by specifying all necessary parameters as command-line arguments.
- In the declarative workflow, you create manifests that describe resources in the YAML or JSON formats, and use commands such as `kubectl apply` to deploy the resources to a cluster.
- Kubernetes provides tools, such as the `kubectl diff` command, to review your changes before applying them.
- You can use Kustomize to create multiple deployments from a single base code with different customizations.
- The `kubectl` command integrates Kustomize into the `apply` subcommand and others.
- Kustomize organizes content around bases and overlays.
- Bases and overlays can create and modify existing resources from other bases and overlays.

Chapter 2

Deploy Packaged Applications

Goal

Deploy and update applications from resource manifests that are packaged for sharing and distribution.

Objectives

- Deploy an application and its dependencies from resource manifests that are stored in an OpenShift template.
- Deploy and update applications from resource manifests that are packaged as Helm charts.

Sections

- OpenShift Templates (and Guided Exercise)
- Helm Charts (and Guided Exercise)

Lab

- Deploy Packaged Applications

OpenShift Templates

Objectives

- Deploy and update applications from resource manifests that are packaged as OpenShift templates.

OpenShift Templates

A template is a Kubernetes custom resource that describes a set of Kubernetes resource configurations. Templates can have parameters. You can create a set of related Kubernetes resources from a template by processing the template, and providing values for the parameters. Templates have varied use cases, and can create any Kubernetes resource. You can create a list of resources from a template by using the CLI or, if a template is uploaded to your project or to the global template library, by using the web console.

The template resource is a Kubernetes extension that Red Hat for OpenShift provides. The Cluster Samples Operator populates templates (and image streams) in the `openshift` namespace. You can opt out of adding templates during installation, and you can restrict the list of templates that the operator populates.

You can also create templates from scratch, or copy and customize a template to suit the needs of your project.

Discovering Templates

The templates that the Cluster Samples Operator provides are in the `openshift` namespace. Use the following `oc get` command to view a list of these templates:

```
[user@host ~]$ oc get templates -n openshift
NAME           DESCRIPTION      PARAMETERS   OBJECTS
cache-service   Red Hat Data Grid...  8 (1 blank)  4
cakephp-mysql-example An example CakePHP...  21 (4 blank)  8
cakephp-mysql-persistent An example CakePHP...  22 (4 blank)  9
...output omitted...
```

To evaluate any template, use the `oc describe template template-name -n openshift` command to view more details about the template, including the description, the labels that the template uses, the template parameters, and the resources that the template generates.

The following example shows the details of the `cache-service` template:

```
[user@host ~]$ oc describe template cache-service -n openshift
Name: cache-service
Namespace: openshift
Created: 2 months ago
Labels: samples.operator.openshift.io/managed=true
template=cache-service
Description: Red Hat Data Grid is an in-memory, distributed key/value store. ①
Annotations: iconClass=icon-datagrid
```

```
...output omitted...

Parameters: ②
  Name: APPLICATION_NAME
  Display Name: Application Name
  Description: Specifies a name for the application.
  Required: true
  Value: cache-service ③

...output omitted...

  Name: APPLICATION_PASSWORD
  Display Name: Client Password
  Description: Sets a password to authenticate client applications.
  Required: false
  Generated: expression ④
  From: [a-zA-Z0-9]{16}

Object Labels: template=cache-service ⑤

Message: <none>

Objects: ⑥
  Secret ${APPLICATION_NAME}
  Service ${APPLICATION_NAME}-ping
  Service ${APPLICATION_NAME}
  StatefulSet.apps ${APPLICATION_NAME}
```

- ① Use the description to determine the purpose of the template.
- ② The parameters provide deployment flexibility.
- ③ The value field provides a default value that you can override.
- ④ The Generated and From fields also generate default values.
- ⑤ The object labels are applied to all resources that the template creates.
- ⑥ The objects section lists the resources that the template creates.

In addition to using the `oc describe` command to view information about a template, the `oc process` command provides a `--parameters` option to view only the parameters that a template uses. For example, use the following command to view the parameters that the `cache-service` template uses:

```
[user@host ~]$ oc process --parameters cache-service -n openshift
NAME          ...  GENERATOR   VALUE
APPLICATION_NAME ...            cache-service
IMAGE          ...            registry.redhat.io/jboss-datagrid-7/...
NUMBER_OF_INSTANCES ...          1
REPLICATION_FACTOR ...          1
EVICTION_POLICY ...           evict
TOTAL_CONTAINER_MEM ...         512
APPLICATION_USER ...
APPLICATION_PASSWORD ...        expression  [a-zA-Z0-9]{16}
```

Use the `-f` option to view the parameters of a template that are defined in a file:

```
[user@host ~]$ oc process --parameters -f my-cache-service.yaml
```

Use the `oc get template template-name -o yaml -n namespace` command to view the manifest for the template. The following example retrieves the template manifest for the cache-service template:

```
[user@host ~]$ oc get template cache-service -o yaml -n openshift
apiVersion: template.openshift.io/v1
kind: Template
labels:
  template: cache-service
metadata:
  ...output omitted...
- apiVersion: v1
  kind: Secret
  metadata:
  ...output omitted...
- apiVersion: v1
  kind: Service
  metadata:
  ...output omitted...
- apiVersion: v1
  kind: Service
  metadata:
  ...output omitted...
- apiVersion: apps/v1
  kind: StatefulSet
  metadata:
  ...output omitted...
parameters:
- description: Specifies a name for the application.
  displayName: Application Name
  name: APPLICATION_NAME
  required: true
  value: cache-service
- description: Sets an image to bootstrap the service.
  name: IMAGE
  ...output omitted...
```

In the template manifest, examine how the template creates resources. The manifest is also a good resource for learning how to create your own templates.

Using Templates

The `oc new-app` command has a `--template` option that can deploy the template resources directly from the openshift project. The following example deploys the resources that are defined in the cache-service template from the openshift project:

```
[user@host ~]$ oc new-app --template=cache-service -p APPLICATION_USER=my-user
```

Using the `oc new-app` command to deploy the template resources is convenient for development and testing. However, for production usage, consume templates in a manner that helps resource and configuration tracking. For example, the `oc new-app` command can only create new resources, not update existing resources.

You can use the `oc process` command to apply parameters to a template, to produce manifests to deploy the templates with a set of parameters. The `oc process` command can process both templates that are stored in files locally, and templates that are stored in the cluster. However, to process templates in a namespace, you must have write permissions on the template namespace. For example, to run `oc process` on the templates in the `openshift` namespace, you must have write permissions on this namespace.



Note

Unprivileged users can read the templates in the `openshift` namespace by default. Those users can extract the template from the `openshift` namespace and create a copy in a project where they have wider permissions. By copying a template to a project, they can use the `oc process` command on the template.

Deploying Applications from Templates

The `oc process` command uses parameter values to transform a template into a set of related Kubernetes resource manifests. For example, the following command creates a set of resource manifests for the `my-cache-service` template. When you use the `-o yaml` option, the resulting manifests are in the YAML format. The example writes the manifests to a `my-cache-service-manifest.yaml` file:

```
[user@host ~]$ oc process my-cache-service \
-p APPLICATION_USER=user1 -o yaml > my-cache-service-manifest.yaml
```

The previous example uses the `-p` option to provide a parameter value to the only required parameter without a default value.

Use the `-f` option with the `oc process` command to process a template that is defined in a file:

```
[user@host ~]$ oc process -f my-cache-service.yaml \
-p APPLICATION_USER=user1 -o yaml > my-cache-service-manifest.yaml
```

Use the `-p` option with *key=value* pairs with the `oc process` command to use parameter values that override the default values. The following example passes three parameter values to the `my-cache-service` template, and overrides the default values of the specified parameters:

```
[user@host ~]$ oc process my-cache-service -o yaml \
-p TOTAL_CONTAINER_MEM=1024 \
-p APPLICATION_USER='cache-user' \
-p APPLICATION_PASSWORD='my-secret-password' \
> my-cache-service-manifest.yaml
```

Instead of specifying parameters on the command line, place the parameters in a file. This option cleans up the command line when many parameter values are required. Save the parameters file in a version control system to keep records of the parameters that are used in production deployments.

For example, instead of using the command-line options in the previous examples, place the key-value pairs in a `my-cache-service-params.env` file. Add the key-value pairs to the file, with each pair on a separate line:

```
TOTAL_CONTAINER_MEM=1024  
APPLICATION_USER='cache-user'  
APPLICATION_PASSWORD='my-secret-password'
```

The corresponding `oc process` command uses the `--param-file` option to pass the parameters as follows:

```
[user@host ~]$ oc process my-cache-service -o yaml \  
--param-file=my-cache-service-params.env > my-cache-service-manifest.yaml
```

Generating a manifest file is not required to use templates. Instead, pipe the output of the `oc process` command directly to the input for the `oc apply -f -` command. The `oc apply` command creates live resources on the Kubernetes cluster.

```
[user@host ~]$ oc process my-cache-service \  
--param-file=my-cache-service-params.env | oc apply -f -
```

Because templates are flexible, you can use the same template to create different resources by changing the input parameters.

Updating Apps from Templates

Because you use the `oc apply` command, after deploying a set of manifests from a template, you can process the template again and use `oc apply` for updates. This procedure can make simple changes to deployed templates, such as changing a parameter. However, many workload updates are not possible with this mechanism. To manage more complex applications, consider using other mechanisms such as Helm charts, which are described elsewhere in this course.

To compare the results of applying a different parameters file to a template against the live resources, pipe the manifest to the `oc diff -f -` command. For example, given a second parameter file named `my-cache-service-params-2.env`, use the following command:

```
[user@host ~]$ oc process my-cache-service -o yaml \  
--param-file=my-cache-service-params-2.env | oc diff -f -  
...output omitted...  
- generation: 1  
+ generation: 2  
  labels:  
    application: cache-service  
    template: cache-service  
@@ -86,10 +86,10 @@  
      timeoutSeconds: 10  
    resources:  
      limits:  
-        memory: 1Gi  
+        memory: 2Gi  
      requests:  
        cpu: 500m  
-        memory: 1Gi
```

```
+     memory: 2Gi
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
volumeMounts:
```

In this case, the configuration change increases the memory usage of the application. The output shows that the second generation uses `2Gi` of memory instead of `1Gi`.

After verifying that the changes are what you intend, you can pipe the output of the `oc process` to the `oc apply -f -` command.

Managing Templates

For production usage, make a customized copy of the template, to change the default values of the template to suitable values for the target project. To copy a template into your project, use the `oc get template` command with the `-o yaml` option to copy the template YAML to a file.

The following example copies the `cache-service` template from the `openshift` project to a YAML file named `my-cache-service.yaml`:

```
[user@host ~]$ oc get template cache-service -o yaml \
-n openshift > my-cache-service.yaml
```

After creating a YAML file for a template, consider making the following changes to the template:

- Give the template a new name that is specific to the target use of the template resources.
- Apply appropriate changes to the parameter default values at the end of the file.
- Remove the `namespace` field of the `template` resource.

You can process templates in other namespaces, if you can create the processed template resource in those namespaces. Processing the template in a different project without changing the template namespace to match the target namespace gives an error. Optionally, you can also delete the `namespace` field from the `metadata` field of the `template` resource.

After you have a YAML file for a template, use the `oc create -f` command to upload the template to the current project. In this case, the `oc create` command is not creating the resources that the template defines. Instead, the command is creating a `template` resource in the project. Using a template that is uploaded to a project clarifies which template provides the resource definitions of a project. After uploading, the template is available to anyone with access to the project.

The following example uploads a customized template that is defined in the `my-cache-service.yaml` file to the current project:

```
[user@host ~]$ oc create -f my-cache-service.yaml
```

Use the `-n namespace` option to upload the template to a different project. The following example uploads the template that is defined in the `my-cache-service.yaml` file to the `shared-templates` project:

```
[user@host ~]$ oc create -f my-cache-service.yaml -n shared-templates
```

Use the `oc get templates` command to view a list of available templates in the project:

```
[user@host ~]$ oc get templates -n shared-templates
NAME                  DESCRIPTION          PARAMETERS      OBJECTS
my-cache-service      Red Hat Data Grid...    8 (1 blank)     4
```



References

For more information, refer to the *Understanding Templates* section in the *Using Templates* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/images/index#templates-overview_using-templates

For more information, refer to the *OpenShift CLI Developer Command Reference* section in the *OpenShift CLI (oc)* chapter in the Red Hat OpenShift Container Platform 4.14 *CLI Tools* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/cli_tools/index#cli-developer-commands

Kubernetes Documentation – kubectl Commands

<https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands>

► Guided Exercise

OpenShift Templates

Deploy and update an application from a template that is stored in another project.

Outcomes

- Deploy and update an application from a template.

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 all resources are available for this exercise.

```
[student@workstation ~]$ lab start packaged-templates
```

Instructions

- 1. Log in to the OpenShift cluster as the developer user with the developer password.

- Log in to the OpenShift cluster.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 2. Examine the available templates in the cluster, in the `openshift` project. Identify an appropriate template to deploy a MySQL database.

- Use the `get` command to retrieve a list of templates in the cluster, in the `openshift` project.

```
[student@workstation ~]$ oc get templates -n openshift
NAME                  DESCRIPTION          PARAMETERS        OBJECTS
...output omitted...
mysql-ephemeral       MySQL database...    8 (3 generated)  3
mysql-persistent       MySQL database...    9 (3 generated)  4
...output omitted...
```

- Use the `oc process --parameters` command to view the parameters of the `mysql-persistent` template.

```
[student@workstation ~]$ oc process --parameters mysql-persistent \
-n openshift
NAME                  DESCRIPTION      GENERATOR      VALUE
MEMORY_LIMIT          ...
```

NAMESPACE	...	openshift
DATABASE_SERVICE_NAME	...	mysql
MYSQL_USER	...	user[A-Z0-9]{3}
MYSQL_PASSWORD	...	[a-zA-Z0-9]{16}
MYSQL_ROOT_PASSWORD	...	[a-zA-Z0-9]{16}
MYSQL_DATABASE	...	sampled
VOLUME_CAPACITY	...	1Gi
MYSQL_VERSION	...	8.0-e18

All the required parameters have either default values or generated values.

- 3. Use the `mysql-persistent` template to deploy a database by processing the template.

3.1. Create the packaged-templates project.

```
[student@workstation ~]$ oc new-project packaged-templates
Now using project "packaged-templates" on server ...
...output omitted...
```

3.2. Use the `oc new-app` command to deploy the application.

```
[student@workstation ~]$ oc new-app --template=mysql-persistent \
-p MYSQL_USER=user1 \
-p MYSQL_PASSWORD=mypasswd
--> Deploying template "packaged-templates/mysql-persistent" to project packaged-templates
...output omitted...
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose service/mysql'
Run 'oc status' to view your app.
```

3.3. Use the `watch` command to verify that the pods are running. Wait for the `mysql-1-deploy` pod to show a `Completed` status. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation ~]$ watch oc get pods
NAME        READY   STATUS    RESTARTS   AGE
mysql-1-5t8h8   1/1     Running      0          83s
mysql-1-deploy   0/1     Completed     0          84s
```

3.4. Connect to the database to verify that it is working.

```
[student@workstation ~]$ oc run query-db -it --rm \
--image registry.ocp4.example.com:8443/rhel8/mysql-80 \
--restart Never --command -- \
/bin/bash -c \
"mysql -uuser1 -pmypasswd --protocol tcp \
-h mysql -P3306 sampledb -e 'SHOW DATABASES;'""
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| Database           |
```

```
+-----+
| information_schema |
| performance_schema |
| sampledb           |
+-----+
pod "query-db" deleted
```

The query-db pod uses the mysql command from the mysql-80 image to send the SHOW DATABASES; query. The --rm option deletes the pod after execution terminates.

- ▶ 4. Deploy the application from the custom template, in the ~/D0280/labs/packaged-templates/custom-template/roster-template.yaml file, to the project. The application initializes and uses the database that the mysql-persistent template deployed.

- 4.1. Upload the custom template to the project.

```
[student@workstation ~]$ oc create -f \
~/D0280/labs/packaged-templates/custom-template/roster-template.yaml
template.template.openshift.io/roster-template created
```

- 4.2. Use oc get templates to view the available templates in the packaged-templates project.

```
[student@workstation ~]$ oc get templates
NAME          DESCRIPTION          PARAMETERS   OBJECTS
roster-template Example application for D0280... 8 (2 blank) 4
```

- 4.3. Use the oc process --parameters command to view the parameter of the roster-template template.

```
[student@workstation ~]$ oc process --parameters roster-template
NAME          DESCRIPTION  GENERATOR  VALUE
IMAGE         ...          registry.../do280-roster:v1
APPNAME       ...          do280-roster
NAMESPACE     ...          packaged-templates
DATABASE_SERVICE_NAME ...          mysql
MYSQL_USER    ...
MYSQL_PASSWORD ...
MYSQL_DATABASE ...
INIT_DB       ...          False
```

- 4.4. Use the oc process command to generate the manifests for the roster-template application resources, and use the oc apply command to create the resources in the Kubernetes cluster.

You must use the same database credentials that you used in an earlier step to configure the database, so that the application can access the database.

```
[student@workstation ~]$ oc process roster-template \
-p MYSQL_USER=user1 -p MYSQL_PASSWORD=mypasswd -p INIT_DB=true | oc apply -f -
...output omitted...
secret/mysql configured
deployment.apps/do280-roster created
service/do280-roster created
route.route.openshift.io/do280-roster created
```

- 4.5. Use the `oc get pods` command to confirm that the application is running.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
do280-roster-c7f596dd8-pqvlv   1/1     Running   0          60s
mysql-1-bl97v           1/1     Running   0          33m
mysql-1-deploy          0/1     Completed  0          33m
```

- 4.6. Use the `oc get routes` command to view the routes.

```
[student@workstation ~]$ oc get routes
NAME            HOST/PORT
do280-roster   do280-roster-packaged-templates.apps.ocp4.example.com ...
```

- 4.7. Open the application URL in the web browser. The header confirms the use of **version 1** of the application.

`http://do280-roster-packaged-templates.apps.ocp4.example.com`

- 4.8. Enter your information in the form and save it to the database.

- 5. Deploy an updated version of the `do280/roster` application from the custom template in the `roster-template` template. Use version 2 of the application and do not overwrite the data in the database.

- 5.1. Create a text file named `roster-parameters.env` with the following content:

```
MYSQL_USER=user1
MYSQL_PASSWORD=mypasswd
IMAGE=registry.ocp4.example.com:8443/redhattraining/do280-roster:v2
```

The option of using a parameter file helps version control software to track changes.

- 5.2. Use the `oc process` command and the `oc diff` command to view the changes in the new manifests when compared to the live application.

```
[student@workstation ~]$ oc process roster-template \
--param-file=roster-parameters.env | oc diff -f -
diff -u -N ...output omitted...
--- /tmp/LIVE-1948327112/apps.v1.Deployment.packaged-templates...
+++ /tmp/MERGED-2797490080/apps.v1.Deployment.packaged-templates...
...output omitted...
      key: database-service
      name: mysql
- name: INIT_DB
```

```
-      value: "true"
-      image: registry.ocp4.example.com:8443/redhattraining/do280-roster:v1
+      value: "False" ①
+      image: registry.ocp4.example.com:8443/redhattraining/do280-roster:v2 ②
      imagePullPolicy: IfNotPresent
      name: do280-roster-image
      ports:
```

- ① The INIT_DB environment variable determines whether the application initializes the database. The default False value is used when the parameter is omitted. In the first deployment, the INIT_DB variable was set to the True value, so the database was initialized. In this second deployment, the deployment does not have to initialize the database again.
- ② The IMAGE parameter changes the image that the template uses.

- 5.3. Use the oc process command to generate the manifests for the roster-template application objects, and use the oc apply command to create the application objects. With the changes from a previous step, you use the IMAGE variable to use a different image for the update and omit the INIT_DB variable.

```
[student@workstation ~]$ oc process roster-template \
--param-file=roster-parameters.env | oc apply -f -
secret/mysql configured
deployment.apps/do280-roster configured
service/do280-roster unchanged
route.route.openshift.io/do280-roster unchanged
```

- 5.4. Use watch to verify that the pods are running. Wait for the mysql-1-deploy pod to show a Completed status. Press Ctrl+C to exit the watch command.

```
[student@workstation ~]$ watch oc get pods
NAME          READY   STATUS    RESTARTS   AGE
do280-roster-c7f596dd8-ktlvl   1/1     Running   0          60s
mysql-1-bl97v        1/1     Running   0          53m
mysql-1-deploy       0/1     Completed  0          53m
```

- 5.5. Open the application URL in the web browser. The route is unchanged, so you can refresh the previous browser page if the page is still open. The header confirms the use of **version 2** of the application. The data that is pulled from the database is unchanged.

<http://do280-roster-packaged-templates.apps.ocp4.example.com>

Finish

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

```
[student@workstation ~]$ lab finish packaged-templates
```

Helm Charts

Objectives

- Deploy and update applications from resource manifests that are packaged as Helm charts.

Helm

Helm is an open source application that helps to manage the lifecycle of Kubernetes applications.

Helm introduces the concept of *charts*. A chart is a package that describes a set of Kubernetes resources that you can deploy. Helm charts define values that you can customize when deploying an application. Helm includes functions to distribute charts and updates.

Many organizations distribute Helm charts to deploy applications. Often, Helm is the supported mechanism to deploy a specific application.

However, Helm does not cover all needs to manage certain kinds of applications. Operators have a more complete model that can handle the lifecycle of more complex applications. For more details about operators, refer to *Kubernetes Operators and the Operator Lifecycle Manager*.

Helm Charts

A Helm chart defines Kubernetes resources that you can deploy. A chart is a collection of files with a defined structure. These files include chart metadata (such as the chart name or version), resource definitions, and supporting material.

Chart authors can use the template feature of the Go language for the resource definitions. For example, instead of specifying the image for a deployment, charts can use user-provided values for the image. By using values to choose an image, cluster administrators can replace a default public image with an image from a private repository.

The following diagram shows the structure of a minimal Helm chart:

```
sample/
  └── Chart.yaml ①
  └── templates ②
    └── example.yaml
  └── values.yaml ③
```

- The `Chart.yaml` file contains chart metadata, such as the name and version of the chart.
- The `templates` directory contains files that define application resources such as deployments.
- The `values.yaml` file contains default values for the chart.

Helm charts can contain hooks that Helm executes at different points during installations and upgrades. Hooks can automate tasks for installations and upgrades. With hooks, Helm charts can manage more complex applications than purely manifest-based processes. Review the chart documentation to learn about the chart hooks and their implications.

Using Helm Charts

Helm is a command-line application. The `helm` command interacts with the following entities:

Charts

Charts are the packaged applications that the `helm` command deploys.

Releases

A *release* is the result of deploying a chart. You can deploy a chart many times to the same cluster. Each deployment is a different release.

Versions

A Helm chart can have many versions. Chart authors can release updates to charts, to adapt to later application versions, introduce new features, or fix issues.

You can use and refer to charts in various ways. For example, if your local file system contains a chart, then you can refer to that chart by using the path to the chart directory. You can also use a path or a URL that contains a chart that is packaged in a tar archive with `gzip` compression.

Inspecting Helm Charts

Use the `helm show` command to display information about a chart. The `show chart` subcommand displays general information, such as the maintainers, or the source URL.

```
[user@host ~]$ helm show chart chart-reference
apiVersion: v1
description: A Helm chart for Kubernetes
name: examplechart
version: 0.1.0
maintainers:
- email: dev@example.com
  name: Developer
sources:
- https://git.example.com/examplechart
```

The `show values` subcommand displays the default values for the chart. The output is in YAML format and comes from the `values.yaml` file in the chart.

```
[user@host ~]$ helm show values chart-reference
image:
  repository: "sample"
  tag: "1.8.10"
  pullPolicy: IfNotPresent
...output omitted...
```

Chart resources use the values from the `values.yaml` file by default. You can override these default values. You can use the output of the `show values` command to discover customizable values.

Installing Helm Charts

After inspecting the chart, you can deploy the resources in the chart by using the `helm install` command. In Helm, *install* refers to deploying the resources in a chart to create a release.

Always refer to the documentation of the chart before installation to learn about prerequisites, extra installation steps, and other information.

To install a chart, you must decide on the following parameters:

- The deployment target namespace
- The values to override
- The release name

Helm charts can contain Kubernetes resources of any kind. These resources can be namespaced or non-namespaced. Like normal resource definitions, namespaced resources in charts can define or omit a namespace declaration.

Most Helm charts that deploy applications do not create a namespace, and namespaced resources in the chart omit a namespace declaration. Typically, when deploying a chart that follows this structure, you create a namespace for the deployment, and Helm creates namespaced resources in this namespace.

After deciding the target namespace, you can design the values to use. Inspect the documentation and the output of the `helm show values` command to decide which values to override.

You can define values by writing a YAML file that contains them. This file can follow the structure from the output of the `helm show values` command, which contains the default values. Specify only the values to override.

Consider the following output from the `helm show values` command for an example chart:

```
image:  
  repository: "sample"  
  tag: "1.8.10"  
  pullPolicy: IfNotPresent
```

Create a `values.yaml` file without the `image` key if you do not want to override any image parameters. Omit the `pullPolicy` key to override the `tag` key but not the pull policy. For example, the following YAML file would override only the image tag:

```
image:  
  tag: "1.8.10-patched"
```

Besides the YAML file, you can override specific values by using command-line arguments.

The final element to prepare a chart deployment is choosing a release name. You can deploy a chart many times to a cluster. Each chart deployment must have a unique release name for identification purposes. Many Helm charts use the release name to construct the name of the created resources.

With the namespace, values, and release name, you can start the deployment process. The `helm install` command creates a release in a namespace, with a set of values.

Rendering Manifests from a Chart

You can use the `--dry-run` option to preview the effects of installing a chart.

```
[user@host ~]$ helm install release-name chart-reference --dry-run \
--values values.yaml
NAME: release-name ①
LAST DEPLOYED: Tue May 30 13:14:57 2023
NAMESPACE: current-namespace
STATUS: pending-install
REVISION: 1
TEST SUITE: None
HOOKS:
MANIFEST:
---
# Source: chart/templates/serviceaccount.yaml ②
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-release-sa
  labels:
...output omitted...

NOTES: ③
The application can be accessed via port 1234.
...output omitted...
```

- ① General information about the new release
- ② A list of the resources that the `helm install` command would create
- ③ Additional information



Note

You define values to use for the installation with the `--values values.yaml` option. In this file, you override the default values from the chart that are defined in the `values.yaml` file that the chart contains.

Often, chart resource names include the release name. In the example output of the `helm install` command, the service account is a combination of the release name and the `-sa` text.

Chart authors can provide installation notes that use the chart values. In the same example, the port number in the notes reflects a value from the `values.yaml` file.

If the preview looks correct, then you can run the same command without the `--dry-run` option to deploy the resources and create the release.

Releases

When the `helm install` command runs successfully, besides creating the resources, Helm creates a release. Helm stores information about the release as a secret of the `helm.sh/release.v1` type.

Inspecting Releases

Use the `helm list` command to inspect releases on a cluster.

```
[user@host ~]$ helm list
NAME      NAMESPACE   REVISION  ... STATUS      CHART          APP VERSION
my-release  example     1          ... deployed  example-4.12.1  1.8.10
```

Similarly to `kubectl` commands, many `helm` commands have the `--all-namespaces` and `--namespace` options. The `helm list` command without options lists releases in the current namespace. If you use the `--all-namespaces` option, then it lists releases in all namespaces. If you use the `--namespace` option, then it lists releases in a single namespace.



Warning

Do not manipulate the release secret. If you remove the secret, then Helm cannot operate with the release.

Upgrading Releases

The `helm upgrade` command can apply changes to existing releases, such as updating values or the chart version.



Important

By default, this command automatically updates releases to use the latest version of the chart.

The `helm upgrade` command uses similar arguments and options to the `helm install` command. However, the `helm upgrade` command interacts with existing resources in the cluster instead of creating resources from a blank state. Therefore, the `helm upgrade` command can have more complex effects, such as conflicting changes. Always review the chart documentation when using a later version of a chart, and when changing values. You can use the `--dry-run` option to preview the manifests that the `helm upgrade` command uses, and compare them to the running resources.

Rolling Back Helm Upgrades

Helm keeps a log of release upgrades, to review changes and roll back to previous releases.

You can review this log by using the `helm history` command:

```
[user@host ~]$ helm history release_name
REVISION  UPDATED        STATUS      CHART          APP VERSION  DESCRIPTION
1          Wed May 31...  superseded  chart-0.0.6  latest       Install complete
2          Wed May 31...  deployed    chart-0.0.7  latest       Upgrade complete
```

You can use the `helm rollback` command to revert to an earlier revision:

```
[user@host ~]$ helm rollback release_name revision
Rollback was a success! Happy Helming!
```

Rolling back can have greater implications than upgrading, because upgrades might not be reversible. If you keep a test environment with the same upgrades as a production environment, then you can test rollbacks before performing them in the production environment to find potential issues.

Helm Repositories

Charts can be distributed as files, archives, or container images, or by using chart repositories.

The `helm repo` command provides the following subcommands to work with chart repositories.

Subcommand	Description
<code>add NAME REPOSITORY_URL</code>	Add a Helm chart repository.
<code>list</code>	List Helm chart repositories.
<code>update</code>	Update Helm chart repositories.
<code>remove REPOSITORY1_NAME REPOSITORY2_NAME ...</code>	Remove Helm chart repositories.

The following command adds a repository:

```
[user@host ~]$ helm repo add \
  openshift-helm-charts https://charts.openshift.io/
"openshift-helm-charts" has been added to your repositories
```

This command and other repository commands change only local configuration, and do not affect any cluster resources. The `helm repo add` command updates the `~/.config/helm/repositories.yaml` configuration file, which keeps the list of configured repositories.

When repositories are configured, other commands can use the list of repositories to perform actions. For example, the `helm search repo` command lists all available charts in the configured repositories:

```
[user@host ~]$ helm search repo
NAME      CHART VERSION   APP VERSION   DESCRIPTION
repo/chart  0.0.7          latest        A sample chart
...output omitted...
```

By default, the `helm search repo` command shows only the latest version of a chart. Use the `--versions` option to list all available versions. By default, the `install` and `upgrade` commands use the latest version of the chart in the repository. You can use the `--version` option to install specific versions.



References

Using Helm

https://helm.sh/docs/intro/using_helm/

Helm Charts

<https://helm.sh/docs/topics/charts/>

Helm Chart Repository Guide

https://helm.sh/docs/topics/chart_repository/

► Guided Exercise

Helm Charts

Deploy and update an application from a chart that is stored in a catalog.

Outcomes

- Deploy an application and its dependencies from a Helm chart.
- Customize the deployment, including scaling and using a custom image.

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.

```
[student@workstation ~]$ lab start packaged-charts
```

Instructions

- 1. Add the classroom Helm repository at the following URL and examine its contents.

`http://helm.ocp4.example.com/charts`

- 1.1. Use the `helm repo list` command to list the repositories that are configured for the student user.

```
[student@workstation ~]$ helm repo list
Error: no repositories to show
```

If the `do280-repo` repository is present, then continue to the next step. Otherwise, add the repository.

```
[student@workstation ~]$ helm repo add do280-repo \
http://helm.ocp4.example.com/charts
"do280-repo" has been added to your repositories
```

- 1.2. Use the `helm search` command to list all the chart versions in the repository.

```
[student@workstation ~]$ helm search repo --versions
NAME          CHART VERSION APP VERSION ...
do280-repo/etherpad  0.0.7      latest      ...
do280-repo/etherpad  0.0.6      latest      ...
...output omitted...
```

The `etherpad` chart has the 0.0.7 and 0.0.6 versions. This chart is a copy of a chart from the <https://github.com/redhat-cop/helm-charts> repository.

- ▶ 2. Install the 0.0.6 version of the etherpad chart to a new packaged-charts-development project, with the example-app release name.

Use the `registry.ocp4.example.com:8443/etherpad/etherpad:1.8.18` image in the offline classroom registry. Expose the application at the `https://development-etherpad.apps.ocp4.example.com` URL.

- 2.1. Examine the values of the chart.

```
[student@workstation ~]$ helm show values do280-repo/etherpad --version 0.0.6
# Default values for etherpad.
replicaCount: 1

defaultTitle: "Labs Etherpad"
defaultText: "Assign yourself a user and share your ideas!"

image:
  repository: etherpad
  name:
  tag:
  pullPolicy: IfNotPresent
...output omitted...
route:
  enabled: true
  host: null
  targetPort: http
...output omitted...
resources: {}
...output omitted...
```

You can configure the image, the replica count, and other values. By default, the chart creates a route. You can customize the route with the `route.host` key.

With the default configuration, the chart uses the `docker.io/etherpad/etherpad:latest` image. The classroom environment is designed for offline use. Use the `registry.ocp4.example.com:8443/etherpad/etherpad:1.8.18` image from the local registry instead.

- 2.2. Create a `values.yaml` file with the following content:

```
image:
  repository: registry.ocp4.example.com:8443/etherpad
  name: etherpad
  tag: 1.8.18
route:
  host: development-etherpad.apps.ocp4.example.com
```

- 2.3. Log in to the cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
...output omitted...
```

- 2.4. Create a packaged-charts-development project.

```
[student@workstation ~]$ oc new-project packaged-charts-development
Now using project "packaged-charts-development" on server ...
...output omitted...
```

- 2.5. Install the etherpad chart to the packaged-charts-development project. Use the values.yaml file that you created in a previous step. Use example-app as the release name.

```
[student@workstation ~]$ helm install example-app do280-repo/etherpad \
-f values.yaml --version 0.0.6
NAME: example-app
LAST DEPLOYED: Mon Jun  5 06:31:26 2023
NAMESPACE: packaged-charts-development
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

- 2.6. Get the route to verify that you customized the route correctly.

```
[student@workstation ~]$ oc get route
NAME           HOST/PORT
example-app-etherpad   development-etherpad.apps.ocp4.example.com ...
```



Note

The route in this example uses 'edge' TLS termination. TLS termination is explained later in this course.

- 2.7. Open a web browser and navigate to <https://development-etherpad.apps.ocp4.example.com>. The application welcome page appears.

▶ 3. Upgrade a Helm chart by installing the 0.0.7 version of the chart.

- 3.1. Use the `helm list` command to verify the installed version.

```
[student@workstation ~]$ helm list
NAME           NAMESPACE          REVISION ... STATUS     CHART
example-app    packaged-charts-dev 1          ... deployed  etherpad-0.0.6
```

- 3.2. Use the `helm search` command to verify that the repository contains a later version.

```
[student@workstation ~]$ helm search repo --versions
NAME           CHART VERSION APP VERSION ...
do280-repo/etherpad  0.0.7      latest      ...
do280-repo/etherpad  0.0.6      latest      ...
...output omitted...
```

- 3.3. Use the `helm upgrade` command to upgrade to the latest version of the chart.

```
[student@workstation ~]$ helm upgrade example-app do280-repo/etherpad \
-f values.yaml --version 0.0.7
Release "example-app" has been upgraded. Happy Helming!
NAME: example-app
LAST DEPLOYED: Mon Jun  5 06:41:00 2023
NAMESPACE: packaged-charts-development
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

- 3.4. Use the `helm list` command to verify the installed version.

```
[student@workstation ~]$ helm list
NAME           NAMESPACE      REVISION ... STATUS     CHART
example-app    packaged-charts-development  2        ... deployed  etherpad-0.0.7
```

- 3.5. Reload the application welcome page in the web browser.

The updates in the new version of the chart do not affect the deployment in this exercise. When you reload the application, the browser displays the same application welcome page.

- ▶ 4. Create a second deployment of the chart to a new `packaged-charts-production` project, with the `example-app` release name.

Expose the application at the `https://etherpad.apps.ocp4.example.com` URL, by customizing the `route.host` key.

- 4.1. Create a `packaged-charts-production` project.

```
[student@workstation ~]$ oc new-project packaged-charts-production
Now using project "packaged-charts-production" on server ...
...output omitted...
```

- 4.2. Edit the `values.yaml` file to configure the host route to `etherpad.apps.ocp4.example.com`.

```
image:
  repository: registry.ocp4.example.com:8443/etherpad
  name: etherpad
  tag: 1.8.18
route:
  host: etherpad.apps.ocp4.example.com
```

- 4.3. Install the 0.0.7 version of the `etherpad` chart to the `packaged-review-production` project.

Use the `values.yaml` file that you edited in a previous step. Use `production` as the release name.

```
[student@workstation ~]$ helm install production do280-repo/etherpad \
-f values.yaml --version 0.0.7
...output omitted...
```

- 4.4. Verify the deployment by opening a web browser and navigating to the application URL `https://etherpad.apps.ocp4.example.com`

This URL corresponds to the host that you specified in the `values.yaml` file. The application welcome page appears in the production URL.

- 5. Reconfigure the production deployment to sustain heavier use. Change the number of replicas to 3.

- 5.1. Verify that the application has a single pod.

```
[student@workstation ~]$ oc get pods
NAME                               READY   STATUS    RESTARTS   AGE
production-etherpad-6b85b94975-qfpqm   1/1     Running   0          12s
```

- 5.2. Edit the `values.yaml` file. Add a `replicaCount` key with the 3 value.

```
image:
  repository: registry.ocp4.example.com:8443/etherpad
  name: etherpad
  tag: 1.8.18
route:
  host: etherpad.apps.ocp4.example.com
replicaCount: 3
```

- 5.3. Use the `helm upgrade` command to update the parameters.

```
[student@workstation ~]$ helm upgrade production do280-repo/etherpad \
-f values.yaml
...output omitted...
```

- 5.4. Verify that the application has three pods.

```
[student@workstation ~]$ oc get pods
NAME                               READY   STATUS    RESTARTS   AGE
production-etherpad-6b85b94975-h9qgz   1/1     Running   0          13s
production-etherpad-6b85b94975-lbr8h    1/1     Running   0          13s
production-etherpad-6b85b94975-qfpqm   1/1     Running   0          94s
```

- 5.5. Reload the application welcome page in the web browser.

The deployment continues working after adding replicas.

- 6. Remove the `values.yaml` file.

```
[student@workstation ~]$ rm values.yaml
```

Finish

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

```
[student@workstation ~]$ lab finish packaged-charts
```


▶ Lab

Deploy Packaged Applications

Deploy and update applications from resource manifests that are packaged for sharing and distribution.

Outcomes

- Deploy an application and its dependencies from resource manifests that are packaged as a Helm chart.
- Update the application to a later version by using the Helm chart.
- Use a container image in a private container registry instead of a public registry.
- Customize the deployment to add resource requests and limits.

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.

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

Instructions

- Log in to the cluster as the developer user with the developer password. Create the `packaged-review` and `packaged-review-prod` projects.
- Add the classroom Helm repository at the `http://helm.ocp4.example.com/charts` URL and examine its contents. Use `do280-repo` for the name of the repository.
- Install the 0.0.6 version of the `etherpad` chart on the `packaged-review` namespace, with the `test` release name. Use the `registry.ocp4.example.com:8443/etherpad/etherpad:1.8.17` image in the offline classroom registry.

Create a `values-test.yaml` file with the image repository, name, and tag.

Field	Value
<code>image.repository</code>	<code>registry.ocp4.example.com:8443/etherpad</code>
<code>image.name</code>	<code>etherpad</code>
<code>image.tag</code>	<code>1.8.17</code>

- Upgrade the `etherpad` application in the `packaged-review` namespace to the 0.0.7 version of the chart. Set the image tag for the deployment in the `values-test.yaml` file.

Field	Value
image.tag	1.8.18

5. Using version 0.0.6, create a second deployment of the chart in the packaged-review-prod namespace, with the prod release name. Copy the `values-test.yaml` file to the `values-prod.yaml` file, and set the route host.

Field	Value
route.host	etherpad.apps.ocp4.example.com

Access the application in the route URL to verify that it is working correctly.

`https://etherpad.apps.ocp4.example.com`

6. Add limits to the etherpad instance in the packaged-review-prod namespace. The chart values example contains comments that show the required format for this change. Set limits and requests for the deployment in the `values-prod.yaml` file. Use version 0.0.7 of the chart.

Field	Value
resources.limits.memory	256Mi
resources.requests.memory	128Mi

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 grade packaged-review
```

Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Deploy Packaged Applications

Deploy and update applications from resource manifests that are packaged for sharing and distribution.

Outcomes

- Deploy an application and its dependencies from resource manifests that are packaged as a Helm chart.
- Update the application to a later version by using the Helm chart.
- Use a container image in a private container registry instead of a public registry.
- Customize the deployment to add resource requests and limits.

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.

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

Instructions

1. Log in to the cluster as the developer user with the developer password. Create the `packaged-review` and `packaged-review-prod` projects.

- 1.1. Log in to the cluster as the developer user with the developer password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
```

- 1.2. Create the `packaged-review` project.

```
[student@workstation ~]$ oc new-project packaged-review
Now using project "packaged-review" on server ...
...output omitted...
```

- 1.3. Create the `packaged-review-prod` project.

```
[student@workstation ~]$ oc new-project packaged-review-prod
Now using project "packaged-review-prod" on server ...
...output omitted...
```

2. Add the classroom Helm repository at the `http://helm.ocp4.example.com/charts` URL and examine its contents. Use `do280-repo` for the name of the repository.

Chapter 2 | Deploy Packaged Applications

- 2.1. Use the `helm repo list` command to list the repositories that are configured for the student user.

```
[student@workstation ~]$ helm repo list
NAME      URL
do280-repo  http://helm.ocp4.example.com/charts
```

If the `do280-repo` repository is present, then continue to the next step. Otherwise, add the repository.

```
[student@workstation ~]$ helm repo add \
do280-repo http://helm.ocp4.example.com/charts
"do280-repo" has been added to your repositories
```

- 2.2. Use the `helm search repo --versions` command to list all the chart versions in the repository.

The `etherpad` chart has versions 0.0.6 and 0.0.7. This chart is a copy of a chart from the <https://github.com/redhat-cop/helm-charts> repository.

```
[student@workstation ~]$ helm search repo --versions
NAME          CHART VERSION APP VERSION DESCRIPTION
do280-repo/etherpad  0.0.7      latest      ...
do280-repo/etherpad  0.0.6      latest      ...
...output omitted...
```

3. Install the 0.0.6 version of the `etherpad` chart on the `packaged-review` namespace, with the `test` release name. Use the `registry.ocp4.example.com:8443/etherpad/etherpad:1.8.17` image in the offline classroom registry.

Create a `values-test.yaml` file with the image repository, name, and tag.

Field	Value
<code>image.repository</code>	<code>registry.ocp4.example.com:8443/etherpad</code>
<code>image.name</code>	<code>etherpad</code>
<code>image.tag</code>	<code>1.8.17</code>

- 3.1. Switch to the `packaged-review` project.

```
[student@workstation ~]$ oc project packaged-review
Now using project "packaged-review" on server ...
```

- 3.2. Examine the values of the chart.

You can configure the image, the deployment resources, and other values. By default, the chart creates a route.

```
[student@workstation ~]$ helm show values do280-repo/etherpad --version 0.0.6
# Default values for etherpad.
replicaCount: 1

defaultTitle: "Labs Etherpad"
```

```

defaultText: "Assign yourself a user and share your ideas!"

image:
  repository: etherpad ①
  name: ②
  tag: ③
  pullPolicy: IfNotPresent

...output omitted...

route:
  enabled: true
  host: null ④
  targetPort: http

...output omitted...

resources: {} ⑤
...output omitted...

```

- ① The registry with the container image.
 - ② Container image name.
 - ③ Container image tag.
 - ④ Hostname for the OpenShift route resource.
 - ⑤ The resource requests and limits for this workload. This value is set by default to {}, which indicates that it is an empty map.
- 3.3. With the default configuration, the chart uses the docker.io/etherpad/etherpad:latest container image. This image is not suitable for the classroom environment. Use the registry.ocp4.example.com:8443/etherpad/etherpad:1.8.17 container image instead.
- Create a values-test.yaml file with the following content:

```

image:
  repository: registry.ocp4.example.com:8443/etherpad
  name: etherpad
  tag: 1.8.17

```

- 3.4. Install the etherpad chart in the packaged-review namespace.

- Use the values-test.yaml file that you created in the previous step.
- Use test as the release name.

Chapter 2 | Deploy Packaged Applications

```
[student@workstation ~]$ helm install test do280-repo/etherpad \
-f values-test.yaml --version 0.0.6
NAME: test
LAST DEPLOYED: Fri Jun 30 01:03:42 2023
NAMESPACE: packaged-review
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

- 3.5. Use the `helm list` command to verify the installed version of the `etherpad` chart.

```
[student@workstation ~]$ helm list
NAME    NAMESPACE      REVISION    ...    STATUS     CHART          APP VERSION
test    packaged-review 1          ...    deployed   etherpad-0.0.6  latest
```

- 3.6. Verify that the pod is running and that the deployment is ready.

```
[student@workstation ~]$ oc get deployments,pods
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test-etherpad  1/1     1           1           27s

NAME                         READY   STATUS    RESTARTS   AGE
pod/test-etherpad-c6657b556-4jh8z 1/1     Running   0          27s
```

- 3.7. Verify that the pod executes the specified container image.

```
[student@workstation ~]$ oc describe pods -n packaged-review | \
grep '^Name:|Image:'
Name:                  test-etherpad-c6657b556-4jh8z
Image:                 registry.ocp4.example.com:8443/etherpad/etherpad:1.8.17
```

- 3.8. Get the route to obtain the application URL.

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT      ...
test-etherpad  test-etherpad-packaged-review.apps.ocp4.example.com ...
```

- 3.9. Open a web browser and navigate to the following URL to view the application page.

<https://test-etherpad-packaged-review.apps.ocp4.example.com>

4. Upgrade the `etherpad` application in the `packaged-review` namespace to the 0.0.7 version of the chart. Set the image tag for the deployment in the `values-test.yaml` file.

Field	Value
<code>image.tag</code>	<code>1.8.18</code>

- 4.1. Edit the `values-test.yaml` file and update the image tag value:

```
image:
  repository: registry.ocp4.example.com:8443/etherpad
  name: etherpad
  tag: 1.8.18
```

- 4.2. Use the `helm search` command to verify that the repository contains a more recent version of the `etherpad` chart.

```
[student@workstation ~]$ helm search repo --versions etherpad
NAME          CHART VERSION APP VERSION DESCRIPTION
do280-repo/etherpad 0.0.7      latest      ...
do280-repo/etherpad 0.0.6      latest      ...
```

- 4.3. Use the `helm upgrade` command to upgrade to the latest version of the chart.

```
[student@workstation ~]$ helm upgrade test do280-repo/etherpad \
-f values-test.yaml --version 0.0.7
Release "test" has been upgraded. Happy Helming!
NAME: test
LAST DEPLOYED: Fri Jun 30 01:05:07 2023
NAMESPACE: packaged-review
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

- 4.4. Use the `helm list` command to verify the installed version of the `etherpad` chart.

```
[student@workstation ~]$ helm list
NAME  NAMESPACE      REVISION  ...  STATUS      CHART           APP VERSION
test  packaged-review 2          ...  deployed  etherpad-0.0.7  latest
```

- 4.5. Verify that the pod is running and that the deployment is ready.

```
[student@workstation ~]$ oc get deployments,pods
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test-etherpad  1/1     1           1           3m31s

NAME                           READY   STATUS    RESTARTS   AGE
pod/test-etherpad-59d775b78f-ftmsz 1/1     Running   0          64s
```

- 4.6. Verify that the pod executes the updated container image.

```
[student@workstation ~]$ oc describe pods -n packaged-review | \
egrep '^Name:|Image:'
Name:                  test-etherpad-59d775b78f-ftmsz
Image:                registry.ocp4.example.com:8443/etherpad/etherpad:1.8.18
```

- 4.7. Reload the `test-etherpad` application welcome page in the web browser.

Chapter 2 | Deploy Packaged Applications

5. Using version 0.0.6, create a second deployment of the chart in the packaged-review-prod namespace, with the prod release name. Copy the values-test.yaml file to the values-prod.yaml file, and set the route host.

Field	Value
route.host	etherpad.apps.ocp4.example.com

Access the application in the route URL to verify that it is working correctly.

<https://etherpad.apps.ocp4.example.com>

- 5.1. Switch to the packaged-review-prod project.

```
[student@workstation ~]$ oc project packaged-review-prod  
Now using project "packaged-review-prod" on server ...
```

- 5.2. Copy the values-test.yaml file to values-prod.yaml.

```
[student@workstation ~]$ cp values-test.yaml values-prod.yaml
```

- 5.3. Set the route host in the values-prod.yaml file.

```
image:  
  repository: registry.ocp4.example.com:8443/etherpad  
  name: etherpad  
  tag: 1.8.18  
route:  
  host: etherpad.apps.ocp4.example.com
```

- 5.4. Install the 0.0.6 version of the etherpad chart on the packaged-review-prod namespace.

Use the values-prod.yaml file that you edited in the previous step. Use prod as the release name.

```
[student@workstation ~]$ helm install prod do280-repo/etherpad \  
-f values-prod.yaml --version 0.0.6  
NAME: prod  
LAST DEPLOYED: Fri Jun 30 01:07:29 2023  
NAMESPACE: packaged-review-prod  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None
```

- 5.5. Use the helm list command to verify the installed version of the etherpad chart.

```
[student@workstation ~]$ helm list  
NAME  NAMESPACE      REVISION  ...  STATUS    CHART          APP VERSION  
prod  packaged-review-prod  1        ...  deployed  etherpad-0.0.6  latest
```

- 5.6. Verify that the pod is running and that the deployment is ready.

```
[student@workstation ~]$ oc get deployments,pods
NAME                      READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/prod-etherpad  1/1     1          1          65s

NAME                      READY  STATUS    RESTARTS  AGE
pod/prod-etherpad-5947dfb987-9dclr  1/1     Running   0          65s
```

- 5.7. Verify that the pod executes the specified container image.

```
[student@workstation ~]$ oc describe pods -n packaged-review-prod | \
egrep '^Name:|Image:'
Name:                  pod/prod-etherpad-5947dfb987-9dclr
Image:                 registry.ocp4.example.com:8443/etherpad/etherpad:1.8.18
```

- 5.8. Verify the deployment by opening a web browser and navigating to the application URL. This URL corresponds to the host that you specified in the `values-prod.yaml` file. The application welcome page appears in the production URL.

<https://etherpad.apps.ocp4.example.com>

6. Add limits to the etherpad instance in the `packaged-review-prod` namespace. The chart values example contains comments that show the required format for this change. Set limits and requests for the deployment in the `values-prod.yaml` file. Use version `0.0.7` of the chart.

Field	Value
<code>resources.limits.memory</code>	<code>256Mi</code>
<code>resources.requests.memory</code>	<code>128Mi</code>

- 6.1. Edit the `values-prod.yaml` file. Configure the deployment to request 128 MiB of RAM, and limit RAM usage to 128 MiB.

```
image:
  repository: registry.ocp4.example.com:8443/etherpad
  name: etherpad
  tag: 1.8.18
route:
  host: etherpad.apps.ocp4.example.com
resources:
  limits:
    memory: 256Mi
  requests:
    memory: 128Mi
```

- 6.2. Use the `helm upgrade` command to upgrade to the latest version of the chart.

```
[student@workstation ~]$ helm upgrade prod do280-repo/etherpad \
-f values-prod.yaml --version 0.0.7
Release "prod" has been upgraded. Happy Helming!
NAME: prod
LAST DEPLOYED: Fri Jun 30 01:09:04 2023
NAMESPACE: packaged-review-prod
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

- 6.3. Verify that the pod is running and that the deployment is ready.

```
[student@workstation ~]$ oc get deployments,pods
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prod-etherpad        1/1     1           1           3m14s

NAME                               READY   STATUS    RESTARTS   AGE
pod/prod-etherpad-6b7d9dffbc-f7cng 1/1     Running   0          36s
```

- 6.4. Examine the application pod from the production instance of the application to verify the configuration change.

```
[student@workstation ~]$ oc describe pods -n packaged-review-prod | \
egrep -A1 '^Name:|Limits|Requests'
Name:                  prod-etherpad-6b7d9dffbc-f7cng
Namespace:             packaged-review-prod
-
Limits:
  memory:  256Mi
Requests:
  memory:  128Mi
```

- 6.5. Examine the pod of the test instance of the application in the packaged-review namespace. This deployment uses the values from the `values-test.yaml` file that did not specify resource limits or requests. The pod in the packaged-review namespace does not have a custom resource allocation.

```
[student@workstation ~]$ oc describe pods -n packaged-review | \
egrep -A1 '^Name:|Limits|Requests'
Name:                  test-etherpad-59d775b78f-ftmsz
Namespace:             packaged-review
```

- 6.6. Use the `helm list` command to verify the installed version of the etherpad chart.

```
[student@workstation ~]$ helm list
NAME  NAMESPACE      REVISION  ...  STATUS  CHART      APP VERSION
prod  packaged-review 2          ...  deployed  etherpad-0.0.7  latest
```

- 6.7. Reload the application welcome page in the web browser. The deployment continues working after you add the limits.

6.8. Remove the `values-test.yaml` and `values-prod.yaml` files.

```
[student@workstation ~]$ rm values-test.yaml values-prod.yaml
```

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 grade packaged-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- Use templates to deploy workloads with parameterization.
- Use the `oc create -f` command to upload a template to a project.
- Use the `oc process` command and the `oc apply -f -` command to deploy template resources to the Kubernetes cluster.
- Provide parameters to customize the template with the `-p` or `--param-file` arguments to the `oc` command.
- View Helm charts with the `helm show chart chart-reference` and `helm show values chart-reference` commands.
- Use the `helm install release-name chart-reference` command to create a release for a chart.
- Inspect releases by using the `helm list` command.
- Use the `helm history release-name` command to view the history of a release.
- Use the `helm repo add repo-name repo-url` command to add a Helm repository to the `~/.config/helm/repositories.yaml` configuration file.
- Use the `helm search repo` command to search repositories in the `~/.config/helm/repositories.yaml` configuration file.

Chapter 3

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.

Sections

- Configure Identity Providers (and Guided Exercise)
- Define and Apply Permissions with RBAC (and Guided Exercise)

Lab

- Authentication and Authorization

Configure Identity Providers

Objectives

- Configure the HTPasswd identity provider for OpenShift authentication.

OpenShift Users and Groups

Several OpenShift resources relate to authentication and authorization. The following list shows 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 that the user is a member of.

Identity

The identity resource keeps a record of successful authentication attempts from a specific user and identity provider. Any data about the source of the authentication is stored on the identity.

Service Account

In OpenShift, applications can communicate with the API independently when user credentials cannot be acquired. To preserve the integrity of the credentials for a regular user, credentials are not shared and service accounts are used instead. With service accounts, you can 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 groups. Authorization policies use groups to assign permissions to multiple users at the same time. For example, to grant 20 users access to objects within a project, it is better to use a group instead of granting access to each user individually. OpenShift Container Platform also provides system groups or virtual groups that are provisioned automatically by the cluster.

Role

A role defines the API operations that a user has permissions to perform on specified 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. OpenShift usually creates these resources automatically after a successful interactive login with OAuth.

Authenticating API Requests

The authentication and authorization security layers enable 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. On successful authentication, the authorization layer either accepts or rejects 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.

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.

Identity Providers

The OpenShift OAuth server can be configured to use many identity providers. The following lists includes the more common identity providers:

HTPasswd

Validates usernames and passwords against a secret that stores credentials that are generated by using the `htpasswd` command.

Keystone

Enables shared authentication with an OpenStack Keystone v3 server.

LDAP

Configures the LDAP identity provider to validate usernames and passwords against an LDAPv3 server, by using simple bind authentication.

GitHub or GitHub Enterprise

Configures a GitHub identity provider to validate usernames and passwords against GitHub or the GitHub Enterprise OAuth authentication server.

OpenID Connect

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

The OAuth custom resource must be updated with your chosen 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. One way is to use the `kubeconfig` file, which embeds an X.509 client certificate that never expires. Another way is to authenticate as the `kubeadmin` virtual user. Successful authentication grants an OAuth access token.

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

Authenticating with 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 for 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. Use the ssh lab@utility command from the workstation machine to access the utility machine.

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 command that requires cluster administrator privileges without logging in to OpenShift.

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

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

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

Authenticating with the kubeadmin 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 to log in to the cluster. The cluster installation logs also provide the login, 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/kubeadmin-password file.

Deleting the Virtual User

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

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

**Warning**

If you delete the `kubeadmin` secret before you configure another user with cluster admin privileges, then you can administer your cluster only by 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 to destroy and reinstall your cluster.

**Warning**

Do not delete the `kubeadmin` user at any time during this course. The `kubeadmin` user is essential to the course lab architecture. If you deleted this user, you would have to delete the lab environment and re-create it.

Managing Users with the HTPasswd Identity Provider

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

Creating an HTPasswd File

The `httpd-tools` package provides the `htpasswd` utility, which must be installed and available on your system.

Create the `htpasswd` file.

```
[user@host ~]$ htpasswd -c -B -b /tmp/htpasswd student redhat123
```

**Important**

Use the `-c` option only when creating a file. The `-c` option replaces all file content if the file already exists.

Add or update credentials.

```
[user@host ~]$ htpasswd -B -b /tmp/htpasswd student redhat1234
```

Delete credentials.

```
[user@host ~]$ 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@host ~]$ oc create secret generic htpasswd-secret \
--from-file htpasswd=/tmp/htpasswd -n openshift-config
```

**Important**

A secret that the HTPasswd identity provider uses requires adding the `htpasswd=` prefix before specifying the path to the file.

Extracting Secret Data

When adding or removing users, use the `oc extract` command to retrieve the secret. Extracting the secret ensures that you work on the current set of users.

By default, the `oc extract` command saves each key within a configuration map or secret as a separate file. Alternatively, you can then redirect all data to a file or display it 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 exists.

```
[user@host ~]$ 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 by using the content of the `/tmp/htpasswd` file.

```
[user@host ~]$ 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 the following command:

```
[user@host ~]$ watch oc get pods -n openshift-authentication
```

Test additions, changes, or deletions to the secret after the new pods finish deploying.

Configuring the HTPasswd Identity Provider

The HTPasswd identity provider validates users against a secret that contains usernames and passwords that are 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 with 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. With the default `claim` value, you cannot log in with different identity providers.
- ③** An existing secret that contains data that is generated by 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@host ~]$ 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 by using the `oc replace` command.

```
[user@host ~]$ oc replace -f oauth.yaml
```

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@host ~]$ htpasswd -D /tmp/htpasswd manager
```

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

```
[user@host ~]$ oc set data secret/htpasswd-secret \
--from-file htpasswd=/tmp/htpasswd -n openshift-config
```

Remove the user resource with the following command:

```
[user@host ~]$ 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@host ~]$ oc get identities | grep manager
my_htpasswd_provider:manager    my_htpasswd_provider    manager      manager    ...
[user@host ~]$ 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. With this role, the user can perform any action on any resources within the cluster. The following example assigns the `cluster-admin` role to the `student` user.

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



References

For more information about identity providers, refer to the *Understanding Identity Provider Configuration* chapter in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#understanding-identity-provider

► Guided Exercise

Configure Identity Providers

Configure the HTPasswd identity provider and create users for cluster administrators.

Outcomes

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

Before You Begin

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

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

Instructions

- 1. Add an entry for two users, `new_admin` and `new_developer`. Assign the `new_admin` user the `redhat` password, and assign the `new_developer` user the `developer` password.

- 1.1. Create an HTPasswd authentication file named `htpasswd` in the `~/D0280/labs/auth-providers/` directory. Add the `new_admin` user with the `redhat` password. The file name is arbitrary; this exercise uses the `~/D0280/labs/auth-providers/htpasswd` file.

Use the `htpasswd` command to populate the HTPasswd authentication file with the usernames and encrypted passwords. The `-B` option uses bcrypt encryption. By default, the `htpasswd` command uses the MD5 hashing algorithm if you do not specify another algorithm.

```
[student@workstation ~]$ htpasswd -c -B -b ~/D0280/labs/auth-providers/htpasswd \
new_admin redhat
Adding password for user new_admin
```

- 1.2. Add the `new_developer` user with the `developer` password to the `~/D0280/labs/auth-providers/htpasswd` file. The password for the `new_developer` user is hashed with the MD5 algorithm, because no algorithm was specified and MD5 is the default hashing algorithm.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-providers/htpasswd \
new_developer developer
Adding password for user new_developer
```

13. Review the contents of the ~/D0280/labs/auth-providers/htpasswd file and verify that it includes two entries with hashed passwords: one for the new_admin user and another for the new_developer user.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-providers/htpasswd
new_admin:$2y$05$qQaFbx4hbf4uZe.SMLSduTN8uN4DNJMj4jE5zXDA57WrTRlpu2QS
new_developer:$apr1$S0TxtLx1$QSRfBIufYP39pKNsIg/nD1
```

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

- 2.1. Log in to the cluster as the admin user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 2.2. Create a secret from the ~/D0280/labs/auth-providers/htpasswd file. To use the HTPasswd identity provider, you must define a secret with a key named htpasswd that contains the HTPasswd user file ~/D0280/labs/auth-providers/htpasswd.

```
[student@workstation ~]$ oc create secret generic localusers \
--from-file htpasswd=~/D0280/labs/auth-providers/htpasswd \
-n openshift-config
secret/localusers created
```

- 2.3. Assign the new_admin user the cluster-admin role. Ignore a warning that the user is not found.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
cluster-admin new_admin
Warning: User 'new_admin' not found
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "new_admin"
```



Note

When you execute the `oc adm policy add-cluster-role-to-user cluster-admin new-admin` command, a naming collision occurs with an existing cluster role binding object. Consequently, the system creates an object and appends to the name `-x`, which is an iterating numeral that starts with `-0`.

To view the new cluster role binding, use the `oc get clusterrolebinding | grep ^cluster-admin` command to list all cluster role bindings that begin with `cluster-admin`. Then, run `oc describe` on the listed item with the highest `-x` value to view the details for your new binding.

- 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-providers` directory.

```
[student@workstation ~]$ oc get oauth cluster \
-o yaml > ~/D0280/labs/auth-providers/oauth.yaml
```

**Note**

For convenience, an `oauth.yaml` file that contains the completed custom resource file is downloaded to `~/D0280/solutions/auth-providers`.

- 3.2. Edit the `~/D0280/labs/auth-providers/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 structure. Ensure that the `htpasswd`, `mappingMethod`, `name`, and `type` strings are at the same indentation level after the `type: LDAP` key/value pair on line 30.

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

- 3.3. Apply the custom resource that was defined in the previous step.

```
[student@workstation ~]$ oc replace -f ~/D0280/labs/auth-providers/oauth.yaml
oauth.config.openshift.io/cluster replaced
```

**Note**

Authentication changes require redeploying pods in the openshift-authentication namespace.

Use the `watch` command to examine the status of workloads in the openshift-authentication namespace.

```
[student@workstation ~]$ watch oc get all -n openshift-authentication
NAME                               READY   STATUS    RESTARTS   AGE
pod/oauth Openshift-6d68ff9dc-6f8dr 1/1     Running   3          2m
...output omitted...
```

A few minutes after you ran the `oc replace` command, the redeployment starts. Wait until new pods are running. Press `Ctrl+C` to exit the `watch` command.

Provided that the previously created secret was created correctly, you can log in by using the HTPasswd identity provider.

- ▶ 4. Log in as the `new_admin` and as the `new_developer` user to verify the HTPasswd user configuration.
 - 4.1. Log in to the cluster as the `new_admin` user to verify that 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, then wait a few moments and try again.

```
[student@workstation ~]$ oc login -u new_admin -p redhat
Login successful.

...output omitted...
```

- 4.2. Use the `oc get nodes` command to verify that the `new_admin` user has the `cluster-admin` role.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES          AGE    VERSION
master01   Ready     control-plane,master,worker 13d   v1.27.6+f67aeb3
```

- 4.3. Log in to the cluster as the `new_developer` user to verify that the HTPasswd authentication is configured correctly.

```
[student@workstation ~]$ oc login -u new_developer -p developer
Login successful.

...output omitted...
```

- 4.4. Use the `oc get nodes` command to verify that the `new_developer` and `new_admin` users do not have the same level of access.

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

- 4.5. Log in as the `new_admin` user.

```
[student@workstation ~]$ oc login -u new_admin -p redhat
Login successful.

...output omitted...
```

- 4.6. List the current users.

```
[student@workstation ~]$ oc get users
NAME          UID          ...  IDENTITIES
admin         6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24  ...
new_admin     489c7402-d318-4805-b91d-44d786a92fc1  ...  myusers:new_admin
new_developer 8dbae772-1dd4-4242-b2b4-955b005d9022  ...  myusers:new_developer
```

**Note**

You might see additional users from previously completed exercises.

- 4.7. Display the list of current identities.

```
[student@workstation ~]$ oc get identity
NAME          IDP NAME    IDP USER NAME  USER NAME
              ...          ...          ...
              ...          ...          ...
              ...          ...          ...
myusers:new_admin   myusers    new_admin    new_admin
                     489c7402-d318-4805-b91d-44d786a92fc1
myusers:new_developer  myusers    new_developer  new_developer
                     8dbae772-1dd4-4242-b2b4-955b005d9022
```

**Note**

You might see additional identities from previously completed exercises.

5. As the `new_admin` user, create a HTPasswd user named `manager` with a password of `redhat`.

- 5.1. Extract the file data from the secret to the `~/D0280/labs/auth-providers/htpasswd` file.

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
--to ~/D0280/labs/auth-providers/ --confirm
/home/student/D0280/labs/auth-providers/htpasswd
```

- 5.2. Add an entry to your ~/D0280/labs/auth-providers/htpasswd file for the additional manager user with the redhat password.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-providers/htpasswd \
manager redhat
Adding password for user manager
```

- 5.3. Review the contents of your ~/D0280/labs/auth-providers/htpasswd file and verify that it includes three entries with hashed passwords: one each for the new_admin, new_developer, and manager users.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-providers/htpasswd
new_admin:$2y$05$qQaFbx4hbf4uZe.SMLSduTN8uN4DNJMj4jE5zXDA57WrTRlpu2QS
new_developer:$apr1$S0TxtLxl$QSRfBIufYP39pKNSIg/nD1
manager:$apr1$HZ/9tC6b$j20cHHg2G02SSu1wyG0ge.
```

- 5.4. You must update the secret after adding additional users. Use the `oc set data secret` command to update the secret. If the command fails, then wait a few moments for the oauth operator to finish reloading, and rerun the command.

```
[student@workstation ~]$ oc set data secret/localusers \
--from-file htpasswd=~/D0280/labs/auth-providers/htpasswd \
-n openshift-config
secret/localusers data updated
```

- 5.5. Use the `watch` command to examine the status of workloads in the `openshift-authentication` namespace.

```
[student@workstation ~]$ watch oc get all -n openshift-authentication
NAME                               READY   STATUS    RESTARTS   AGE
pod/oauth-openshift-6d68ffb9dc-6f8dr   1/1     Running   3          2m
...output omitted...
```

A few minutes after you ran the `oc set data` command, the redeployment starts. Wait until new pods are running. Press `Ctrl+C` to exit the `watch` command.

- 5.6. Log in to the cluster as the manager user.



Note

If the authentication fails, then wait a few moments and try again.

```
[student@workstation ~]$ oc login -u manager -p redhat
Login successful.

...output omitted...
```

- 6. Create an auth-providers project, and then verify that the new_developer user cannot access the project.

6.1. As the manager user, create an auth-providers project.

```
[student@workstation ~]$ oc new-project auth-providers
Now using project "auth-providers" on server https://api.ocp4.example.com:6443".
...output omitted...
```

6.2. Log in as the new_developer user.

```
[student@workstation ~]$ oc login -u new_developer -p developer
Login successful.

...output omitted...
```

6.3. Attempt to delete the auth-providers project.

```
[student@workstation ~]$ oc delete project auth-providers
Error from server (Forbidden): projects.project.openshift.io "auth-providers" is
forbidden: User "new_developer" cannot delete resource "projects" in API group
"project.openshift.io" in the namespace "auth-providers"
```

- 7. Change the password for the manager user.

7.1. Log in as the new_admin user.

```
[student@workstation ~]$ oc login -u new_admin -p redhat
Login successful.

...output omitted...
```

7.2. Extract the file data from the secret to the ~/D0280/labs/auth-providers/htpasswd file.

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
--to ~/D0280/labs/auth-providers/ --confirm
/home/student/D0280/labs/auth-providers/htpasswd
```

7.3. Generate a random user password and assign it to the MANAGER_PASSWD variable.

```
[student@workstation ~]$ MANAGER_PASSWD="$(openssl rand -hex 15)"
```

- 7.4. Update the `manager` user to use the stored password in the `MANAGER_PASSWD` variable.

```
[student@workstation ~]$ htpasswd -b ~/D0280/labs/auth-providers/htpasswd \
  manager ${MANAGER_PASSWD}
Updating password for user manager
```

- 7.5. Update the secret.

```
[student@workstation ~]$ oc set data secret/localusers \
  --from-file htpasswd=~/D0280/labs/auth-providers/htpasswd \
  -n openshift-config
secret/localusers data updated
```

- 7.6. Use the `watch` command to examine the status of workloads in the `openshift-authentication` namespace.

```
[student@workstation ~]$ watch oc get all -n openshift-authentication
NAME                           READY   STATUS    RESTARTS   AGE
pod/oauth-openshift-6d68ffb9dc-6f8dr   1/1     Running   3          2m
...output omitted...
```

A few minutes after you ran the `oc set data` command, the redeployment starts. Wait until new pods are running. Press `Ctrl+C` to exit the `watch` command.

- 7.7. Log in as the `manager` user to verify the updated password.

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

...output omitted...
```



Note

If the authentication fails, then wait a few moments and try again.

- ▶ 8. Remove the `manager` user.

- 8.1. Log in as the `new_admin` user.

```
[student@workstation ~]$ oc login -u new_admin -p redhat
Login successful.

...output omitted...
```

- 8.2. Extract the file data from the secret to the `~/D0280/labs/auth-providers/htpasswd` file.

```
[student@workstation ~]$ oc extract secret/localusers -n openshift-config \
  --to ~/D0280/labs/auth-providers/ --confirm
/home/student/D0280/labs/auth-providers/htpasswd
```

- 8.3. Delete the `manager` user from the `~/DO280/labs/auth-providers/htpasswd` file.

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

- 8.4. Update the secret.

```
[student@workstation ~]$ oc set data secret/localusers \  
--from-file htpasswd=~/DO280/labs/auth-providers/htpasswd \  
-n openshift-config  
secret/localusers data updated
```

- 8.5. Use the `watch` command to examine the status of workloads in the `openshift-authentication` namespace.

```
[student@workstation ~]$ watch oc get all -n openshift-authentication  
NAME READY STATUS RESTARTS AGE  
pod/oauth-openshift-6d68ffb9dc-6f8dr 1/1 Running 3 2m  
...output omitted...
```

A few minutes after you ran the `oc set data` command, the redeployment starts. Wait until new pods are running. Press `Ctrl+C` to exit the `watch` command.

- 8.6. Log in as the `manager` user. If the login succeeds, then try again until the login fails.

```
[student@workstation ~]$ oc login -u manager -p ${MANAGER_PASSWD}  
Login failed (401 Unauthorized)  
Verify you have provided correct credentials.
```

- 8.7. Log in as the `new_admin` user.

```
[student@workstation ~]$ oc login -u new_admin -p redhat  
Login successful.  
...output omitted...
```

- 8.8. Delete the identity resource for the `manager` user.

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

- 8.9. Delete the user resource for the `manager` user.

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

- 8.10. List the current users to verify that you deleted the `manager` user.

```
[student@workstation ~]$ oc get users
NAME          UID          ... IDENTITIES
admin         6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24 ...
new_admin     489c7402-d318-4805-b91d-44d786a92fc1 ... myusers:new_admin
new_developer 8dbae772-1dd4-4242-b2b4-955b005d9022 ... myusers:new_developer
```

- 8.11. Display the list of current identities to verify that you deleted the manager identity.

```
[student@workstation ~]$ oc get identity
NAME           IDP NAME   IDP USER NAME  USER NAME
               USER   UID
...
...           ...      ...       admin
...           ...      ...       6126c5a9-4d18-4cdf-95f7-b16c3d3e7f24
myusers:new_admin    myusers  new_admin    new_admin
                     489c7402-d318-4805-b91d-44d786a92fc1
myusers:new_developer myusers  new_developer new_developer
                     8dbae772-1dd4-4242-b2b4-955b005d9022
```

- 8.12. Extract the secret and verify that only the new_admin and new_developer users 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
new_admin:$2y$05$qQaFbx4hbf4uZe.SMLSduTN8uN4DNJMj4jE5zXDA57WrTRlpu2QS
new_developer:$apr1$S0TxtLXl$QSRfBIufYP39pKNsIg/nD1
```

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

- 9.1. Log in as the admin user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
Login successful.

...output omitted...
```

- 9.2. Delete the auth-providers project.

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

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

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

Delete all the lines after the ldap identity provider definition on line 34. Your file should match the following example:

```

apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - ldap:
      ...output omitted...
      type: LDAP
    # Delete all lines below
    - htpasswd:
      fileData:
        name: localusers
      mappingMethod: claim
      name: myusers
      type: HTPasswd

```

Save your changes, and then verify that the `oc edit` command applied those changes:

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

- 9.4. Use the `watch` command to examine the status of workloads in the `openshift-authentication` namespace.

```
[student@workstation ~]$ watch oc get all -n openshift-authentication
NAME                  READY   STATUS    RESTARTS   AGE
pod/oauth-openshift-6d68ffb9dc-6f8dr  1/1     Running   3          2m
...output omitted...
```

A few minutes after you ran the `oc edit` command, the redeployment starts. Wait until new pods are running. Press `Ctrl+C` to exit the `watch` command.

- 9.5. Delete the `localusers` secret from the `openshift-config` namespace.

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

- 9.6. Delete all identity resources.

```
[student@workstation ~]$ oc delete identity --all
identity.user.openshift.io "Red Hat Identity Management:dwlk...jb20" deleted
identity.user.openshift.io "myusers:new_admin" deleted
identity.user.openshift.io "myusers:new_developer" deleted
```



Note

You might see additional identities from previously completed exercises.

- 9.7. Delete all user resources.

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



Note

You might see additional users from previously completed exercises.

Finish

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

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

Define and Apply Permissions with RBAC

Objectives

- 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 whether a user can perform certain actions within the cluster or project. You can choose between two role types, 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.

RBAC Level	Description
Cluster RBAC	Roles and bindings that apply across all projects.
Local RBAC	Roles and bindings that are scoped to a given project. Local role bindings can reference both cluster and local roles.



Note

This two-level hierarchy enables reuse across multiple projects through the cluster roles, and enables customization inside individual projects through local roles. Authorization evaluation uses both the cluster role bindings and the local role bindings to allow or deny an action on a resource.

Managing RBAC with the CLI

Cluster administrators can use the `oc adm policy` command to 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@host ~]$ 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@host ~]$ 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@host ~]$ 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@host ~]$ 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 whether a user can execute an action on a resource. For example:

```
[user@host ~]$ 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. Other required steps 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 access cluster status information.
cluster-reader	Users with this role can access or view most of the objects but cannot modify them.

Default roles	Description
edit	Users with this role can create, change, and delete common application resources on the project, such as services and deployments. 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 their own projects.
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 applications.

The `self-provisioner` cluster role enables a user to create `projectrequests` resources. By default, the `self-provisioners` cluster role binding associates the `self-provisioner` cluster role with the `system:authenticated:oauth` group. Add users to this group so they can create projects in RHOPC.

The `edit` role gives a user sufficient access to act as a developer inside the project, but working under the constraints that a project administrator configured.

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@host ~]$ oc policy add-role-to-user role-name username -n project
```

For example, run the following command to add the `dev` user to the `basic-user` cluster role in the `wordpress` project.

```
[user@host ~]$ oc policy add-role-to-user basic-user dev -n wordpress
```

Even though `basic-user` is a cluster role, the `add-role-to-user` subcommand limits the scope of the role to the `wordpress` namespace for the `dev` user.

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 role bindings.

Regular users

Most interactive OpenShift Container Platform users are regular users, and are represented with the `User` object. This type of user represents a person with access to the platform.

System users

Many system users are created automatically when the infrastructure is defined, mainly for 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, and various others. An anonymous system user is used by default for unauthenticated requests.

System user names start with a `system:` prefix, such as `system:admin`, `system:openshift-registry`, and `system:node:node1.example.com`.

Service accounts

Service accounts are system users that are associated with projects. Workloads can use service accounts to invoke Kubernetes APIs.

Some service account users are created automatically during project creation. Project administrators can create more service accounts to grant extra privileges to workloads. By default, service accounts have no roles. Grant roles to service accounts to enable workloads to use specific APIs.

Service accounts are represented with the `ServiceAccount` object.

System account user names start with a `system:serviceaccount:namespace:` prefix, such as `system:serviceaccount:default:deployer` and `system:serviceaccount:accounting: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, the policy determines what the user is authorized to do.

Group Management

A group resource represents a set of users. Cluster administrators can use the `oc adm groups` command to add groups or to add users to groups. For example, run the following command to add the `lead-developers` group to the cluster:

```
[user@host ~]$ oc adm groups new lead-developers
```

Likewise, the following command adds the `user1` user to the `lead-developers` group:

```
[user@host ~]$ oc adm groups add-users lead-developers user1
```



References

For more information about RBAC, refer to the *Using RBAC to Define and Apply Permissions* chapter in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#authorization-overview_using-rbac

For more information about groups, refer to the *Understanding Authentication* chapter in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#understanding-authentication

Kubernetes Namespaces

<https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

► Guided Exercise

Define and Apply Permissions with RBAC

Define role-based access controls and apply permissions to users.

Outcomes

- 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 start auth-rbac
```

Instructions

- 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 redhatocp \
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 'ROLE|self-provisioner'
NAME          ROLE           ... GROUPS ...
self-provisioners ClusterRole/self-provisioner ... system:authenticated:oauth
```

- 2. Remove the privilege to create 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:   rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  self-provisioner
Subjects:
  Kind  Name          Namespace
  ----  ---          -----
  Group system:authenticated:oauth
```



Important

Do not confuse the `self-provisioner` cluster role with the `self-provisioners` cluster role binding.

- 2.2. Remove the `self-provisioner` cluster role from the `system:authenticated:oauth` virtual group, which deletes the `self-provisioners` role binding.

```
[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"
```



Note

You can safely ignore the warning about your changes being lost.

- 2.3. Verify that the role is 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 whether any other cluster role bindings reference the `self-provisioner` cluster role.

```
[student@workstation ~]$ oc get clusterrolebinding -o wide | \
  grep -E 'ROLE|self-provisioner'
NAME      ROLE      AGE      USERS      GROUPS      SERVICEACCOUNTS
```

- 2.5. Log in as the **leader** user with the **redhat** password.

```
[student@workstation ~]$ oc login -u leader -p redhat
Login successful.

...output omitted...
```

- 2.6. Try to create a project. The operation should fail.

```
[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.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
Login successful.

...output omitted...
```

- 3.2. Create the **auth-rbac** project.

```
[student@workstation ~]$ oc new-project auth-rbac
Now using project "auth-rbac" on server "https://api.ocp4.example.com:6443".

...output omitted...
```

- 3.3. 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 named **dev-group**.

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

- 4.2. Add the **developer** user to the group that you created in the previous step.

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

- 4.3. Create a second group named **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 the group that you created in the previous step.

```
[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
Default SMB Group
admins        admin
dev-group    developer
developer
editors
ocpadmins     admin
ocpdevs       developer
qa-group     qa-engineer
```



Note

The lab environment already contains groups from the lab LDAP directory.

- 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 the dev-group 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 the qa-group 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 that OpenShift assigns to service accounts.

```
[student@workstation ~]$ oc get rolebindings -o wide | grep -v '^system:'
NAME      ROLE          AGE     USERS   GROUPS      SERVICEACCOUNTS
admin     ClusterRole/admin  60s    admin
admin-0   ClusterRole/admin 45s    leader
edit      ClusterRole/edit  30s
view      ClusterRole/view  15s    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 by using the standard image stream from OpenShift.

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

- 6.3. Try to grant write privileges to the qa-engineer user. The operation 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 can view objects in the auth-rbac project, but not modify anything.

- 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. The operation 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 redhatocp
Login successful.

...output omitted...
```

8.2. Restore project creation privileges for all users by re-creating the `self-provisioners` cluster role binding that the OpenShift installer created.

```
[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"
```



Note

You can safely ignore the warning that the group was not found.

Finish

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

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

► Lab

Authentication and Authorization

Configure the HTPasswd identity provider, create groups, and assign roles to users and groups.

Outcomes

- 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 start auth-review
```

The command ensures that the cluster API is reachable, and that the cluster uses the initial lab authentication settings.

Instructions

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 the `L@bR3v!ew` password. Add two entries to the file for the `new_admin` and `new_developer` users. Use the `L@bR3v!ew` password for each new user.
2. Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password. Configure your cluster to use the HTPasswd identity provider by using the defined user names and passwords in the `~/D0280/labs/auth-review/tmp_users` file. For grading, use the `auth-review` name for the secret.
3. Make the `new_admin` user a cluster administrator. Log in as both the `new_admin` and `new_developer` users to verify HTPasswd user configuration and cluster privileges.
4. As the `new_admin` user, prevent users from creating projects in the cluster.
5. Create a `managers` group, 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 `developers` group and grant edit privileges on the `auth-review` project. Add the `new_developer` user to the group.
7. Create a `qa` group and grant view privileges on the `auth-review` project. Add the `tester` user to the group.

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 grade auth-review
```

Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Authentication and Authorization

Configure the HTPasswd identity provider, create groups, and assign roles to users and groups.

Outcomes

- 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 start auth-review
```

The command ensures that the cluster API is reachable, and that the cluster uses the initial lab authentication settings.

Instructions

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 the `L@bR3v!ew` password. Add two entries to the file for the **new_admin** and **new_developer** users. Use the `L@bR3v!ew` 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 to use the `L@bR3v!ew` password. Add entries for the **new_admin** and **new_developer** users with the `L@bR3v!ew` password.

```
[student@workstation ~]$ for NAME in tester leader new_admin new_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 new_admin
Adding password for user new_developer
```

- 1.3. Review the contents of the ~/D0280/labs/auth-review/tmp_users file. This file does not contain a line for the analyst user. The file includes two new entries with hashed passwords for the new_admin and new_developer users.

```
[student@workstation ~]$ cat ~/D0280/labs/auth-review/tmp_users
tester:$apr1$EyWSDib4$uLoUMPwohNWURU5L5ogkB/
leader:$apr1$/08SyNdp$gjr.P7FMJbK2IebFU0QQn/
new_admin:$apr1$M5WHRPR2$GbGDKTK8QTrW2S/f2/1Kt1
new_developer:$apr1$dXdG8tWd$N8HA0SUe3TbqAhI049gOH0
```

2. Log in to your OpenShift cluster as the admin user with the redhatocp password. Configure your cluster to use the HTPasswd identity provider by using the defined user names and passwords in the ~/D0280/labs/auth-review/tmp_users file. For grading, use the auth-review name for the secret.

- 2.1. Log in to the cluster as the admin user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 2.2. Create an auth-review secret by 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
```

- 2.3. 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
```

- 2.4. Edit the ~/D0280/labs/auth-review/oauth.yaml file to add an htpasswd identity provider in addition to the existing ldap identity provider on line 30. The following excerpt shows the text to add in bold. Ensure that the htpasswd, mappingMethod, name, and type fields are at the same indentation level.

```

apiVersion: config.openshift.io/v1
kind: OAuth
...output omitted...
spec:
  identityProviders:
    - ldap:
      ...output omitted...
      type: LDAP
      # Add the text after this comment.
    - htpasswd:
      fileData:
        name: auth-review
      mappingMethod: claim
      name: htpasswd
      type: HTPasswd

```

**Note**

For convenience, the `~/D0280/solutions/auth-review/oauth.yaml` file contains a minimal version of the OAuth configuration with the specified customizations.

2.5. Apply the customized resource that you defined in the previous step.

```
[student@workstation ~]$ oc replace -f ~/D0280/labs/auth-review/oauth.yaml
oauth.config.openshift.io/cluster replaced
```

2.6. A successful update to the `oauth/cluster` resource re-creates the `oauth-openshift` pods in the `openshift-authentication` namespace.

```
[student@workstation ~]$ watch oc get pods -n openshift-authentication
```

Wait until the new `oauth-openshift` pods are ready and running, and the previous pods have terminated.

```
Every 2.0s: oc get pods -n openshift-authentication      ...
NAME                           READY   STATUS    RESTARTS   AGE
oauth-openshift-68d6f666fd-z746p   1/1     Running   0          42s
```

Press **Ctrl+C** to exit the `watch` command.

**Note**

Pods in the `openshift-authentication` namespace redeploy when the `oc replace` command succeeds.

In this exercise, changes to authentication might require a few minutes to apply.

You can examine the status of pods and deployments in the `openshift-authentication` namespace to monitor the authentication status. You can also examine the `authentication` cluster operator for further status information.

Provided that the previously created secret was created correctly, you can log in by using the HTPasswd identity provider.

3. Make the `new_admin` user a cluster administrator. Log in as both the `new_admin` and `new_developer` users to verify HTPasswd user configuration and cluster privileges.

- 3.1. Assign the `new_admin` user the `cluster-admin` role.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user \
    cluster-admin new_admin
Warning: User 'new_admin' not found
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "new_admin"
```

**Note**

You can safely ignore the warning that the `new_admin` user is not found.

- 3.2. Log in to the cluster as the `new_admin` user to verify that HTPasswd authentication is configured correctly.

```
[student@workstation ~]$ oc login -u new_admin -p 'L@bR3v!ew'
Login successful.

...output omitted...
```

- 3.3. Use the `oc get nodes` command to verify that the `new_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      control-plane, master, worker   14d      v1.27.6+f67aeb3
```

- 3.4. Log in to the cluster as the `new_developer` user to verify that the HTPasswd authentication is configured correctly.

```
[student@workstation ~]$ oc login -u new_developer -p 'L@bR3v!ew'
Login successful.

...output omitted...
```

- 3.5. Use the `oc get nodes` command to verify that the `new_developer` user does not have cluster administration privileges.

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

4. As the `new_admin` user, prevent users from creating projects in the cluster.

- 4.1. Log in to the cluster as the `new_admin` user.

```
[student@workstation ~]$ oc login -u new_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-provisioners
'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 `managers` group, 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 `managers` group.

```
[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 `developers` group and grant edit privileges on the `auth-review` project. Add the `new_developer` user to the group.

- 6.1. Log in to the cluster as the `new_admin` user.

```
[student@workstation ~]$ oc login -u new_admin -p 'L@bR3v!ew'  
Login successful.  
...output omitted...
```

- 6.2. Create a `developers` group.

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

- 6.3. Add the `new_developer` user to the `developers` group.

```
[student@workstation ~]$ oc adm groups add-users developers new_developer  
group.user.openshift.io/developers added: "new_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 `qa` group and grant view privileges on the `auth-review` project. Add the `tester` user to the group.

- 7.1. Create a `qa` group.

```
[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

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 grade auth-review
```

Finish

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- A newly installed OpenShift cluster provides two authentication methods that grant administrative access: the `kubeconfig` file and the `kubeadm` virtual user.
- The HTPasswd identity provider authenticates users against credentials that are stored in a secret. The secret name and other settings for the identity provider are stored inside the OAuth custom resource.
- To manage user credentials by 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, which an identity provider manages, 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 manage 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 role's associations to users and groups.
- To grant a user cluster administrator privileges, assign the `cluster-admin` role to that user.

Chapter 4

Network Security

Goal

Protect network traffic between applications inside and outside the cluster.

Objectives

- Allow and protect network connections to applications inside an OpenShift cluster.
- Restrict network traffic between projects and pods.
- Configure and use automatic service certificates.

Sections

- Protect External Traffic with TLS (and Guided Exercise)
- Configure Network Policies (and Guided Exercise)
- Protect Internal Traffic with TLS (and Guided Exercise)

Lab

- Network Security

Protect External Traffic with TLS

Objectives

- Allow and protect network connections to applications inside an OpenShift cluster.

Accessing Applications 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, whereas others use their own API resource, such as Ingress and Route.

With OpenShift routes, you can expose your applications to external networks, to reach the applications with a unique, publicly accessible hostname. 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 that runs as pods in your cluster:

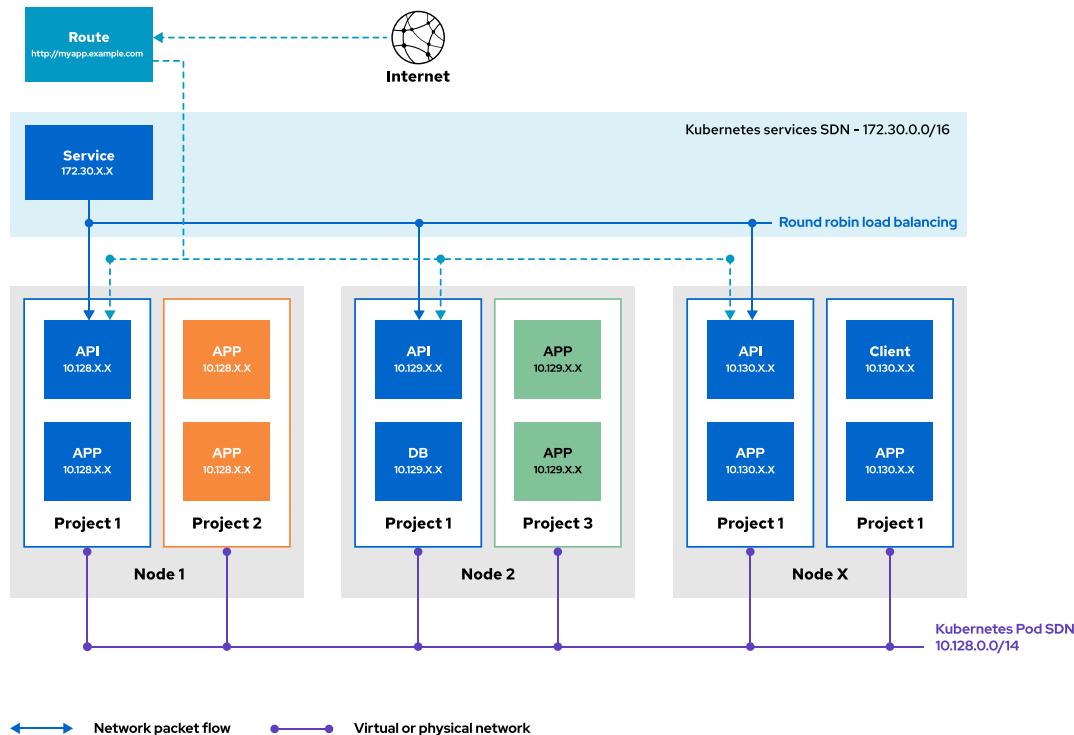


Figure 4.1: Using routes to expose applications

**Note**

For performance reasons, routers send requests directly to pods based on service configuration.

The dotted line in the diagram indicates this implementation. The router accesses the pods through the services network.

Encrypting Routes

Routes can be either encrypted or unencrypted. Encrypted routes support several types of transport layer security (TLS) termination to serve certificates to the client. Unencrypted routes are the simplest to configure, because they require no key or certificates. By contrast, encrypted routes encrypt traffic to and from the pods.

An encrypted route specifies the TLS termination of the route. The following termination types are available:

OpenShift Route Encryption

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 TLS termination from the router. In this mode, the application is responsible for serving certificates for the traffic. Passthrough is a common method for supporting 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 an encrypted route, you need a TLS certificate. The following command shows how to create an encrypted edge route with a custom TLS certificate:

```
[user@host ~]$ oc create route edge \
--service api-frontend --hostname api.apps.acme.com \
--key api.key --cert api.crt ① ②
```

① The --key option requires the certificate private key.

② The --cert option requires the signed certificate.

If the --key and --cert options are omitted, then the RHOCP ingress operator provides a certificate from the internal Certificate Authority (CA). In this case, the route will not reference

the certificate directly. Use the `oc get secrets/router-ca -n openshift-ingress-operator -o yaml` command to view the certificate that the internal CA provides.

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 encrypted:

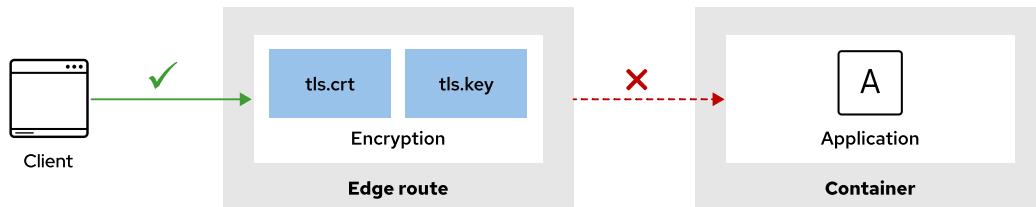


Figure 4.2: Securing applications with edge routes



Note

Network policies can help you to protect the internal traffic between your applications or between projects.

Securing Applications with Passthrough Routes

The previous example demonstrates how to create an edge route, which means an OpenShift route that presents a certificate at the edge. Passthrough routes offer a more 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 provide the certificate 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.

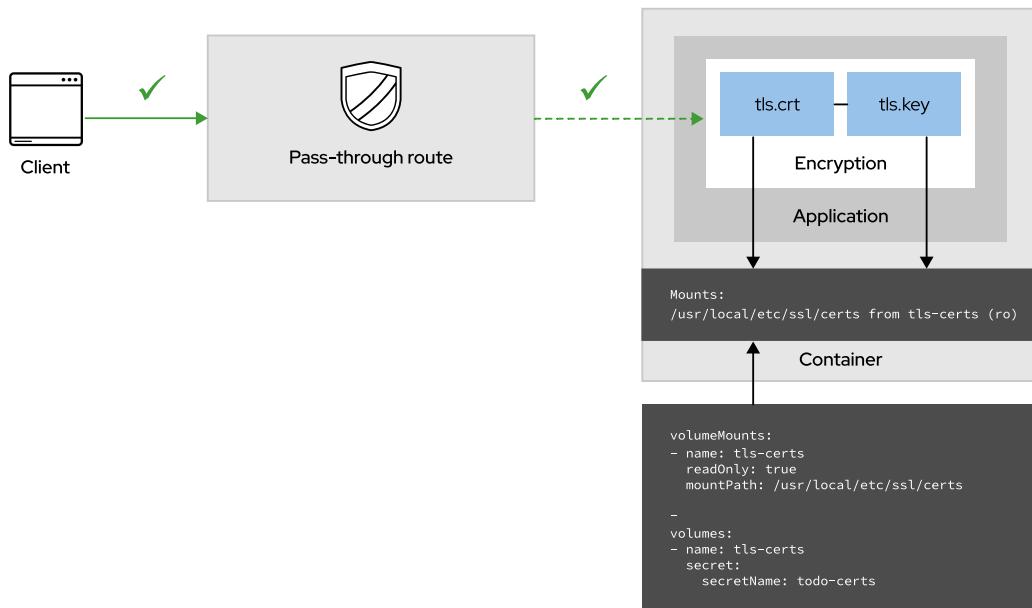


Figure 4.3: Securing applications with passthrough routes

Securing Applications with Re-encrypt Routes

Re-encrypt routes provide end-to-end encryption. First, re-encrypt routes terminate the encryption between an external client and the router. This encryption uses a certificate with a fully qualified domain name (FQDN) that is trusted by the client, such as the `my-app.example.com` hostname.

Then, the router re-encrypts the connection when accessing an internal cluster service. This internal communication requires a certificate for the target service with an OpenShift FQDN, such as the `my-app.namespace.svc.cluster.local` hostname.

The certificates for internal TLS connections require a public key infrastructure (PKI) to sign the certificate. OpenShift provides the `service-ca` controller to generate and sign service certificates for internal traffic. The `service-ca` controller creates a secret that it populates with a signed certificate and key. A deployment can mount this secret as a volume to use the signed certificate. Using the `service-ca` controller is explained later in this chapter.



References

For more information about how to manage routes, refer to the *Configuring Routes* chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#configuring-routes

For more information about how to configure ingress cluster traffic, refer to the *Configuring Ingress Cluster Traffic* chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#configuring-ingress-cluster-traffic

Self-Serviced End-to-end Encryption Approaches for Applications Deployed in OpenShift

<https://cloud.redhat.com/blog/self-serviced-end-to-end-encryption-approaches-for-applications-deployed-in-openshift>

► Guided Exercise

Protect External Traffic with TLS

Expose an application that is secured by TLS certificates.

Outcomes

- 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. The command also gives the `developer` user edit access on the project.

```
[student@workstation ~]$ lab start network-ingress
```

Instructions

As an application developer, you are ready to deploy your application in OpenShift. In this activity, you deploy two versions of the application: one that is exposed over unencrypted traffic (HTTP), and one that is exposed over encrypted traffic (HTTPS).

The container image, which is accessible at `https://registry.ocp4.example.com:8443/redhat-training/todo-angular`, has two tags: `v1.1`, which is the plain text version of the application, and `v1.2`, which is the encrypted version. Your organization uses its own certificate authority (CA) that can sign certificates for the following domains:

- `*.apps.ocp4.example.com`
- `*.ocp4.example.com`

The CA certificate is accessible at `~/D0280/labs/network-ingress/certs/training-CA.pem`. The `passphrase.txt` file contains a unique password that protects the CA key. The `certs` directory 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 `registry.ocp4.example.com:8443/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 few minutes, so that the application can start, and then review the resources in the project.

```
[student@workstation ~]$ oc status
...output omitted...
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 registry.ocp4.example.com:8443/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 hostname 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
todo-http todo-http.apps.ocp4.example.com
```

- 2.5. On the **workstation** machine, open Firefox and access the application URL. Confirm that you can see the application.
- `http://todo-http.apps.ocp4.example.com`
- 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, use the `grep` command to filter on JavaScript resources.

Start by retrieving the name of the main interface, whose IP is `172.25.250.9`.

```
[student@workstation ~]$ ip addr | 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 "angular"
```



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...
<script type="text/javascript" src="assets/js/libs/angular/angular.min.js">
<script type="text/javascript" src="assets/js/libs/angular-route.min.js">
<script type="text/javascript" src="assets/js/libs/angular/angular-animate.min.js">
...output omitted...
```

- 3. Create an encrypted 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 that 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.

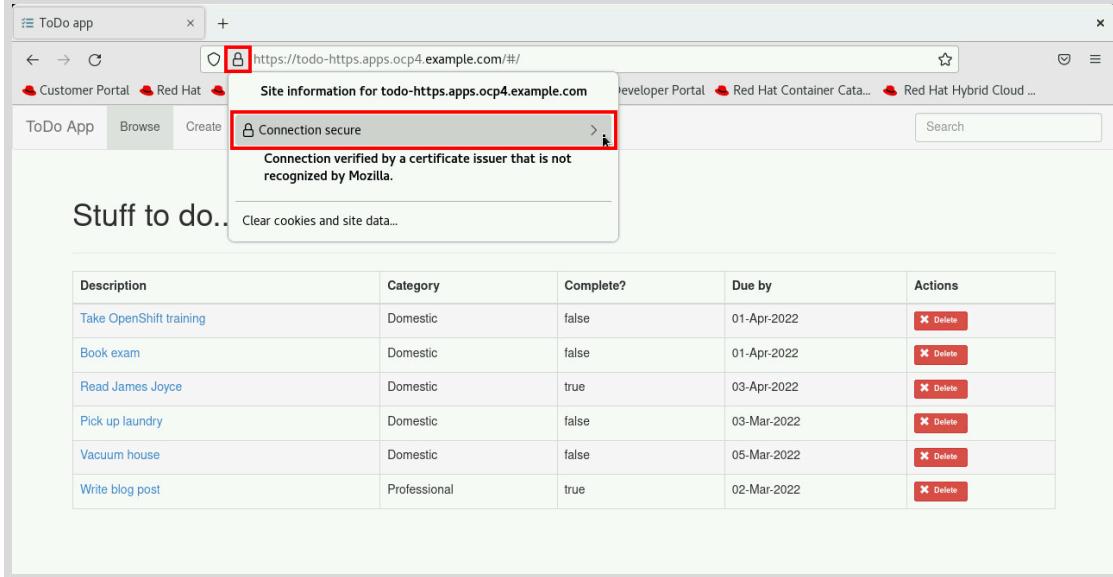
Give the route a hostname of `todo-https.apps.ocp4.example.com`.

```
[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
```

When the `--key` and `--cert` options are omitted, the RHOCP ingress operator creates the required certificate with its own Certificate Authority (CA).

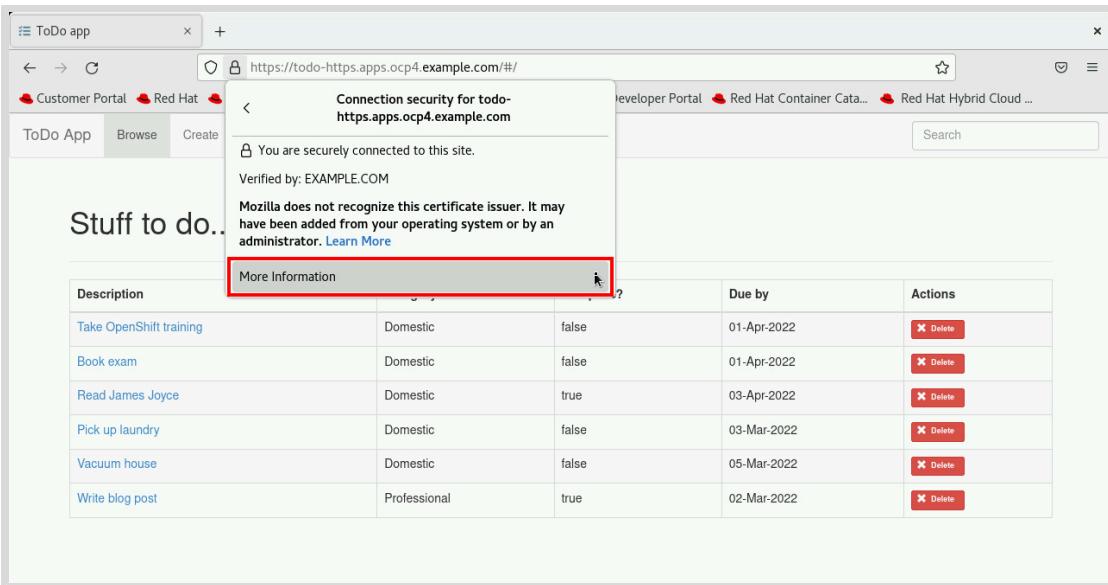
- 3.2. To test the route and read the certificate, open Firefox and access the application URL.
 - <https://todo-https.apps.ocp4.example.com>

Click the **padlock**, and then click the arrow next to **Connection secure**.

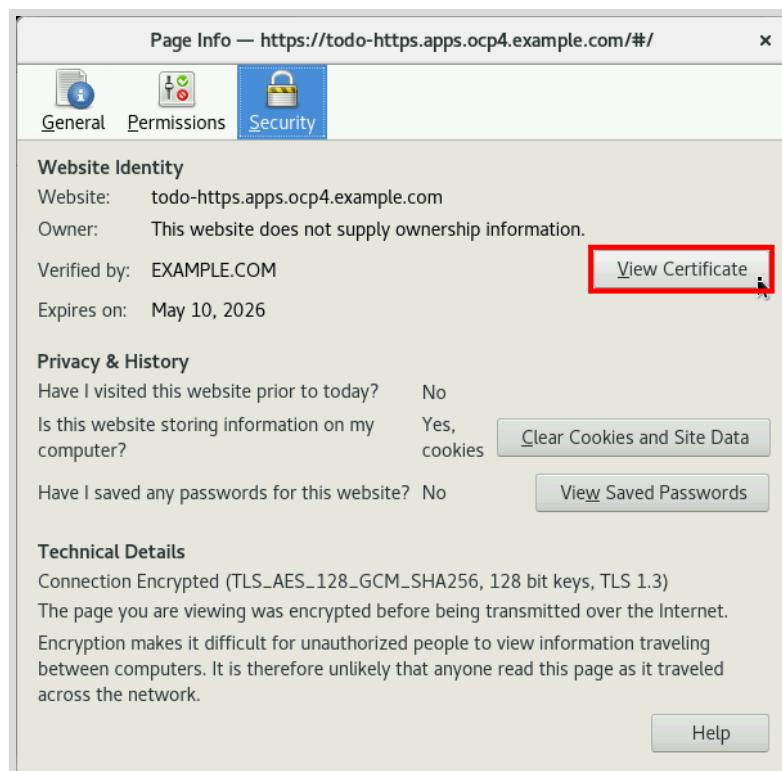


Firefox displays a message that the connection is verified by a certificate issuer that Mozilla does not recognize. This message is displayed because the route signed certificate comes from an internal CA that is installed on the classroom OS. This CA, although not recognized by Mozilla, is valid for the lab environment purposes. If your organization uses a custom public key infrastructure (PKI), then you might see the same message.

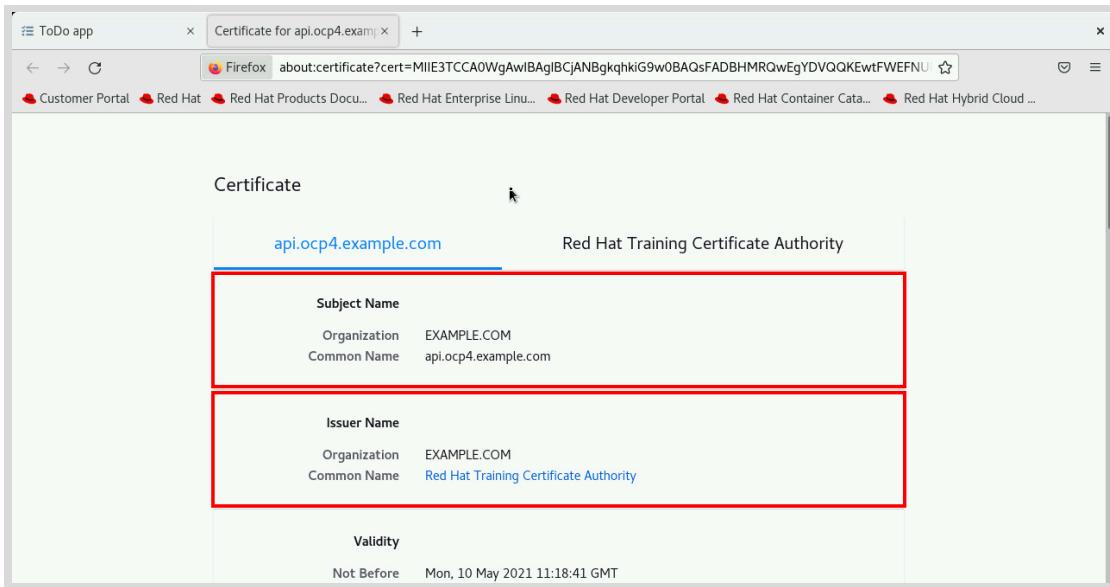
Click **More Information** to display the page information window.



Click **View Certificate** to display the certificate information.



Locate the CN entry to see that the OpenShift ingress operator created the certificate with its own CA.



- 3.3. From the terminal, use the `curl` command with the `-I` and `-v` options to retrieve the connection headers.

The `Server certificate` section shows some information about the certificate. The alternative name matches the name of the route. The output indicates that the remote certificate is trusted because it matches the CA.

```
[student@workstation network-ingress]$ curl -I -v \
  https://todo-https.apps.ocp4.example.com
...output omitted...
* Server certificate:
*  subject: O=EXAMPLE.COM; CN=.api.ocp4.example.com
*  start date: May 10 11:18:41 2021 GMT
*  expire date: May 10 11:18:41 2026 GMT
*  subjectAltName: host "todo-https.apps.ocp4.example.com" matched cert's
  "*.apps.ocp4.example.com"
*  issuer: O=EXAMPLE.COM; CN=Red Hat Training Certificate Authority
*  SSL certificate verify ok.
...output omitted...
```

- 3.4. Although the traffic is encrypted at the edge with a certificate, you can still access the plain text 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
```

- 3.5. Create a debug pod in the `todo-http` deployment. Use the Red Hat Universal Base Image (UBI), which contains tools to interact with containers.

```
[student@workstation network-ingress]$ oc debug -t deployment/todo-http \
--image registry.ocp4.example.com:8443/ubi8/ubi:8.4
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$
```

- 3.6. 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...
```

- 3.7. Exit the debug pod.

```
sh-4.4$ exit
Removing debug pod ...
```

- 3.8. Delete the edge route. In the following 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 an encrypted route in passthrough mode and let the application expose that certificate.

- 4.1. Go to the `~/D0280/labs/network-ingress/certs` directory and list the files.

```
[student@workstation network-ingress]$ cd certs
[student@workstation certs]$ ls -l
total 20
-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 ~/DO280/labs/network-ingress/certs/openssl-commands.txt file.

```
[student@workstation certs]$ openssl genrsa -out training.key 4096
```

- 4.3. Generate the certificate signing request (CSR) for the todo-
https.apps.ocp4.example.com hostname.

```
[student@workstation certs]$ openssl req -new \
-key training.key -out training.csr \
-subj "/C=US/ST=North Carolina/L=Raleigh/O=Red Hat/\
CN=todo-https.apps.ocp4.example.com"
```



Warning

Type the request **subject** on one line. Alternatively, remove the **-subj** option and its content. Without the **-subj** option, the **openssl** command prompts you for the values; indicate a common name (CN) of todo-
https.apps.ocp4.example.com.

- 4.4. Finally, generate the signed certificate. Notice the use of the **-CA** and **-CAkey** options for signing the certificate against the CA. Use the **-passin** option to reuse the password of the CA. Use the **extfile** option 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
Certificate request self-signature ok
subject=C = US, ST = North Carolina, L = Raleigh, O = Red Hat, CN = todo-
https.apps.ocp4.example.com
```

- 4.5. Ensure that the newly created certificate and key are present in the current directory.

```
[student@workstation certs]$ ls -lrt
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-r--r--. 1 student student 352 Aug 3 12:38 training.ext
-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-----. 1 student student 1675 Aug 3 13:38 training.key
-rw-rw-r--. 1 student student 1017 Aug 3 13:39 training.csr
-rw-rw-r--. 1 student student 41 Aug 3 13:40 training-CA.srl
-rw-rw-r--. 1 student student 1399 Aug 3 13:40 training.crt
```

- 4.6. Return to the `network-ingress` directory. This step is important, because the next step involves creating a route that uses the self-signed certificate.

```
[student@workstation certs]$ cd ~/D0280/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.crt` 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 at `~/D0280/labs/network-ingress/todo-app-v2.yaml` points to version 2 of the container image. Examine how the new version of the application is configured to support SSL certificates.

```
[student@workstation network-ingress]$ cat todo-app-v2.yaml
apiVersion: apps/v1
kind: Deployment
...output omitted...
  volumeMounts:
  - name: tls-certs
    readOnly: true
    mountPath: /usr/local/etc/ssl/certs
...output omitted...
  volumes:
  - name: tls-certs
    secret:
      secretName: todo-certs
---
apiVersion: v1
kind: Service
...output omitted...
  ports:
  - name: https
    port: 8443
    protocol: TCP
    targetPort: 8443
...output omitted...
```

The `todo-certs` secret with the SSL certificate is mounted in the container in the `/usr/local/etc/ssl/certs` directory to enable TLS for the application. Additionally, the `todo-app-v2` deployment changes the service to include port 8443.

- 5.3. Run the `oc create` command to create a deployment that uses that image.

```
[student@workstation network-ingress]$ oc create -f todo-app-v2.yaml
deployment.apps/todo-https created
service/todo-https created
```

- 5.4. Wait a couple of minutes to ensure that the application pod is running. Use the `oc set volumes` command to review the volumes that are mounted inside the pod.

```
[student@workstation network-ingress]$ oc set volumes deployment/todo-https
todo-https
secret/todo-certs as tls-certs
mounted at /usr/local/etc/ssl/certs
```

▶ 6. Create the encrypted route.

- 6.1. Run the `oc create route` command to define the new route.

Give the route a hostname 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 to 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 forwards only packets that are encrypted by the application web server certificate.

```
[student@workstation network-ingress]$ curl -vv -I \
--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: Jun 15 01:53:30 2021 GMT
*  expire date: Jun 14 01:53:30 2026 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 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 container image.

```
[student@workstation network-ingress]$ oc debug -t deployment/todo-https \
--image registry.ocp4.example.com:8443/ubi8/ubi:8.4
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 encrypted version.

```
sh-4.4$ curl -I http://172.30.121.154
HTTP/1.1 301 Moved Permanently
Server: nginx/1.14.1
Date: Tue, 15 Jun 2021 02:01:19 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. Clean up the exercise directory and project.

- 8.1. Change to the home directory.

```
[student@workstation network-ingress]$ cd
```

- 8.2. Delete the `network-ingress` project.

```
[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 step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish network-ingress
```

Configure Network Policies

Objectives

- Restrict network traffic between projects and pods.

Managing Network Policies in OpenShift

With network policies, you can configure isolation policies for individual pods. Network policies do not require administrative privileges, and give 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 with network policies, you can separate traffic regardless of where it originates.

In contrast to traditional firewalls, Kubernetes network policies control network traffic between pods by using labels instead of IP addresses. To manage network communication between pods in two namespaces, assign a label to the namespace that needs access to another namespace, and create a network policy that selects these labels. You can also use a network policy to select labels on individual pods to create ingress or egress rules. In network policies, use selectors under spec to assign which destination pods are affected by the policy, and selectors under spec.ingress to assign which source pods are allowed. The following command assigns the `network=network-1` label to the `network-1` namespace:

```
[user@host ~]$ oc label namespace network-1 network=network-1
```

The following examples describe network policies that allow communication between pods in the `network-1` and `network-2` namespaces:

- The following network policy applies to any pods with the `deployment="product-catalog"` label in the `network-1` namespace. The `network-2` namespace has the `network=network-2` label. The policy allows TCP traffic over port 8080 from pods whose label is `role="qa"` in namespaces with the `network=network-2` label.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: network-1-policy
  namespace: network-1
spec:
  podSelector: ①
    matchLabels:
      deployment: product-catalog
  ingress: ②
    - from: ③
      - namespaceSelector:
          matchLabels:
            network: network-2
  podSelector:
    matchLabels:
```

```
role: qa
ports: ④
- port: 8080
  protocol: TCP
```

- ❶ The top-level `podSelector` field is required and defines which pods use the network policy. If the `podSelector` is empty, then all pods in the namespace are matched.
- ❷ The `ingress` field defines a list of ingress traffic rules to apply to the matched pods from the top-level `podSelector` field.
- ❸ 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.
- ❹ The `ports` field is a list of destination ports that allow traffic to reach the selected pods.
- The following network policy allows traffic from any pods in namespaces with the `network=network-1` label into any 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
  namespace: network-2
spec:
  podSelector: {}
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network: network-1
```



Note

Network policies are Kubernetes resources. As such, you can manage them with `oc` commands.

Network Policies Between Projects

One benefit of using network policies is to manage security between projects (tenants), which you cannot do with layer 2 technologies such as VLANs. With this approach, you can create tailored policies between projects to ensure that users can access only 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 can be 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`. With the logic options, you can create specific policy rules. The following examples highlight the differences that the syntax can make:

- This example combines the selectors into one rule, and thereby allows access only from pods with the `app=mobile` label in namespaces with the `network=dev` label. This sample shows a logical `AND` statement.

```
...output omitted...
ingress:
- from:
  - namespaceSelector:
    matchLabels:
      network: dev
  podSelector:
    matchLabels:
      app: mobile
```

- By changing the `podSelector` field in the previous example to be an item in the `from` list, any pods in namespaces with the `network=dev` label or any pods with the `app=mobile` label from any namespace can reach the pods that match the top-level `podSelector` field. This sample shows a logical *OR* statement.

```
...output omitted...
ingress:
- from:
  - namespaceSelector:
    matchLabels:
      network: dev
  - podSelector:
    matchLabels:
      app: mobile
```

Deny-all Network Policies

If a pod is matched by selectors in one or more network policies, then the pod accepts only connections that at least one of those network policies allows. 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: {}
```



Important

If a pod does not match any network policies, then OpenShift does not restrict traffic to that pod. When creating an environment to allow network traffic only explicitly, you must include a deny-all policy.

Allowing Access from OpenShift Cluster Services

When you protect your pods by using network policies, OpenShift cluster services might need explicit policies to access pods. Several common scenarios require explicit policies, including the following ones:

- The router pods that enable access from outside the cluster by using ingress or route resources
- The monitoring service, if your application exposes metrics endpoints

The following policies allow ingress from OpenShift monitoring and ingress pods:

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/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

Network policies do not block traffic from pods that use host networking to pods in the same node.

For example, on a single-node cluster, a deny-all network policy does not prevent ingress pods that use the host network strategy from accessing application pods.

Inside a node, traffic from pods that use host networking is treated differently from traffic from other pods. Network policies control only internal traffic from pods that do not use host networking.

When traffic leaves a node, no such different treatment exists, and network policies control all traffic from other nodes.

For more information about this topic, refer to Network Policies [<https://kubernetes.io/docs/concepts/services-networking/network-policies/#what-you-can-t-do-with-network-policies-at-least-not-yet>]



References

For more information about network policy, refer to the *Network Policy* chapter in the Red Hat OpenShift Container Platform 4.14 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#network-policy

► Guided Exercise

Configure Network Policies

Create network policies and review pod isolation that these network policies created.

Outcomes

- Create network policies to control communication between pods.
- Verify that 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 necessary resource files for the exercise.

```
[student@workstation ~]$ lab start network-policy
```

Instructions

- 1. Log in to the OpenShift cluster and create the `network-policy` project.

- 1.1. Log in to the cluster as the `developer` user with the `developer` password.

```
[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 identical deployments named `hello` and `test`. Create a route to the `hello` deployment.

- 2.1. Create the `hello` deployment that uses the `registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0` container image.

```
[student@workstation ~]$ oc new-app --name hello \
    --image registry.ocp4.example.com:8443/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 the test deployment that uses the `registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0` container image.

```
[student@workstation ~]$ oc new-app --name test \
    --image registry.ocp4.example.com:8443/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...
```

- 2.3. 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 that the test pod can access the hello pod by using the `oc rsh` and `curl` commands.
- 3.1. Open a second terminal and run the script at `~/D0280/labs/network-policy/display-project-info.sh`. This script provides information about the pods, service, and route that are 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. Access the `hello` pod IP address from the `test` pod by using the `oc rsh` and `curl` commands.

```
[student@workstation ~]$ oc rsh test-c4d74c9d5-5pq9s \
curl 10.8.0.13:8080 | grep Hello
<h1>Hello, world from nginx!</h1>
```

- 3.3. Access the `hello` service IP address from the `test` pod by using the `oc rsh` and `curl` commands.

```
[student@workstation ~]$ oc rsh test-c4d74c9d5-5pq9s \
curl 172.30.137.226:8080 | grep Hello
<h1>Hello, world from nginx!</h1>
```

- 3.4. Access the `hello route` hostname by using the `curl` command.

```
[student@workstation ~]$ curl -s hello-network-policy.apps.ocp4.example.com | \
grep Hello
<h1>Hello, world from nginx!</h1>
```

- ▶ 4. Create a project named `different-namespace` that contains a deployment named `sample-app`.

- 4.1. Create the `different-namespace` project.

```
[student@workstation ~]$ oc new-project different-namespace
Now using project "different-namespace" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

- 4.2. Create the `sample-app` deployment from the `registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0` image. The web app listens on port 8080.

```
[student@workstation ~]$ oc new-app --name sample-app \
--image registry.ocp4.example.com:8443/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. Access the `hello` and `test` pods in the `network-policy` project from the `sample-app` pod in the `different-namespace` project.

- 5.1. In the second terminal, view the full name of the `sample-app` pod with the `display-project-info.sh` script.

```
[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
=====
PROJECT: different-namespace

POD NAME
sample-app-d5f945-spx9q
=====
```

- 5.2. In the first terminal, access the `hello` pod IP address from the `sample-app` pod by using the `oc rsh` and `curl` commands.

```
[student@workstation ~]$ oc rsh sample-app-d5f945-spx9q \
curl 10.8.0.13:8080 | grep Hello
<h1>Hello, world from nginx!</h1>
```

- 5.3. Access the `test` pod IP address from the `sample-app` pod by using the `oc rsh` and `curl` commands. Target the IP address that was 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. In the `network-policy` project, create a `deny-all` network policy by using the resource file 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. Change 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` field to target all pods in the `network-policy` project.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
spec:
  podSelector: {}
```

**Note**

A solution is provided at `~/D0280/solutions/network-policy/deny-all.yaml`.

- 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 that the `deny-all` network policy forbids network access to pods in the `network-policy` project.

- 7.1. 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
^C
command terminated with exit code 130
```

- 7.2. Switch to the `different-namespace` project.

```
[student@workstation network-policy]$ oc project different-namespace
Now using project "different-namespace" on server "https://
api.ocp4.example.com:6443".
```

- 7.3. Verify 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 rsh sample-app-d5f945-spx9q \
  curl 10.8.0.14:8080 | grep Hello
^C
command terminated with exit code 130
```

- 8. Create a network policy to allow traffic to the `hello` pod in the `network-policy` project from the `sample-app` pod in the `different-namespace` project via TCP on port 8080. Use the resource file at `~/D0280/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:

```
...output omitted...
spec:
  podSelector:
    matchLabels:
      deployment: hello
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network: different-namespace
  podSelector:
    matchLabels:
      deployment: sample-app
  ports:
    - port: 8080
      protocol: TCP
```

**Note**

A solution is provided at `~/D0280/solutions/network-policy/allow-specific.yaml`.

- 8.2. Apply the network policy from the `allow-specific.yaml` file 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` project.

```
[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 `different-namespace` namespace with the `network=different-namespace` label.

- 9.1. Log in as the `admin` user.

```
[student@workstation network-policy]$ oc login -u admin -p redhatocp
Login successful.

...output omitted...
```

- 9.2. Apply the `network=different-namespace` label with the `oc label` command.

```
[student@workstation network-policy]$ oc label namespace different-namespace \
    network=different-namespace
namespace/different-namespace labeled
```

**Important**

The allow-specific network policy uses labels to match the different-namespace namespace. By default, namespaces and projects do not get any labels automatically.

9.3. Confirm that the different-namespace label was applied.

```
[student@workstation network-policy]$ oc describe namespace different-namespace
Name:           different-namespace
Labels:         network=different-namespace
...output omitted...
```

9.4. Log in as the developer user.

```
[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. Switch to the different-namespace project.

```
[student@workstation network-policy]$ oc project different-namespace
Already on project "different-namespace" on server "https://
api.ocp4.example.com:6443".
```

10.2. Access the hello pod in the network-policy namespace with the `oc rsh` and `curl` commands via the 8080 port.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q \
    curl 10.8.0.13:8080 | grep Hello
<h1>Hello, world from nginx!</h1>
```

- 10.3. Verify that the hello pod cannot be accessed on another port. Because the network policy allows access only to port 8080 on the hello pod, requests to any other port are ignored and eventually time out. Wait a few seconds, and then press `Ctrl+C` to exit the `curl` command when no response occurs.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q \
    curl 10.8.0.13:8181 | grep Hello
^C
command terminated with exit code 130
```

- 10.4. Verify that the `test` pod cannot be accessed from the `sample-app` pod. Wait a few seconds, and then press `Ctrl+C` to exit the `curl` command when no response occurs.

```
[student@workstation network-policy]$ oc rsh sample-app-d5f945-spx9q \
  curl 10.8.0.14:8080 | grep Hello
^C
command terminated with exit code 130
```

- 11. Verify if the `hello` route cannot access the `hello` pod.

- 11.1. Verify if the `hello` pod cannot be accessed via its exposed route.

```
[student@workstation network-policy]$ curl -s \
  hello-network-policy.apps.ocp4.example.com
<h1>Hello, world from nginx!</h1>
```

The lab environment is a single-node cluster. Because the ingress pods use host networking and the application pods are in the same node, the network policy does not block the traffic.

- 12. Create a network policy that allows traffic to the `hello` deployment via the exposed route. Use the resource file at `~/D0280/labs/network-policy/allow-from-openshift-ingress.yaml`.

This step does not have an effect on the lab environment, because the lab environment is a single-node cluster. On a cluster with multiple nodes, this step is required for correct ingress operation.

- 12.1. Use a text editor to replace the `CHANGE_ME` values in the `allow-from-openshift-ingress.yaml` file as follows:

```
...output omitted...
spec:
  podSelector:
    matchLabels:
      deployment: 'hello'
  ingress:
  - from:
    - namespaceSelector:
      matchLabels:
        policy-group.network.openshift.io/ingress: ""
```



Note

A solution is provided at `~/D0280/solutions/network-policy/allow-from-openshift-ingress.yaml`.

- 12.2. Apply the network policy from the `allow-from-openshift-ingress.yaml` file 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
```

12.3. 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
```

12.4. Log in as the `admin` user.

```
[student@workstation network-policy]$ oc login -u admin -p redhatocp
Login successful.

...output omitted...
```

12.5. Access the `hello` pod via the exposed route with the `curl` command.

```
[student@workstation network-policy]$ curl -s \
hello-network-policy.apps.ocp4.example.com | grep Hello
<h1>Hello, world from nginx!</h1>
```

- 13. Close the terminal window that contains the output of the `display-project-info.sh` script, and navigate to the home directory.

```
[student@workstation network-policy]$ cd
```

Finish

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

```
[student@workstation ~]$ lab finish network-policy
```

Protect Internal Traffic with TLS

Objectives

- Configure and use automatic service certificates.

Zero-trust Environments

Zero-trust environments assume that every interaction begins in an untrusted state. Users can access only files or objects that are specifically allowed; communication must be encrypted; and client applications must verify the authenticity of servers.

By default, OpenShift encrypts network traffic between nodes and the control plane, and prevents external entities from reading internal traffic. This encryption provides stronger security than default Kubernetes, which does not automatically encrypt internal traffic. Although the control plane traffic is encrypted, applications in OpenShift do not necessarily verify the authenticity of other applications or encrypt application traffic.

Zero-trust environments require that a trusted certificate authority (CA) signs the certificates that are used to encrypt traffic. By referencing the CA certificate, an application can cryptographically verify the authenticity of another application with a signed certificate.

Service Certificates

OpenShift provides the `service-ca` controller to generate and sign service certificates for internal traffic. The `service-ca` controller creates a secret that it populates with a signed certificate and key. A deployment can mount this secret as a volume to use the signed certificate. Additionally, client applications need to trust the `service-ca` controller CA.

Service Certificate Creation

To generate a certificate and key pair, apply the `service.beta.openshift.io/serving-cert-secret-name=your-secret` annotation to a service. The `service-ca` controller creates the `your-secret` secret in the same namespace if it does not exist, and populates it with a signed certificate and key pair for the service.

```
[user@host ~]$ oc annotate service hello \ ①
  service.beta.openshift.io/serving-cert-secret-name=hello-secret ②
service/hello annotated
```

① The `hello` service is annotated.

② The secret that contains the certificate and key pair is named `hello-secret`.

After OpenShift generates the secret, you must mount the secret in the application deployment. The location to place the certificate and key is application-dependent. The following YAML patch is for an NGINX deployment:

```

spec:
  template:
    spec:
      containers:
        - name: hello
          volumeMounts:
            - name: hello-volume ❶
              mountPath: /etc/pki/nginx/ ❷
      volumes:
        - name: hello-volume ❸
          secret:
            defaultMode: 420 ❹
            secretName: hello-secret ❺
            items:
              - key: tls.crt ❻
                path: server.crt ❼
              - key: tls.key ❽
                path: private/server.key ❾

```

- ❶❸ Defining the volume as hello-volume.
- ❷ The application-specific mount path.
- ❹ The read-write permissions that the application recommends.
- ❺ The secret that the earlier annotation defined.
- ❻❼ The secret has `tls.crt` as the signed certificate and `tls.key` as the key.
- ❼❾ The application-specific destinations for the certificate and key.

After mounting the secret to the application container, the application can use the signed certificate for TLS traffic.

Client Service Application Configuration

For a client service application to verify the validity of a certificate, the application needs the CA bundle that signed that certificate. The `service-ca` controller injects the CA bundle when you apply the `service.beta.openshift.io/inject-cabundle=true` annotation to an object. You can apply the annotation to configuration maps, API services, custom resource definitions (CRD), mutating webhooks, and validating webhooks.

Configuration Maps

Apply the `service.beta.openshift.io/inject-cabundle=true` annotation to a configuration map to inject the CA bundle into the `data: { service-ca.crt }` field. The `service-ca` controller replaces all data in the selected configuration map with the CA bundle. You must therefore use a dedicated configuration map to prevent overwriting existing data.

```
[user@host ~]$ oc annotate configmap ca-bundle \
  service.beta.openshift.io/inject-cabundle=true
configmap/ca-bundle annotated
```

API service

Applying the annotation to an API service injects the CA bundle into the `spec.caBundle` field.

CRD

Applying the annotation to a CRD injects the CA bundle into the `spec.conversion.webhook.clientConfig.caBundle` field.

Mutating or validating webhook

Applying the annotation to a mutating webhook or validating webhook injects the CA bundle into the `clientConfig.caBundle` field.

Key Rotation

The service CA certificate is valid for 26 months by default and is automatically rotated after 13 months. After rotation is a 13-month grace period where the original CA certificate is still valid. During this grace period, each pod that is configured to trust the original CA certificate must be restarted in some way. A service restart automatically injects the new CA bundle.

You can also manually rotate the certificate for the service CA and for generated service certificates. To rotate a generated service certificate, delete the existing secret, and the `service-ca` controller automatically generates a new one.

```
[user@host ~]$ oc delete secret certificate-secret
secret/certificate-secret deleted
```

To manually rotate the service CA certificate, delete the `signing-key` secret in the `openshift-service-ca` namespace.

```
[user@host ~]$ oc delete secret/signing-key -n openshift-service-ca
secret/signing-key deleted
```

This process immediately invalidates the former service CA certificate. You must restart all pods that use it, for TLS to function.

Alternatives to Service Certificates

Other options can handle TLS encryption inside an OpenShift cluster, such as a service mesh or the `certmanager` operator.

You can use the `certmanager` operator to delegate the certificate signing process to a trusted external service, and also to renew a certificate.

You can also use Red Hat OpenShift Service Mesh for encrypted service-to-service communication and for other advanced features. Service mesh is an advanced topic and is not covered in the course.



References

For more information about service certificates, refer to the *Securing Service Traffic Using Service Serving Certificate Secrets* section in the *Configuring Certificates* chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/security_and_compliance/index#understanding-service-serving_service-serving-certificate

For more information about service mesh, refer to the *About OpenShift Service Mesh* section in the *Service Mesh 2.x* chapter in the Red Hat OpenShift Container Platform 4.14 Service Mesh documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/service_mesh/index#ossm-about

For more information about the `cert-manager` operator, refer to the *cert-manager Operator for Red Hat OpenShift* chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/security_and_compliance/index#cert-manager-operator-about

Red Hat Topics - What Is Zero Trust?

<https://www.redhat.com/en/topics/security/what-is-zero-trust>

► Guided Exercise

Protect Internal Traffic with TLS

Configure two applications to connect securely inside the cluster by using TLS certificates that OpenShift manages.

Outcomes

- Generate service certificates with the `service-ca` controller.
- Mount a service certificate by using secrets.
- Use a configuration map to inject a service certificate into a pod.

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 OpenShift cluster is ready and creates the `network-svccerts` project and `server` deployment for the exercise. The command also creates a test pod named `no-ca-bundle` for use later in the exercise.

```
[student@workstation ~]$ lab start network-svccerts
```

Instructions

In this exercise, you work with the `server` deployment, which has an NGINX container that serves a "Hello World!" page with the HTTPS protocol. This deployment differs from earlier NGINX deployments, because it allows only the HTTPS protocol. The `server` application expects the existence of a certificate that you create in the exercise steps.

► 1. Log in to the OpenShift cluster as the `admin` user and switch to the `network-svccerts` project.

1.1. Use the `oc login` command to log in to `api.ocp4.example.com:6443` as the `admin` user with the `redhatocp` password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

1.2. Use the `oc project` command to switch to the `network-svccerts` project.

```
[student@workstation ~]$ oc project network-svccerts
Now using project "network-svccerts" on server "https://
api.ocp4.example.com:6443".
```

- 2. Generate a service certificate and secret that are named `server-secret` for the `server` service, and then mount the secret in the `server` deployment.
- 2.1. Annotate the `server` service with `service.beta.openshift.io/serving-cert-secret-name=server-secret` by using the `oc annotate` command. It automatically creates a secret named `server-secret`, which is populated with a signed TLS key and certificate.

```
[student@workstation ~]$ oc annotate service server \
  service.beta.openshift.io/serving-cert-secret-name=server-secret
service/server annotated
```

- 2.2. Use the `oc describe` command to view the service and secret descriptions to verify that OpenShift created the secret.

```
[student@workstation ~]$ oc describe service server
...output omitted...
Annotations:      service.beta.openshift.io/serving-cert-secret-name: server-
secret
                  service.beta.openshift.io/serving-cert-signed-by: openshift-
service-serving-signer@1667565598
...output omitted...

[student@workstation ~]$ oc describe secret server-secret
Name:            server-secret
Namespace:       network-svccerts
...output omitted...
Type:  kubernetes.io/tls

Data
=====
tls.key:  1675 bytes
tls.crt:  2615 bytes
```

- 2.3. Use a text editor to create a patch file to mount the `server-secret` secret in the `server` deployment. Edit the resource file at `~/D0280/labs/network-svccerts/server-secret.yaml`. Replace the `CHANGE_ME` sections as shown in the following example:

```
spec:
  template:
    spec:
      containers:
        - name: server
          volumeMounts:
            - name: server-secret
              mountPath: /etc/pki/nginx/
      volumes:
        - name: server-secret
          secret:
            defaultMode: 420
            secretName: server-secret
            items:
              - key: tls.crt
```

```
path: server.crt
- key: tls.key
  path: private/server.key
```

- 2.4. Apply the patch file to the server deployment with the `oc patch` command.

```
[student@workstation ~]$ oc patch deployment server \
--patch-file ~/D0280/labs/network-svccerts/server-secret.yaml
deployment.apps/server patched
```

- 2.5. Use the `openssl s_client` command in the no-ca-bundle pod to verify that OpenShift supplied the server deployment with a certificate. Verify that the no-ca-bundle pod needs to configure the CA that issued the OpenShift service certificate for certificate validation.

```
[student@workstation ~]$ oc exec no-ca-bundle -- \
  openssl s_client -connect server.network-svccerts.svc:443
depth=1 CN = openshift-service-serving-signer@1667565598
CONNECTED(00000004)
---
Certificate chain
  0 s:CN = server.network-svccerts.svc
    i:CN = openshift-service-serving-signer@1667565598
  1 s:CN = openshift-service-serving-signer@1667565598
    i:CN = openshift-service-serving-signer@1667565598
---
...output omitted...
verify error:num=19:self signed certificate in certificate chain
DONE
```



Note

The output shows the `verify error:num=19:self signed certificate in certificate chain` error, because the no-ca-bundle pod is not configured with the OpenShift cluster's CA bundle.

- 3. Generate the ca-bundle configuration map that contains the service CA bundle, and use it to create the client pod.

- 3.1. Create an empty configuration map named `ca-bundle` by using the `oc create` command.

```
[student@workstation ~]$ oc create configmap ca-bundle
configmap/ca-bundle created
```

- 3.2. Annotate the `ca-bundle` configuration map with `service.beta.openshift.io/inject-cabundle=true` by using the `oc annotate` command.

```
[student@workstation ~]$ oc annotate configmap ca-bundle \
  service.beta.openshift.io/inject-cabundle=true
configmap/ca-bundle annotated
```

- 3.3. View the YAML output of the `ca-bundle` configuration map to verify that the CA bundle is present.

```
[student@workstation ~]$ oc get configmap ca-bundle -o yaml
...output omitted...
data:
  service-ca.crt: |
    -----BEGIN CERTIFICATE-----
...output omitted...
```

- 3.4. Use a text editor to add the `ca-bundle` configuration map to the `client.yaml` pod definition. Edit the resource file at `~/D0280/labs/network-svccerts/client.yaml`. Replace the `CHANGE_ME` sections of the file as shown in the following example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
  labels:
    app: client
  name: client
  namespace: network-svccerts
spec:
  replicas: 1
  selector:
    matchLabels:
      deployment: client
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      annotations:
        openshift.io/generated-by: OpenShiftNewApp
      labels:
        deployment: client
    spec:
      containers:
        - image: registry.ocp4.example.com:8443/redhattraining/hello-world-nginx
          imagePullPolicy: IfNotPresent
          name: client-deploy
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - mountPath: /etc/pki/ca-trust/extracted/pem
              name: trusted-ca
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
        dnsPolicy: ClusterFirst
        restartPolicy: Always
```

```
schedulerName: default-scheduler
terminationGracePeriodSeconds: 30
volumes:
  - configMap:
      defaultMode: 420
      name: ca-bundle
      items:
        - key: service-ca.crt
          path: tls-ca-bundle.pem
    name: trusted-ca
  name: trusted-ca
```

- 3.5. Apply the `client.yaml` file with the `oc apply` command to create the `client` pod.

```
[student@workstation ~]$ oc apply -f ~/DO280/labs/network-svccerts/client.yaml
...output omitted...
pod/client created
```

- ▶ 4. Show that the `server` service is now accessible over HTTPS with a certificate that is signed by the OpenShift cluster.
- 4.1. Use the `curl` command within the `client` pod to test that the `server` service is accessible on HTTPS.

```
[student@workstation ~]$ oc exec deploy/client -- \
  curl -s https://server.network-svccerts.svc
<html>
<body>
  <h1>Hello, world from nginx!</h1>
</body>
</html>
```

- 4.2. Use the `openssl s_client` command within the `client` pod to verify that the certificate is signed by the OpenShift cluster.

```
[student@workstation ~]$ oc exec deploy/client -- \
  openssl s_client -connect server.network-svccerts.svc:443
CONNECTED(00000004)
---
Certificate chain
  0 s:CN = server.network-svccerts.svc
    i:CN = openshift-service-serving-signer@1667565598
  1 s:CN = openshift-service-serving-signer@1667565598
    i:CN = openshift-service-serving-signer@1667565598
---
...output omitted...
verify return:1
DONE
```

Finish

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

```
[student@workstation ~]$ lab finish network-svccerts
```

▶ Lab

Network Security

Configure firewall rules to protect microservice communication, and also configure TLS encryption between those microservices and for external access.

Outcomes

- Encrypt internal traffic between pods by using TLS service secrets that OpenShift generates.
- Route external traffic to terminate TLS within the cluster.
- Restrict ingress traffic for a group of pods by using network policies.

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 start network-review
```

This command ensures that the environment is ready and copies the necessary files for this exercise.

This command also deploys an API that is composed of a product and a stock microservice to the `network-review` project.

The product microservice is the entry point to the API. The stock microservice provides only additional information to the product response. If the product microservice cannot reach the stock microservice, then the product microservice returns the `-1` value.

The developer deployed the API without the security configuration. You must configure TLS for end-to-end communications and restrict the ingress to pods for both microservices.

To complete the exercise, the following URLs must respond without errors:

- `https://stock.network-review.svc.cluster.local/product/1`
- `https://product.apps.ocp4.example.com/products`



Note

The `lab start` deploys solution files in the `~/D0280/solutions/network-review/` directory.

Instructions

1. Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.
2. Create the `stock-service-cert` secret for the OpenShift service certificate to encrypt communications between the `product` and the `stock` microservices.

3. Configure TLS on the stock microservice by using the stock-service-cert secret that OpenShift generates.

Use the following settings in the deployment to configure TLS:

- Set the path for the certificate and key to /etc/pki/stock/.
- Set the TLS_ENABLED environment variable to "true".
- Update the liveness and readiness probes to use TLS.
- Change the service to listen on the standard HTTPS 443 port.

4. Configure TLS between the product and the stock microservices by using the internal Certificate Authority (CA) from OpenShift.

The product microservice requires the following settings:

- The CERT_CA environment variable that is set to /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem to access the OpenShift CA
- The STOCK_URL environment variable with the HTTPS protocol

5. Configure TLS on the product microservice by using a signed certificate by a corporate CA to accept TLS connections from outside the cluster.

You have the CA certificate and the signed certificate for the product.apps.ocp4.example.com domain in the certs directory of the lab.

Use the following settings in the deployment to configure TLS:

- Set the path for the certificate and key to /etc/pki/product/.
- Set the TLS_ENABLED environment variable to the "true" value.
- Update the liveness and readiness probes to use TLS.

6. Expose the product microservice to outer cluster access by using the FQDN in the signed certificate by the corporate CA. Use the product.apps.ocp4.example.com hostname.

7. Configure network policies to accept only ingress connections to the stock pod on the 8085 port that come from a pod with the app=product label.

8. Configure network policies to accept only ingress connections to the product pod on the 8080 port that come from the OpenShift router pods.

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 grade network-review
```

Finish

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Network Security

Configure firewall rules to protect microservice communication, and also configure TLS encryption between those microservices and for external access.

Outcomes

- Encrypt internal traffic between pods by using TLS service secrets that OpenShift generates.
- Route external traffic to terminate TLS within the cluster.
- Restrict ingress traffic for a group of pods by using network policies.

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 start network-review
```

This command ensures that the environment is ready and copies the necessary files for this exercise.

This command also deploys an API that is composed of a product and a stock microservice to the `network-review` project.

The product microservice is the entry point to the API. The stock microservice provides only additional information to the product response. If the product microservice cannot reach the stock microservice, then the product microservice returns the `-1` value.

The developer deployed the API without the security configuration. You must configure TLS for end-to-end communications and restrict the ingress to pods for both microservices.

To complete the exercise, the following URLs must respond without errors:

- `https://stock.network-review.svc.cluster.local/product/1`
- `https://product.apps.ocp4.example.com/products`



Note

The `lab start` deploys solution files in the `~/D0280/solutions/network-review/` directory.

Instructions

1. Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.
 - 1.1. Use the `oc login` command to log in to your OpenShift cluster as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

2. Create the `stock-service-cert` secret for the OpenShift service certificate to encrypt communications between the product and the stock microservices.

- 2.1. Change to the `network-review` project.

```
[student@workstation ~]$ oc project network-review
Now using project "network-review" on server "https://api.ocp4.example.com:6443"
```

- 2.2. Change to the `~/D0280/labs/network-review` directory to access the lab files.

```
[student@workstation ~]$ cd ~/D0280/labs/network-review
```

- 2.3. Edit the `stock-service.yaml` manifest to configure the `stock` service with the `service.beta.openshift.io/serving-cert-secret-name: stock-service-cert` annotation. This annotation creates the `stock-service-cert` secret with the service certificate and the key.

```
apiVersion: v1
kind: Service
metadata:
  name: stock
  namespace: network-review
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: stock-service-cert
spec:
...output omitted...
```

- 2.4. Apply the stock service changes by using the `oc apply` command.

```
[student@workstation network-review]$ oc apply -f stock-service.yaml
service/stock configured
```

- 2.5. Verify that the `stock-service-cert` secret contains a valid certificate for the `stock.network-review.svc` hostname in the `tls.crt` secret key. Decode the secret output with the `base64` command by using the `-d` option. Then, use the `openssl x509` command to read the output from standard input, and use the `-text` option to print the certificate in text form.

```
[student@workstation network-review]$ oc get secret stock-service-cert \
--output="jsonpath={.data.tls\.crt}" \
| base64 -d \
| openssl x509 -text
...output omitted...
X509v3 Subject Alternative Name:
DNS:stock.network-review.svc, DNS:stock.network-review.svc.cluster.local
...output omitted...
```

3. Configure TLS on the stock microservice by using the stock-service-cert secret that OpenShift generates.

Use the following settings in the deployment to configure TLS:

- Set the path for the certificate and key to /etc/pki/stock/.
- Set the TLS_ENABLED environment variable to "true".
- Update the liveness and readiness probes to use TLS.
- Change the service to listen on the standard HTTPS 443 port.

- 3.1. Edit the stock-deployment.yaml file to mount the stock-service-cert secret on the /etc/pki/stock/ path.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stock
  namespace: network-review
spec:
  ...output omitted...
  spec:
    containers:
      - name: stock
    ...output omitted...
    env:
      - name: TLS_ENABLED
        value: "false"
    volumeMounts:
      - name: stock-service-cert
        mountPath: /etc/pki/stock/
  volumes:
    - name: stock-service-cert
      secret:
        defaultMode: 420
        secretName: stock-service-cert
```

- 3.2. Edit the stock deployment in the stock-deployment.yaml file to configure TLS for the application and for the liveness and readiness probes on lines 26, 31, and 34.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stock
  namespace: network-review
spec:
```

```
...output omitted...
spec:
  containers:
    - name: stock
  ...output omitted...
  ports:
    - containerPort: 8085
  readinessProbe:
    httpGet:
      port: 8085
      path: /readyz
      scheme: HTTPS
  livenessProbe:
    httpGet:
      port: 8085
      path: /livez
      scheme: HTTPS
  env:
    - name: TLS_ENABLED
      value: "true"
  ...output omitted...
```

- 3.3. Apply the stock deployment updates by using the `oc apply` command.

```
[student@workstation network-review]$ oc apply -f stock-deployment.yaml
deployment/stock configured
```

- 3.4. Edit the `stock-service.yaml` file to configure the `stock` service to listen on the standard HTTPS 443 port.

```
apiVersion: v1
kind: Service
metadata:
  name: stock
  namespace: network-review
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: stock-service-cert
spec:
  selector:
    app: stock
  ports:
    - port: 443
      targetPort: 8085
      name: https
```

- 3.5. Apply the `stock` service changes by using the `oc apply` command.

```
[student@workstation network-review]$ oc apply -f stock-service.yaml
service/stock configured
```

4. Configure TLS between the `product` and the `stock` microservices by using the internal Certificate Authority (CA) from OpenShift.

The `product` microservice requires the following settings:

- The CERT_CA environment variable that is set to /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem to access the OpenShift CA
 - The STOCK_URL environment variable with the HTTPS protocol
- 4.1. Edit the configuration map in the service-ca-configmap.yaml file to add the service.beta.openshift.io/inject-cabundle: "true" annotation. This annotation injects the OpenShift internal CA into the service-ca configuration map.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: service-ca
  namespace: network-review
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
data: {}
```

- 4.2. Create the service-ca configuration map by using the oc create command.

```
[student@workstation network-review]$ oc create -f service-ca-configmap.yaml
configmap/service-ca created
```

- 4.3. Verify that OpenShift injects the CA certificate by describing the service-ca configuration map with the oc describe command.

```
[student@workstation network-review]$ oc describe configmap service-ca
Name:       service-ca
Namespace:   network-review
Labels:     <none>
Annotations: service.beta.openshift.io/inject-cabundle: true

Data
=====
service-ca.crt:
-----
-----BEGIN CERTIFICATE-----
```

- 4.4. Edit the product-deployment.yaml file to configure the product deployment to use the service-ca configuration map, to add the CERT_CA environment variable, and to update the STOCK_URL environment variable to use the HTTPS protocol after line 32.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product
  namespace: network-review
spec:
  ...output omitted...
  spec:
    containers:
```

```

- name: product
...output omitted...
env:
  - name: CERT_CA
    value: "/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem"
  - name: TLS_ENABLED
    value: "false"
  - name: STOCK_URL
    value: "https://stock.network-review.svc"
volumeMounts:
  - name: trusted-ca
    mountPath: /etc/pki/ca-trust/extracted/pem
volumes:
  - name: trusted-ca
    configMap:
      defaultMode: 420
      name: service-ca
      items:
        - key: service-ca.crt
          path: tls-ca-bundle.pem

```

- 4.5. Apply the product deployment updates by using the `oc apply` command.

```
[student@workstation network-review]$ oc apply -f product-deployment.yaml
deployment/product configured
```

- 4.6. Send a request to the `https://stock.network-review.svc/product/1` URL from product deployment to verify that you can query the stock microservice by using HTTPS. Run the `oc exec` command to run the `curl` command to send a request to the stock microservice.

```
[student@workstation network-review]$ oc exec deployment/product \
-- curl -s https://stock.network-review.svc/product/1
10
```

5. Configure TLS on the product microservice by using a signed certificate by a corporate CA to accept TLS connections from outside the cluster.

You have the CA certificate and the signed certificate for the `product.apps.ocp4.example.com` domain in the `certs` directory of the lab.

Use the following settings in the deployment to configure TLS:

- Set the path for the certificate and key to `/etc/pki/product/`.
- Set the `TLS_ENABLED` environment variable to the "true" value.
- Update the liveness and readiness probes to use TLS.

- 5.1. Create the `passthrough-cert` secret by using the `product.pem` certificate and the `product.key` key from the lab directory.

```
[student@workstation network-review]$ oc create secret tls passthrough-cert \
--cert certs/product.pem --key certs/product.key
secret/passthrough-cert created
```

- 5.2. Edit the `product-deployment.yaml` file to mount the `passthrough-cert` secret on the `/etc/pki/product/` path after line 39.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product
spec:
  ...output omitted...
  spec:
    containers:
      - name: product
    ...output omitted...
    volumeMounts:
      - name: passthrough-cert
        mountPath: /etc/pki/product/
      - name: trusted-ca
        mountPath: /etc/pki/ca-trust/extracted/pem
    volumes:
      - name: passthrough-cert
        secret:
          defaultMode: 420
          secretName: passthrough-cert
      - name: trusted-ca
        configMap:
          defaultMode: 420
          name: service-ca
          items:
            - key: service-ca.crt
              path: tls-ca-bundle.pem
```

- 5.3. Edit the `product-deployment.yaml` file to configure TLS for the application and for the liveness and readiness probes on lines 26, 31, and 36.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product
spec:
  ...output omitted...
  spec:
    containers:
      - name: product
    ...output omitted...
    ports:
      - containerPort: 8080
    readinessProbe:
      httpGet:
        port: 8080
        path: /readyz
        scheme: HTTPS
    livenessProbe:
      httpGet:
        port: 8080
```

```

    path: /livez
    scheme: HTTPS
  env:
    - name: TLS_ENABLED
      value: "true"
    - name: STOCK_URL
      value: "https://stock.network-review.svc"
...output omitted...

```

- 5.4. Apply the product deployment updates by using the `oc apply` command.

```
[student@workstation network-review]$ oc apply -f product-deployment.yaml
deployment.apps/product configured
```

6. Expose the product microservice to outer cluster access by using the FQDN in the signed certificate by the corporate CA. Use the `product.apps.ocp4.example.com` hostname.

- 6.1. Create a passthrough route for the product service by using the `product.apps.ocp4.example.com` hostname.

```
[student@workstation network-review]$ oc create route passthrough product-https \
--service product --port 8080 \
--hostname product.apps.ocp4.example.com
route.route.openshift.io/product-https created
```

- 6.2. Verify that you can query the product microservice from outside the cluster by using the `curl` command with the `ca.pem` CA certificate.

```
[student@workstation network-review]$ curl --cacert certs/ca.pem \
https://product.apps.ocp4.example.com/products
[{"id":1,"name":"rpi4_4gb","stock":10}, {"id":2,"name":"rpi4_8gb","stock":5}]
```

7. Configure network policies to accept only ingress connections to the `stock` pod on the 8085 port that come from a pod with the `app=product` label.

- 7.1. Edit the `stock-ingresspolicy.yaml` to add the network policy specification.

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ingress-stock-policy
spec:
  podSelector:
    matchLabels:
      app: stock
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: product
  ports:
    - protocol: TCP
      port: 8085

```

7.2. Create the network policy.

```
[student@workstation network-review]$ oc create -f stock-ingresspolicy.yaml
networkpolicy.networking.k8s.io/stock-ingress-policy created
```

8. Configure network policies to accept only ingress connections to the **product** pod on the 8080 port that come from the OpenShift router pods.
 - 8.1. Edit the **product-ingresspolicy.yaml** file to accept ingress connections from router pods by adding a namespace selector with the **policy-group.network.openshift.io/ingress** label.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: product-ingress-policy
spec:
  podSelector:
    matchLabels:
      app: product
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              policy-group.network.openshift.io/ingress: ""
  ports:
    - protocol: TCP
      port: 8080
```

8.2. Create the network policy.

```
[student@workstation network-review]$ oc create -f product-ingresspolicy.yaml
networkpolicy.networking.k8s.io/product-ingress-policy created
```

8.3. Change to the home directory.

```
[student@workstation network-ingress]$ 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 grade network-review
```

Finish

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

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

Summary

- With OpenShift routes, you can expose your applications to external networks securely.
- The types of route encryption are edge, passthrough, and re-encryption.
- With network policies, you can configure isolation policies for individual pods.
- You can use network policies to create logical zones in the SDN that map to your organization network zones.
- In contrast to traditional firewalls, Kubernetes network policies control network traffic between pods by using labels instead of IP addresses.
- OpenShift provides the `service-ca` controller to generate and sign service certificates for internal traffic.
- To generate a certificate and key pair, apply the `service.beta.openshift.io/serving-cert-secret-name=your-secret` annotation to a service.
- OpenShift can inject its CA into configuration maps with a custom annotation. Client applications can use these configuration maps to validate connections to services that run in the cluster.

Chapter 5

Expose non-HTTP/SNI Applications

Goal

Expose applications to external access without using an ingress controller.

Objectives

- Expose applications to external access by using load balancer services.
- Expose applications to external access by using a secondary network.

Sections

- Load Balancer Services (and Guided Exercise)
- Multus Secondary Networks (and Guided Exercise)

Lab

- Expose non-HTTP/SNI Applications

Load Balancer Services

Objectives

- Expose applications to external access by using load balancer services.

Exposing Non-HTTP Services

When you use Kubernetes, you run workloads that provide services to users. You create resources such as deployments to run workloads, for example a web application. Ingresses and routes provide a way to expose the services that these workloads implement. However, in some scenarios, ingresses and routes are not sufficient to expose the service that a pod provides.

Many internet services implement a process that listens on a given port and IP address. For example, a service that uses the 1.2.3.4 IP address runs an SSH server that listens on port 22. Clients connect to port 22 on that IP address to use the SSH service.

Web servers implement the HTTP protocol and other related protocols such as HTTPS.

Kubernetes ingresses and OpenShift routes use the virtual hosting property of the HTTP protocol to expose web services that are running on the cluster. Ingresses and routes run a single web server that uses virtual hosting to route each incoming request to a Kubernetes service by using the request hostname.

For example, ingresses can route requests for the `https://a.example.com` URL to a Kubernetes service in the cluster, and can route requests for the `https://b.example.com` URL to a different service in the cluster.

However, many protocols do not have equivalent features. Ingress and route resources can expose only HTTP services. To expose non-HTTP services, you must use a different resource. Because these resources cannot expose multiple services on the same IP address and port, they require more setup effort, and might require more resources, such as IP addresses.



Important

Preferably use ingresses and routes to expose services when possible.

Kubernetes Services

Kubernetes workloads are flexible resources that can create many pods. By creating multiple pods for a workload, Kubernetes can provide increased reliability and performance. If a pod fails, then other pods can continue providing a service. With multiple pods, which possibly run on different systems, workloads can use more resources for increased performance.

However, if many pods provide a workload service, then users of the service can no longer access the service by using the combination of a single IP address and a port. To provide transparent access to workload services that run on multiple pods, Kubernetes uses resources of the Service type. A service resource contains the following information:

- A selector that describes the pods that run the service

- A list of the ports that provide the service on the pods

Different types of Kubernetes services exist, each with different purposes:

Internal communication

Services of the `ClusterIP` type provide service access within the cluster.

Exposing services externally

Services of the `NodePort` and `LoadBalancer` types, as well as the use of the external IP feature of `ClusterIP` services, expose services that are running in the cluster to outside the cluster.

Different providers can implement Kubernetes services, by using the `type` field of the service resource.

Although these services are useful in specific scenarios, some services require extra configuration, and they can pose security challenges. Load balancer services have fewer limitations and provide load balancing.

Load Balancer Services

Load balancer services require the use of network features that are not available in all environments.

For example, cloud providers typically provide their own load balancer services. These services use features that are specific to the cloud provider.

If you run a Kubernetes cluster on a cloud provider, controllers in Kubernetes use the cloud provider's APIs to configure the required cloud provider resources for a load balancing service. On environments where managed load balancer services are not available, you must configure a load balancer component according to the specifics of your network.

The MetalLB Component

MetalLB is a load balancer component that provides a load balancing service for clusters that do not run on a cloud provider, such as a bare metal cluster, or clusters that run on hypervisors. MetalLB operates in two modes: layer 2 and Border Gateway Protocol (BGP), with different properties and requirements. You must plan the use of MetalLB to consider your requirements and your network design.

MetalLB is an operator that you can install with the Operator Lifecycle Manager. After installing the operator, you must configure MetalLB through its custom resource definitions. In most situations, you must provide MetalLB with an IP address range.

Using Load Balancer Services

When a cluster has a configured load balancer component, you can create services of the `LoadBalancer` type to expose non-HTTP services outside the cluster.

For example, the following resource definition exposes port 1234 on pods with the `example` value for the `name` label.

```
apiVersion: v1
kind: Service
metadata:
  name: example-lb
  namespace: example
```

```

spec:
  ports:
    - port: 1234 ①
      protocol: TCP
      targetPort: 1234
  selector:
    name: example ②
  type: LoadBalancer ③

```

- ①** Exposed port
- ②** Pod selector
- ③** LoadBalancer service type You can also use the `kubectl expose` command with the `--type LoadBalancer` argument to create load balancer services imperatively.

After you create the service, the load balancer component updates the service resource with information such as the public IP address where the service is available.

```

[user@host ~]$ kubectl get service
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
example-lb  LoadBalancer  172.30.21.79  192.168.50.20  1234:31265/TCP  4m7s

```

You can now connect to the service on port 1234 of the 192.168.50.20 address.

You can also obtain the address from the `status` field of the resource.

```

[user@host ~]$ oc get example-lb -o jsonpath=".status.loadBalancer.ingress"
[{"ip":"192.168.50.20"}]

```

Each load balancer service allocates IP addresses for services by following different processes. For example, when installing MetalLB, you must provide ranges of IPs that MetalLB assigns to services.

After exposing a service by using a load balancer, always verify that the service is available from your intended network locations. Use a client for the exposed protocol to ensure connectivity, and test that load balancing works as expected. Some protocols might require further adjustments to work correctly behind a load balancer. You can also use network debugging tools, such as the `ping` and `traceroute` commands to examine connectivity.



References

For more information, refer to the *Load Balancing with MetalLB* chapter in the Red Hat OpenShift Container Platform 4.14 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#load-balancing-with-metallb

Kubernetes Services

<https://kubernetes.io/docs/concepts/services-networking/service/>

MetalLB on OpenShift

<https://metallb.universe.tf/installation/clouds/#metallb-on-openshift-ocp>

► Guided Exercise

Load Balancer Services

Expose a deployment to external access by using a load balancer service.

Outcomes

- Use load balancer services to expose the video streams that the application produces.
- Access the video streams with a media player.
- Realize that external factors can cause a load balancer to fail.

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 start non-http-lb
```

Instructions

- 1. Log in as the `developer` user, and list the YAML resource manifests for the video streaming application in the `~/D0280/labs/non-http-lb` directory.

- 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 `non-http-lb` project.

```
[student@workstation ~]$ oc new-project non-http-lb
Now using project "non-http-lb" on server ...
...output omitted...
```

- 1.3. Change to the `~/D0280/labs/non-http-lb` directory.

```
[student@workstation ~]$ cd ~/D0280/labs/non-http-lb
[student@workstation non-http-lb]$
```

- 1.4. List the contents of the directory. The YAML resource manifests represent three instances of the video streaming application.

```
[student@workstation non-http-lb]$ ls -l
total 12
-rw-rw-r--. 1 student student 1561 Jun 21 16:29 virtual-rtsp-1.yaml
-rw-rw-r--. 1 student student 1563 Jun 21 16:29 virtual-rtsp-2.yaml
-rw-rw-r--. 1 student student 1565 Jun 21 16:21 virtual-rtsp-3.yaml
```

15. Each deployment emulates the video stream from a security camera on port 8554.

Deployment	Video stream	Location	Image
virtual-rtsp-1	Camera 1	Downtown	 Camera 1 - Downtown
virtual-rtsp-2	Camera 2	Roundabout	 Camera 2 - Roundabout
virtual-rtsp-3	Camera 3	Intersection	 Camera 3 - Intersection

- 2. Deploy the first instance of the application, and expose the video stream from the **downtown** camera by using a load balancer service.

- 2.1. Create the first instance of the video stream deployment. This application produces the video stream of the **downtown** camera.

```
[student@workstation non-http-lb]$ oc apply -f virtual-rtsp-1.yaml
deployment.apps/virtual-rtsp-1 created
```

- 2.2. Wait until the pod is running and the deployment is ready. Press **Ctrl+C** to exit the **watch** command.

```
[student@workstation non-http-lb]$ watch oc get deployments,pods
Every 2.0s: oc get deployments,pods          workstation: Wed Jun 21 16:25:26 2023

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/virtual-rtsp-1   1/1      1           1          59s

NAME                      READY   STATUS     RESTARTS   AGE
pod/virtual-rtsp-1-98cd84d79a-qhn9r   1/1      Running    0          59s
```

2.3. Create a load balancer service to expose the first deployment.

```
[student@workstation non-http-lb]$ oc expose deployment/virtual-rtsp-1 \
--type=LoadBalancer --target-port=8554
service/virtual-rtsp-1 exposed
```

2.4. Get the external IP address of the load balancer service.

```
[student@workstation non-http-lb]$ oc get services
NAME            TYPE      CLUSTER-IP   EXTERNAL-IP     PORT(S)        AGE
virtual-rtsp-1  LoadBalancer  172.30.4.18  192.168.50.20  8554:32170/TCP  59s
```

2.5. Verify that you can connect to the external IP address of the load balancer service on port 8554.

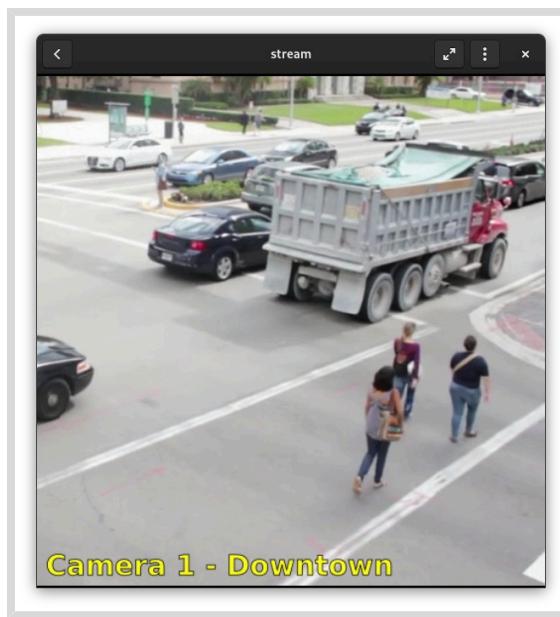
```
[student@workstation non-http-lb]$ nc -vz 192.168.50.20 8554
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Connected to 192.168.50.20:8554.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds
```

2.6. Open the URL in the media player to confirm that the video stream of the **downtown** camera is working correctly.

- `rtsp://192.168.50.20:8554/stream`

```
[student@workstation non-http-lb]$ totem rtsp://192.168.50.20:8554/stream
...output omitted...
```

Close the media player window after confirming that the video stream works correctly.



Chapter 5 | Expose non-HTTP/SNI Applications

- 3. Deploy the remaining instances of the video stream application. Expose the video streams from the **roundabout** and **intersection** cameras by using a load balancer service. Understand that the classroom is configured to provide only two IP addresses.

- 3.1. Create the second instance of the video stream deployment. This application produces the video stream of the **roundabout** camera.

```
[student@workstation non-http-lb]$ oc apply -f virtual-rtsp-2.yaml
deployment.apps/virtual-rtsp-2 created
```

- 3.2. Create the third instance of the video stream deployment. This application produces the video stream of the **intersection** camera.

```
[student@workstation non-http-lb]$ oc apply -f virtual-rtsp-3.yaml
deployment.apps/virtual-rtsp-3 created
```

- 3.3. Wait until the pods are running and the deployments are ready. Press **Ctrl+C** to exit the **watch** command.

```
[student@workstation non-http-lb]$ watch oc get deployments,pods
Every 2.0s: oc get deployments,pods          workstation: Wed Jun 21 16:30:33 2023

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/virtual-rtsp-1   1/1     1           1           5m
deployment.apps/virtual-rtsp-2   1/1     1           1           60s
deployment.apps/virtual-rtsp-3   1/1     1           1           30s

NAME                      READY   STATUS    RESTARTS   AGE
pod/virtual-rtsp-1-98cd84d79a-qhn9r   1/1     Running   0          5m
pod/virtual-rtsp-2-769b5bcb89-r8csp   1/1     Running   0          60s
pod/virtual-rtsp-3-6cdb9f7ffb-g6d9d   1/1     Running   0          30s
```

- 3.4. Create a load balancer service to expose the second deployment.

```
[student@workstation non-http-lb]$ oc expose deployment/virtual-rtsp-2 \
--type=LoadBalancer --target-port=8554
service/virtual-rtsp-2 exposed
```

- 3.5. Create a load balancer service to expose the third deployment.

```
[student@workstation non-http-lb]$ oc expose deployment/virtual-rtsp-3 \
--type=LoadBalancer --target-port=8554
service/virtual-rtsp-3 exposed
```

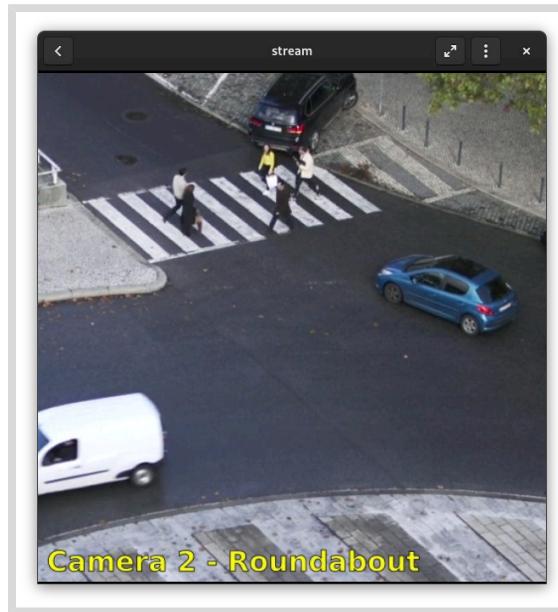
- 3.6. Get the external IP address of the second load balancer service.

```
[student@workstation non-http-lb]$ oc get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      ...
virtual-rtsp-1  LoadBalancer  172.30.94.188  192.168.50.20  8554:32325/TCP  ...
virtual-rtsp-2  LoadBalancer  172.30.15.148  192.168.50.21  8554:31640/TCP  1
virtual-rtsp-3  LoadBalancer  172.30.228.35  <pending>        8554:32089/TCP  2
```

- ➊ The second load balancer service has an associated external IP address.
 - ➋ No IP address is assigned to the third load balancer, and it is displayed as <pending> because all available load balancer IP addresses are in use. The MetalLB operator in the classroom uses the IPAddressPools configuration to restrict the available load balancer IP addresses to 192.168.50.20 and 192.168.50.21.
- 3.7. Open the URL in the media player to confirm that the video stream of the **roundabout** camera is working correctly.
- `rtsp://192.168.50.21:8554/stream`

```
[student@workstation non-http-lb]$ totem rtsp://192.168.50.21:8554/stream  
...output omitted...
```

Close the media player window after confirming that the video stream works correctly.



- ▶ 4. Delete the first service to reallocate the IP address to the third service, and view the video stream of the **intersection** camera.
- 4.1. Delete the first service to release the assigned IP address.

```
[student@workstation non-http-lb]$ oc delete service/virtual-rtsp-1  
service "virtual-rtsp-1" deleted
```

- 4.2. Verify that the third service has an assigned external IP address.

```
[student@workstation non-http-lb]$ oc get services  
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)        ...  
virtual-rtsp-2  LoadBalancer  172.30.15.148  192.168.50.21  8554:31640/TCP  ...  
virtual-rtsp-3  LoadBalancer  172.30.228.35   192.168.50.20  8554:32089/TCP  ❶
```

- ➊ The IP address is now allocated to the third service.

- 4.3. Open the URL in the media player to confirm that the video stream of the **intersection** camera is working correctly.
- `rtsp://192.168.50.20:8554/stream`

```
[student@workstation non-http-lb]$ totem rtsp://192.168.50.20:8554/stream  
...output omitted...
```

Close the media player window after confirming that the video stream works correctly.



▶ 5. Clean up the resources.

- 5.1. Change to the student HOME directory.

```
[student@workstation non-http-lb]$ cd  
[student@workstation ~]$
```

- 5.2. Delete all the services in the namespace.

```
[student@workstation ~]$ oc delete services --all  
service "virtual-rstp-2" deleted  
service "virtual-rstp-3" deleted
```

- 5.3. Delete all the deployments in the namespace.

```
[student@workstation ~]$ oc delete deployments --all  
deployment.apps "virtual-rstp-1" deleted  
deployment.apps "virtual-rstp-2" deleted  
deployment.apps "virtual-rstp-3" deleted
```

5.4. Delete the `non-http-lb` project.

```
[student@workstation ~]$ oc delete project/non-http-lb  
project.project.openshift.io "non-http-lb" deleted
```

Finish

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

```
[student@workstation ~]$ lab finish non-http-lb
```

Multus Secondary Networks

Objectives

- Expose applications to external access by using a secondary network.

Using Different Networks

Kubernetes manages a pod network and a service network. The pod network provides network interfaces to each pod, and by default, provides network communication between all pods. The service network provides stable addressing for services that run on pods. Furthermore, other facilities provide mechanisms to expose services outside the cluster.

However, in some cases, connecting some pods to a different network can provide benefits or help to address requirements.

For example, using a dedicated network with dedicated resources can improve the performance of specific traffic. Additionally, a dedicated network can have different security properties from the default network and help to achieve security requirements.

In addition to these advantages, using extra interfaces can also simplify some tasks, such as controlling outgoing traffic from pods.

The Multus CNI (container network interface) plug-in helps to attach pods to custom networks.

These custom networks can be either existing networks outside the cluster, or custom networks that are internal to the cluster.

Configuring Secondary Networks

To use existing custom networks, first you must make available the network on cluster nodes.

You can use operators, such as the Kubernetes NMState operator or the SR-IOV (Single Root I/O Virtualization) network operator, to customize node network configuration. With these operators, you define custom resources to describe the intended network configuration, and the operator applies the configuration.

The SR-IOV network operator configures SR-IOV network devices for improved bandwidth and latency on certain platforms and devices.

Attaching Secondary Networks

To configure secondary networks, create a `NetworkAttachmentDefinition` resource. Alternatively, update the configuration of the cluster network operator to add a secondary network. Some network attachment definitions create and manage virtual network devices, including virtual bridges. The virtual network devices attach to existing networks that are configured and managed outside OpenShift. Other network attachment definitions use existing network interfaces on the cluster nodes. Network attachment definitions can also perform additional network configuration.

Pod Annotations

Network attachment resources are namespaced, and are available only to pods in their namespace.

When the cluster has additional networks, you can add the `k8s.v1.cni.cncf.io/networks` annotation to the pod's template to use one of the additional networks. The value of the annotation is the name of the network attachment definition to use, or a list of maps with additional configuration options. Besides network attachments, you can also add pods to networks that the SR-IOV network operator configures.

For example, the following deployment uses the `example` network:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example
  namespace: example
spec:
  selector:
    matchLabels:
      app: example
      name: example
  template:
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: example
      labels:
        app: example
        name: example
    spec:
      ...output omitted...
```

Multus updates the `k8s.v1.cni.cncf.io/network-status` annotation with the status of the additional networks.

```
[user@host ~]$ oc get pod example \
-o jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status}'
[{
  "name": "ovn-kubernetes",
  "interface": "eth0",
  "ips": [
    "10.8.0.59"
  ],
  "mac": "0a:58:0a:08:00:3b",
  "default": true,
  "dns": {}
}, {
  "name": "non-http-multus/example", ①
  "interface": "net1", ②
  "ips": [
    "1.2.3.4"
  ],
  "mac": "0a:58:0a:08:00:3c"
}]
```

```
"mac": "52:54:00:01:33:0a",
"dns": {}
}]
```

- ➊ The example pod is attached to the default pod network and to the example custom network.
- ➋ To access the custom network, Multus creates a network interface in the pod. Multus uses the net string followed by a number to name these network interfaces.



Note

The period is the JSONPath field access operator. Normally, you use the period to access parts of the resource, such as in the `.metadata.annotations` JSONPath expression. To access fields that contain periods with JSONPath, you must escape the periods with a backslash (\).

Network Attachment Custom Resource

You can create network attachment definitions of the following types:

Host device

Attaches a network interface to a single pod.

Bridge

Uses an existing bridge interface on the node, or configures a new bridge interface. The pods that are attached to this network can communicate with each other through the bridge, and to any other networks that are attached to the bridge.

IPVLAN

Creates an IPVLAN-based network that is attached to a network interface.

MACVLAN

Creates an MACVLAN-based network that is attached to a network interface.

Bridges are network interfaces that can forward packets between different network interfaces that are attached to the bridge. Virtualization environments often use bridges to connect the network interfaces of virtual machines to the network.

IPVLAN and MACVLAN are Linux network drivers that are designed for container environments. Container environments often use these network drivers to connect pods to the network.

Although bridge interfaces, IPVLAN, and MACVLAN have similar purposes, they have different characteristics, such as different usage of MAC addresses, filtering capabilities, and other features. For example, you might need to use IPVLAN instead of MACVLAN in networks with a limit of MAC addresses, because IPVLAN uses fewer MAC addresses.

The following resource definition shows a `NetworkAttachmentDefinition` resource for a host device.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: example ①
spec:
  config: |-
```

```
{  
    "cniVersion": "0.3.1",  
    "name": "example", 2  
    "type": "host-device", 3  
    "device": "ens4",  
    "ipam": { 4  
        "type": "dhcp"  
    }  
}
```

- 1** The network name
- 2** The same value for the name parameter that you provided previously for this network attachment definition
- 3** The network type
- 4** Additional network configuration

Network Operator Settings

You can also create the same network attachment by editing the cluster network operator configuration:

```
apiVersion: operator.openshift.io/v1  
kind: Network  
metadata:  
  name: cluster  
spec:  
  ...output omitted...  
  additionalNetworks:  
    - name: example 1  
      namespace: example 2  
      rawCNIConfig: |- 3  
        {  
          "cniVersion": "0.3.1",  
          "name": "example", 4  
          "type": "host-device", 5  
          "device": "ens4",  
          "ipam": { 6  
              "type": "dhcp"  
          }  
        }  
    type: Raw
```

- 1** The network name
- 2** The namespace
- 3** The CNI plug-in configuration in JSON format
- 4** The same value for the name parameter that you provided previously for this additional network
- 5** The network type

6 Additional network configuration

The IP Address Management (IPAM) CNI plug-in provides IP addresses for other CNI plug-ins.

In the previous examples, the `ipam` key contains a network configuration that uses DHCP. You can provide more complex network configurations in the `ipam` key. For example, the following configuration uses a static address.

```
"ipam": {  
    "type": "static",  
    "addresses": [  
        {"address": "192.168.x.x/24"}  
    ]  
}
```

Although all the pods in the cluster still use the cluster-wide default network to maintain connectivity across the cluster, you can define more than one additional network for your cluster. The added networks give you flexibility when you configure pods that deliver network functions.

The network isolation that an additional network provides is useful for enhanced performance or for security, depending on your needs.



References

For more information, refer to the *Multiple Networks* chapter in the Red Hat OpenShift Container Platform 4.14 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#multiple-networks

For more information about the SR-IOV network operator, including supported platforms and devices, refer to the *About Single Root I/O Virtualization (SR-IOV) Hardware Networks* section in the *Hardware Networks* chapter in the Red Hat OpenShift Container Platform 4.14 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#about-sriov

For more information, refer to the *About the Kubernetes NMState Operator* section in the *About Networking* chapter in the Red Hat OpenShift Container Platform 4.14 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/networking/index#kubernetes-nmstate

► Guided Exercise

Multus Secondary Networks

Expose a PostgreSQL database to external access by using a secondary network.

Outcomes

- Make a PostgreSQL database accessible outside the cluster on an isolated network by using an existing node network interface.

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.

```
[student@workstation ~]$ lab start non-http-multus
```

Instructions

► 1. Deploy a sample database.

- 1.1. Log in to the OpenShift cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 1.2. Create a `non-http-multus` project.

```
[student@workstation ~]$ oc new-project non-http-multus
...output omitted...
```

- 1.3. Create the resources that the `~/D0280/labs/non-http-multus/deployment.yaml` file contains.

```
[student@workstation ~]$ oc apply -f ~/D0280/labs/non-http-multus/deployment.yaml
secret/database created
persistentvolumeclaim/database created
deployment.apps/database created
```

- 1.4. Wait until all resources are ready.

```
[student@workstation ~]$ oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/database-654db5f958-8p6m5   1/1     Running   0          3m36s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/database   1/1      1           1          3m36s

NAME                      DESIRED   CURRENT   READY   AGE
replicaset.apps/database-654db5f958   1         1         1        3m36s
```

This application contains only a deployment, a persistent volume claim, and a secret. The application does not contain any services, so the database is not accessible outside the pod network.

This application uses a database that requires exclusive access to the database data. On the database deployment, only one pod must be running at a time. To prevent multiple pods from running at a time, the deployment uses the recreate strategy.

This scenario is part of the scenarios where you assign a network interface exclusively to a pod. In these scenarios, the host device strategy is suitable. A network attachment with the host device strategy is suitable only for a single pod.

In other scenarios, you must use more complex network attachments.

► 2. Examine the cluster nodes and inspect the network interface that you use in this exercise.

- 2.1. Log in to the OpenShift cluster as the `admin` user with the `redhatocp` password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 2.2. Use the `oc get node` command to list the cluster nodes.

```
[student@workstation ~]$ oc get node
NAME      STATUS    ROLES          AGE      VERSION
master01   Ready     control-plane,master,worker   36d     v1.27.6+f67aeb3
```

The cluster has a single node with the control plane and worker roles.

- 2.3. Run the `ip addr` command in the node, by using the `oc debug` command to execute commands in the node.

```
[student@workstation ~]$ oc debug node/master01 -- chroot /host ip addr
Temporary namespace openshift-debug-mrchh is created for debugging node...
Starting pod/master01-debug ...
To use host binaries, run: chroot /host
Pod IP: 192.168.50.10
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    qlen 1000
...output omitted...
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master ovs-
    system state UP group default qlen 1000 ❶
```

```

link/ether 52:54:00:00:32:0a brd ff:ff:ff:ff:ff:ff
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
    default qlen 1000 ②
        link/ether 52:54:00:01:33:0a brd ff:ff:ff:ff:ff:ff
        inet 192.168.51.10/24 brd 192.168.51.255 scope global dynamic noprefixroute
    ens4
        valid_lft 461179517sec preferred_lft 461179517sec
        inet6 fe80::b9dd:9436:4fc7:738/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
4: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
    qlen 1000
        link/ether 22:31:45:7a:e2:e3 brd ff:ff:ff:ff:ff:ff
5: ovn-k8s-mp0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state
    UNKNOWN group default qlen 1000
    ...output omitted...
6: br-int: <BROADCAST,MULTICAST> mtu 1400 qdisc noop state DOWN group default qlen
    1000
        link/ether 52:db:28:19:51:94 brd ff:ff:ff:ff:ff:ff
8: br-ex: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        link/ether 52:54:00:00:32:0a brd ff:ff:ff:ff:ff:ff
        inet 192.168.50.10/24 brd 192.168.50.255 scope global dynamic noprefixroute
    br-ex
    ...output omitted...

```

- ➊ The ens3 interface is the main network interface of the cluster.
- ➋ The ens4 interface is an additional network interface for exercises that require an additional network. This interface is attached to a 192.168.51.0/24 network, with the 192.168.51.10 IP address.

The system has other interfaces, including bridges and pod network interfaces.

- 3. Examine the networking configuration of the **workstation** machine. The **workstation** machine has no access to the 192.168.51.0/24 network, which is the **ens4** interface in the cluster node.
- 3.1. Use the **ip addr** command to examine the network interfaces in the **workstation** machine.

```

[student@workstation ~]$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
    default qlen 1000 ①
        link/ether 52:54:00:00:fa:09 brd ff:ff:ff:ff:ff:ff
        inet 172.25.250.9/24 brd 172.25.250.255 scope global noprefixroute eth0
            valid_lft forever preferred_lft forever

```

```
inet6 fe80::5054:ff:fe00:fa09/64 scope link
  valid_lft forever preferred_lft forever
...output omitted...
```

- ➊ The **workstation** machine has a single Ethernet interface. This interface is on a different network from the **ens4** interface in the cluster node.

3.2. Use the **route** command to view the routing table in the **workstation** machine.

```
[student@workstation ~]$ ip route
default via 172.25.250.254 dev eth0 proto static metric 100
10.88.0.0/16 dev podman0 proto kernel scope link src 10.88.0.1
172.25.250.0/24 dev eth0 proto kernel scope link src 172.25.250.9 metric 100
192.168.50.0/24 via 172.25.250.253 dev eth0 proto static metric 100
```

The **workstation** routing table does not have a route to the 192.168.51.0/24 network.

3.3. Use the **ping** command to check connectivity to the **ens4** interface in the cluster.

```
[student@workstation ~]$ ping 192.168.51.10
PING 192.168.51.10 (192.168.51.10) 56(84) bytes of data.
^C
--- 192.168.51.10 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5137ms
```

The command does not produce any output after printing the first line. Wait a few seconds, and then press **Ctrl+C** to interrupt the **ping** command. The **ping** command prints that after transmitting some packets, no response is received.

The **workstation** machine cannot connect to the additional cluster network.



Note

The network diagram in the Orientation to the Classroom Environment [<https://rol.redhat.com/rol/app/courses/do280-4.14/pages/pr01s02>] lecture shows the VMs and the networks that are available to the **workstation** machine.

- ▶ 4. Examine the networking configuration of the **utility** machine. The **utility** machine has access to the 192.168.51.0/24 network.

4.1. Use the **ssh** command to connect to the **utility** machine.

```
[student@workstation ~]$ ssh utility
...output omitted...
[student@utility ~]$
```

4.2. Use the **ip addr** command to examine the network interfaces in the **utility** machine.

```
[student@utility ~]$ ip addr
...output omitted...
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
    default qlen 1000 ①
        link/ether 52:54:00:02:33:fe brd ff:ff:ff:ff:ff:ff
        inet 192.168.51.254/24 brd 192.168.51.255 scope global noprefixroute eth2
    ...output omitted...
```

- ① The `eth2` interface is attached to the `192.168.51.0/24` network, with the `192.168.51.254` IP address.

4.3. Use the `ping` command to check connectivity to the `ens4` interface in the cluster.

```
[student@utility ~]$ ping 192.168.51.10
PING 192.168.51.10 (192.168.51.10) 56(84) bytes of data.
64 bytes from 192.168.51.10: icmp_seq=1 ttl=64 time=0.687 ms
64 bytes from 192.168.51.10: icmp_seq=2 ttl=64 time=0.169 ms
^C
--- 192.168.51.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1058ms
rtt min/avg/max/mdev = 0.169/0.428/0.687/0.259 ms
```

Wait a few seconds, and then press `Ctrl+C` to interrupt the `ping` command. The `ping` command shows that the `utility` machine can connect to the additional cluster network.

4.4. Exit the SSH session to go back to the `workstation` machine.

```
[student@utility ~]$ exit
logout
Connection to utility closed.
[student@workstation ~]$
```

- 5. Configure a network attachment definition for the `ens4` interface, so that the custom network can be attached to a pod.
- 5.1. Edit the `~/DO280/labs/non-http-multus/network-attachment-definition.yaml` file. Use the `custom` name, the `host-device` type, and the `ens4` device. Configure IP address management to use the `static` type, with the `192.168.51.0/24` address.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: custom
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "custom",
      "type": "host-device",
      "device": "ens4",
      "ipam": {
```

```

    "type": "static",
    "addresses": [
        {"address": "192.168.51.10/24"}
    ]
}
}

```

- 5.2. Use the `diff` command to compare your network attachment definition with the solution in the `~/D0280/solutions/non-http-multus/network-attachment-definition.yaml` file. If the files are identical, then the `diff` command does not return any output.

```
[student@workstation ~]$ diff \
~/D0280/labs/non-http-multus/network-attachment-definition.yaml \
~/D0280/solutions/non-http-multus/network-attachment-definition.yaml
```

- 5.3. Use the `oc create` command to create the network attachment definition.

```
[student@workstation ~]$ oc create \
-f ~/D0280/labs/non-http-multus/network-attachment-definition.yaml
networkattachmentdefinition.k8s.cni.cncf.io/custom created
```

- ▶ 6. Edit the deployment to add the `k8s.v1.cni.cncf.io/networks` annotation with the `custom` value.

- 6.1. Log in to the OpenShift cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 6.2. Edit the `~/D0280/labs/non-http-multus/deployment.yaml` file to add the `k8s.v1.cni.cncf.io/networks` annotation with the `custom` value after line 37.

```

apiVersion: v1
...output omitted...
- apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: database
  spec:
    replicas: 1
    strategy:
      type: Recreate
    selector:
      matchLabels:
        name: database
        app: database
    template:
```

```
metadata:  
  labels:  
    name: database  
    app: database  
  annotations:  
    k8s.v1.cni.cncf.io/networks: custom  
  spec:  
...output omitted...
```

6.3. Use the `oc apply` command to add the annotation.

```
[student@workstation ~]$ oc apply -f ~/DO280/labs/non-http-multus/deployment.yaml  
secret/database configured  
persistentvolumeclaim/database unchanged  
deployment.apps/database configured
```

6.4. Wait until all resources are ready.

```
[student@workstation ~]$ oc get all  
NAME                      READY   STATUS    RESTARTS   AGE  
pod/database-74d79685f7-8p6m5  1/1     Running   0          3m36s  
  
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE  
deployment.apps/database  1/1     1           1          3m36s  
  
NAME                      DESIRED  CURRENT  READY   AGE  
replicaset.apps/database-654db5f958  0        0        0      15m  
replicaset.apps/database-74d79685f7    1        1        1      3m36s
```

6.5. Examine the `k8s.v1.cni.cncf.io/network-status` annotation in the pod.

```
[student@workstation ~]$ oc get pod database-74d79685f7-6schp \  
-o jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status}'  
[  
  {  
    "name": "ovn-kubernetes",  
    "interface": "eth0",  
    "ips": [  
      "10.8.0.92"  
    ],  
    "mac": "0a:58:0a:08:00:5c",  
    "default": true,  
    "dns": {}  
  }, {  
    "name": "non-http-multus/custom",  
    "interface": "net1",  
    "ips": [  
      "192.168.51.10"  
    ],  
    "mac": "52:54:00:01:33:0a",  
    "dns": {}  
  }]
```

**Note**

The period is the JSONPath field access operator. Normally, you use the period to access parts of the resource, such as in the `.metadata.annotations` JSONPath expression. To access fields that contain periods with JSONPath, you must escape the periods with a backslash (\).

- 7. Verify that you can access the database from the `utility` machine.

- 7.1. Use the `ssh` command to connect to the `utility` machine.

```
[student@workstation ~]$ ssh utility
...output omitted...
[student@utility ~]$
```

- 7.2. Run a command to execute a query on the database. Use the IP address on the custom network to connect to the database. Use `password` as the password for the user.

```
[student@utility ~]$ psql -h 192.168.51.10 \
-U user sample -c 'SELECT 1;'
Password for user user: password
?column?
-----
1
(1 row)
```

- 7.3. Exit the SSH session to return to the `workstation` machine.

```
[student@utility ~]$ exit
logout
Connection to utility closed.
[student@workstation ~]$
```

- 8. Verify that you cannot use the same process to access the database from the `workstation` machine, because the `workstation` machine cannot access the custom network.

- 8.1. Run a command to execute a query on the database. Use the IP address on the custom network to connect to the database.

```
[student@workstation ~]$ psql -h 192.168.51.10 \
-U user sample -c 'SELECT 1;'

psql: error: could not connect to server: Connection refused
Is the server running on host "192.168.51.10" and accepting
TCP/IP connections on port 5432?
```

After the image is downloaded, the command pauses for over a minute, because you cannot access the custom network from the `workstation` machine.

The deployment uses the custom network, and you can access the database only through the custom network.

Finish

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

```
[student@workstation ~]$ lab finish non-http-multus
```

▶ Lab

Expose non-HTTP/SNI Applications

Expose applications to external access without using an ingress controller.

Outcomes

- Expose a non-http application to external access by using the LoadBalancer type service.
- Configure a network attachment definition for an isolated network.
- Make an application accessible outside the cluster on an isolated network by using an existing node network interface.

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 configures the MetalLB operator to provide a single IP address, `192.168.50.20`, for the load balancer services.

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

Instructions

- Deploy the `virtual-rtsp` application to a new `non-http-review-rtsp` project as the `developer` user with the `developer` password, and verify that the `virtual-rtsp` pod is running.
The application consists of the `~/D0280/labs/non-http-review/virtual-rtsp.yaml` file.
- Expose the `virtual-rtsp` deployment by using the `LoadBalancer` service.
- Access the `virtual-rtsp` application by using the URL in the media player. Run the `totem rtsp://EXTERNAL-IP:8554/stream` command to play the stream in the media player.
- Deploy the `nginx` deployment to a new `non-http-review-nginx` project as the `developer` user with the `developer` password, and verify that the `nginx` pod is running.
The application consists of the `~/D0280/labs/non-http-review/nginx.yaml` file.



Important

The exercise is using an HTTP application as a stand-in for testing connectivity to an external network.

- Configure a network attachment definition for the `ens4` interface, so that the isolated network can be attached to a pod.

The `master01` node has two Ethernet interfaces. The `ens3` interface is the main network interface of the cluster. The `ens4` interface is an additional network interface for exercises

that require an additional network. The `ens4` interface is attached to a `192.168.51.0/24` network, with the `192.168.51.10` IP address.

You can modify the `~/DO280/labs/non-http-review/network-attachment-definition.yaml` file to configure a network attachment definition by using the following parameters:

Parameter	Value
<code>name</code>	<code>custom</code>
<code>type</code>	<code>host-device</code>
<code>device</code>	<code>ens4</code>
<code>ipam.type</code>	<code>static</code>
<code>ipam.addresses</code>	<code>{"address": "192.168.51.10/24"}</code>

- The `nginx` application does not contain any services, so the application is not accessible outside the pod network.

Assign the `ens4` network interface exclusively to the `nginx` pod, by using the `custom` network attachment definition. Edit the `nginx` deployment to add the `k8s.v1.cni.cncf.io/networks` annotation with the `custom` value as the developer user with the `developer` password.

- Verify that you can access the `nginx` application from the `utility` machine by using the following URL:
`http://isolated-network-IP-address:8080`
- Verify that you cannot access the `nginx` application from the `workstation` machine, because the `workstation` machine cannot access the isolated network.

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 grade non-http-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Expose non-HTTP/SNI Applications

Expose applications to external access without using an ingress controller.

Outcomes

- Expose a non-http application to external access by using the LoadBalancer type service.
- Configure a network attachment definition for an isolated network.
- Make an application accessible outside the cluster on an isolated network by using an existing node network interface.

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 configures the MetalLB operator to provide a single IP address, `192.168.50.20`, for the load balancer services.

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

Instructions

- Deploy the `virtual-rtsp` application to a new `non-http-review-rtsp` project as the `developer` user with the `developer` password, and verify that the `virtual-rtsp` pod is running.

The application consists of the `~/D0280/labs/non-http-review/virtual-rtsp.yaml` file.

- Log in to your OpenShift cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- Change to the `~/D0280/labs/non-http-review` directory.

```
[student@workstation ~]$ cd ~/D0280/labs/non-http-review
```

- Create a `non-http-review-rtsp` project.

```
[student@workstation non-http-review]$ oc new-project non-http-review-rtsp
Now using project "non-http-review-rtsp" on server ...
...output omitted...
```

- 1.4. Use the `oc create` command to create the `virtual-rtsp` deployment by using the `virtual-rtsp.yaml` file.

```
[student@workstation non-http-review]$ oc create -f virtual-rtsp.yaml
deployment.apps/virtual-rtsp created
```

- 1.5. List the deployments and pods. Wait for the `virtual-rtsp` pod to be ready. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation non-http-review]$ watch oc get deployments,pods
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/virtual-rtsp   1/1     1           1          21s

NAME                  READY   STATUS    RESTARTS   AGE
pod/virtual-rtsp-54d8d6b57d-6jsvm   1/1     Running   0          21s
```

2. Expose the `virtual-rtsp` deployment by using the `LoadBalancer` service.

- 2.1. Create a load balancer service for the `virtual-rtsp` deployment.

```
[student@workstation non-http-review]$ oc expose deployment/virtual-rtsp \
--name=virtual-rtsp-loadbalancer --type=LoadBalancer
service/virtual-rtsp-loadbalancer exposed
```

- 2.2. Retrieve the external IP address of the `virtual-rtsp-loadbalancer` service.

```
[student@workstation non-http-review]$ oc get svc/virtual-rtsp-loadbalancer
NAME            TYPE        ...   EXTERNAL-IP      PORT(S)
virtual-rtsp-loadbalancer   LoadBalancer  ...   192.168.50.20   8554:32570/TCP
```

The `virtual-rtsp-loadbalancer` has the `192.168.50.20` external IP address.

3. Access the `virtual-rtsp` application by using the URL in the media player. Run the `totem rtsp://EXTERNAL-IP:8554/stream` command to play the stream in the media player.

- 3.1. Open the URL in the media player to confirm that the video stream is working correctly.
`rtsp://192.168.50.20:8554/stream`

```
[student@workstation non-http-review]$ totem rtsp://192.168.50.20:8554/stream
...output omitted...
```



Close the media player window after confirming that the video stream works correctly.

- Deploy the `nginx` deployment to a new `non-http-review-nginx` project as the `developer` user with the `developer` password, and verify that the `nginx` pod is running. The application consists of the `~/DO280/labs/non-http-review/nginx.yaml` file.



Important

The exercise is using an HTTP application as a stand-in for testing connectivity to an external network.

- Create a `non-http-review-nginx` project.

```
[student@workstation non-http-review]$ oc new-project non-http-review-nginx
Now using project "non-http-review-nginx" on server ...
...output omitted...
```

- Use the `oc apply` command to create the `nginx` deployment by using the `nginx.yaml` file.

```
[student@workstation non-http-review]$ oc apply -f nginx.yaml
deployment.apps/nginx created
```

- List the deployments and pods. Wait for the `nginx` pod to be ready. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation non-http-review]$ watch oc get deployments,pods
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx   1/1     1           1          53s

NAME          READY   STATUS    RESTARTS   AGE
pod/nginx-649779cbd-d6sbv   1/1     Running   0          53s
```

- Configure a network attachment definition for the ens4 interface, so that the isolated network can be attached to a pod.

The master01 node has two Ethernet interfaces. The ens3 interface is the main network interface of the cluster. The ens4 interface is an additional network interface for exercises that require an additional network. The ens4 interface is attached to a 192.168.51.0/24 network, with the 192.168.51.10 IP address.

You can modify the ~/D0280/labs/non-http-review/network-attachment-definition.yaml file to configure a network attachment definition by using the following parameters:

Parameter	Value
name	custom
type	host-device
device	ens4
ipam.type	static
ipam.addresses	{"address": "192.168.51.10/24"}

- Log in to your OpenShift cluster as the admin user with the redhatocp password.

```
[student@workstation non-http-review]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- Edit the ~/D0280/labs/non-http-review/network-attachment-definition.yaml file. Use the custom name, the host-device type, and the ens4 device. Configure IP address management to use the static type, with the 192.168.51.10/24 address.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: custom
spec:
  config: |- 
    {
      "cniVersion": "0.3.1",
      "name": "custom",
      "type": "host-device",
      "device": "ens4",
      "ipam": {
        "type": "static",
        "addresses": [
          {"address": "192.168.51.10/24"}
        ]
      }
    }
```

- 5.3. Use the `oc create` command to create the network attachment definition.

```
[student@workstation non-http-review]$ oc create -f \
  network-attachment-definition.yaml
networkattachmentdefinition.k8s.cni.cncf.io/custom created
```

6. The `nginx` application does not contain any services, so the application is not accessible outside the pod network.

Assign the `ens4` network interface exclusively to the `nginx` pod, by using the `custom` network attachment definition. Edit the `nginx` deployment to add the `k8s.v1.cni.cncf.io/networks` annotation with the `custom` value as the developer user with the developer password.

- 6.1. Log in to the OpenShift cluster as the developer user with the developer password.

```
[student@workstation non-http-review]$ oc login -u developer -p developer \
  https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 6.2. Edit the `~/D0280/labs/non-http-review/nginx.yaml` file to add the `k8s.v1.cni.cncf.io/networks` annotation with the `custom` value after line 18.

```
...output omitted...
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nginx
    annotations:
      k8s.v1.cni.cncf.io/networks: custom
  spec:
    containers:
...output omitted...
```

- 6.3. Use the `oc apply` command to add the annotation.

```
[student@workstation non-http-review]$ oc apply -f nginx.yaml
deployment.apps/nginx configured
```

- 6.4. Wait for the `nginx` pod to be ready. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation non-http-review]$ watch oc get deployments,pods
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx  1/1     1            1           34m

NAME                READY   STATUS    RESTARTS   AGE
pod/nginx-6f45d9f89-wp2gg  1/1     Running   0          53s
```

- 6.5. Examine the `k8s.v1.cni.cncf.io/network-status` annotation in the pod.

```
[student@workstation ~]$ oc get pod nginx-6f45d9f89-wp2gg \
-o jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status}'
[{
  "name": "ovn-kubernetes",
  "interface": "eth0",
  "ips": [
    "10.8.0.82"
  ],
  "mac": "0a:58:0a:08:00:52",
  "default": true,
  "dns": {}
}, {
  "name": "non-http-review-nginx/custom",
  "interface": "net1",
  "ips": [
    "192.168.51.10"
  ],
  "mac": "52:54:00:01:33:0a",
  "dns": {}
}]
```



Note

The period is the JSONPath field access operator. Normally, you use the period to access parts of the resource, such as in the `.metadata.annotations` JSONPath expression. To access fields that contain periods with JSONPath, you must escape the periods with a backslash (\).

7. Verify that you can access the `nginx` application from the `utility` machine by using the following URL:

`http://isolated-network-IP-address:8080`

- 7.1. Use the `ssh` command to connect to the `utility` machine.

```
[student@workstation non-http-review]$ ssh utility
...output omitted...
[student@utility ~]$
```

- 7.2. Verify that the `nginx` application is accessible. Use the IP address on the isolated network to access the `nginx` application.

```
[student@utility ~]$ curl 'http://192.168.51.10:8080/'  
<html>  
  <body>  
    <h1>Hello, world from nginx!</h1>  
  </body>  
</html>
```

7.3. Exit the SSH session to go back to the `workstation` machine.

```
[student@utility ~]$ exit  
logout  
Connection to utility closed.  
[student@workstation non-http-review]$
```

8. Verify that you cannot access the `nginx` application from the `workstation` machine, because the `workstation` machine cannot access the isolated network.

8.1. Verify that the `nginx` application is not accessible from the `workstation` machine.

```
[student@workstation non-http-review]$ curl 'http://192.168.51.10:8080/'  
curl: (7) Failed to connect to 192.168.51.10 port 8080: Connection timed out
```

8.2. Change to the student HOME directory.

```
[student@workstation non-http-review]$ cd  
[student@workstation ~]$
```

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 grade non-http-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- Kubernetes ingresses and OpenShift routes use the virtual hosting property of the HTTP protocol to expose web services that are running on the cluster.
- Different providers can implement Kubernetes services, by using the type field of the service resource.
- When a load balancer component is configured for a cluster, you can create services of the `LoadBalancer` type to expose non-HTTP services outside the cluster.
- The Multus CNI (container network interface) plug-in helps to attach pods to custom networks.
- You can configure the additional network by using a network attachment definition resource.

Chapter 6

Enable Developer Self-Service

Goal

Configure clusters for safe self-service by developers from multiple teams, and disallow self-service if operations staff must provision projects.

Objectives

- Configure compute resource quotas and Kubernetes resource count quotas per project and cluster-wide.
- Configure default and maximum compute resource requirements for pods per project.
- Configure default quotas, limit ranges, role bindings, and other restrictions for new projects, and the allowed users to self-provision new projects.
- Project and Cluster Quotas (and Guided Exercise)
- Per-Project Resource Constraints: Limit Ranges (and Guided Exercise)
- The Project Template and the Self-Provisioner Role (and Guided Exercise)
- Enable Developer Self-Service

Sections

Project and Cluster Quotas

Objectives

- Configure compute resource quotas and Kubernetes resource count quotas per project and cluster-wide.

Limiting Workloads

Kubernetes clusters can run heterogeneous workloads across many compute nodes. By using Kubernetes role-based access control (RBAC), cluster administrators can allow users to create workloads on their own. Although RBAC can limit the kinds of resources that users can create, administrators might want further measures to ensure correct operation of the cluster.

Clusters have limited resources, such as CPU, RAM, and storage. If workloads on a cluster exceed the available resources, then workloads might not work correctly. A cluster that is configured to autoscale might also incur unwanted economic costs if the cluster scales to accommodate unexpected workloads.

To help with this issue, Kubernetes workloads can reserve resources and declare resource limits. Workloads can specify the following properties:

Resource limits

Kubernetes can limit the resources that a workload consumes. Workloads can specify an upper bound of the resources that they expect to use under normal operation. If a workload malfunctions or has unexpected load, then resource limits prevent the workload from consuming an excessive amount of resources and impacting other workloads.

Resource requests

Workloads can declare their minimum required resources. Kubernetes tracks requested resources by workloads, and prevents deployments of new workloads if the cluster has insufficient resources. Resource requests ensure that workloads get their needed resources.

These measures prevent workloads from affecting other workloads. However, cluster administrators might need to prevent other risks.

For example, users might mistakenly create unwanted workloads. The resource requests of those unwanted workloads can prevent legitimate workloads from executing.

By dividing workloads into namespaces, Kubernetes can offer enhanced protection features. The namespace structure often mirrors the organization that runs the cluster. Kubernetes introduces resource quotas to limit resource usage by the combined workloads in a namespace.

Resource Quotas

Kubernetes administrators can create resources of the `ResourceQuota` type in a namespace for this purpose. When a resource quota exists in a namespace, Kubernetes prevents the creation of workloads that exceed the quota.

Whereas quota features in other systems often act on users or groups of users, Kubernetes resource quotas act on namespaces.

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: memory
  namespace: example
spec:
  hard: ①
    limits.memory: 4Gi
    requests.memory: 2Gi
  scopes: {} ②
  scopeSelector: {} ③

```

① The `hard` key lists restrictions.

②③ The `scopes` and `scopeSelector` keys define which namespace resources the quota applies to. This course does not cover those keys.

The following sections describe the compute and object count quotas that you can include in the `hard` key. Other components can define other quotas and enforce them.

Compute Resource Quotas

You can set the following compute quotas:

- `limits.cpu`
- `limits.memory`
- `requests.cpu`
- `requests.memory`

Limit quotas interact with resource limits, and request quotas interact with resource requests.

Limit quotas control the maximum compute resources that the workloads in a namespace can consume. Consider a namespace where all workloads have a memory limit. No individual workload can consume enough memory to cause a problem. However, because users can create any number of workloads, the workloads of a namespace can consume enough memory to cause a problem for workloads in other namespaces. If you set a namespace memory usage limit, then the workloads in the namespace cannot consume more memory than this limit.

Request quotas control the maximum resources that workloads in a namespace can reserve. If you do not set namespace request quotas, then a single workload can request any quantity of resources, such as RAM or CPU. This request can cause further requests in other namespaces to fail. By setting namespace request quotas, the total requested resources by workloads in a namespace cannot exceed the quota.

Excessive quotas can cause resource underutilization and can limit workload performance unnecessarily.

After setting any compute quota, all workloads must define the corresponding request or resource limit. For example, if you create a `limits.cpu` quota, then the workloads that you create require the `resources.limits.cpu` key.

Object Count Quotas

A quota can also limit the number of resources of a given type in a namespace. For example, you can create a quota that prevents the creation of more than 10 deployments in a namespace.

Clusters store resource definitions in a backing store. Kubernetes backing stores are databases, and like any other database, the more data that they store, the more resources are needed for adequate performance. Namespaces with many resources can impact Kubernetes performance. Additionally, any process that creates cluster resources might malfunction and create unwanted resources.

Setting object count quotas can limit the damage from accidents, and maintain adequate cluster performance.



Note

Red Hat validates the performance of OpenShift up to a specific number of objects in a set of configurations. If you are planning a large cluster, then these results can help you to size the cluster and to establish object count quotas.

See the references section for more information.

Some Kubernetes resources might affect external systems. For example, creating a persistent volume might create an entity in the storage provider. Many persistent volumes might cause issues in the storage provider. Examine the systems that your cluster interacts with to learn about possible resource constraints, and establish object count quotas to prevent issues.

Use the `count/resource_type` syntax to set a quota for resources of the core group. Use the `oc api-resources` command with an empty `api-group` parameter to list resources of the core group.

```
[user@host ~]$ oc api-resources --api-group="" --namespaced=true
NAME           SHORTNAMES   APIVERSION   NAMESPACED   KIND
bindings       v1           true          Binding
...output omitted...
```

For resources in other groups, use the `count/resource_type.group` syntax.

Kubernetes initially supported quotas for a limited set of resource types. These quotas do not use the `count/resource_type` syntax. You might find a `services` quota instead of a `count/services` quota. The `Resource Quotas` reference further describes these quotas.

Applying Project Quotas

Navigate to Administration > ResourceQuotas to create a resource quota from the web console. The YAML editor loads an example resource quota that you can edit for your needs.

You can also use the `oc` command to create a resource quota. The `oc` command can create resource quotas without requiring a complete resource definition. Execute the `oc create resourcequota --help` command to display examples and help for creating resource quotas without a complete resource definition.

For example, execute the following command to create a resource quota that limits the number of pods in a namespace:

```
[user@host ~]$ oc create resourcequota example --hard=count/pods=1
resourcequota/example created
```

The previous command is equivalent to creating a resource quota with the following definition:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example
spec:
  hard:
    count/pods: "1"
```

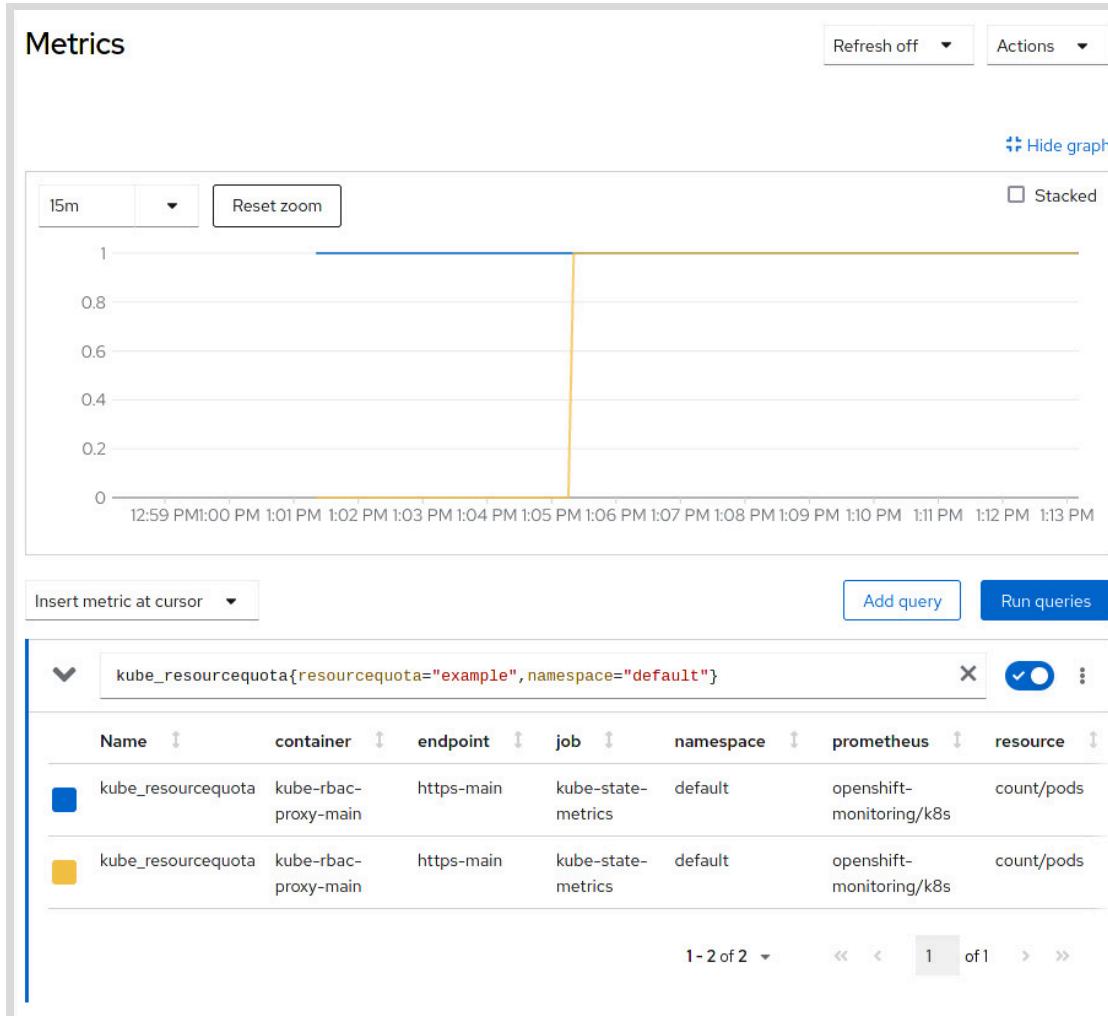
After creating a resource quota, the `status` key in the resource describes the current values and limits in the quota.

```
[user@host ~]$ oc get quota example -o yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2024-01-30T17:59:52Z"
  name: example
  namespace: default
  resourceVersion: "193658"
  uid: df12b484-4e78-4920-acb4-e04ab286a4a1
spec:
  hard:
    count/pods: "1"
status:
  hard:
    count/pods: "1"
  used:
    count/pods: "0"
```

The `oc get` and `oc describe` commands show resource quota information in a custom format. The `oc get` command displays the status of the quota in resource lists:

```
[user@host ~]$ oc get quota
NAME      AGE      REQUEST          LIMIT
example   9m54s   count/pods: 1/1
```

Resource quotas generate the `kube_resourcequota` metric. You can examine this metric for planning and trend analysis.

Figure 6.1: The `kube_resourcequota` metric

Troubleshooting Resource Quotas

Because resource quotas are extensible, Kubernetes cannot verify that a resource quota is correct. For example, the following command creates a resource quota that has no effect:

```
[user@host ~]$ oc create resourcequota example --hard=count/deployment=1
resourcequota/example created
```

The correct syntax for limiting the number of deployments is `count/deployment.apps`.

To ensure that a resource quota is correct, you can use the following procedures:

- Create a quota with an artificially low value in a testing environment, and ensure that the resource quota has an effect.
- Review the quota status.

For example, if a namespace contains a deployment, then an incorrectly defined resource quota shows 0 deployments:

```
[user@host ~]$ oc get resourcequota
NAME      AGE      REQUEST           LIMIT
example   2m47s   count/deployment: 0/1
```

However, a correctly defined resource quota shows the deployment:

```
[user@host ~]$ oc get resourcequota
NAME      AGE      REQUEST           LIMIT
example   4s      count/deployments.apps: 1/1
```

Exceeding a quota often produces an error immediately. For example, if you create a deployment that exceeds the deployment quota, then the deployment creation fails.

```
[user@host ~]$ oc create deployment --image=nginx hello
error: failed to create deployment: deployments.apps "hello" is forbidden:
exceeded quota: example, requested: count/deployments.apps=1, used: count/
deployments.apps=1, limited: count/deployments.apps=1
```

However, some quotas do not cause operations to fail immediately. For example, if you set a resource quota for pods, then creating a deployment appears to succeed, but the deployment never becomes available. When a resource quota is acting indirectly, namespace events might provide further information.

```
[user@host ~]$ oc get event --sort-by .metadata.creationTimestamp
LAST SEEN    TYPE      REASON           OBJECT          MESSAGE
...output omitted...
10s          Normal    ScalingReplicaSet  deployment/hello  Scaled up
replica set hello-5cdfd9c858 to 1
9s           Warning   FailedCreate     replicaset/hello-5cdfd9c858  Error
creating: pods "hello-5cdfd9c858-zsgn9" is forbidden: exceeded quota: example,
requested: count/pods=1, used: count/pods=1, limited: count/pods=1
5s           Warning   FailedCreate     replicaset/hello-5cdfd9c858  (combined
from similar events): Error creating: pods "hello-5cdfd9c858-h2dv4" is forbidden:
exceeded quota: example, requested: count/pods=1, used: count/pods=1, limited:
count/pods=1
```

The web console also shows quota information. Navigate to **Administration > ResourceQuotas** to view resource quotas and their status. The project pages on both the developer and administrator perspectives also show the quotas that apply to a specific project.

Creating Quotas Across Multiple Projects

Cluster administrators can use resource quotas to apply restrictions to namespaces.

Resource restrictions often follow organization structure. Although namespaces often reflect organization structure, cluster administrators might apply restrictions to resources without being limited to a single namespace.

For example, a group of developers manages many namespaces. Namespace quotas can limit RAM usage per namespace. However, a cluster administrator cannot limit total RAM usage by all workloads that the group of developers manages.

OpenShift introduces cluster resource quotas for those scenarios.

Cluster resource quotas follow a similar structure to namespace resource quotas. However, cluster resource quotas use selectors to choose which namespaces the quota applies to.

Cluster resource quotas selectors use set-based requirements.

The following example shows a cluster resource quota:

```
apiVersion: quota.openshift.io/v1
kind: ClusterResourceQuota
metadata:
  name: example
spec:
  quota: ①
    hard:
      limits.cpu: 4
  selector: ②
    annotations: {}
    labels:
      matchLabels:
        kubernetes.io/metadata.name: example
```

- ① The `quota` key contains the quota definition. This key follows the structure of the `ResourceQuota` specification. The `hard` key is nested inside the `quota` key, instead of being directly nested inside the `spec` key as in resource quotas.
- ② The `selector` key defines which namespaces the cluster resource quota applies to. Other Kubernetes features, such as services and network policies, use the same selectors.

Navigate to **Administration > CustomResourceDefinitions** to create a cluster resource quota with the web console.

You can also use the `oc` command to create a cluster quota. The `oc` command can create quotas without requiring a complete resource definition. Execute the `oc create clusterresourcequota --help` command to display examples and help about creating cluster resource quotas without a complete resource definition.

For example, execute the following command to create a resource quota that limits total CPU requests. The quota limits the total CPU requests on namespaces that have the `group` label with the `dev` value.

```
[user@host ~]$ oc create clusterresourcequota example
--project-label-selector=group=dev --hard=requests.cpu=10
clusterresourcequota/example created
```

Cluster resource quotas collect total resource usage across namespaces and enforce the limits. The following example shows the status of the previous cluster resource quota:

```
apiVersion: quota.openshift.io/v1
kind: ClusterResourceQuota
metadata:
  name: example
spec:
  quota:
```

```

hard:
  requests.cpu: "10"
selector:
  annotations: null
  labels:
    matchLabels:
      group: dev
status:
  namespaces: ❶
    - namespace: example-3
      status:
        hard:
          requests.cpu: "10"
        used:
          requests.cpu: 500m
    - namespace: example-2
      status:
        hard:
          requests.cpu: "10"
        used:
          requests.cpu: 250m
  ...output omitted...
  total: ❷
    hard:
      requests.cpu: "10"
    used:
      requests.cpu: 2250m

```

- ❶ The `namespaces` key lists the namespaces that the quota applies to. For each namespace, the `used` key shows the current utilization.
- ❷ The `total` key aggregates the data in the `namespaces` key.

Users might not have read access to cluster resource quotas. OpenShift creates resources of the `AppliedClusterResourceQuota` type in namespaces that are affected by cluster resource quotas. Project administrators can review quota usage by reviewing the `AppliedClusterResourceQuota` resources. For example, use the `oc describe` command to view the cluster resource quotas that apply to a specific namespace:

```
[user@host ~]$ oc describe AppliedClusterResourceQuota -n example-2
Name: example
Created: 9 minutes ago
Labels: <none>
Annotations: <none>
Namespace Selector: ["example-3" "example-2" "example-4" "example-1"]
Label Selector: group=dev
AnnotationSelector: map[]
Resource Used Hard
-----
requests.cpu 2250m 10
```

**Note**

The `--all-namespaces` argument to `oc` commands such as the `get` and `describe` commands does not work with `AppliedClusterResourceQuota` resources. These resources are listed only when you select a namespace.

Navigate to **Administration > ResourceQuotas** to view quotas and their status. This page displays cluster quotas along with namespace quotas. Although you can view resources of the `ClusterResourceQuota` type and create resources of the `ResourceQuota` type in the `ResourceQuotas` page, you cannot create objects of the `ClusterResourceQuota` in this page.

The project pages on both the developer and administrator perspectives also show the cluster quotas that apply to a specific project.

**References**

For more information, refer to the *Quotas* chapter in the Red Hat OpenShift Container Platform 4.14 *Building Applications* documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/building_applications/index#quotas-setting-per-project

For more information about object counts, refer to the *Planning Your Environment According to Object Maximums* chapter in the Red Hat OpenShift Container Platform 4.14 *Scalability and Performance* documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/scalability_and_performance/index#ibm-z-platform

Requests and Limits

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits>

Resource Quotas

<https://kubernetes.io/docs/concepts/policy/resource-quotas/>

► Guided Exercise

Project and Cluster Quotas

Configure quotas for a project so that applications cannot scale to consume all capacity of a cluster node.

Outcomes

- Verify that requesting resources in one namespace can prevent creation of workloads in different namespaces.
- Set a quota to prevent workloads in a namespace from requesting excessive resources.
- Verify that you can continue to create workloads in different namespaces.

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 deletes the namespaces that you use in this exercise.

```
[student@workstation ~]$ lab start selfservice-quotas
```

Instructions

- 1. Log in to your OpenShift cluster as the `developer` user with the `developer` password.

- 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...
```

- 2. Create a `selfservice-quotas` project.

- 2.1. Use the `oc new-project` command to create the project.

```
[student@workstation ~]$ oc new-project selfservice-quotas
Now using project "selfservice-quotas" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

- 3. Create a deployment with a container that requests one CPU.

- 3.1. Use the `oc create` command to create the deployment.

```
[student@workstation ~]$ oc create deployment test \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx
deployment.apps/test created
```

- 3.2. Use the `oc set resources` command to request one CPU in the container specification.

```
[student@workstation ~]$ oc set resources deployment test --requests=cpu=1
deployment.apps/test resource requirements updated
```

- 3.3. Use the `oc get` command to ensure that the deployment starts a pod correctly.

```
[student@workstation ~]$ oc get pod,deployment
NAME                      READY   STATUS    RESTARTS   AGE
pod/test-8b9fdfbd9-bltlc  1/1     Running   0          13s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test  1/1     1           1           49s
```

Execute the command until the deployment and the pod are ready.

► 4. Try to scale the deployment to eight replicas.

- 4.1. Use the `oc scale` command to scale the deployment.

```
[student@workstation ~]$ oc scale deployment test --replicas=8
deployment.apps/test scaled
```

- 4.2. Use the `oc get` command to view pods and deployments.

```
[student@workstation ~]$ oc get pod,deployment
NAME                      READY   STATUS    RESTARTS   AGE
pod/test-6c66b55cb5-2kclt  1/1     Running   0          48m
pod/test-6c66b55cb5-5n58r  0/1     Pending   0          5s
pod/test-6c66b55cb5-8x929  0/1     Pending   0          5s
pod/test-6c66b55cb5-blgms  0/1     Pending   0          5s
pod/test-6c66b55cb5-d6z42  1/1     Running   0          6s
pod/test-6c66b55cb5-fc8bk  0/1     Pending   0          5s
pod/test-6c66b55cb5-t29dh  0/1     Pending   0          6s
pod/test-6c66b55cb5-xqr66  0/1     Pending   0          6s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test  2/8     8           2           54m
```

Out of eight pods that the deployment creates, only some of them change to `Running` status. The other pods stay in `Pending` status. Not all replicas of the deployment are ready and available.

- 4.3. Use the `oc get` command to list events. Sort the events by their creation timestamp.

```
[student@workstation ~]$ oc get event --sort-by .metadata.creationTimestamp
LAST SEEN      TYPE        REASON          OBJECT                MESSAGE
...output omitted...
3m58s       Normal      ScalingReplicaSet deployment/test
replica set test-6c66b55cb5 to 8
3m58s       Normal      Scheduled        pod/test-6c66b55cb5-d6z42
Successfully assigned selfservice-quotas/test-6c66b55cb5-d6z42 to master01
3m57s       Warning     FailedScheduling pod/test-6c66b55cb5-5n58r   0/1 nodes
are available: 1 Insufficient cpu. preemption: 0/1 nodes are available: 1 No
preemption victims found for incoming pod..
...output omitted...
```

Replicas fail to schedule, because the cluster has insufficient CPU.

► 5. Examine the cluster as an administrator.

5.1. Log in to the cluster as the `admin` user with the `redhatocp` password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
Login successful.

...output omitted...
```

5.2. Use the `oc adm top node` command to display the resource usage of nodes.

```
[student@workstation ~]$ oc adm top node
NAME      CPU(cores)  CPU%    MEMORY(bytes)  MEMORY%
master01  772m        14%    10185Mi        68%
```

The cluster does not show high CPU usage.

5.3. Use the `oc describe` command to view the node details.

```
[student@workstation ~]$ oc describe node/master01
Name:                     master01
...output omitted...
Capacity:
  cpu:                   6
...output omitted...
Allocatable:
  cpu:                   5500m
...output omitted...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
Resource      Requests      Limits
-----
cpu           4627m (84%)  0 (0%)
memory        12102Mi (81%) 0 (0%)
ephemeral-storage 0 (0%)    0 (0%)
hugepages-1Gi 0 (0%)      0 (0%)
hugepages-2Mi 0 (0%)      0 (0%)
...output omitted...
```

Chapter 6 | Enable Developer Self-Service

The node has a capacity of six CPUs, and has more than five allocatable CPUs. However, over five CPUs are requested, so less than one CPU is available for new workloads.

- 6. Create a `test` project as an administrator, and verify that you cannot create new workloads that request a CPU.

- 6.1. Use the `oc new-project` command to create the project.

```
[student@workstation ~]$ oc new-project test
Now using project "test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 6.2. Use the `oc create` command to create the deployment.

```
[student@workstation ~]$ oc create deployment test \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx
deployment.apps/test created
```

- 6.3. Use the `oc set resources` command to request one CPU in the container specification.

```
[student@workstation ~]$ oc set resources deployment test --requests=cpu=1
deployment.apps/test resource requirements updated
```

- 6.4. Use the `oc get` command to review the pods and deployments in the `test` namespace.

```
[student@workstation ~]$ oc get pod,deployment
NAME                  READY   STATUS    RESTARTS   AGE
pod/test-8b9fdfbd9-rrn7t   0/1     Pending   0          8s
pod/test-c454765f-vkt96   1/1     Running   0          100s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test   1/1      1           1          100s
```

The deployment created one pod before adding the CPU request. When you updated the deployment to request a CPU, the deployment tried to replace the pod to add the CPU request. The new pod is in the `Pending` state, because the cluster has less than one CPU available to request.

The workload in the `selfservice-quotas` namespace prevents the creation of workloads in other namespaces.

- 6.5. Use the `oc delete` command to delete the `test` namespace.

```
[student@workstation ~]$ oc delete namespace test
namespace "test" deleted
```

- 7. As an administrator, scale the deployment to one replica.

- 7.1. Use the `oc project` command to switch to the `selfservice-quotas` project.

```
[student@workstation ~]$ oc project selfservice-quotas
Now using project "selfservice-quotas" on server "https://
api.ocp4.example.com:6443".
```

- 7.2. Use the `oc scale` command to scale the `test` deployment to one replica.

```
[student@workstation ~]$ oc scale deployment test --replicas=1
deployment.apps/test scaled
```

- 8. Create a quota to prevent workloads in the `selfservice-quotas` namespace from requesting more than one CPU.

- 8.1. Use the `oc create` command to create the quota.

```
[student@workstation ~]$ oc create quota one-cpu --hard=requests.cpu=1
resourcequota/one-cpu created
```

- 8.2. Use the `oc get` command to verify the quota.

```
[student@workstation ~]$ oc get quota one-cpu -o yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2024-01-30T18:26:49Z"
  name: one-cpu
  namespace: selfservice-quotas
  ...output omitted...
spec:
  hard:
    requests.cpu: "1"
status:
  hard:
    requests.cpu: "1"
  used:
    requests.cpu: "1"
```

The `test` deployment already requests one CPU.

- 9. Try to scale the deployment to eight replicas and to create a second deployment.

- 9.1. Use the `oc scale` command to scale the deployment.

```
[student@workstation ~]$ oc scale deployment test --replicas=8
deployment.apps/test scaled
```

- 9.2. Use the `oc create` command to create a second deployment.

```
[student@workstation ~]$ oc create deployment test2 \
  --image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx
deployment.apps/test2 created
```

- 9.3. Use the `oc get` command to review pods and deployments.

```
[student@workstation ~]$ oc get pod,deployment
NAME                 READY   STATUS    RESTARTS   AGE
pod/test-6c66b55cb5-mdxjl  1/1     Running   0          2m58s

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test  1/8      1           1           3m20s
deployment.apps/test2  0/1      0           0           14s
```

The test deployment creates only two pods. The second deployment does not create any pods.

- 9.4. Use the `oc get` command to examine the quota status.

```
[student@workstation ~]$ oc get quota one-cpu -o yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: one-cpu
  namespace: selfservice-quotas
...output omitted...
spec:
  hard:
    requests.cpu: "1"
status:
  hard:
    requests.cpu: "1"
  used:
    requests.cpu: "1"
```

The used status is kept at 1 because the `test2` deployment can't request more resources in the quota.

- 9.5. Use the `oc get` command to list events. Sort the events by their creation timestamp.

```
[student@workstation ~]$ oc get event --sort-by .metadata.creationTimestamp
LAST SEEN   TYPE      REASON          OBJECT                MESSAGE
...output omitted...
4m42s       Warning   FailedCreate    replicaset/test-6c66b55cb5
(combined from similar events): Error creating: pods "test-6c66b55cb5-djrr9" is forbidden: exceeded quota: one-cpu, requested: requests.cpu=1, used: requests.cpu=2, limited: requests.cpu=2
9m3s        Warning   FailedCreate    replicaset/test2-7b9df44445  Error
creating: pods "test2-7b9df44445-98wxp" is forbidden: failed quota: one-cpu: must
specify requests.cpu for: hello-world-nginx
...output omitted...
```

The `test` deployment cannot create further pods, because the new pods would exceed the quota. The `test2` deployment cannot create pods, because the deployment does not set a CPU request.

- ▶ 10. Create a `test` project to verify that you can create new workloads in other namespaces that request CPU resources.

- 10.1. Use the `oc new-project` command to create the project.

```
[student@workstation ~]$ oc new-project test
Now using project "test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 10.2. Use the `oc create` command to create the deployment.

```
[student@workstation ~]$ oc create deployment test --image \
  registry.ocp4.example.com:8443/redhattraining/hello-world-nginx
deployment.apps/test created
```

- 10.3. Use the `oc set resources` command to request one CPU in the container specification.

```
[student@workstation ~]$ oc set resources deployment test --requests=cpu=1
deployment.apps/test resource requirements updated
```

- 10.4. Use the `oc get` command to review the pods and deployments in the `test` namespace.

```
[student@workstation ~]$ oc get pod,deployment
NAME                      READY   STATUS    RESTARTS   AGE
pod/test-8b9fdfbd9-447w9   1/1     Running   0          21s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/test   1/1     1           1           51s
```

Even though you cannot create further workloads in the `selfservice-quotas` namespace, you can create workloads that request CPUs in other namespaces when the node has CPUs and memory available.

Finish

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

```
[student@workstation ~]$ lab finish selfservice-quotas
```

Per-Project Resource Constraints: Limit Ranges

Objectives

- Configure default and maximum compute resource requirements for pods per project.

Managing Namespace Resources

Cluster administrators can set resource quotas on namespaces. Namespace quotas limit the resources that workloads in a namespace use. Quotas address resource management at the cluster level.

Kubernetes users might have further resource management needs within a namespace.

- Users might accidentally create workloads that consume too much of the namespace quota. These unwanted workloads might prevent other workloads from running.
- Users might forget to set workload limits and requests, or might find it time-consuming to configure limits and requests. When a namespace has a quota, creating workloads fails if the workload does not define values for the limits or requests in the quota.

Kubernetes introduces limit ranges to help with these issues. Limit ranges are namespaced objects that define limits for workloads within the namespace.

Limit Ranges

The following YAML file shows an example limit range:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
  namespace: default
spec:
  limits:
    - default:
        memory: 512Mi
      defaultRequest:
        memory: 256Mi
      type: Container
```

Limit ranges can specify the following limit types:

Default limit

Use the `default` key to specify default limits for workloads.

Default request

Use the `defaultRequest` key to specify default requests for workloads.

Maximum

Use the `max` key to specify the maximum value of both requests and limits.

Minimum

Use the `min` key to specify the minimum value of both requests and limits.

Limit-to-request ratio

The `maxLimitRequestRatio` key controls the relationship between limits and requests. If you set a ratio of two, then the resource limit cannot be more than twice the request.

This course does not cover limit-to-request ratios in detail.

Limit ranges can apply to containers, pods, images, image streams, and persistent volume claims.

Setting Maximum and Minimum Limit Ranges

When you set the `max` key, users cannot create workloads that declare limits or that make resource requests over the maximum.

Use maximums to prevent accidentally high resource requests and limits. These situations can exhaust quotas and cause other issues.

Consider allowing users who create workloads to edit maximum limit ranges. Although maximum limit ranges act as a convenient safeguard, excessively low limits can prevent users from creating legitimate workloads.

Minimum limit ranges are useful to ensure that users create workloads with enough requests and limits. If users create such workloads often, then consider adding minimums.

Setting Defaults

Defaults are convenient in namespaces with quotas, and eliminate a need to declare limits explicitly in each workload. When a quota is present, all workloads must specify the corresponding limits and requests. When you set the `default` and `defaultRequest` keys, workloads use the requests and limits from the limit range by default.

Defaults are especially convenient in scenarios where many workloads are created dynamically. For example, continuous integration tools might run tests for each change to a source code repository. Each test can create multiple workloads. Because many tests can run concurrently, the resource usage of testing workloads can be significant. Setting quotas for testing workloads is often needed to limit resource usage. If you set CPU and RAM quotas for requests and limits, then the continuous integration tool must set the corresponding limits in every testing workload. Setting defaults can save time with configuring limits. However, determining appropriate defaults might be complex for namespaces with varied workloads.

Creating Limit Ranges

Consider a namespace with the following quota:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example
  namespace: example
spec:
  hard:
    limits.cpu: "8"
```

```
limits.memory: 8Gi
requests.cpu: "4"
requests.memory: 4Gi
```

The following command creates a deployment:

```
[user@host ~]$ oc create deployment example --image=image
deployment.apps/example created
```

The quota prevents the deployment from creating pods:

```
[user@host ~]$ oc get event --sort-by .metadata.creationTimestamp
LAST SEEN      TYPE      REASON          OBJECT          MESSAGE
...output omitted...
13s           Warning   FailedCreate    replicaset/example-74c57c8dff   Error
  creating: pods "example-74c57c8dff-rzl7w" is forbidden: failed quota: example:
    must specify limits.cpu for: hello-world-nginx; limits.memory for: hello-world-
    nginx; requests.cpu for: hello-world-nginx; requests.memory for: hello-world-nginx
...output omitted...
```

The following limit range includes all types of limits:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: example
  namespace: example
spec:
  limits:
    - default:
        cpu: 500m
        memory: 512Mi
      defaultRequest:
        cpu: 250m
        memory: 256Mi
    max:
      cpu: "1"
      memory: 1Gi
    min:
      cpu: 125m
      memory: 128Mi
  type: Container
```

Limit ranges do not affect existing pods. If you delete the deployment and run the `oc create` command again, then the deployment creates a pod with the applied limit range.

```
[user@host ~]$ oc describe pod
...output omitted...
Containers:
  hello-world-nginx:
    Limits:
      cpu:      500m
```

```
memory: 512Mi
Requests:
  cpu:        250m
  memory:    256Mi
...output omitted...
```

The values correspond to the `default` and `defaultRequest` keys in the limit range.

The deployment does not contain any limits in the specification. The Kubernetes API server includes an admission controller that enforces limit ranges. The controller affects pod definitions, but not deployments, stateful sets, or other workloads.

You can replace the CPU limit, or add other resource specifications, by using the `oc set resources` command:

```
[user@host ~]$ oc set resources deployment example --limits(cpu=new-cpu-limit)
```

You can experiment with different CPU limits.

If you request CPU values outside the range that the `min` and `max` keys define, then Kubernetes does not create the pods, and it logs warnings.

```
[user@host ~]$ oc get event --sort-by .metadata.creationTimestamp
LAST SEEN      TYPE      REASON          OBJECT          MESSAGE
...output omitted...
5m43s          Warning   FailedCreate    replicaset/example-7c4dfc5fb8   Error
  creating: pods "example-7c4dfc5fb8-q7x94" is forbidden: maximum cpu usage per
  Container is 1, but limit is 1200m
...output omitted...
5m26s          Warning   FailedCreate    replicaset/example-798d65c854   Error
  creating: pods "example-798d65c854-b94k8" is forbidden: minimum cpu usage per
  Container is 125m, but request is 100m
...output omitted...
```



Note

When you experiment with deployments and resource quotas, consider what happens when you modify a deployment. Modifications create a replacement replica set, and the existing replica set also continues to run until the rollout completes.

The pods of both replica sets count towards the resource quota.

If the new replica set satisfies the quota, but the combined replica sets exceed the quota, then the rollout cannot complete.

When creating a limit range, you can specify any combination of the `default`, `defaultRequest`, `min`, and `max` keys. However, if you do not specify the `default` or `defaultRequest` keys, then Kubernetes modifies the limit range to add these keys. These keys are copied from the `min` or `max` keys. For more predictable behavior, always specify the `default` and `defaultRequest` keys if you specify the `min` or `max` keys.

Also, the values for CPU or memory keys must follow these rules:

- The `max` value must be higher than or equal to the `default` value.

- The `default` value must be higher than or equal to the `defaultRequest` value.
- The `defaultRequest` value must be higher than or equal to the `min` value.

Do not create conflicting limit ranges in a namespace. For example, if two default CPU values are specified, then it would be unclear which one is applied.



References

For more information, refer to the *Restrict Resource Consumption with Limit Ranges* section in the *Working with Clusters* chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/nodes/index#nodes-cluster-limit-ranges

Limit Ranges

<https://kubernetes.io/docs/concepts/policy/limit-range/>

► Guided Exercise

Per-Project Resource Constraints: Limit Ranges

Configure a project with default compute resource limits so pods do not run unconstrained.

Outcomes

- Verify that workloads have no limits by default.
- Create a limit range.
- Create a workload and inspect the limits that the limit range adds to the containers.

Before You Begin

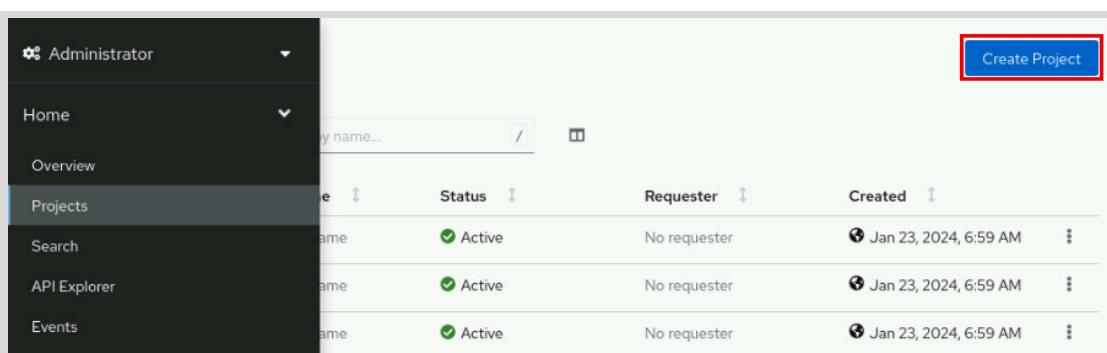
As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and deletes the namespace that you use in this exercise.

```
[student@workstation ~]$ lab start selfservice-ranges
```

Instructions

- ▶ 1. As the `admin` user, navigate and log in to the OpenShift web console.
 - 1.1. Open a web browser and navigate to `https://console-openshift-console.apps.ocp4.example.com`.
 - 1.2. Click **Red Hat Identity Management** and log in as the `admin` user with the `redhatocp` password.
- ▶ 2. Create a `selfservice-ranges` project.
 - 2.1. Navigate to **Home > Projects**, and then click **Create Project**.



The screenshot shows the OpenShift web console interface. On the left, there is a sidebar with a dark theme. The 'Administrator' dropdown is open, showing 'Home' and 'Projects'. The 'Projects' option is selected and highlighted in blue. Below the sidebar, there is a search bar with the placeholder 'Search by name...'. To the right of the search bar is a 'Create Project' button, which is also highlighted with a red box. The main area displays a table of existing projects. The columns are labeled 'Name', 'Status', 'Requester', and 'Created'. There are three entries in the table, all of which are 'Active' and have 'No requester'. The 'Created' column shows dates from January 23, 2024, at 6:59 AM.

Name	Status	Requester	Created
selfservice-ranges	Active	No requester	Jan 23, 2024, 6:59 AM
selfservice-ranges	Active	No requester	Jan 23, 2024, 6:59 AM
selfservice-ranges	Active	No requester	Jan 23, 2024, 6:59 AM

- 2.2. Type `selfservice-ranges` in the **Name** field, and then click **Create**.

Create Project

An OpenShift project is an alternative representation of a Kubernetes namespace.

[Learn more about working with projects](#)

Name * (?)

Display name

Description

[Cancel](#) [Create](#)

► 3. Create an example deployment.

3.1. Navigate to **Workloads > Deployments**, and then click **Create Deployment**.

The screenshot shows the OpenShift navigation sidebar on the left with options like Home, Operators, Workloads (selected), Pods, and Deployments. The main content area displays a message 'No Deployments found'. In the bottom right corner of the main area, there is a blue 'Create Deployment' button, which is highlighted with a red rectangular box.

3.2. Ensure that **Form view** is selected, and then type **example** in the **Name** field.

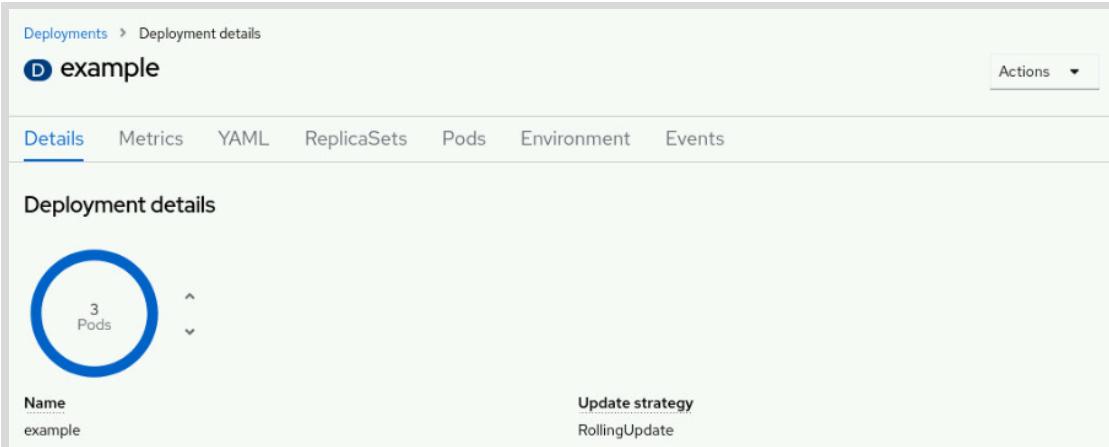
The screenshot shows the 'Create Deployment' form. At the top, it says 'Project: selfservice-ranges'. Below that, 'Configure via:' has 'Form view' selected (indicated by a blue radio button). A note at the top left says 'DeploymentConfig is being deprecated with OpenShift 4.14'. It continues: 'Feature development of DeploymentConfigs will be deprecated in OpenShift Container Platform 4.14. DeploymentConfigs will continue to be supported for security and critical fixes, but you should migrate to Deployments wherever it is possible.' Below this note is a link 'Learn more about Deployments'. Another note below says 'Note: Some fields may not be represented in this form view. Please select "YAML view" for full control.' At the bottom, the 'Name *' field contains 'example'.

You create this deployment several times during this exercise. To use the terminal instead for the exercise, copy the deployment definition from the YAML editor.

3.3. Click **Create** to create the deployment.

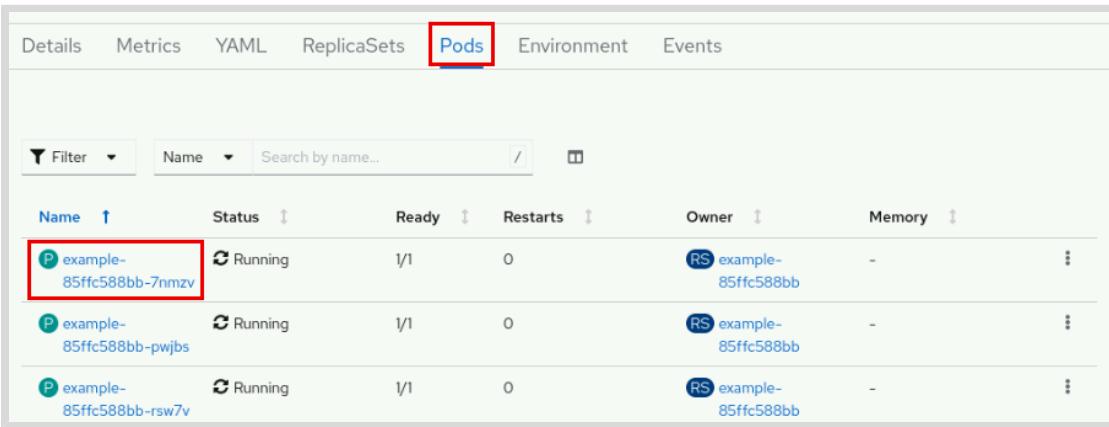
► 4. Examine the containers in the deployment.

- 4.1. Wait a few seconds until the **Deployment Details** section shows that the deployment scaled to three pods.



The screenshot shows the 'Deployment details' page for a deployment named 'example'. At the top, there's a summary card with a large blue circle containing the number '3' and the word 'Pods'. Below this, under the 'Deployment details' heading, there are two sections: 'Name' (example) and 'Update strategy' (RollingUpdate). The 'Details' tab is selected in the navigation bar.

- 4.2. Click the **Pods** tab, and then click the name of any of the pods in the example deployment.



The screenshot shows the 'Pods' list page for the 'example' deployment. The 'Pods' tab is selected in the navigation bar. A table lists three pods:

Name	Status	Ready	Restarts	Owner	Memory	...
P example-85ffc588bb-7nmzv	Running	1/1	0	RS example-85ffc588bb	-	...
P example-85ffc588bb-pwjbs	Running	1/1	0	RS example-85ffc588bb	-	...
P example-85ffc588bb-rsw7v	Running	1/1	0	RS example-85ffc588bb	-	...

The first pod, 'example-85ffc588bb-7nmzv', is highlighted with a red box.

The name of the pods might differ from the ones you get.

- 4.3. Scroll to the **Containers** section, and then click **container**.

The screenshot shows the 'Containers' section of a pod details page. A container named 'container' is listed, with its name highlighted by a red box. The table columns are Name, Image, State, Restarts, and Started. The 'container' row shows the image 'image-registry.openshift-image-stream-index:latest', state 'Running', 0 restarts, and started at 'Feb 6, 2024, 3:36 PM'.

Name	Image	State	Restarts	Started
container	image-registry.openshift-image-stream-index:latest	Running	0	Feb 6, 2024, 3:36 PM

Figure 6.8: The container link in the pod details page

- 4.4. Verify that the **Resource requests** and **Resource limits** fields show a hyphen.
Containers do not have resource requests nor limits by default.

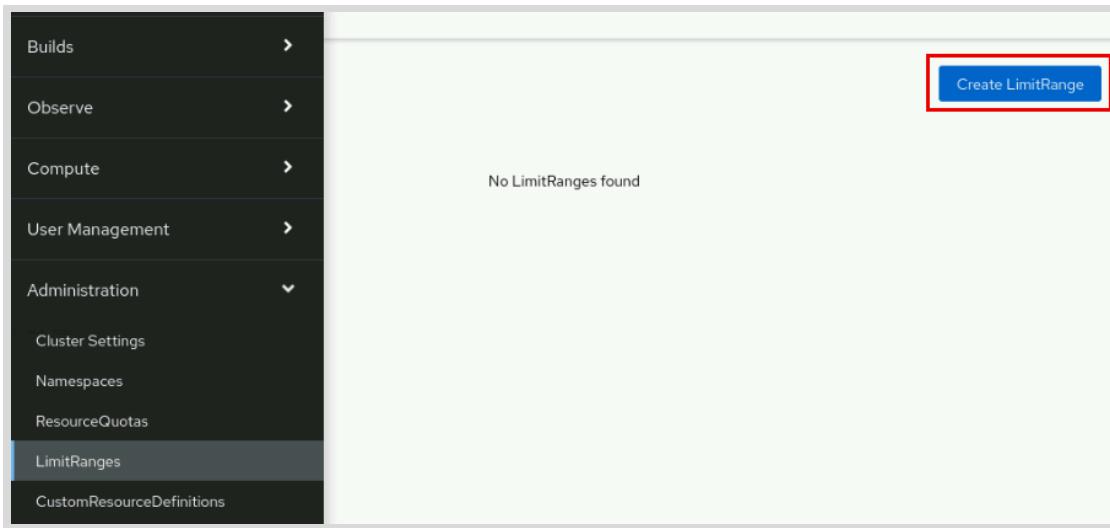
The screenshot shows the 'Container details' section of a container details page. A container named 'container' is listed, with its name highlighted by a red box. The table columns are State, ID, Restarts, Resource requests, and Resource limits. The 'Resource requests' and 'Resource limits' rows are highlighted by a red box.

State	ID	Restarts	Resource requests	Resource limits
Running	cri-o://437elaf437dc6cc69ee56a1f2801814560696b8f5de26b5falb6d02de542f04e	0	-	-

Figure 6.9: Container resources

► 5. Create a limit range.

- 5.1. Navigate to **Administration > LimitRanges**, and then click **Create LimitRange**.



- 5.2. The YAML editor displays a template that defines a limit range for containers. The limit range sets a default memory request of 256 Mi and a default memory limit of 512 Mi.

```

apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
  namespace: selfservice-ranges
spec:
  limits:
    default:
      memory: 512Mi
      defaultRequest:
        memory: 256Mi
      type: Container

```

LimitRange

Schema

LimitRange sets resource usage limits for each kind of resource in a Namespace.

- apiVersion string

APIVersion defines the versioned schema of this representation of an object. Servers should convert

Click **Create** to create a limit range with the template definition.

- 6. Examine the containers of the original deployment to verify that the limit range did not add resource requests nor limits.
- 6.1. Navigate to **Workloads > Pods**, and then click the name of any of the pods in the example deployment.

Name	Status	Ready	Restarts	Owner	Memory
example-85ffc588bb-7nmzv	Running	1/1	0	RS example-85ffc588bb	-
example-85ffc588bb-pwjbs	Running	1/1	0	RS example-85ffc588bb	-
example-85ffc588bb-rsw7v	Running	1/1	0	RS example-85ffc588bb	-

6.2. Scroll to the **Containers** section, and then click **container**.

Name	Image	State	Restarts	Started
container	image-registry.openshift-image... oci-dockerd://85ffc588bb-7nmzv	Running	0	Feb 6, 2024, 3:36 PM

6.3. The **Resource requests** and **Resource limits** fields continue to show a hyphen.

Resource requests
-

Resource limits
-

► **7.** Delete the deployment.

- 7.1. Navigate to **Workloads > Deployment**. Click the vertical ellipsis (⋮) menu at the end of the **example** row, and then click **Delete Deployment**. Click **Delete** to confirm.

The screenshot shows a table of Deployments. One row is selected, labeled 'example'. To the right of the row is a vertical ellipsis (⋮) menu. A red box highlights the 'Delete Deployment' option in this menu.

Name	Status	Labels	Pod selector
example	3 of 3 pods	No labels	app=example

► **8.** Create the example deployment again.

- 8.1. Navigate to **Workloads > Deployments**, and then click **Create Deployment**.

The screenshot shows the left-hand navigation sidebar with 'Administrator' selected. Under 'Workloads', 'Deployments' is selected and highlighted with a blue bar. The main content area shows the message 'No Deployments found'. In the bottom right corner of the main area, there is a blue button labeled 'Create Deployment', which is also highlighted with a red box.

- 8.2. Ensure that **Form view** is selected, and then type **example** in the **Name** field.

8.3. Click **Create** to create the deployment.

► **9.** Examine the containers in the deployment.

9.1. Wait a few seconds until the **Deployment Details** section shows that the deployment scaled to three pods.

9.2. Click the **Pods** tab, and then click the name of any of the pods.

Name	Status	Ready	Restarts	Owner	Memory
example-85ffc588bb-7hlcc	Running	1/1	0	RS example-85ffc588bb	-
example-85ffc588bb-swk5f	Running	1/1	0	RS example-85ffc588bb	-
example-85ffc588bb-x9z4h	Running	1/1	0	RS example-85ffc588bb	-

9.3. Scroll to the **Containers** section, and then click **container**.

The screenshot shows the deployment details for the 'example' deployment. It includes sections for Annotations (5 annotations), Created at (Feb 6, 2024, 7:30 PM), Owner (RS example-85ffc58bb), Node (master01), Image pull secret (default-dockercfg-j9477), PodDisruptionBudget (No PodDisruptionBudget), and Receiving Traffic (0). The Containers section lists one container named 'container' with the following details:

Name	Image	State	Restarts	Started
container	image-registry.openshift-image... (image ID: cri-o://d9e2a2994f3e7d092c1874a7ff8a9ed43c4be57bf313a8bc71953bae277daa06)	Running	0	Feb 6, 2024, 7:30 PM

- 9.4. Note that the **Resource requests** and **Resource limits** fields now have values that correspond to the limit range.

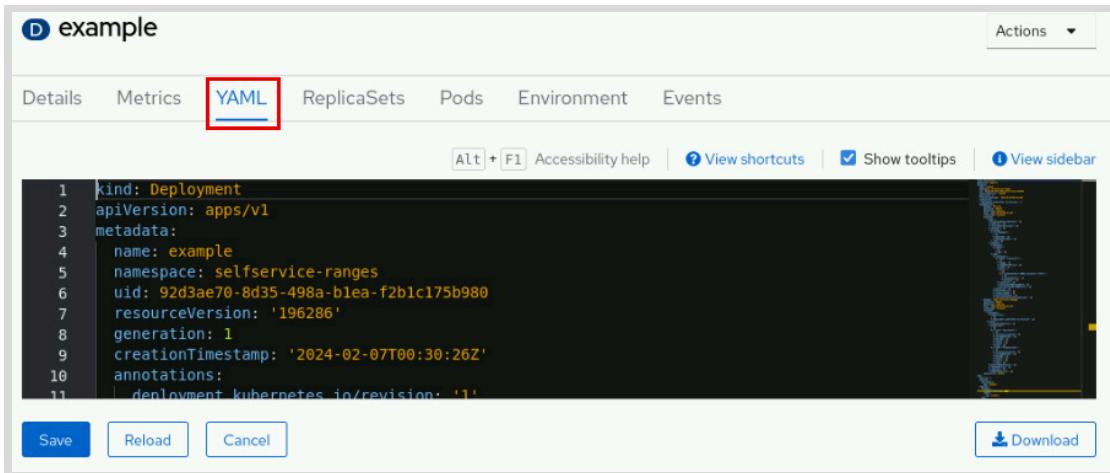
The screenshot shows the container details for the 'container' pod. It includes fields for State (Running), ID (cri-o://d9e2a2994f3e7d092c1874a7ff8a9ed43c4be57bf313a8bc71953bae277daa06), Restarts (0), and Resource requests (memory: 256Mi). The Resource limits field (memory: 512Mi) is also shown. The Resource requests and Resource limits fields are highlighted with a red box.

► 10. Examine the deployment.

- 10.1. Navigate to Workloads > Deployments, and then click **example**.

The screenshot shows the list of deployments. The 'example' deployment is selected and highlighted with a red box. It has a status of '3 of 3 pods' and a pod selector of 'app=example'.

- 10.2. Click **YAML** to show the YAML editor.



The screenshot shows the OpenShift YAML editor interface for a deployment named 'example'. The 'YAML' tab is selected, highlighted with a red box. The editor displays the following YAML code:

```

1 kind: Deployment
2 apiVersion: apps/v1
3 metadata:
4   name: example
5   namespace: selfservice-ranges
6   uid: 92d3ae70-8d35-498a-b1ea-f2b1c175b980
7   resourceVersion: '196206'
8   generation: 1
9   creationTimestamp: '2024-02-07T00:30:26Z'
10  annotations:
11    deployment.kubernetes.io/revision: '1'

```

Below the editor are buttons for 'Save', 'Reload', 'Cancel', and 'Download'.

10.3. The YAML editor displays the resource definition of the deployment.

```

kind: Deployment
apiVersion: apps/v1
metadata:
  name: example
  namespace: selfservice-ranges
...output omitted...
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: example
    spec:
      containers:
        - name: container
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/
httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          resources: {}
          terminationMessagePath: /dev/termination-log
...output omitted...

```

Although the containers have resource limits and requests, the `resources` key in the deployment is empty. Limit ranges modify containers to add resource limits and requests, but not deployments.

► 11. Evaluate the limit range by examining pod metrics.

11.1. Navigate to **Workloads > Pods**, and then click the name of any of the pods.

11.2. Click Metrics.



Figure 6.25: Pod metrics

The **Memory usage** graph displays the memory usage of the pod (about 50 MiB), the request (256 MiB), and the limit (512 MiB).

The template deployment in the web console uses an `httpd` image that consumes little memory. In this case, the limit range requests more memory than the container requires to work. If you create many similar deployments, then the limit range can cause the deployments to request more memory than they need. If the namespace has resource quotas, then you might not be able to create workloads even if the cluster has enough available resources.

Most real workloads have larger memory usage that varies with load. Evaluate the resource usage of your workloads to decide whether limit ranges can help you to manage cluster resources, and examine resource usage to find adequate values.

Finish

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

```
[student@workstation ~]$ lab finish selfservice-ranges
```

The Project Template and the Self-Provisioner Role

Objectives

- Configure default quotas, limit ranges, role bindings, and other restrictions for new projects, and the allowed users to self-provision new projects.

Project Creation

Kubernetes provides namespaces to isolate workloads.

Namespace metadata has security implications in clusters. For example, policy controllers might use namespace labels to limit capabilities in a namespace. If users can modify namespaces, then malicious users can modify namespace metadata to override security measures.

Additionally, namespaces are not namespaced. Therefore, granting granular access to namespaces poses some challenges. For example, with Kubernetes role-based access control, you cannot allow users to list a subset of namespaces. However, to allow users to list their namespaces, you must allow them to list all namespaces.



Note

Listing resources and viewing individual resources are different operations. You can grant users permissions to view specific namespaces, but listing namespaces requires a separate permission.

OpenShift introduces projects to improve security and users' experience of working with namespaces. The OpenShift API server adds the `Project` resource type. When you make a query to list projects, the API server lists namespaces, filters the visible namespaces to your user, and returns the visible namespaces in project format.

Additionally, OpenShift introduces the `ProjectRequest` resource type. When you create a project request, the OpenShift API server creates a namespace from a template. By using a template, cluster administrators can customize namespace creation. For example, cluster administrators can ensure that new namespaces have specific permissions, resource quotas, or limit ranges.

These features provide self-service management of namespaces. Cluster administrators can allow users to create namespaces without allowing users to modify namespace metadata. Administrators can also customize the creation of namespaces to ensure that namespaces follow organizational requirements.

Planning a Project Template

You can add any namespaced resource to the project template. For example, you can add resources of the following types:

Roles and role bindings

Add roles and role bindings to the template to grant specific permissions in new projects. The default template grants the `admin` role to the user who requests the project. You can keep this permission or use another similar permission, such as granting the `admin` role to

a group of users. You can also add different permissions, such as more granular permissions over specific resource types.

Resource quotas and limit ranges

Add resource quotas to the project template to ensure that all new projects have resource limits. If you add resource quotas, then creating workloads requires explicit resource limit declarations. Consider adding limit ranges to reduce the effort for workload creation.

Even with quotas in all namespaces, users can create projects to continue adding workloads to a cluster. If this scenario is a concern, then consider adding cluster resource quotas to the cluster.

Network policies

Add network policies to the template to enforce organizational network isolation requirements.

Creating a Project Template

The `oc adm create-bootstrap-project-template` command prints a template that you can use to create your own project template.

This template has the same behavior as the default project creation in OpenShift. The template adds a role binding that grants the `admin` cluster role over the new namespace to the user who requests the project.

Project templates use the same template feature as the `oc new-app` command.

Execute the following command to create a file with an initial template:

```
[user@host ~]$ oc adm create-bootstrap-project-template -o yaml > file
```

This initial template has the following content:

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects: ①
- apiVersion: project.openshift.io/v1 ②
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: rbac.authorization.k8s.io/v1 ③
  kind: RoleBinding
  metadata:
    creationTimestamp: null
    name: admin
```

```

namespace: ${PROJECT_NAME}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: ${PROJECT_ADMIN_USER}
parameters: ④
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
- name: PROJECT_DESCRIPTION
- name: PROJECT_ADMIN_USER
- name: PROJECT_REQUESTING_USER

```

- ① The resources that OpenShift creates in new namespaces
- ② The project resource
- ③ A role binding to grant the admin role to the requesting user
- ④ The parameters that are available to the template

When a user requests a project, OpenShift replaces the `${VARIABLE}` syntax with the parameters of the project request, and creates the objects in the `objects` key.

Modify the object list to add the required resources for new namespaces.

The YAML output of `oc` commands that return lists of objects is formatted similarly to the template `objects` key.

```

[user@host ~]$ oc get limitrange,resourcequota -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: LimitRange
  metadata:
    creationTimestamp: "2024-01-31T17:48:23Z"
    name: example
    namespace: example
    resourceVersion: "881771"
    uid: d0c19c60-00a9-4028-acc5-22680f1ea658
  spec:
    limits:
      - default:
          cpu: 500m
          memory: 512Mi
        defaultRequest:
          cpu: 250m
          memory: 256Mi
      max:
        cpu: "1"
        memory: 1Gi
      min:

```

```

cpu: 125m
memory: 128Mi
type: Container
- apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2024-01-31T17:48:04Z"
  name: example
  namespace: example
  resourceVersion: "881648"
  uid: 108f0771-dc11-4289-ae76-6514d58bbece
spec:
  hard:
    count/pods: "1"
status:
...output omitted...
kind: List
metadata:
resourceVersion: ""

```

Some common resources in project templates, such as quotas, do not have strict validation. For example, if the previous template contains the `count/pod` text instead of the `count/pods` text, then the quota does not work. You can create the project template, and new namespaces contain the quota, but the quota does not have an effect. To define a project template and to reduce the risk of errors, you can perform the following steps:

- Create a namespace.
- Create your chosen resources and test until you get the intended behavior.
- List the resources in YAML format.
- Edit the resource listing to ensure that the definitions create the correct resources. For example, remove elements that do not apply to resource creation, such as the `creationTimestamp` or `status` keys.
- Replace the namespace name with the `${PROJECT_NAME}` value.
- Add the list of resources to the project template that the `oc adm create-bootstrap-project-template` command generates.



Note

Extracting a resource definition from an existing resource might not always produce correct results. Besides including elements that do not apply to resource creation, existing definitions might contain attributes that generate unexpected behavior. For example, a controller might add to resources some annotations that are unsuitable for template definitions.

Even after testing the resources in a test namespace, always verify that the projects that are created from your template have only the required behavior.

Use the `oc create` command to create the template resource in the `openshift-config` namespace:

```
[user@host ~]$ oc create -f template -n openshift-config
template.template.openshift.io/project-request created
```

Configuring the Project Template

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 `projects.config.openshift.io/cluster` resource rolls out a new version of the `apiserver` deployment in the `openshift-apiserver` namespace. After the new `apiserver` deployment completes, new projects create the resources in the customized project template.



Note

During the `apiserver` deployment rollout, API requests can produce unexpected results.

To revert to the original project template, modify the `projects.config.openshift.io/cluster` resource to clear the `spec` resource to match the `spec: {}` format.

Managing Self-provisioning Permissions

Users with the `self-provisioner` cluster role can create projects. By default, the `self-provisioner` role is bound to all authenticated users.

Control the binding of the role to limit which users can request new projects.



Important

Remember that users with namespace permissions can create namespaces that do not use the project template.

Use the `oc describe` command to view the role bindings.

```
[user@host ~]$ oc describe clusterrolebinding.rbac self-provisioners
Name:          self-provisioners
Labels:        <none>
Annotations:   rbac.authorization.kubernetes.io/autoupdate: true
Role:
```

```
Kind: ClusterRole
Name: self-provisioner
Subjects:
  Kind   Name           Namespace
  ----  ---            -----
  Group  system:authenticated:oauth
```

This role binding has an `rbac.authorization.kubernetes.io/autoupdate` annotation. This annotation protects roles and bindings from modifications that can interfere with the working of clusters. When the API server starts, the cluster restores resources with this annotation automatically, unless you set the annotation to the `false` value.

To make changes, disable automatic updates with the annotation, and edit the subjects in the binding.



Important

The `oc adm policy remove-cluster-role-from-group` command removes the cluster role binding when you remove the last subject.

Use extra caution or avoid this command to manage protected role bindings. The command removes the permission, but only until the API server restarts. Removing the permission permanently after deleting the binding is a lengthier process than changing the subjects.

For example, to disable self-provisioning, execute the following commands:

```
[user@host ~]$ oc annotate clusterrolebinding/self-provisioners \
  --overwrite rbac.authorization.kubernetes.io/autoupdate=false
clusterrolebinding.rbac.authorization.k8s.io/self-provisioners annotated
[user@host ~]$ oc patch clusterrolebinding.rbac self-provisioners \
  -p '{"subjects": null}'
clusterrolebinding.rbac.authorization.k8s.io/self-provisioners patched
```

You can also use the `oc edit` command to modify any value of a resource. The command launches the `vi` editor to apply your modifications. For example, to change the subject of the role binding from the `system:authenticated:oauth` group to the `provisioners` group, execute the followign command:

```
[user@host ~]$ oc edit clusterrolebinding/self-provisioners
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  ...output omitted...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: self-provisioner
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: provisioners
```



References

For more information, refer to the *Configuring Project Creation* section in the *Projects* chapter in the Red Hat OpenShift Container Platform 4.14 *Building Applications* documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/building_applications/index#configuring-project-creation

Customizing OpenShift Project Creation

<https://developers.redhat.com/blog/2020/02/05/customizing-openshift-project-creation/>

► Guided Exercise

The Project Template and the Self-Provisioner Role

Restrict the ability to self-provision projects to a group of users, and ensure that all users from that group have write privileges on all projects that any of them creates. Also, ensure that their new projects are constrained by a limit range that restricts memory usage.

Outcomes

- Limit project creation to a group of users.
- Customize project creation.

Before You Begin

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

This command performs the following actions:

- Ensure that the cluster API is reachable.
- Create the `provisioner1` and `provisioner2` users with the `redhat` password.

```
[student@workstation ~]$ lab start selfservice-projtemplate
```

Instructions

In this exercise, you configure the cluster so that only members of the `provisioners` group can create projects. Members of the `provisioners` group have full permissions on new projects. Users cannot create workloads that request more than 1 GiB of RAM in new projects.

► 1. Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.

1.1. Log in to the cluster as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

► 2. Allow only members of the `provisioners` group to create projects.

2.1. Examine the `provisioners` group.

```
[student@workstation ~]$ oc describe group provisioners
Name:          provisioners
Created:       12 seconds ago
Labels:        <none>
Annotations:   <none>
Users:         provisioner1
               provisioner2
```

The provisioners group contains the provisioner1 and provisioner2 users.

- 2.2. Use the `oc edit` command to edit the `self-provisioners` cluster role binding.

```
[student@workstation ~]$ oc edit clusterrolebinding self-provisioners
```

The `oc edit` command launches the `vi` editor to apply your modifications. Change the subject of the role binding from the `system:authenticated:oauth` group to the `provisioners` group.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
...output omitted...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: self-provisioner
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: provisioners
```



Note

The `rbac.authorization.kubernetes.io/autoupdate` annotation protects this cluster role binding. If the API server restarts, then Kubernetes restores this cluster role binding.

In this exercise context, you are not required to make the change permanent.

Not in this exercise, but in a real-world context, you would make the change permanent by using the following command:

```
[user@host ~]$ oc annotate clusterrolebinding/self-provisioners \
--overwrite rbac.authorization.kubernetes.io/autoupdate=false
```

- 3. Verify that users outside the `provisioners` group cannot create projects.

- 3.1. Log in to the cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer  
Login successful.
```

You don't have any projects. Contact your system administrator to request a project.

After the role binding is changed, the `oc login` command reports that you must contact your system administrator to request a project, because the `developer` user cannot create projects.

- 3.2. Verify that the `developer` user cannot create projects.

```
[student@workstation ~]$ oc new-project test  
Error from server (Forbidden): You may not request a new project via this API.
```

- 4. Verify that members of the `provisioners` group can create projects.

- 4.1. Log in to the cluster as the `provisioner1` user with the `redhat` password.

```
[student@workstation ~]$ oc login -u provisioner1 -p redhat  
Login successful.
```

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

- 4.2. Create a project by using the `oc new-project` command.

```
[student@workstation ~]$ oc new-project test  
Now using project "test" on server "https://api.ocp4.example.com:6443".  
...output omitted...
```

- 4.3. Verify that you can create resources in the `test` project.

```
[student@workstation ~]$ oc create configmap test  
configmap/test created
```

- 5. Verify that another member of the `provisioners` group cannot access the `test` project.

- 5.1. Log in to the cluster as the `provisioner2` user with the `redhat` password.

```
[student@workstation ~]$ oc login -u provisioner2 -p redhat  
Login successful.
```

You don't have any projects. You can try to create a new project, by running
`oc new-project <projectname>`

The `oc login` command reports that the `provisioner2` user does not have any projects.

- 5.2. Try to change to the `test` project with the `oc project` command.

```
[student@workstation ~]$ oc project test
error: You are not a member of project "test".
You are not a member of any projects. You can request a project to be created with
the 'new-project' command.
```

- 6. Log in to the cluster as the `admin` user with the `redhatocp` password, to clean up.

- 6.1. Log in as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
...output omitted...
```

- 6.2. Delete the `test` project.

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

- 7. Create a namespace to design a project template. Add a limit range that prevents users from creating workloads that request more than 1 GiB of RAM.

- 7.1. Use the `oc create namespace` command to create the `template-test` namespace.

```
[student@workstation ~]$ oc create namespace template-test
namespace/template-test created
```

- 7.2. Edit the `~/D0280/labs/selfservice-projtemplate/limitrange.yaml` file to add the limit. The file must match the following content:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: max-memory
  namespace: template-test
spec:
  limits:
    - max:
        memory: 1Gi
      type: Container
```

- 7.3. Use the `oc create` command to create the limit range that the `~/D0280/labs/selfservice-projtemplate/limitrange.yaml` file defines.

```
[student@workstation ~]$ oc create \
-f ~/D0280/labs/selfservice-projtemplate/limitrange.yaml
limitrange/max-memory created
```

- 7.4. Examine the `~/D0280/labs/selfservice-projtemplate/deployment.yaml` file. This file defines a deployment that requests 2 GiB of RAM.

```

apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
  name: test
spec:
...output omitted...
  template:
...output omitted...
    spec:
      containers:
        - image: registry.ocp4.example.com:8443/redhattraining/hello-world-
nginx:v1.0
          name: hello-world-nginx
        resources:
          limits:
            memory: 2Gi

```

- 7.5. Create the deployment by using the ~/D0280/labs/selfservice-projtemplate/deployment.yaml file.

```
[student@workstation ~]$ oc create \
-f ~/D0280/labs/selfservice-projtemplate/deployment.yaml \
-n template-test
deployment.apps/test created
```

- 7.6. Examine the pods and events in the template-test namespace.

```
[student@workstation ~]$ oc get pod -n template-test
No resources found in template-test namespace.
```

```
[student@workstation ~]$ oc get event -n template-test \
--sort-by=metadata.creationTimestamp
LAST SEEN      TYPE      REASON          OBJECT                  MESSAGE
...output omitted...
39s           Warning   FailedCreate    replicaset/test-846769884c  Error
creating: pods "test-846769884c-5zjhw" is forbidden: maximum memory usage per
Container is 1Gi, but limit is 2Gi
```

The limit range maximum prevents the deployment from creating pods.

► 8. Define the project template.

The ~/D0280/solutions/selfservice-projtemplate/template.yaml file contains a solution.

- 8.1. Use the `oc adm create-bootstrap-project-template` command to print an initial project template. Redirect the output to the `template.yaml` file.

```
[student@workstation ~]$ oc adm create-bootstrap-project-template \
-o yaml >template.yaml
```

- 8.2. Use the `oc` command to list the limit range in YAML format. Redirect the output to append to the template.yaml file.

```
[student@workstation ~]$ oc get limitrange -n template-test \
-o yaml >>template.yaml
```

- 8.3. Edit the template.yaml file to perform the following operations:

- Apply the following changes to the `subjects` key in the `admin` role binding:
 - Change the `kind` key to `Group`.
 - Change the `name` key to `provisioners`.
- Move the limit range to immediately after the role binding definition.
- Replace the namespace: `template-test` text with the namespace: `${PROJECT_NAME}` text.
- Remove any left-over content after the `parameters` block.
- Remove the following keys from the limit range and quota definitions:
 - `creationTimestamp`
 - `resourceVersion`
 - `uid`

If you use the `vi` editor, then you can use the following procedure to move a block of text:

- Move to the beginning of the block.
- Press `V` to enter visual line mode. This mode selects entire lines for manipulation.
- Move to the end of the block. The editor highlights the selected lines.
- Press `d` to delete the lines and to store them in a register for later use.
- Move to the destination.
- Press `P` to insert the lines that are stored in the register.

You can also press `dd` to delete entire lines, and press `.` to repeat the operation.

The resulting file should match the following content:

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: project.openshift.io/v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
```

```

creationTimestamp: null
name: ${PROJECT_NAME}
spec: {}
status: {}
- apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: admin
  namespace: ${PROJECT_NAME}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: provisioners
- apiVersion: v1
kind: LimitRange
metadata:
  name: max-memory
  namespace: ${PROJECT_NAME}
spec:
  limits:
    - default:
        memory: 1Gi
      defaultRequest:
        memory: 1Gi
      max:
        memory: 1Gi
      type: Container
parameters:
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
- name: PROJECT_DESCRIPTION
- name: PROJECT_ADMIN_USER
- name: PROJECT_REQUESTING_USER

```



Note

The limit range has `default` and `defaultRequest` limits, although the definition does not contain these keys. When creating a limit range, always set the `default` and `defaultRequest` limits for more predictable behavior.

► 9. Create and configure the project template.

9.1. Use the oc command to create the project template.

```
[student@workstation ~]$ oc create -f template.yaml -n openshift-config
template.template.openshift.io/project-request created
```

9.2. Use the oc edit command to change the global cluster project configuration.

```
[student@workstation ~]$ oc edit projects.config.openshift.io cluster
```

Edit the resource to match the following content:

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
...output omitted...
  name: cluster
...output omitted...
spec:
  projectRequestTemplate:
    name: project-request
```

9.3. Use the `watch` command to view the API server pods.

```
[student@workstation ~]$ watch oc get pod -n openshift-apiserver
NAME      READY   STATUS    RESTARTS   AGE
apiserver-6b7b...   2/2     Running   0          2m30s
```

Wait until new pods are rolled out. The rollout can take a few minutes to start. Press `Ctrl+C` to exit the `watch` command.

▶ 10. Create a project as the `provisioner1` user.

10.1. Log in to the cluster as the `provisioner1` user with the `redhat` password.

```
[student@workstation ~]$ oc login -u provisioner1 -p redhat
Login successful.
...output omitted...
```

10.2. Create a project by using the `oc new-project` command.

```
[student@workstation ~]$ oc new-project test
Now using project "test" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

▶ 11. Verify that the `provisioner2` user can access the `test` project and create resources. Verify that the limit range has the intended effect.

11.1. Log in to the cluster as the `provisioner2` user with the `redhat` password.

```
[student@workstation ~]$ oc login -u provisioner2 -p redhat
Login successful.

You have one project on this server: "test"

Using project "test".
```

The `oc login` command reports that the `provisioner2` user has the `test` project. The command selects the project.

- 11.2. Create a resource on the test project.

```
[student@workstation ~]$ oc create configmap test  
configmap/test created
```

The provisioner2 user can create resources in a project that the provisioner1 user created.

- 11.3. Create a deployment that exceeds the limit range by using the ~/D0280/labs/selfservice-projtemplate/deployment.yaml file.

```
[student@workstation ~]$ oc create \  
-f ~/D0280/labs/selfservice-projtemplate/deployment.yaml  
deployment.apps/test created
```

- 11.4. Examine the pods and events in the template-test namespace.

```
[student@workstation ~]$ oc get pod  
No resources found in test namespace.
```

```
[student@workstation ~]$ oc get event --sort-by=metadata.creationTimestamp  
LAST SEEN      TYPE      REASON          OBJECT          MESSAGE  
...output omitted...  
39s           Warning   FailedCreate    replicaset/test-846769884c   Error  
creating: pods "test-846769884c-5zjhw" is forbidden: maximum memory usage per  
Container is 1Gi, but limit is 2Gi
```

The limit range works as expected.

Finish

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

```
[student@workstation ~]$ lab finish selfservice-projtemplate
```

► Lab

Enable Developer Self-Service

Configure a project with restrictions that prevent its users and applications from consuming all capacity of a cluster.

Outcomes

- Configure project creation to use a custom project template.
- Limit resource usage for new projects.

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.

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

Instructions

- Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.
- Design a project template with the following properties:
 - The user who requests the project has the default `admin` role binding.
 - The workloads in the project cannot request a total of more than 2 GiB of RAM, and they cannot use more than 4 GiB of RAM.

Each workload in the project has the following properties:

- Default memory request of 256 MiB
- Default memory limit of 512 MiB
- Minimum memory request of 128 MiB
- Maximum memory usage of 1 GiB

You can use the `oc create quota` command to create the resource quota without creating a YAML definition. A template for the limit range is available at `~/D0280/labs/selfservice-review/limitrange.yaml`.

You can create a `template-test` namespace to design your project template.



Note

The next steps assume that you design the template in a `template-test` namespace. The lab scripts clean and grade the design namespace only if you create it with the `template-test` name.

3. Verify that the quota and limit range have the intended effect.

For example, create a deployment that uses the `registry.ocp4.example.com:8443/redhat-training/hello-world-nginx:v1.0` image without resource specifications. Verify that the limit range adds requests and limits to the pods. Scale the deployment to 10 replicas. Examine the deployment and the quota to verify that they have the intended effect.

If you design your template without creating a test namespace, then you must verify your design by other means.

4. Create a project template definition with the same properties.

**Note**

The solution for this step assumes that you designed your template in a `template-test` namespace. If you do not create a `template-test` namespace to design the template, then you must create the project template by other means.

The `~/D0280/solutions/selfservice-review/template.yaml` file contains a solution.

5. Create and configure the project template.
6. Create a project to verify that the template works as intended.

**Note**

The lab scripts clean up only a `template-validate` namespace.

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 grade selfservice-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Enable Developer Self-Service

Configure a project with restrictions that prevent its users and applications from consuming all capacity of a cluster.

Outcomes

- Configure project creation to use a custom project template.
- Limit resource usage for new projects.

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.

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

Instructions

- Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.

- 1.1. Log in to the cluster as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

2. Design a project template with the following properties:

- The user who requests the project has the default `admin` role binding.
- The workloads in the project cannot request a total of more than 2 GiB of RAM, and they cannot use more than 4 GiB of RAM.

Each workload in the project has the following properties:

- Default memory request of 256 MiB
- Default memory limit of 512 MiB
- Minimum memory request of 128 MiB
- Maximum memory usage of 1 GiB

You can use the `oc create quota` command to create the resource quota without creating a YAML definition. A template for the limit range is available at `~/D0280/labs/selfservice-review/limitrange.yaml`.

You can create a `template-test` namespace to design your project template.



Note

The next steps assume that you design the template in a `template-test` namespace. The lab scripts clean and grade the design namespace only if you create it with the `template-test` name.

- 2.1. Use the `oc` command to create a `template-test` namespace.

```
[student@workstation ~]$ oc create namespace template-test  
namespace/template-test created
```

- 2.2. Use the `oc create quota` command to create the memory quota in the `template-test` namespace.

```
[student@workstation ~]$ oc create quota memory \  
--hard=requests.memory=2Gi,limits.memory=4Gi \  
-n template-test  
resourcequota/memory created
```

- 2.3. Edit the limit range definition at `~/D0280/labs/selfservice-review/limitrange.yaml`. Replace the `CHANGE_ME` text to match the following file:

```
apiVersion: v1  
kind: LimitRange  
metadata:  
  name: memory  
  namespace: template-test  
spec:  
  limits:  
    - min:  
        memory: 128Mi  
    defaultRequest:  
        memory: 256Mi  
    default:  
        memory: 512Mi  
    max:  
        memory: 1Gi  
  type: Container
```

- 2.4. Use the `oc` command to create the limit range.

```
[student@workstation ~]$ oc create \  
-f ~/D0280/labs/selfservice-review/limitrange.yaml  
limitrange/memory created
```

3. Verify that the quota and limit range have the intended effect.

Chapter 6 | Enable Developer Self-Service

For example, create a deployment that uses the `registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0` image without resource specifications. Verify that the limit range adds requests and limits to the pods. Scale the deployment to 10 replicas. Examine the deployment and the quota to verify that they have the intended effect.

If you design your template without creating a test namespace, then you must verify your design by other means.

- 3.1. Use the `oc create deployment` command to create a deployment without resource specifications.

```
[student@workstation ~]$ oc create deployment -n template-test test-limits \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0
deployment.apps/test-limits created
```

- 3.2. Use the `oc` command to view the `resources` key of the container specification. Optionally, use the `jq` command to indent the output.

```
[student@workstation ~]$ oc get pod -n template-test \
-o jsonpath='{.items[0].spec.containers[0].resources}'
{"limits":{"memory":"512Mi"}, "requests":{"memory":"256Mi"}}
```

Although you create the deployment without specifying resources, the limit range applies RAM requests and limits.

- 3.3. Use the `oc scale` command to scale the deployment to verify that the quota has an effect.

```
[student@workstation ~]$ oc scale deployment -n template-test test-limits \
--replicas=10
deployment.apps/test-limits scaled
```

- 3.4. Examine the deployment and the quota.

```
[student@workstation ~]$ oc get deployment -n template-test
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
test-limits  8/10     8           8          8m41s
```

```
[student@workstation ~]$ oc describe resourcequota -n template-test memory
Name:          memory
Namespace:     template-test
Resource       Used   Hard
-----
limits.memory  4Gi    4Gi
requests.memory 2Gi   2Gi
```

The deployment uses the quota completely, and scales only to eight pods. Each pod requests 256 MiB of RAM, and eight pods request 2 GiB of RAM. Each pod has a 512 MiB RAM limit, and eight pods have a 4 GiB RAM limit.

4. Create a project template definition with the same properties.

**Note**

The solution for this step assumes that you designed your template in a `template-test` namespace. If you do not create a `template-test` namespace to design the template, then you must create the project template by other means.

The `~/DO280/solutions/selfservice-review/template.yaml` file contains a solution.

- 4.1. Use the `oc adm create-bootstrap-project-template` to print an initial project template. Redirect the output to the `template.yaml` file.

```
[student@workstation ~]$ oc adm create-bootstrap-project-template \
-o yaml >template.yaml
```

- 4.2. Use the `oc` command to list the limit ranges and quotas in YAML format. Redirect the output to append to the `template.yaml` file.

```
[student@workstation ~]$ oc get limitrage,quota -n template-test \
-o yaml >>template.yaml
```

- 4.3. Edit the `template.yaml` file to perform the following operations:

- Move the limit range and quota definitions immediately after the role binding definition.
- Remove any left-over content after the `parameters` block.
- Remove the following keys from the limit range and quota definitions:
 - `creationTimestamp`
 - `resourceVersion`
 - `uid`
 - `status`
- Replace the `namespace: template-test` text with the `namespace: ${PROJECT_NAME}` text.

If you use the `vi` editor, then you can use the following procedure to move a block of text:

- Move to the beginning of the block.
- Press `V` to enter visual line mode. This mode selects entire lines for manipulation.
- Move to the end of the block. The editor highlights the selected lines.
- Press `d` to delete the lines and to store them in a register for later usage.
- Move to the destination.
- Press `P` to insert the lines that are stored in the register.

You can also press `dd` to delete entire lines, and `.` to repeat the operation.

The resulting file should match the following content:

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: project.openshift.io/v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: null
    name: admin
    namespace: ${PROJECT_NAME}
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: admin
  subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: User
      name: ${PROJECT_ADMIN_USER}
- apiVersion: v1
  kind: LimitRange
  metadata:
    name: memory
    namespace: ${PROJECT_NAME}
  spec:
    limits:
      - default:
          memory: 512Mi
        defaultRequest:
          memory: 256Mi
      max:
        memory: 1Gi
      min:
        memory: 128Mi
    type: Container
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: memory
    namespace: ${PROJECT_NAME}
  spec:
```

```
hard:  
  limits.memory: 4Gi  
  requests.memory: 2Gi  
parameters:  
  - name: PROJECT_NAME  
  - name: PROJECT_DISPLAYNAME  
  - name: PROJECT_DESCRIPTION  
  - name: PROJECT_ADMIN_USER  
  - name: PROJECT_REQUESTING_USER
```

5. Create and configure the project template.

5.1. Use the `oc` command to create the project template.

```
[student@workstation ~]$ oc create -f template.yaml -n openshift-config  
template.template.openshift.io/project-request created
```

5.2. Use the `oc edit` command to change the cluster project configuration.

```
[student@workstation ~]$ oc edit projects.config.openshift.io cluster
```

Edit the resource to match the following content:

```
apiVersion: config.openshift.io/v1  
kind: Project  
metadata:  
  ...output omitted...  
  name: cluster  
spec:  
  projectRequestTemplate:  
    name: project-request
```

To edit the file, you use the default `vi` editor.

5.3. Use the `watch` command to view the API server pods.

```
[student@workstation ~]$ watch oc get pod -n openshift-apiserver  
NAME      READY     STATUS    RESTARTS   AGE  
apiserver-5cf...  2/2      Running  0          2m50s
```

Wait until new pods are rolled out. Press `Ctrl+C` to exit the `watch` command.

6. Create a project to verify that the template works as intended.



Note

The lab scripts clean up only a `template-validate` namespace.

6.1. Use the `oc new-project` command to create a `template-validate` project.

```
[student@workstation ~]$ oc new-project template-validate
Now using project "template-validate" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

6.2. Describe the quotas in the current namespace.

```
[student@workstation ~]$ oc describe quota
Name:           memory
Namespace:      template-validate
Resource        Used   Hard
-----
limits.memory   0     4Gi
requests.memory 0     2Gi
```

6.3. Describe the limit ranges in the current namespace.

```
[student@workstation ~]$ oc describe limitrange
Name:           memory
Namespace:      template-validate
Type          Resource  Min    Max  Default Request  Default Limit ...
-----
Container     memory    128Mi  1Gi  256Mi          512Mi       ...
```

6.4. Optionally, execute again the commands that you used earlier to create a deployment.
Scale the deployment, and verify the limits.

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 grade selfservice-review
```

Finish

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

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

Summary

- Cluster administrators can create quotas to limit resource usage by namespace.
- Cluster resource quotas implement resource limits across groups of namespaces that namespace selectors define.
- Limit ranges provide resource defaults, minimums, and maximums for workloads in a namespace.
- Cluster administrators can configure project templates to add resources to all new projects. These resources can implement permissions, quotas, network policies, and others.
- The **self-provisioner** role grants permissions to create projects. By default, this role is bound to all authenticated users.

Chapter 7

Manage Kubernetes Operators

Goal

Install and update operators that the Operator Lifecycle Manager and the Cluster Version Operator manage.

Objectives

- Explain the operator pattern and different approaches for installing and updating Kubernetes operators.
- Install and update operators by using the web console and the Operator Lifecycle Manager.
- Install and update operators by using the Operator Lifecycle Manager APIs.

Sections

- Kubernetes Operators and the Operator Lifecycle Manager (and Matching Quiz)
- Install Operators with the Web Console (and Guided Exercise)
- Install Operators with the CLI (and Guided Exercise)

Lab

- Manage Kubernetes Operators

Kubernetes Operators and the Operator Lifecycle Manager

Objectives

- Describe the operator pattern and different approaches for installing and updating Kubernetes operators.

The Operator Pattern

You can deploy workloads to Kubernetes with resources such as deployments, replica sets, stateful sets, daemon sets, jobs, and cron jobs. All of these resources create a workload that runs software that is packaged as a container image, in different modalities. For example, jobs execute a one-off task; cron jobs execute tasks periodically; and the other resources create persistent workloads. Resources such as deployments, stateful sets, or daemon sets differ on how the workload is distributed in a cluster.

These resources are sufficient to deploy many workloads. However, more complex workloads might require significant work to deploy with only these resources. For example, a workload can involve different component workloads, such as a database server, a back-end service, and a front-end service.

A workload might have maintenance tasks that can be automated, such as backing up data or updating the workload.

The operator pattern is a way to implement reusable software to manage such complex workloads.

An operator typically defines custom resources (CRs). The operator CRs contain the needed information to deploy and manage the workload. For example, an operator that deploys database servers defines a database resource where you can specify the database name, sizing requirements, and other parameters.

The operator watches the cluster for instances of the CRs, and then creates the Kubernetes resources to deploy the custom workload. For example, when you create a database resource, the database operator creates a stateful set and a persistent volume that provide the database that is described in the database resource. If the database resource describes a backup schedule and target, then the operator creates a cron job that backs up the database to the target according to the schedule.

By using operators, cluster administrators create CRs that describe a complex workload, and the operator creates and manages the workload.

Deploying Operators

Many pieces of software implement the operator pattern, in different ways.

Cluster operators

Cluster operators provide the platform services of OpenShift, such as the web console and the OAuth server.

Add-on operators

OpenShift includes the Operator Lifecycle Manager (OLM). The OLM helps users to install and update operators in a cluster. Operators that the OLM manages are also known as *add-on operators*, in contrast with cluster operators that implement platform services.

Other operators

Software providers can create software that follows the operator pattern, and then distribute the software as manifests, Helm charts, or any other software distribution mechanism.

Cluster Operators

The Cluster Version Operator (CVO) installs and updates cluster operators as part of the OpenShift installation and update processes.

The CVO provides cluster operator status information as resources of the `ClusterOperator` type. Inspect the cluster operator resources to examine cluster health.

```
[user@host ~]$ oc get clusteroperator
NAME           VERSION  AVAILABLE  PROGRESSING  DEGRADED  ...
authentication  4.14.0   True       False        False      ...
baremetal      4.14.0   True       False        False      ...
cloud-controller-manager 4.14.0   True       False        False      ...
...output omitted...
```

The status of cluster operator resources includes conditions to help with identifying cluster issues. The `oc` command shows the message that is associated with the latest condition. This message can provide further information about cluster issues.

To view cluster operator resources in the web console, navigate to **Administration > Cluster Settings**, and then click the **ClusterOperators** tab.

The Operator Lifecycle Manager and the OperatorHub

Administrators can use the OLM to install, update, and remove operators.

You can use the web console to interact with the OLM. The OLM also follows the operator pattern, and so the OLM provides CRs to manage operators with the Kubernetes API.

The OLM uses operator catalogs to find available operators to install. Operator catalogs are container images that provide information about available operators, such as descriptions and available versions.

OpenShift includes several default catalogs:

Red Hat

Red Hat packages, ships, and supports operators in this catalog.

Certified

Independent software vendors support operators in this catalog.

Community

Operators without official support.

Marketplace

Commercial operators that you can buy from Red Hat Marketplace.

You can also create your own catalogs, or mirror catalogs for offline clusters.

**Note**

The lab environment includes a single catalog with the operators you use in the course. The lab environment hosts the contents of this catalog, so that the course can be completed without internet access.

The OLM creates a resource of the `PackageManifest` type for each available operator. The web console also displays available operators and provides a wizard to install operators. You can also install operators by using the `Subscription` CR and other CRs.

**Note**

Operators that are installed with the OLM have a different lifecycle from cluster operators. The CVO installs and updates cluster operators in lockstep with the cluster. Administrators use the OLM to install, update, and remove operators independently from cluster updates.

Some operators might require additional steps to install, update, or remove.

Implementing Operators

An operator is composed of a set of custom resource definitions and a Kubernetes workload. The operator workload uses the Kubernetes API to watch instances of the CRs and to create matching workloads.

**Note**

A cluster contains two workload sets for each operator.

- The operator workload, which the OLM manages
- The workloads that are associated with the custom resources, and which the operator manages

You can implement operators to automate any manual Kubernetes task that fits the operator pattern. You can use most software development platforms to create operators. The following SDKs provide components and frameworks to help with developing operators:

The Operator SDK

The Operator SDK contains tools to develop operators with the Go programming language, and Ansible. The Operator SDK also contains tools to package Helm charts as operators.

The Java Operator SDK

The Java Operator SDK contains tools to develop operators with the Java programming language. The Java Operator SDK has a Quarkus extension to develop operators with the Quarkus framework.



References

For more information, refer to the *Operators* guide in the Red Hat OpenShift Container Platform 4.14 documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index

Operator SDK

<https://sdk.operatorframework.io/>

Java Operator SDK

<https://javaoperatorsdk.io/>

Quarkus Operator SDK

<https://github.com/quarkiverse/quarkus-operator-sdk>

► Quiz

Kubernetes Operators and the Operator Lifecycle Manager

Match the following items to their counterparts in the table.

A component that manages add-on operators

A component that manages cluster operators

A component that provides cluster platform services

A component that the Operator Lifecycle Manager installs

A way to implement reusable software to manage complex workloads

Term	Definition
Operator pattern	
Cluster operator	
Add-on operator	
Cluster Version Operator	
Operator Lifecycle Manager	

► Solution

Kubernetes Operators and the Operator Lifecycle Manager

Match the following items to their counterparts in the table.

Term	Definition
Operator pattern	A way to implement reusable software to manage complex workloads
Cluster operator	A component that provides cluster platform services
Add-on operator	A component that the Operator Lifecycle Manager installs
Cluster Version Operator	A component that manages cluster operators
Operator Lifecycle Manager	A component that manages add-on operators

Install Operators with the Web Console

Objectives

- Install and update operators by using the web console and the Operator Lifecycle Manager.

Installing Operators with the Web Console

The OpenShift web console provides a graphical interface to the Operator Lifecycle Manager (OLM). The **OperatorHub** page lists available operators and provides an interface for installing them. The **Installed Operators** page lists installed operators. You can inspect and uninstall operators from the **Installed Operators** page.

The Install Operator Wizard

Navigate to Operators > OperatorHub to display the list of available operators. The **OperatorHub** page displays operators, and has filters to locate operators by category, source, provider, subscription required, and other criteria.

The screenshot shows the Red Hat OpenShift web console interface. On the left is a sidebar with a navigation menu. The 'Operators' section is expanded, showing 'OperatorHub' as the active tab. Other tabs include 'Installed Operators', 'Workloads', 'Networking', 'Storage', 'Builds', 'Observe', and 'Compute'. The main content area is titled 'OperatorHub' and contains a search bar with the placeholder 'Filter by keyword...'. Below the search bar, there are two sections: 'All Items' and 'All Items'. Under 'All Items', there are two cards: 'do280 Operator Catalog Red Hat' (Compliance Operator) and 'do280 Operator Catalog Red Hat' (DevWorkspace Operator). Under 'Source', there is a list of categories: 'do280 Operator Catalog Red Hat (6)', 'Red Hat (5)', and 'Devfile (1)'. Under 'Provider', there is a list of categories: 'Red Hat (5)' and 'Devfile (1)'. At the bottom of the page, there are two more cards: 'do280 Operator Catalog Red Hat' (Compliance Operator) and 'do280 Operator Catalog Red Hat' (DevWorkspace Operator).

Figure 7.1: Available operators

Click an operator to display further information.



Important

Before installing an operator, review the operator information and consult the operator documentation. You might need to configure the operator further for successful deployment.

Click **Install** to begin the **Install Operator** wizard.

You can choose installation options in the **Install Operator** wizard.

Update channel

You can choose the most suitable operator update channel for your requirements. For more information, refer to *Operator Update Channels*.

Installation mode

The default **All namespaces on the cluster (default)** installation mode should be suitable for most operators. This mode configures the operator to monitor all namespaces for custom resources.

For example, an operator that deploys database servers defines a custom resource that describes a database server. When using the **All namespaces on the cluster (default)** installation mode, users can create those custom resources in their namespaces. Then, the operator deploys database servers in the same namespaces, along with other user workloads.

Cluster administrators can combine this mode with self-service features and other namespace-based features, such as role-based access control and network policies, to control user usage of operators.

Installed namespace

The OLM installs the operator workload to the selected namespace in this option. Some operators install by default to the `openshift-operators` namespace. Other operators suggest creating a namespace.

Although users might require access to the workloads that the operator manages, typically only cluster administrators require access to the operator workload.

Update approval

The OLM updates operators automatically when new versions are available. Choose manual updates to prevent automatic updates.

For an operator that includes monitoring in its definition, the wizard displays a further option to enable the monitoring. Adding monitoring from non-Red Hat operators is not supported.

The installation mode and installed namespace options are related. Review the documentation of the operator to learn the supported options.

After you configure the installation, click **Install**. The web console creates subscription and operator group resources according to the selected options in the wizard. After the installation starts, the web console displays progress information.

Viewing Installed Operators

When the OLM finishes installing an operator, click **View Operator** to display the **Operator details** page. You can also view information about installed operators by navigating to **Operators > Installed Operators**.

The **Installed Operators** page lists the installed cluster service version (CSV) resources that correspond to installed operators.

Every version of an operator has a CSV. The OLM uses information from the CSV to install the operator. The OLM updates the **status** key of the CSV with installation information.

CSVs are namespaced, so the **Installed Operator** page has a similar namespace filter to other web console pages. Operators that were installed with the "all namespaces" mode have a CSV in all namespaces.



Note

The operator installation mode determines which namespaces the operator monitors for custom resources. This mode is a distinct option from the installed namespace option, which determines the operator workload namespace.

The **Installed Operators** page shows information such as the operator status and available updates. Click an operator to navigate to the **Operator details** page.

The **Operator details** page contains the following tabs, where you can view further details and perform other actions.

Details

Displays information about the CSV.

YAML

Displays the CSV in YAML format.

Subscription

In this tab, you can change installation options, such as the update channel and update approval. This tab also links to the install plans of the operator. When you configure an operator for manual updates, you approve install plans for updates in this tab.

Events

Lists events that are related to the operator.

The **Operator details** page also has tabs for custom resources. For each custom resource that the operator defines, a web console tab lists all resources of that type. Additionally, the **All instances** tab aggregates all resources of types that the operator defines.

Using Operators

Custom resources are the most common way to interact with operators. You can create custom resources by using the custom resource tabs on the **Installed Operators** page. Select the tab to correspond to the custom resource type to create, and then click the create button.

Custom resources use the same creation page as other Kubernetes resources. You can choose either the YAML view or the form view to configure the new resource.

In the YAML view, you use the YAML editor to compose the custom resource. The editor provides a starting template that you can customize. The YAML view also displays documentation about the custom resource schema. The `oc explain` command provides the same documentation.

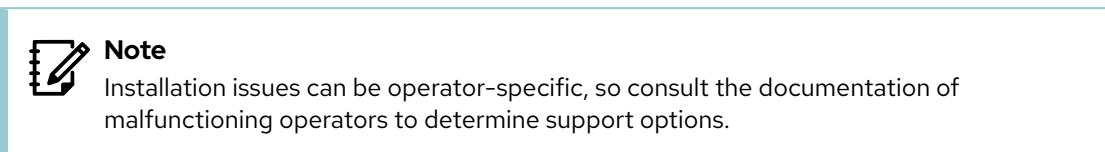
The form view presents a set of fields in a resource. Instead of composing a full YAML definition, you can edit the fields individually. When complete, OpenShift creates a resource from the values in the form.

Fields might provide help text and further configuration help. For example, fields with a limited set of values might provide a drop-down list with the possible values. The form view might provide more guidance, but might not contain fields to customize all possible options of a custom resource.

Troubleshooting Operators

The OLM might fail to install or update operators, or operators might not work correctly.

To identify operator installation issues, examine the status and conditions of the CSV, subscription, and install plan resources.



To troubleshoot further issues that cause operators to work incorrectly, first identify the operator workload. The **Operator Deployments** field in the **Operator details** page shows operator workloads. Operators might create further workloads, including workloads that follow the definitions that you provide in custom resources.

Project: openshift-operators ▾

Installed Operators > Operator details

Web Terminal
1.6.0 provided by Red Hat Actions ▾

Details YAML Subscription Events

Provided APIs
No Kubernetes APIs are being provided by this Operator.

Description
Start a Web Terminal in your browser with common CLI tools for interacting with the cluster.
...

ClusterServiceVersion details

Name web-terminal.v1.6.0	Status ✓ Succeeded
Namespace NS openshift-operators	Status reason install strategy completed with no errors
Labels operator...	Operator Deployments D web-terminal-controller

Figure 7.2: Operator deployments

Identify and troubleshoot the operator workload as with any other Kubernetes workload. The following resources are common starting points when troubleshooting:

- The status of Kubernetes workload resources, such as deployments or stateful sets
- Pod logs and their status
- Events



References

For more information, refer to the *Installing from OperatorHub Using the Web Console* section in the *Administrator Tasks* chapter in the Red Hat OpenShift Container Platform 4.14 Operators documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-installing-from-operatorhub-using-web-console_olm-adding-operators-to-a-cluster

For more information about monitoring configuration, refer to the *Maintenance and Support for Monitoring* section in the *Configuring the Monitoring Stack* chapter in the Red Hat OpenShift Container Platform 4.14 Monitoring documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-installing-from-operatorhub-using-web-console_olm-adding-operators-to-a-cluster

► Guided Exercise

Install Operators with the Web Console

Install an operator by using the web console.

Outcomes

- Install and uninstall an operator with the web console.
- Examine the resources that the web console creates for the installation, and the operator workloads.

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.

```
[student@workstation ~]$ lab start operators-web
```

Instructions

► 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 redhatocp \
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 **Red Hat Identity Management** and log in as the **admin** user with the **redhatocp** password.

► 2. Install the File Integrity operator.

- 2.1. Click **Operators > OperatorHub**. In the **Filter by keyword** field, type **integrity** to locate the File Integrity operator, and then click **File Integrity Operator**.

The screenshot shows the Red Hat OpenShift web console's OperatorHub. The left sidebar is collapsed, and the main area is titled "OperatorHub". A search bar at the top right contains the text "Integrity". Below the search bar, the results are displayed under the heading "All Items". A single result, "File Integrity Operator", is shown in a card. The card includes a thumbnail icon, the name "File Integrity Operator", the provider "do280 Operator Catalog Cs", and a brief description: "An operator that manages file integrity checks on nodes." A red box highlights this card.

- 2.2. The web console displays information about the File Integrity operator. Click **Install** to proceed to the **Install Operator** page.

The screenshot shows the "File Integrity Operator" install page. The title is "File Integrity Operator" with a subtext "1.33 provided by Red Hat". A large blue "Install" button is highlighted with a red box. Below the button, there are sections for "Channel" (set to "stable"), "Version" (set to "1.33"), and "Capability level" (which includes "Basic Install" and "Seamless Upgrades" checked). There are also sections for "Source" (set to "do280 Operator Catalog Cs") and "Provider".

- 2.3. The **Install Operator** page contains installation options. You can use the default options.

The lab environment cluster is a disconnected cluster to ensure that exercises are reproducible. The Operator Lifecycle Manager is configured to use a mirror registry with only the required operators for the course. In this registry, the File Integrity operator has a single available update channel.

By default, the File Integrity operator installs to all namespaces and creates the `openshift-file-integrity` namespace. The operator workload resides in this namespace.

Do not enable monitoring, which this exercise does not cover.

The screenshot shows the Red Hat OpenShift web console. The left sidebar has a dropdown for 'Administrator'. Under 'Operators', 'OperatorHub' is selected. The main content area is titled 'OperatorHub > Operator Installation' and shows the 'Install Operator' page for the 'File Integrity Operator'. The operator is provided by Red Hat. The 'Update channel' is set to 'stable' and the 'Version' is '1.3.3'. The 'Installation mode' is set to 'All namespaces on the cluster (default)'. In the 'Installed Namespace' section, 'operator recommended Namespace: openshift-file-integrity' is selected. A note at the bottom says 'Namespace creation' and 'Namespace openshift-file-integrity does not exist and will be created.' The top right of the screen shows a notification count of 2 and the user 'Administrator'.

For more information about the File Integrity operator, refer to the *File Integrity Operator* chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/security_and_compliance/index#file-integrity-operator-release-notes

2.4. Click **Install** to install the operator.

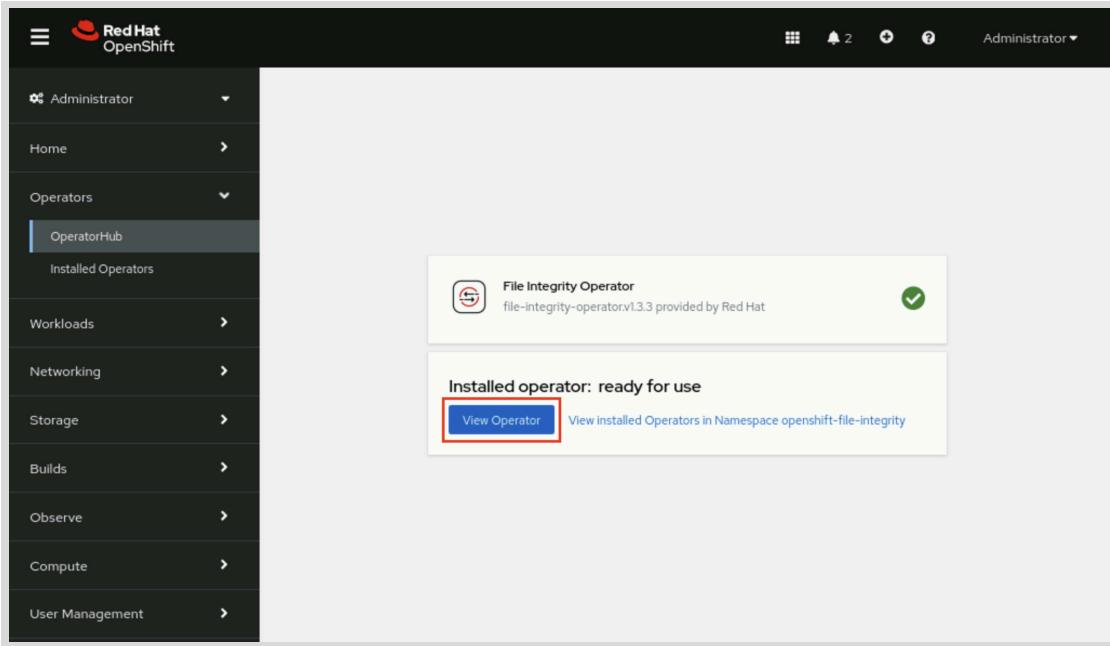
The web console displays some progress information. Click **View Operator**.



Note

The web console might display the **View Operator** button briefly before the OLM finishes the installation. The web console can also display errors briefly.

Wait until the web console displays **View Operator** for more than a few seconds.



The screenshot shows the Red Hat OpenShift web interface. The left sidebar has a dark theme with white text. The 'Operators' section is expanded, and 'OperatorHub' is selected. In the main content area, a card for the 'File Integrity Operator' is displayed. The card includes the operator's icon, name, version ('file-integrity-operator.v1.3.3'), provider ('Red Hat'), and a green checkmark icon. Below the card, a message says 'Installed operator: ready for use'. Underneath this message are two buttons: a blue 'View Operator' button with a red border, and a smaller blue 'View installed Operators in Namespace openshift-file-integrity' button.

Ensure that the `openshift-file-integrity` project is selected in the Project list.

- 3. The web console displays details about the installed operator.

The **Details** tab displays information about the operator and the related cluster service version resource.

The screenshot shows the Red Hat OpenShift web interface. The left sidebar is titled "Administrator" and includes links for Home, Operators (which is selected), Workloads, Networking, Storage, Builds, Observe, Compute, User Management, and Administration. The main content area is titled "Project: openshift-file-integrity". It shows the "Installed Operators" section with "File Integrity Operator" listed as "1.3.3 provided by Red Hat". The "Operator details" tab is selected. Below it, there are tabs for Details, YAML, Subscription, Events, All instances, File Integrity, and FileIntegrityNodeStatus. The "Details" tab is active. Under "Provided APIs", there are two entries: "File Integrity" and "FileIntegrityNodeStatus". Each entry has a description, a "Create instance" button, and a "Provider" field set to "Red Hat". To the right of these, there are fields for "Created at" (Jan 25, 2024, 9:27 AM) and "Links" (File Integrity Operator upstream repo: https://github.com/openshift/file-integrity-operator). Below this, there is a "Description" section with the text: "An operator that manages file integrity checks on nodes." At the bottom right, there is a link to "File Integrity Operator documentation" with the URL https://docs.openshift.com/container-platform/latest/security/file_integrity_operator/file-integrity-operator-understanding.html.

Scroll down to the **Conditions** section to review the evolution of the installation process. The last condition is for the **Succeeded** phase, because the installation completed correctly.

The **YAML** tab displays the cluster service version resource API resource in YAML format.

Click the **Subscription** tab to view information about the operator subscription resource. In this tab, you can change the update channel and the update approval configuration. The tab also links to the install plan. The install plan further describes the operator installation process. When the OLM finds an update for an operator that is configured for manual updates, then the OLM creates an install plan for the update. You approve the update in the install plan details page.

- 4. Optionally, test the File Integrity operator.

The File Integrity operator watches resources of the **FileIntegrity** type. When you create a file integrity resource, the operator creates a workload that verifies the file integrity of nodes. The results of the verification are presented as resources of the **FileIntegrityNodeStatus** type.

- 4.1. Click the **File Integrity** tab, and click **Create FileIntegrity**.

The screenshot shows the Red Hat OpenShift web interface. The left sidebar is titled 'Administrator' and includes links for Home, Operators, Workloads, Networking, Storage, Builds, Observe, Compute, User Management, and Administration. The 'Operators' link is currently selected. The main content area is titled 'File Integrity Operator' and shows the 'File Integrity' tab is active. Below the tabs are buttons for Details, YAML, Subscription, Events, All instances, and File Integrity. A red box highlights the 'Create FileIntegrity' button.

- 4.2. Use YAML view and modify the gracePeriod to 60. Then, click Create to create a file integrity resource.

The screenshot shows the 'Create FileIntegrity' dialog in the Red Hat OpenShift web console. The left sidebar shows the 'Installed Operators' section is selected. The main area has a title 'Create FileIntegrity' and instructions to enter YAML or JSON. It shows 'Configure via:' options for 'Form view' (radio button) and 'YAML view' (radio button, selected). The YAML editor contains a configuration file with a 'gracePeriod' field highlighted with a red box. At the bottom are 'Create' and 'Cancel' buttons, with 'Create' highlighted by a red box. To the right is a 'FileIntegrity' schema panel with tabs for 'Schema' and 'FileIntegrity'. The schema definition is provided:

```

apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: example-fileintegrity
  namespace: openshift-file-integrity
spec:
  config:
    gracePeriod: 60
    maxBackups: 5
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      operator: Exists
    debug: false

```

- 4.3. Click the FileIntegrityNodeStatus tab. After a few minutes, the list shows a new example-fileintegrity-master01 resource.

The screenshot shows the Red Hat OpenShift web console interface. The left sidebar is titled 'Administrator' and includes links for Home, Operators, Workloads, Networking, Storage, Builds, Observe, Compute, User Management, and Administration. The main content area is titled 'Project: openshift-file-integrity'. Under 'Installed Operators', the 'File Integrity Operator' is listed as '1.3.3 provided by Red Hat'. The 'FileIntegrityNodeStatus' tab is selected, showing a table with one row:

Name	Kind	Namespace	Status	Labels
FINS example-fileintegrity-master01	FileIntegrityNodeStatus	openshift-file-integrity	-	file-integrity... -example...



Note

The first file integrity resource that you create might not work correctly.

If the operator does not create the `FileIntegrityNodeStatus` resource in a few minutes, then delete and create again the `FileIntegrity` resource.

The exercise outcome does not depend on obtaining a `FileIntegrityNodeStatus` resource.

- 4.4. After `FileIntegrityNodeStatus` has successfully been created, run this as the `admin` user to modify the node's filesystem: `oc debug node/master01 -- touch /host/etc/foobar`

```
[student@workstation ~]$ oc debug node/master01 -- touch /host/etc/foobar
Starting pod/master01-debug-l92pd ...
To use host binaries, `run chroot /host`

Removing debug pod ...
```

- 4.5. Click `Workloads > ConfigMaps` to list configmaps in the `openshift-file-integrity` namespace.

The screenshot shows the Red Hat OpenShift web interface. The left sidebar is collapsed. The main header says "Red Hat OpenShift". The top right has icons for notifications (3), help, and administrator settings. The title bar says "Project: openshift-file-integrity". The main content area is titled "ConfigMaps" with a "Create ConfigMap" button. A search bar at the top of the list says "Search by name...". The table lists several ConfigMaps:

Name	Size	Created	Actions
CM aide-example-fileintegrity-master01-failed	1	Jan 25, 2024, 12:08 PM	...
CM aide-pause	1	Jan 25, 2024, 11:39 AM	...
CM aide-reinit	1	Jan 25, 2024, 11:39 AM	...
CM example-fileintegrity	1	Jan 25, 2024, 11:39 AM	...
CM kube-root-ca.crt	1	Jan 25, 2024, 11:38 AM	...
CM openshift-service-ca.crt	1	Jan 25, 2024, 11:38 AM	...

The sidebar on the left has sections for Home, Operators (with OperatorHub and Installed Operators), Workloads (with Pods, Deployments, DeploymentConfigs, StatefulSets, Secrets, and ConfigMaps selected), CronJobs, Jobs, and DaemonSets.

- 4.6. Select **aide-example-fileintegrity-master01-failed** and view the report below Data

The screenshot shows the Red Hat OpenShift web interface. The left sidebar is collapsed. The main header says "Red Hat OpenShift". The top right has icons for notifications (3), help, and administrator settings. The title bar says "Project: openshift-file-integrity". The main content area shows the details of a selected ConfigMap:

- Owner:** No owner
- Data:**
 - integritylog:**

The integritylog content is as follows:

```

Start timestamp: 2024-01-25 17:16:58 +0000 (AIDE 0.16)
AIDE found differences between database and filesystem!!

Summary:
Total number of entries:      32358
Added entries:                1
Removed entries:              0
Changed entries:              0

-----
Added entries:
-----
f+++++:+ /hostroot/etc/foobar

-----
The attributes of the (uncompressed) database(s):
-----

```

The sidebar on the left has sections for Home, Operators (with OperatorHub and Installed Operators), Workloads (with Pods, Deployments, DeploymentConfigs, StatefulSets, Secrets, and ConfigMaps selected), CronJobs, Jobs, DaemonSets, ReplicaSets, ReplicationControllers, HorizontalPodAutoscalers, and PodDisruptionBudgets.

- 5. Examine and differentiate the File Integrity operator workloads from the operator-managed workloads.

- 5.1. Click **Workloads > Deployments** to list deployments in the **openshift-file-integrity** namespace.

The **file-integrity-operator** deployment is the operator workload that the OLM creates. This deployment watches file integrity resources, and creates the workloads to verify file integrity.

- 5.2. Click **Workloads > DaemonSets** to list daemon sets in the **openshift-file-integrity** namespace.

If you create a file integrity resource, then the operator creates an **aide-example-fileintegrity** daemon set to verify file integrity.

- 6. Uninstall the File Integrity operator.

- 6.1. Click **Operators > Installed Operators**.

6.2. In the list of installed operators, click **File Integrity Operator**.

6.3. Select **Uninstall Operator** from the Actions list, and then click **Uninstall**.

The screenshot shows the Red Hat OpenShift web interface. On the left is a navigation sidebar with options like Home, Operators, Workloads, Networking, Storage, Builds, Observe, Compute, User Management, and Administration. The main content area is titled 'File Integrity Operator' under 'Installed Operators'. It shows the operator version as 1.3.3 provided by Red Hat. The 'Actions' dropdown menu is open, and the 'Uninstall Operator' option is highlighted with a red box. Other actions shown in the dropdown include 'Edit Subscription'. Below the operator details, there are sections for 'Provided APIs' (File Integrity and FileIntegrityNodeStatus), 'Description' (An operator that manages file integrity checks on nodes), and links to 'Provider' (Red Hat) and 'Created at' (Jan 25, 2024, 9:27 AM). There are also 'Links' to the upstream repository and documentation.

► 7. Delete the `openshift-file-integrity` namespace.

The OLM creates the `openshift-file-integrity` namespace when installing the File Integrity operator.

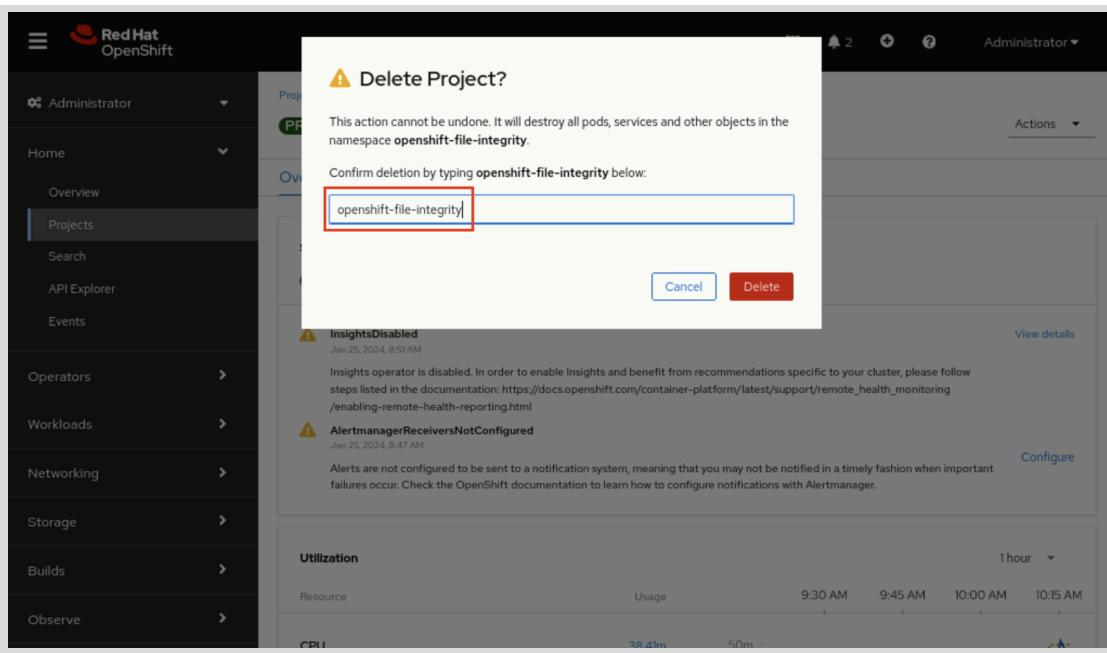
Before deleting an operator, always review the operator documentation to learn specific deletion actions.

7.1. Click **Home > Projects**.

7.2. Type **integrity** in the **Name** filter field.

7.3. Click **openshift-file-integrity**.

7.4. Select **Delete Project** from the **Actions** list. Then, type `openshift-file-integrity` and click **Delete**.



Finish

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

```
[student@workstation ~]$ lab finish operators-web
```

Install Operators with the CLI

Objectives

- Install and update operators by using the Operator Lifecycle Manager APIs.

Installing Operators

To install an operator, you must perform the following steps:

- Locate the operator to install.
- Review the operator and its documentation for installation options and requirements.
 - Decide the update channel to use.
 - Decide the installation mode. For most operators, you should make them available to all namespaces.
 - Decide to deploy the operator workload to an existing namespace or to a new namespace.
 - Decide whether the Operator Lifecycle Manager (OLM) applies updates automatically, or requires an administrator to approve updates.
- Create an operator group if needed for the installation mode.
- Create a namespace for the operator workload if needed.
- Create the operator subscription.
- Review and test the operator installation.

Operator Resources

The OLM uses the following resource types:

Catalog source

Each catalog source resource references an operator repository. Periodically, the OLM examines the catalog sources in the cluster and retrieves information about the operators in each source.

Package manifest

The OLM creates a package manifest for each available operator. The package manifest contains the required information to install an operator, such as the available channels.

Operator group

Operator groups define how the OLM presents operators across namespaces.

Subscription

Cluster administrators create subscriptions to install operators.

Operator

The OLM creates operator resources to store information about installed operators.

Install plan

The OLM creates install plan resources as part of the installation and update process. When requiring approvals, administrators must approve install plans.

Cluster service version (CSV)

Each version of an operator has a corresponding CSV. The CSV contains the information that the OLM requires to install the operator.

When installing an operator, an administrator must create only the subscription and the operator group. The OLM generates all other resources automatically.

Examining Available Operators

Examine catalog sources in the openshift-marketplace namespace to know which catalog sources are available in a cluster.

```
[user@host ~]$ oc get catalogsource -n openshift-marketplace
NAME          DISPLAY           TYPE    ... AGE
do280-catalog-cs   do280 Operator Catalog Cs  grpc   ... 7d6h
```

The OLM creates a package manifest for each available operator that a catalog source references. List the package manifests to know which operators are available for installation.

```
[user@host ~]$ oc get packagemanifests
NAME          CATALOG           AGE
lvms-operator  do280 Operator Catalog Cs  7d6h
kubevirt-hyperconverged  do280 Operator Catalog Cs  7d6h
file-integrity-operator  do280 Operator Catalog Cs  7d6h
compliance-operator    do280 Operator Catalog Cs  7d6h
metallb-operator      do280 Operator Catalog Cs  7d6h
```

To gather the required information to install an operator, view the details of a specific package manifest. Use the `oc describe` command on a package manifest to view details about an operator.

```
[user@host ~]$ oc describe packagemanifest lvms-operator -n openshift-marketplace
Name:          lvms-operator
...output omitted...
Spec:
Status:
  Catalog Source:      do280-catalog-cs ①
  Catalog Source Display Name: do280 Operator Catalog Cs
  Catalog Source Namespace:   openshift-marketplace ②
  Catalog Source Publisher:
  Channels:
    Current CSV:  lvms-operator.v4.14.1 ③
    Current CSV Desc:
      Annotations:
        ...output omitted...
      Capabilities:          Seamless Upgrades
      Categories:            Storage
      Container Image:       registry.redhat.io/lvms4/lvms-
rhel9-operator@sha256:545a...67e9
```

```

Description: Logical volume manager storage
provides dynamically provisioned local storage for container workloads
...output omitted...
operatorframework.io/suggested-namespace: openshift-storage
operators.openshift.io/infrastructure-features: ["csi", "disconnected"]
operators.openshift.io/valid-subscription: ["OpenShift Container
Platform", "OpenShift Platform Plus"]
operators.operatorframework.io/builder: operator-sdk-v1.23.0
...output omitted...
Apiservicedefinitions:
Customresourcedefinitions:
Owned:
Kind: LogicalVolume
Name: logicalvolumes.topolvm.io
Version: v1
Description: LVMCluster is the Schema for the lvmclusters API
Display Name: LVMCluster
Kind: LVMCluster
Name: lvmclusters.lvm.topolvm.io
Version: v1alpha1
Kind: LVMVolumeGroupNodeStatus
Name: lvmvolumegroupnodestatuses.lvm.topolvm.io
Version: v1alpha1
Kind: LVMVolumeGroup
Name: lvmvolumegroups.lvm.topolvm.io
Version: v1alpha1
Description: Logical volume manager storage provides dynamically
provisioned local storage. ④
Display Name: LVM Storage
Install Modes: ⑤
Supported: true
Type: OwnNamespace
Supported: true
Type: SingleNamespace
Supported: false
Type: MultiNamespace
Supported: false
Type: AllNamespaces
...output omitted...
Links: ⑥
Name: Source Repository
URL: https://github.com/openshift/lvm-operator
...output omitted...
Maturity: alpha
Provider:
Name: Red Hat
...output omitted...
Version: 4.14.1
Name: stable-4.14 ⑦
Default Channel: stable-4.14
Package Name: lvms-operator
Provider:
Name: Red Hat
Events: <none>

```

- ❶❷ The catalog source and namespace for the operator, which are required to identify the operator when creating the subscription.
- ❸❹ Examine the available channels and CSVs to decide which upgrade path to use.
- ❺❻ The description and links provide useful information and documentation for installation and uninstallation procedures.
- ❽ The install modes provide information about supported namespace operation modes.

Installing Operators

After you examine the package manifest, review the operator documentation. Operators might require specific installation procedures.

If you decide to deploy the operator workload to a new namespace, then create the namespace. Many operators recommend to use the existing `openshift-operators` namespace, or require specific namespaces.

Determine whether you need to create an operator group. Operators use the operator group in their namespace. Operators monitor custom resources in the namespaces that the operator group targets.

The `openshift-operators` namespace contains a `global-operators` operator group. Operators that are installed in the `openshift-operators` namespace use this operator group and monitor all namespaces.

If the `global-operators` operator group is not suitable, then create another operator group. The following YAML definition describes the structure of an operator group:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: name
  namespace: namespace ❶
spec:
  targetNamespaces: ❷
    - namespace
```

- ❶ Operators follow the operator group in the namespace that they are deployed in.
- ❷ List the namespaces that the operator monitors for custom resources. You can also use the `spec.selector` field to select namespaces by using labels.

After creating the necessary namespaces or operator groups, you create a subscription. The following YAML file is an example of a subscription:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: lvms-operator
  namespace: openshift-storage ❶
spec:
  channel: stable-4.14 ❷
  name: lvms-operator ❸
```

```
source: do280-catalog-cs ④
installPlanApproval: Automatic ⑤
sourceNamespace: openshift-marketplace
```

- ①** The namespace for the operator workload
- ②** The update channel, from the discovered information from the `oc describe packagemanifest` command
- ③** The package manifest to subscribe to
- ④** The source catalog, from the discovered information from the `oc describe packagemanifest` command
- ⑤** The install plan approval mode, either `Automatic` or `Manual`

Install Plans

The OLM creates an install plan resource to represent the required process to install or update an operator. The OLM updates the operator resource to reference the install plan in the `status.components.refs` field. You can view the reference by using the `oc describe` command on the operator resource.

```
[user@host ~]$ oc describe operator file-integrity-operator
Name:           file-integrity-operator.openshift-file-integrity
Namespace:
Labels:         <none>
Annotations:   <none>
API Version:  operators.coreos.com/v1
Kind:          Operator
...output omitted...
Status:
Components:
...output omitted...
Refs:
  API Version: operators.coreos.com/v1alpha1
  Kind:        InstallPlan
  Name:        install-pmh78
  Namespace:   openshift-file-integrity
  API Version: operators.coreos.com/v1alpha1
Conditions:
  Last Transition Time: 2024-01-26T17:53:27Z
  Message:            all available catalogsources are healthy
  Reason:             AllCatalogSourcesHealthy
  Status:              False
  Type:                CatalogSourcesUnhealthy
  Last Transition Time: 2024-01-26T17:53:49Z
  Reason:             RequiresApproval
  Status:              True
  Type:                InstallPlanPending
  Kind:                Subscription
  Name:                file-integrity-operator
  Namespace:          openshift-file-integrity
Events:
```

If the install plan mode is set to `Manual` in the subscription, then you must manually approve the install plan. To approve an install plan, change the `spec.approved` field to `true`. For example, you can use the `oc patch` command to approve an install plan:

```
[user@host ~]$ oc patch installplan install-pmh78 --type merge -p \
'{"spec":{"approved":true}}' -n openshift-file-integrity
installplan.operators.coreos.com/install-pmh78 patched
```

With an `Automatic` install plan mode, the OLM applies updates as soon as they are available.

Using Operators

Typically, operators create custom resource definitions. You create instances of those custom resources to use the operator. Review the operator documentation to learn how to use an operator.

Additionally, you can learn about the available custom resource definitions by examining the operator. The CSV contains a list of the custom resource definitions in the `spec.customresourcedefinitions` field. For example, use the following command to list the custom resource definitions:

```
[user@host ~]$ oc get csv metallb-operator.v4.14.0-202401151553 \
-o jsonpath=".spec.customresourcedefinitions.owned[*].name}{'\n'}"
addresspools.metallb.io addresspools.metallb.io bfdprofiles.metallb.io
bgpadvertisements.metallb.io bgppeers.metallb.io bgppeers.metallb.io
communities.metallb.io ipaddresspools.metallb.io l2advertisements.metallb.io
metallbs.metallb.io
```

You can also use the `oc explain` command to view the description of individual custom resource definitions.

Troubleshooting Operators

Some operators require additional steps to install or update. Review the documentation to validate whether you performed all necessary steps, and to learn about support options.

You can examine the status of the operator, install plan, and CSV resources. When installing or updating operators, the OLM updates those resources with progress information.

Even if the OLM installs an operator correctly, the operator might not function correctly.

Operators typically contain two kinds of workloads:

- The operator workload, which monitors custom resources.
- The workloads that individual instances of the custom resources created.

The `spec.install.spec.deployments` in the CSV contains the deployments that the OLM creates when installing an operator. These deployments often correspond to the operator workload. However, the operator might create further deployments either for its own workload, or for the workloads that are associated with custom resources.



References

For more information about operators, refer to the *Operators Overview* chapter in the Red Hat OpenShift Container Platform 4.14 *Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#operators-overview

For more information about installing operators, refer to the *Installing from OperatorHub Using the CLI* section in the *Administrator Tasks* chapter in the Red Hat OpenShift Container Platform 4.14 *Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-installing-operator-from-operatorhub-using-cli_olm-adding-operators-to-a-cluster

For more information about operator groups, refer to the *Operator Groups* section in the *Understanding Operators* chapter in the Red Hat OpenShift Container Platform 4.14 *Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-operatorgroups-about_olm-understanding-olm

► Guided Exercise

Install Operators with the CLI

Install an operator by using the command-line interface and Kubernetes manifests.

Outcomes

- Install operators from the CLI with manual updates.

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 is ready, and removes the `openshift-file-integrity` namespace and File Integrity operator if they exist.

```
[student@workstation ~]$ lab start operators-cli
```

Instructions

In this exercise, you install the File Integrity operator with manual updates. The documentation of the File Integrity operator contains specific installation instructions.

For more information, refer to the *Installing the File Integrity Operator Using the CLI* section in the *File Integrity Operator* chapter in the Red Hat OpenShift Container Platform 4.14 *Security and Compliance* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/security_and_compliance/index#installing-file-integrity-operator-using-cli_file-integrity-operator-installation

- 1. Log in to the OpenShift cluster as the `admin` user with the `redhatocp` password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 2. Find the details of the File Integrity operator within the OpenShift package manifests.

- 2.1. View the available operators within the OpenShift Marketplace by using the `oc get` command.

```
[student@workstation ~]$ oc get packagemanifests
NAME          CATALOG          AGE
file-integrity-operator do280 Operator Catalog Cs 37h
lvms-operator      do280 Operator Catalog Cs 37h
compliance-operator do280 Operator Catalog Cs 37h
metallb-operator   do280 Operator Catalog Cs 37h
kubevirt-hyperconverged do280 Operator Catalog Cs 37h
```

- 2.2. Examine the File Integrity operator package manifest by using the `oc describe` command.

```
[student@workstation ~]$ oc describe packagemanifest file-integrity-operator
Name:           file-integrity-operator
...output omitted...
Spec:
Status:
  Catalog Source:      do280-catalog-cs
  Catalog Source Display Name: do280 Operator Catalog Cs
  Catalog Source Namespace:  openshift-marketplace
  Catalog Source Publisher:
  Channels:
  ...output omitted...
  Install Modes:
    Supported: true
    Type:      OwnNamespace
    Supported: true
    Type:      SingleNamespace
    Supported: false
    Type:      MultiNamespace
    Supported: true
    Type:      AllNamespaces
  ...output omitted...
  Name:        stable
  Default Channel: stable
  Package Name:  file-integrity-operator
  ...output omitted...
```

The operator is in the `do280-catalog-cs` catalog source in the `openshift-marketplace` namespace. The operator has a single channel with the `stable` name. The operator has the `file-integrity-operator` name.

- 3. Install the File Integrity operator. By following the operator installation instructions, you must install the operator in the `openshift-file-integrity` namespace. Also, you must make the operator available only in that namespace. The File Integrity operator requires you to create a namespace with specific labels.
- 3.1. The operator documentation provides a YAML definition of the required namespace. The definition is available in the `~/D0280/labs/operators-cli/namespace.yaml` path. Examine the definition and create the namespace.

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged
  name: openshift-file-integrity
```

```
[student@workstation ~]$ oc create -f ~/D0280/labs/operators-cli/namespace.yaml
namespace/openshift-file-integrity created
```

- 3.2. Create an operator group in the operator namespace. The operator group targets the same namespace. You can use the template in the `~/D0280/labs/operators-cli/operator-group.yaml` path. Edit the file and configure the namespaces.

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  targetNamespaces:
    - openshift-file-integrity
```

Create the operator group.

```
[student@workstation ~]$ oc create \
-f ~/D0280/labs/operators-cli/operator-group.yaml
operatorgroupoperators.coreos.com/file-integrity-operator created
```

- 3.3. Create the subscription in the operator namespace. You can use the template in the `~/D0280/labs/operators-cli/subscription.yaml` path. Edit the file with the data that you obtained in a previous step. Set the approval policy to **Manual**.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  channel: "stable"
  installPlanApproval: Manual
  name: file-integrity-operator
  source: do280-catalog-cs
  sourceNamespace: openshift-marketplace
```

Create the subscription.

```
[student@workstation ~]$ oc create -f ~/D0280/labs/operators-cli/subscription.yaml
subscriptionoperators.coreos.com/file-integrity-operator created
```

► 4. Approve the install plan.

- 4.1. Examine the operator resource that the OLM created.

```
[student@workstation ~]$ oc describe operator file-integrity-operator
Name:           file-integrity-operator.openshift-file-integrity
...output omitted...
Status:
Components:
  Label Selector:
    Match Expressions:
      Key:      operators.coreos.com/file-integrity-operator.openshift-file-
integrity
      Operator: Exists
  Refs:
...output omitted...
  Kind:          InstallPlan
  Name:          install-4wsq6
  Namespace:    openshift-file-integrity
  API Version: operators.coreos.com/v1alpha1
  Conditions:
    Last Transition Time: 2024-01-26T10:38:22Z
    Message:             all available catalogsources are healthy
    Reason:              AllCatalogSourcesHealthy
    Status:               False
    Type:                CatalogSourcesUnhealthy
    Last Transition Time: 2024-01-26T10:38:21Z
    Reason:              RequiresApproval
    Status:               True
    Type:                InstallPlanPending
  Kind:          Subscription
  Name:          file-integrity-operator
  Namespace:    openshift-file-integrity
Events:
```

Verify that the operator has a condition of the `InstallPlanPending` type. The operator can have other conditions, and they do not indicate a problem. The operator references the install plan. You use the install plan name in a later step. If the install plan is not generated, then wait a few moments and run the `oc describe` command again.

- 4.2. View the install plan specification with the `oc get` command. Replace the name with the install plan name that you obtained in a previous step.

```
[student@workstation ~]$ oc get installplan -n openshift-file-integrity \
install-4wsq6 -o jsonpath='{.spec}{"\n"}'
{"approval":"Manual","approved":false,"clusterServiceVersionNames":["file-
integrity-operator.v1.3.3","file-integrity-operator.v1.3.3"],"generation":1}
```

The install plan is set to manual approval, and the `approved` field is set to false.

- 4.3. Approve the install plan with the `oc patch` command. Replace the name with the install plan name that you obtained in a previous step.

```
[student@workstation ~]$ oc patch installplan install-4wsq6 --type merge -p \
'{"spec":{"approved":true}}' -n openshift-file-integrity
installplan.operator.coreos.com/install-4wsq6 patched
```

- 4.4. Verify that the operator installs successfully, by using the `oc describe` command. Check the latest transaction for the current status. The installation might not complete immediately. If the installation is not complete, then wait a few minutes and view the status again.

```
[student@workstation ~]$ oc describe operator file-integrity-operator
...output omitted...
Status:
Components:
  Label Selector:
    Match Expressions:
      Key: operators.coreos.com/file-integrity-operator.openshift-file-
integrity
      Operator: Exists
  Refs:
    ...output omitted...
  Conditions:
    Last Transition Time: 2024-01-26T18:21:03Z
    Last Update Time: 2024-01-26T18:21:03Z
    Message: install strategy completed with no errors
    Reason: InstallSucceeded
    Status: True
    Type: Succeeded
  Kind: ClusterServiceVersion
  Name: file-integrity-operator.v1.0.0
  Namespace: openshift-file-integrity
  ...output omitted...
```

- 4.5. Examine the workloads in the `openshift-file-integrity` namespace.

```
[student@workstation ~]$ oc get all -n openshift-file-integrity
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+,
unavailable in v4.10000+
NAME                                     READY   STATUS    RESTARTS   AGE
pod/file-integrity-operator-6985588576-x2k49   1/1     Running   1 (50s ago)  56s
...output omitted...

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/file-integrity-operator   1/1     1           1           56s

NAME                                     DESIRED   CURRENT   READY
replicaset.apps/file-integrity-operator-6985588576   1         1         1
56s
```

The namespace has a ready deployment.

- 5. Test the operator to ensure that it is functional. The operator watches `FileIntegrity` resources, runs file integrity checks on nodes, and creates `FileIntegrityNodeStatus` with the results of the checks.
- 5.1. Create a `FileIntegrity` custom resource by applying the file at `~/D0280/labs/operators-cli/worker-fileintegrity.yaml` with the `oc apply` command.

```
[student@workstation ~]$ oc apply -f \
~/D0280/labs/operators-cli/worker-fileintegrity.yaml
fileintegrity.fileintegrity.openshift.io/worker-fileintegrity created
```

- 5.2. Verify that the operator functions, by viewing the `worker-fileintegrity` object with the `oc describe` command.

```
[student@workstation ~]$ oc describe fileintegrity worker-fileintegrity \
-n openshift-file-integrity
Name:           worker-fileintegrity
Namespace:      openshift-file-integrity
Labels:          <none>
Annotations:    <none>
API Version:   fileintegrity.openshift.io/v1alpha1
Kind:           FileIntegrity
...output omitted...
Spec:
Config:
Grace Period:  900
Max Backups:   5
Node Selector:
node-role.kubernetes.io/worker:
Tolerations:
Effect:        NoSchedule
Key:           node-role.kubernetes.io/master
Operator:      Exists
Effect:        NoSchedule
Key:           node-role.kubernetes.io/infra
Operator:      Exists
Events:        <none>
```

- 5.3. Use `oc edit` to edit the `Grace Period` to `60` in the `FileIntegrity` custom resource to trigger a failure.

```
[student@workstation ~]$ oc edit fileintegrity worker-fileintegrity \
-n openshift-file-integrity
Name:           worker-fileintegrity
Namespace:      openshift-file-integrity
Labels:          <none>
Annotations:    <none>
API Version:   fileintegrity.openshift.io/v1alpha1
Kind:           FileIntegrity
...output omitted...
Spec:
Config:
Grace Period:  60
Max Backups:   5
```

```

Node Selector:
  node-role.kubernetes.io/worker:
Tolerations:
  Effect:    NoSchedule
  Key:       node-role.kubernetes.io/master
  Operator:  Exists
  Effect:    NoSchedule
  Key:       node-role.kubernetes.io/infra
  Operator:  Exists
Events:   <none>

```

- 5.4. Verify that the operator automatically creates a `FileIntegrityNodeStatus` object, by using the `oc get` command. You might need to wait a few minutes for the object to generate.



Note

The first file integrity resource that you create might not work correctly.

If the operator does not create the `FileIntegrityNodeStatus` resource in a few minutes, then delete the `FileIntegrity` resource and create it again.

The exercise outcome does not depend on obtaining a `FileIntegrityNodeStatus` resource.

```
[student@workstation ~]$ oc get fileintegritynodestatuses \
-n openshift-file-integrity
NAME          NODE      STATUS
worker-fileintegrity-master01 master01 Succeeded
```

- 5.5. After `FileIntegrityNodeStatus` has successfully been created, run this as the admin user to modify the node's filesystem: `oc debug node/master01 -- touch /host/etc/foobar`

```
[student@workstation ~]$ oc debug node/master01 -- touch /host/etc/foobar
Starting pod/master01-debug-l92pd ...
To use host binaries, `run chroot /host`

Removing debug pod ...
```

- 5.6. Run `oc get configmaps -n openshift-file-integrity` to list configmaps in the `openshift-file-integrity` namespace.

```
[student@workstation ~]$ oc get configmaps -n openshift-file-integrity --watch
NAME          DATA  AGE
aide-pause        1   109m
aide-reinit       1   109m
aide-worker-fileintegrity-master01-failed  1   108m
kube-root-ca.crt 1   117m
openshift-service-ca.crt 1   117m
worker-fileintegrity 1   109m
```

**Note**

It may take several minutes for `aide-worker-fileintegrity-master01-failed` to show. Use the `--watch` flag and wait a few minutes until the failed configmap shows to move on to the next step. Press `Ctrl+C` to exit.

- 5.7. Run `oc describe` to view the report in `aide-worker-fileintegrity-master01-failed` configmap in the `openshift-file-integrity` namespace.

```
[student@workstation ~]$ oc describe \
  configmap/aide-worker-fileintegrity-master01-failed \
  -n openshift-file-integrity
Name:      aide-worker-fileintegrity-master01-failed
Namespace:  openshift-file-integrity
Labels:    file-integrity.openshift.io/node=master01
          file-integrity.openshift.io/owner=worker-fileintegrity
          file-integrity.openshift.io/result-log=
Annotations:  file-integrity.openshift.io/files-added: 1
              file-integrity.openshift.io/files-changed: 0
              file-integrity.openshift.io/files-removed: 0

Data
\-----
integritylog:
\-----

Start timestamp: 2024-01-26 18:31:16 +0000 (AIDE 0.16)
AIDE found differences between database and filesystem!!

Summary:
  Total number of entries: 32359
  Added entries:           1
  Removed entries:         0
  Changed entries:         0

-----
Added entries:
-----

f+++++++: /hostroot/etc/cni/multus/certs/multus-
client-2024-01-26-15-14-01.pem
f+++++++: /hostroot/etc/foobar

-----
The attributes of the (uncompressed) database(s):
-----

/hostroot/etc/kubernetes/aide.db.gz
MD5      : UswXQ1Va/Vpj1XF1rCP0vA==
SHA1     : s6t06MCRrDgc4x0WnX6vk5rfLGU=
RMD160   : jvDdvAOC7/tI0TjDe7Kzmy5nUk8=
TIGER    : TjW192YTQBmG4oGza7siI6CBRnztgrp6
SHA256   : E8rWurdI9HgGP6402qWY+lDAaLoGiNs
PEka/siI1F0=
```

```
SHA512    : JPDhgoEnNiTaDLqawkGtHplRW8f6zm3g  
           jDB3E6X6XM4+13yhjwh/pokFAp5BhRSC  
           0C4XXibXsS40YxYiE5hBaw==
```

```
End timestamp: 2024-01-26 18:31:45 +0000 (run time: 0m 29s)
```

```
BinaryData  
\====
```

```
Events: <none>
```

Finish

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

```
[student@workstation ~]$ lab finish operators-cli
```

► Lab

Manage Kubernetes Operators

Install an operator and verify that it is healthy.

Outcomes

- Install the Compliance operator on the command line.

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 the operator that is used in this exercise is not present.

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

Instructions

In this exercise, you install the Compliance operator. For more information, refer to the *Compliance Operator* chapter in the Red Hat OpenShift Container Platform 4.14 *Security and Compliance* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/security_and_compliance/index#co-overview.

- Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.
- Examine the package manifest for the Compliance operator to discover the operator name, catalog name, suggested namespace, and channel.
- Create the recommended `openshift-compliance` namespace.
- Create an operator group with the `compliance-operator` name in the `openshift-compliance` namespace. The target namespace of the operator group is the `openshift-compliance` namespace. You can use the `~/D0280/labs/operators-review/operator-group.yaml` file as a template.
- Create a `compliance-operator` subscription in the `openshift-compliance` namespace. The subscription has the following parameters:

Field	Value
channel	stable
spec.name	compliance-operator
source	do280-catalog-cs
sourceNamespace	openshift-marketplace

You can use the `~/DO280/labs/operators-review/subscription.yaml` file as a template.

You can configure automatic install plan approvals.

6. Wait until the operator is installed.

The Operator Lifecycle Manager creates a cluster service version in the `openshift-compliance` namespace. Wait until the cluster service version resource (CSV) is in the `Succeeded` phase.

Although the CSV defines a single `compliance-operator` deployment, the operator has two additional deployments. Wait until the `compliance-operator`, `ocp4-openshift-compliance-pp`, and `rhcose4-openshift-compliance-pp` deployments are ready.

7. Verify that the operator works correctly.

This operator watches custom resources of the `ScanSettingBinding` type and runs file integrity checks on cluster nodes. The operator reports results with custom resources of the `ComplianceSuite` type.

Create a scan setting binding in the `openshift-compliance` namespace. You can use the `~/DO280/labs/operators-review/scan-setting-binding.yaml` file as a template.

You can also use the web console to create the scan setting binding. The YAML editor in the web console provides the same scan setting binding resource as an example.

Wait until a resource of the `ComplianceSuite` type in the `DONE` phase is present in the `openshift-compliance` namespace.

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 grade operators-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Manage Kubernetes Operators

Install an operator and verify that it is healthy.

Outcomes

- Install the Compliance operator on the command line.

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 the operator that is used in this exercise is not present.

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

Instructions

In this exercise, you install the Compliance operator. For more information, refer to the *Compliance Operator* chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/security_and_compliance/index#co-overview.

- Log in to your OpenShift cluster as the `admin` user with the `redhatocp` password.

- Log in to the cluster as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
  https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- Examine the package manifest for the Compliance operator to discover the operator name, catalog name, suggested namespace, and channel.

- Use the `oc` command to list the package manifest resources.

```
[student@workstation ~]$ oc get packagemanifest
NAME          CATALOG          AGE
lvms-operator do280 Operator Catalog Cs 2d5h
file-integrity-operator do280 Operator Catalog Cs 2d5h
metallb-operator do280 Operator Catalog Cs 2d5h
compliance-operator do280 Operator Catalog Cs 2d5h
kubevirt-hyperconverged do280 Operator Catalog Cs 2d5h
```

- Examine the `compliance-operator` package manifest.

```
[student@workstation ~]$ oc get packagemanifest compliance-operator -o yaml
apiVersion: packages.coreos.com/v1
kind: PackageManifest
metadata:
  creationTimestamp: "2024-01-24T14:05:27Z"
  labels:
    catalog: do280-catalog-cs
    catalog-namespace: openshift-marketplace
  ...output omitted...
  name: compliance-operator
  namespace: default
spec: {}
status:
  ...output omitted...
  channels:
    - currentCSV: compliance-operator.v1.4.0
      currentCSVDesc:
        annotations:
          alm-examples: |- ...
      ...output omitted...
      operatorframework.io/suggested-namespace: openshift-compliance
    ...output omitted...
      version: 1.4.0
      name: stable
    defaultChannel: stable
    packageName: compliance-operator
  ...output omitted...
```

The package manifest contains the following information:

Field	Value
catalog	do280-catalog-cs
catalog-namespace	openshift-marketplace
suggested-namespace	openshift-compliance
defaultChannel	stable
packageName	compliance-operator

3. Create the recommended openshift-compliance namespace.

3.1. Use the oc command to create the namespace.

```
[student@workstation ~]$ oc create namespace openshift-compliance
namespace/openshift-compliance created
```

4. Create an operator group with the compliance-operator name in the openshift-compliance namespace. The target namespace of the operator group is the openshift-compliance namespace. You can use the ~/D0280/labs/operators-review/operator-group.yaml file as a template.

- 4.1. Create an `operator-group.yaml` file with the following content:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  targetNamespaces:
    - openshift-compliance
```

- 4.2. Use the `oc` command to create the operator group:

```
[student@workstation ~]$ oc create -f operator-group.yaml
operatorgroup.operator.coreos.com/compliance-operator created
```

5. Create a `compliance-operator` subscription in the `openshift-compliance` namespace. The subscription has the following parameters:

Field	Value
channel	stable
spec.name	compliance-operator
source	do280-catalog-cs
sourceNamespace	openshift-marketplace

You can use the `~/D0280/labs/operators-review/subscription.yaml` file as a template.

You can configure automatic install plan approvals.

- 5.1. Create a `subscription.yaml` file with the following content:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  channel: stable
  installPlanApproval: Automatic
  name: compliance-operator
  source: do280-catalog-cs
  sourceNamespace: openshift-marketplace
```

- 5.2. Use the `oc` command to create the operator group:

```
[student@workstation ~]$ oc create -f subscription.yaml
subscription.operator.coreos.com/compliance-operator created
```

6. Wait until the operator is installed.

The Operator Lifecycle Manager creates a cluster service version in the `openshift-compliance` namespace. Wait until the cluster service version resource (CSV) is in the `Succeeded` phase.

Although the CSV defines a single `compliance-operator` deployment, the operator has two additional deployments. Wait until the `compliance-operator`, `ocp4-openshift-compliance-pp`, and `rhcose4-openshift-compliance-pp` deployments are ready.

6.1. Select the `openshift-compliance` project.

```
[student@workstation ~]$ oc project openshift-compliance
Now using project "openshift-compliance" on server "https://api.ocp4.example.com:6443".
```

6.2. Wait until the CSV is in the `Succeeded` phase.

```
[student@workstation ~]$ oc get csv
NAME                  DISPLAY          VERSION   ...  PHASE
compliance-operator.v1.4.0  Compliance Operator  1.4.0     ...  Succeeded
...output omitted...
```

The available CSV version in the lab might change. Commands in the following steps require you to replace the available version in the lab.

- 6.3. Inspect the CSV to view the operator deployment. Replace the version that you obtained in a previous step. The `.spec.install.spec.deployments` JSONPath expression describes the location of the operator deployments in the CSV resource. Optionally, use the `jq` command to indent the output.

```
[student@workstation ~]$ oc get csv compliance-operator.v1.4.0 \
-o jsonpath={.spec.install.spec.deployments} | jq
[
  {
    "name": "compliance-operator",
    "spec": {
      ...
    }
  }
]
```

The Compliance operator describes a single deployment with the `compliance-operator` name.

6.4. Use the `oc` command to list the workloads in the operator namespace.

```
[student@workstation ~]$ oc get all
NAME
pod/compliance-operator-...  ...
pod/ocp4-openshift-compliance-pp-...  ...
pod/rhcose4-openshift-compliance-pp-...  ...

...output omitted...

NAME          READY  ...
deployment.apps/compliance-operator  1/1   ...
deployment.apps/ocp4-openshift-compliance-pp  1/1   ...
```

```
deployment.apps/rhcos4 Openshift-compliance-pp 1/1 ...  
...output omitted...
```

Besides the `compliance-operator` deployment, the Compliance operator creates two other deployments.

Wait until all deployments are ready.

7. Verify that the operator works correctly.

This operator watches custom resources of the `ScanSettingBinding` type and runs file integrity checks on cluster nodes. The operator reports results with custom resources of the `ComplianceSuite` type.

Create a scan setting binding in the `openshift-compliance` namespace. You can use the `~/DO280/labs/operators-review/scan-setting-binding.yaml` file as a template.

You can also use the web console to create the scan setting binding. The YAML editor in the web console provides the same scan setting binding resource as an example.

Wait until a resource of the `ComplianceSuite` type in the `DONE` phase is present in the `openshift-compliance` namespace.

7.1. Examine the `alm-examples` annotation in the CSV. Replace the version that you obtained in a previous step.

```
[student@workstation ~]$ oc get csv compliance-operator.v1.4.0 \
-o jsonpath='{.metadata.annotations.alm-examples}' | jq
[
  ...
  ...output omitted...
  {
    "apiVersion": "compliance.openshift.io/v1alpha1",
    "kind": "ScanSettingBinding",
    "metadata": {
      "name": "nist-moderate"
    },
    "profiles": [
      {
        "apiGroup": "compliance.openshift.io/v1alpha1",
        "kind": "Profile",
        "name": "rhcos4-moderate"
      }
    ],
    "settingsRef": {
      "apiGroup": "compliance.openshift.io/v1alpha1",
      "kind": "ScanSetting",
      "name": "default"
    }
  },
  ...
  ...output omitted...
]
```

The annotation contains an example scan setting binding that you can use. The example is in JSON format. When creating a scan setting binding in the web console, the YAML editor loads the same example.

You can also use the `oc explain` command to describe the scan setting binding resource.

- 7.2. Create the scan setting binding resource by using the example file in the ~/D0280/labs/operators-review/scan-setting-binding.yaml path.

```
[student@workstation ~]$ oc create \
-f ~/D0280/labs/operators-review/scan-setting-binding.yaml
scansettingbinding.compliance.openshift.io/nist-moderate created
```

- 7.3. Use the oc command to list compliance suite and pod resources. Execute the command repeatedly until the compliance suite resource is in the DONE phase.

```
[student@workstation ~]$ oc get compliancesuite,pod
NAME                                         PHASE   RESULT
compliancesuite.compliance.openshift.io/nist-moderate   DONE    NON-COMPLIANT

NAME
pod/compliance-operator-...      ...
pod/ocp4 Openshift-compliance-pp-...  ...
pod/rhcos4 Openshift-compliance-pp-...  ...
```

To execute the scan, the compliance operator creates extra pods. The pods disappear when the scan completes.

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 grade operators-review
```

Finish

As the student user on the workstation machine, use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- Operators extend the capabilities of a Kubernetes cluster.
- Cluster operators provide the platform services of OpenShift, such as the web console.
- The Operator Lifecycle Manager manages add-on operators, which are sourced from catalogs such as the OperatorHub.
- Most operators create and manage complex workloads based on declarative custom resources.
- Users can view, install, update, and troubleshoot add-on operators by using the web console.
- Users can use the package manifest, subscription, operator group, and install plan resources to manage add-on operators from the command line or from the API.

Chapter 8

Application Security

Goal

Run applications that require elevated or special privileges from the host operating system or Kubernetes.

Objectives

- Create service accounts and apply permissions, and manage security context constraints.
- Run an application that requires access to the Kubernetes API of the application's cluster.
- Automate regular cluster and application management tasks by using Kubernetes cron jobs.

Sections

- Control Application Permissions with Security Context Constraints (and Guided Exercise)
- Allow Application Access to Kubernetes APIs (and Guided Exercise)
- Cluster and Node Maintenance with Kubernetes Cron Jobs (and Guided Exercise)

Lab

- Application Security

Control Application Permissions with Security Context Constraints

Objectives

- Create service accounts and apply permissions, and manage security context constraints.

Security Context Constraints (SCCs)

Red Hat OpenShift provides *security context constraints* (SCCs), a security mechanism that limits the access from a running pod in OpenShift to the host environment. SCCs control the following host resources:

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

Cluster administrators can run the following command to list the SCCs that OpenShift defines:

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

OpenShift provides the following default SCCs:

- anyuid
- hostaccess
- hostmount-anyuid
- hostnetwork
- hostnetwork-v2
- lvms-topolvm-node
- lvms-vgmanager
- machine-api-termination-handler
- node-exporter
- nonroot
- nonroot-v2
- privileged
- restricted
- restricted-v2

For additional information about an SCC, use the `oc describe` command:

```
[user@host ~]$ oc describe scc anyuid
Name:          anyuid
Priority:      10
Access:
Users:         <none>
Groups:        system:cluster-admins
Settings:
...output omitted...
```

Most pods that OpenShift creates use the `restricted-v2` SCC, which provides limited access to resources that are external to OpenShift. Use the `oc describe` command to view the security context constraint that a pod uses.

```
[user@host ~]$ oc describe pod console-5df4fcbb47-67c52 \
-n openshift-console | grep scc
openshift.io/scc: restricted-v2
```

Container images that are downloaded from public container registries, such as Docker Hub, might fail to run when using the `restricted-v2` SCC. For example, a container image that requires running as a specific user ID can fail because the `restricted-v2` SCC runs the container by using a random user ID. A container image that listens on port 80 or on port 443 can fail for a related reason. The random user ID that the `restricted-v2` SCC uses cannot start a service that listens on a privileged network port (port numbers that are less than 1024). Use the `scc-subject-review` subcommand to list all the security context constraints that can overcome the limitations that hinder the container:

```
[user@host ~]$ oc get deployment deployment-name -o yaml | \
oc adm policy scc-subject-review -f -
```

The `anyuid` SCC defines the `run as user` strategy to be `RunAsAny`, which means that the pod can run as any available user ID in the container. With this strategy, containers that require a specific user can run the commands by using a specific user ID.

To change the container to run with a different SCC, you must create a service account that is 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 different namespace from the current one:

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

To associate the service account with an SCC, use the `oc adm policy` command. Identify a service account by using the `-z` option, and use the `-n` option if the service account exists in a different namespace from the current one:

```
[user@host ~]$ oc adm policy add-scc-to-user SCC -z service-account
```



Important

Only cluster administrators can assign an SCC to a service account or remove an SCC from a service account. Allowing pods to run with a less restrictive SCC can make your cluster less secure. Use with caution.

Change an existing deployment to use the service account by using the `oc set serviceaccount` command:

```
[user@host ~]$ oc set serviceaccount deployment/deployment-name \
service-account-name
```

If the command succeeds, then the pods that are associated with the deployment redeploy.

Privileged Containers

Some containers might need to access the runtime environment of the host. For example, the S2I builder class of privileged containers requires access beyond the limits of its 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 4.14 *Authentication and Authorization* documentation at
https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#managing-pod-security-policies

► Guided Exercise

Control Application Permissions with Security Context Constraints

Deploy applications that require pods with extended permissions.

Outcomes

- Create service accounts and assign security context constraints (SCCs) to them.
- Assign a service account to a deployment.
- 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 start appsec-scc
```

Instructions

- 1. Log in to the OpenShift cluster and create the `appsec-scc` project.

1.1. Log in to the cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

1.2. Create the `appsec-scc` project.

```
[student@workstation ~]$ oc new-project appsec-scc
Now using project "appsec-scc" on server ...
...output omitted...
```

- 2. Deploy an application named `gitlab` by using the container image at `registry.ocp4.example.com:8443/redhattraining/gitlab-ce:8.4.3-ce.0`. This image is a copy of the container image at `docker.io/gitlab/gitlab-ce:8.4.3-ce.0`. Verify that the reason for the pod failure is because the container image needs `root` privileges.

2.1. Deploy the `gitlab` application.

```
[student@workstation ~]$ oc new-app --name gitlab \
--image registry.ocp4.example.com:8443/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 whether the application is successfully deployed. It should give an error, because this image needs `root` privileges to deploy.

NAME	READY	STATUS	RESTARTS	AGE
gitlab-d89cd88f8-jwqbp	0/1	Error	0	36s

**Note**

It might take some time for the image to reach the `Error` state. You might also see the `CrashLoopBackOff` status when you validate the health of the pod.

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

```
[student@workstation ~]$ oc logs pod/gitlab-d89cd88f8-jwqbp
...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...
```

The application tries to write to the `/etc` directory. To allow the application to write to the `/etc` directory, you can make the application run as the `root` user. To run the application as the `root` user, you can grant the `anyuid` SCC to a service account.

- 3. Create a service account and assign the `anyuid` SCC to it.

- 3.1. Log in as the `admin` user with the `redhatocp` password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 3.2. Verify the appropriate SCC to use with this deployment.

```
[student@workstation]$ oc get deploy
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
gitlab    0/1     1           0           109s

[student@workstation]$ oc get deploy/gitlab -o yaml | oc adm policy \
  scc-subject-review -f -
RESOURCE          ALLOWED BY
Deployment/gitlab  anyuid
```

The output confirms that the anyuid SCC allows the gitlab deployment to create and update pods.

3.3. Create a service account named gitlab-sa.

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

3.4. 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 to use 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.

- 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 -sL http://gitlab.apps.ocp4.example.com/ | \
grep '<title>'>
<title>Sign in · GitLab</title>
```

► 6. Delete the appsec-scc project.

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

Finish

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

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

Allow Application Access to Kubernetes APIs

Objectives

- Run an application that requires access to the Kubernetes API of the application's cluster.

Securing Kubernetes APIs

With the Kubernetes APIs, a user or an application can query and modify the cluster state. To protect your cluster from malicious interactions, you must grant access to the different Kubernetes APIs.

Role-based access control (RBAC) authorization is preconfigured in OpenShift. An application requires explicit RBAC authorization to access restricted Kubernetes APIs.

Application Authorization with Service Accounts

A service account is a Kubernetes object within a project. The service account represents the identity of an application that runs in a pod.

To grant an application access to a Kubernetes API, take these actions:

- Create an application service account.
- Grant the service account access to the Kubernetes API.
- Assign the service account to the application pods.

If the pod definition does not specify a service account, then the pod uses the `default` service account. OpenShift grants no rights to the `default` service account, which is expected for business workloads. It is not recommended to grant additional permissions to the `default` service account, because it grants those additional permissions to all pods in the project, which might not be intended.

Use Cases for Kubernetes API Access

Regular business applications can successfully use the `default` service account, without requiring access to the Kubernetes APIs. On the contrary, infrastructure applications need access to monitor or to modify the cluster resources. These infrastructure applications might be classified into the following use cases:

Monitoring Applications

Applications in this category need read access to watch cluster resources or to verify cluster health. For example, a service such as Red Hat Advanced Cluster Security (ACS) needs read access to scan your cluster containers for vulnerabilities.

Controllers

Controllers are applications that constantly watch and try to reach the intended state of a resource.

For example, GitOps tools, such as ArgoCD, have controllers that watch cluster resources that are stored in a repository, and update the cluster to react to changes in that repository.

Operators

Operators automate creating, configuring, and managing instances of Kubernetes-native applications. Therefore, operators need permissions for configuration and maintenance tasks.

For example, a database operator might create a deployment when it detects a CR that defines a new database.

Application Kubernetes API Authorization with Roles

To provide the application with the needed permissions only, you can create roles or cluster roles that describe the application requirements. Roles grant permissions to Kubernetes API resources within a single namespace. Cluster roles grant permissions, either within one or more namespaces, or to all the cluster.

For example, you can create a cluster role for an application to read secrets.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

- ➊ The API groups, where an empty string represents the core API
- ➋ The resources that the role refers to
- ➌ The verbs or actions that the role allows the application to perform on the resource

You can also use the default cluster roles that OpenShift defines, which have wider permissions. For example, you can use the `edit` cluster role to get read access on secrets, as in the previous `secret-reader` cluster role.

The `edit` cluster role is less restrictive, and allows the application to create or update most objects.

Binding Roles to Service Accounts

For an application to use the role permissions, you must bind the role or cluster role to the application service account.

To bind a role or cluster role to a service account in a namespace, you can use the `oc adm policy` command with the `add-role-to-user` subcommand.

This command assigns a cluster role to a service account that exists in the current project:

```
[user@host ~]$ oc adm policy add-role-to-user cluster-role -z service-account
```

You can optionally use `-z` to avoid specifying the `system:serviceaccount:project` prefix when you assign the role to a service account that exists in the current project.

To create a cluster role binding, you can use the `oc adm policy` command with the `add-cluster-role-to-user` subcommand.

The following command assigns a cluster role to a service account with a cluster scope:

```
[user@host ~]$ oc adm policy add-cluster-role-to-user cluster-role service-account
```

Assigning an Application Service Account to Pods

OpenShift uses RBAC authorization by using the roles that are associated to the service account to grant or deny access to the resource. You specify the service account name in the `spec.serviceAccountName` pod definition field.

Applications must use the service account token internally when accessing a Kubernetes API. In earlier OpenShift versions than 4.11, OpenShift generated a secret with a token when creating a service account. Starting from OpenShift 4.11, tokens are no longer generated automatically. You must use the TokenRequest API to generate the service account token. You must mount the token as a pod volume for the application to access it.

Scoping Application Access to Kubernetes API Resources

An application might require access to a resource in the same namespace, or in a different namespace, or in all namespaces.

Accessing API Resources in the Same Namespace

To grant an application access to resources in the same namespace, you need a role or a cluster role and a service account in that namespace. You then create a role binding that associates to the service account the actions that the role grants on the resource. Using a role binding with a cluster role grants access only to the resource within the namespace.

Accessing API Resources in a Different Namespace

To give an application access to a resource in a different namespace, you must create the role binding in the project with the resource. The subject for the binding references the application service account that is in a different namespace from the binding.

You can use the following syntax to refer to service accounts from other projects:

```
system:serviceaccount:project:service-account
```

For example, if you have an application pod in the `project-1` project that requires access to `project-2` secrets, then you must take these actions:

- Create an `app-sa` service account in the `project-1` project.
- Assign the `app-sa` service account to your application pod.
- Create a role binding on the `project-2` project that references the `app-sa` service account and the `secret-reader` role or cluster role.

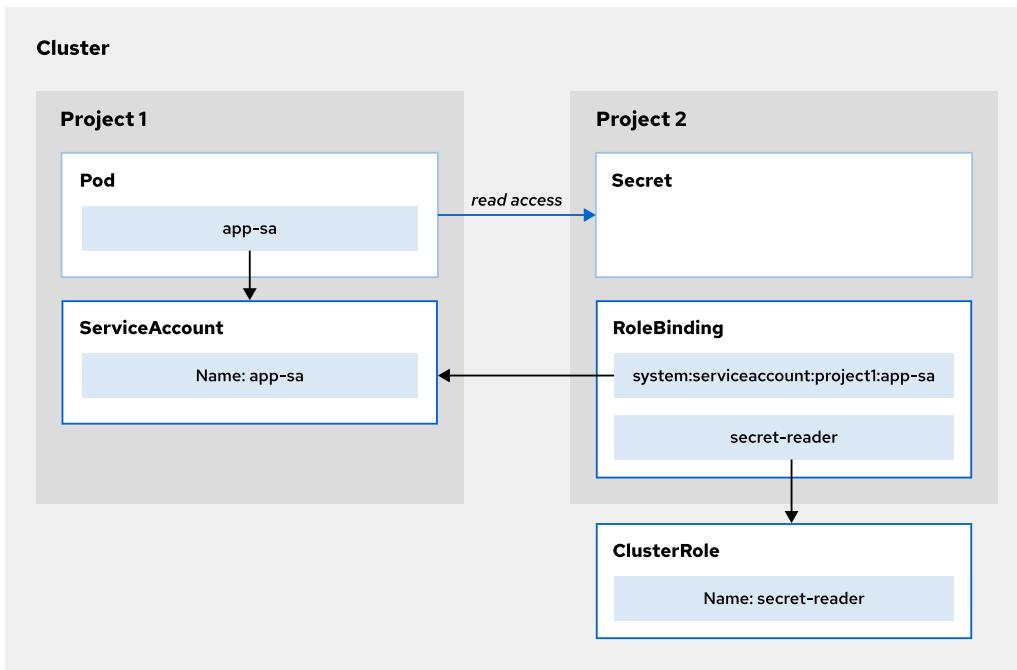


Figure 8.1: Grant access to a service account to a different project

In this way, you restrict an application's access to a Kubernetes API to specified namespaces.

Accessing API Resources in All Namespaces

Grant your application service account the cluster role by using a cluster role binding. The cluster role binding grants the application cluster access to the API.



References

For more information, refer to the *Using RBAC to Define and Apply Permissions* chapter in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#authorization-overview_using-rbac

For more information, refer to the *Understanding and Creating Service Accounts* chapter in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#service-accounts-overview_understanding-service-accounts

For more information, refer to the *Using Service Accounts in Applications* chapter in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#service-accounts-overview_using-service-accounts

For more information, refer to the *About Automatically-generated Service Account Token Secrets* section in the Red Hat OpenShift Container Platform 4.14 *Authentication and Authorization* documentation at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#auto-generated-sa-token-secrets_using-service-accounts

► Guided Exercise

Allow Application Access to Kubernetes APIs

Configure an application with limited access to Kubernetes API resources.

Outcomes

You should be able to grant Kubernetes API access to an application by using a service account that has a role with the required privileges.

Before You Begin

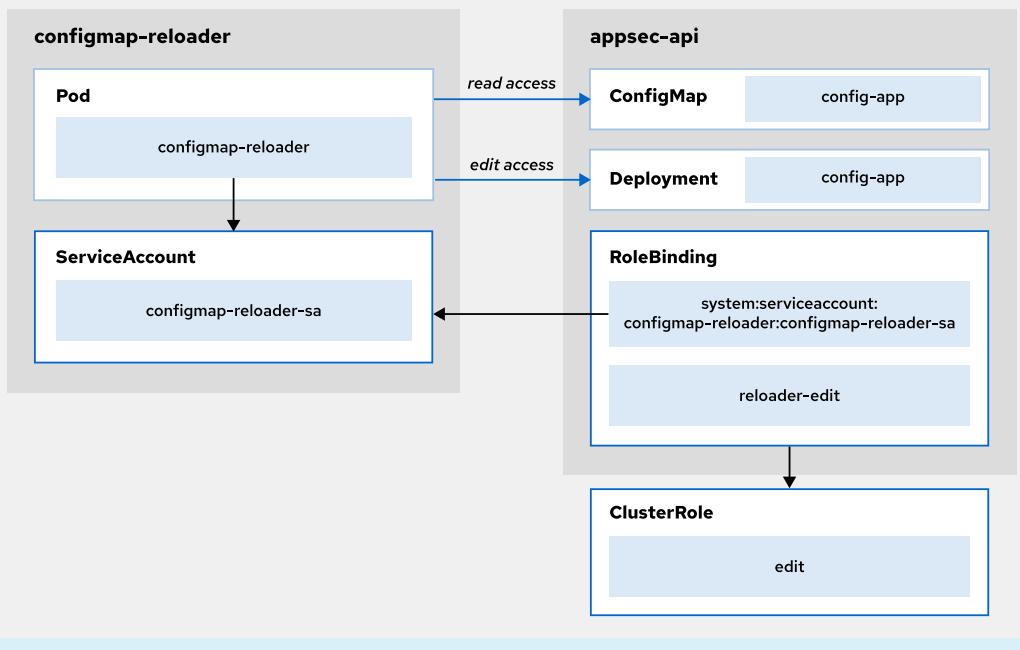
As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

The lab command copies the following files to the lab directory:

- The deployment manifest to install the Stakater Reloader application, at <https://github.com/stakater/Reloader>. This application is a controller that watches for changes in configuration maps and does rolling upgrades on associated deployments.
- The manifests to install the `config-app` API, which has an endpoint to show its internal configuration. The deployment manifest mounts the API configuration from a configuration map.

In this exercise, you grant permissions on the `appsec-api` project to the Reloader application, for read access to the configuration map API and edit access to the deployment API.

Cluster



**Warning**

Using a controller to update a Kubernetes resource by reacting to changes is an alternative to using GitOps. However, do not use both a controller and GitOps for such changes because it might cause conflicts.

```
[student@workstation ~]$ lab start appsec-api
```

Instructions

- 1. Change to the lab directory.

- 1.1. Change to the ~/D0280/labs/appsec-api directory.

```
[student@workstation ~]$ cd ~/D0280/labs/appsec-api
```

- 2. Log in as the `admin` user and change to the `configmap-reloader` project.

- 2.1. Open a terminal window and log in as the `admin` user with the `redhatocp` password.

```
[student@workstation appsec-api]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 2.2. Use the `oc project` command to change to the `configmap-reloader` namespace.

```
[student@workstation appsec-api]$ oc project configmap-reloader
Now using project "configmap-reloader" on server ...
```

- 3. Create the `configmap-reloader` service account to hold the permissions for the `Reloaded` application. Then, assign the `configmap-reloader` service account to the `configmap-reloader` deployment.

- 3.1. Create the `configmap-reloader` service account.

```
[student@workstation appsec-api]$ oc create sa configmap-reloader-sa
serviceaccount/configmap-reloader-sa created
```

- 3.2. Add the `configmap-reloader-sa` service account to the deployment in the `reloader-deployment.yaml` file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: configmap-reloader
```

```

name: configmap-reloader
namespace: configmap-reloader
spec:
  selector:
    matchLabels:
      app: configmap-reloader
      release: "reloader"
  template:
    metadata:
      labels:
        app: configmap-reloader
    spec:
      serviceAccountName: configmap-reloader-sa
      containers:
...output omitted...

```

- 3.3. Use the `oc` command to create the `configmap-reloader` deployment from the `reloader-deployment.yaml` file.

```
[student@workstation appsec-api]$ oc apply -f reloader-deployment.yaml
deployment.apps/configmap-reloader created
```

► 4. As the `developer` user, create the `appsec-api` project.

- 4.1. Log in to the cluster as the `developer` user with the `developer` password.

```
[student@workstation appsec-api]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

- 4.2. Use the `oc new-project` command to create the `appsec-api` project.

```
[student@workstation appsec-api]$ oc new-project appsec-api
Now using project "appsec-api" on server ...
```

► 5. Grant permissions to the `configmap-reloader-sa` service account to watch configuration map resources and roll out deployments on the `appsec-api` project.

- 5.1. Assign the `edit` cluster role to the `configmap-reloader-sa` service account in the `appsec-api` project. To assign the cluster role, create a local role binding by using the `oc policy add-role-to-user` command with the following options:

- The `edit` default cluster role.
- The `system:serviceaccount:configmap-reloader:configmap-reloader-sa` username to reference the `configmap-reloader-sa` service account in the `configmap-reloader` project.
- The `--rolebinding-name` option to use the `reloader-edit` name for the role binding.
- The `-n appsec-api`, which is optional because you are already in the `appsec-api` project.

```
[student@workstation appsec-api]$ oc policy add-role-to-user edit \
    system:serviceaccount:configmap-reloader:configmap-reloader-sa \
    --rolebinding-name=reloader-edit \
    -n appsec-api
clusterrole.rbac.authorization.k8s.io/edit added:
"system:serviceaccount:configmap-reloader:configmap-reloader-sa"
```



Note

The `edit` cluster role with the local role binding allows the `configmap-reloader-sa` service account to modify most objects in the `appsec-api` project. In a production scenario, it is best to grant access only to the APIs that your application requires.

- ▶ 6. Install the `config-app` API by using the manifest files in the `config-app` directory.

- 6.1. Use the `oc apply` command with the `-f` option to create all the manifests in the `config-app` directory.

```
[student@workstation appsec-api]$ oc apply -f ./config-app
configmap/config-app created
deployment.apps/config-app created
route.route.openshift.io/config-app created
service/config-app created
```

- 6.2. Read the `config.yaml` content from the `config-app` configuration map by running the `oc get` command.

```
[student@workstation appsec-api]$ oc get configmap config-app \
    --output="jsonpath={.data.config\\.yaml}"
application:
  name: "config-app"
  description: "config-app"
```

- 6.3. Run the `curl` command to verify that the exposed route, `https://config-app-appsec-api.apps.ocp4.example.com/config`, shows the `config-app` configuration map content.

```
[student@workstation appsec-api]$ curl -s \
  https://config-app-appsec-api.apps.ocp4.example.com/config | jq
{
  "application": {
    "description": "config-app",
    "name": "config-app"
  }
}
```

- ▶ 7. Update the `config-app` configuration map `description` key and query `/config` endpoint to verify that the `Reload` controller upgrades the `config-app` deployment.

- 7.1. Update the description data in the configuration map in the config-app/configmap.yaml file to the API that exposes its configuration value.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-app
  namespace: appsec-api
data:
  config.yaml: |
    application:
      name: "config-app"
      description: "API that exposes its configuration"
```

- 7.2. Use the oc command to apply the changes to the config-app/configmap.yaml file.

```
[student@workstation appsec-api]$ oc apply -f config-app/configmap.yaml
configmap/config-app configured
```

- 7.3. Use the watch command to query the API /config endpoint by using the curl command to verify that the API configuration changes. Press Ctrl+C to exit.

```
[student@workstation appsec-api]$ watch \
  "curl -s https://config-app-appsec-api.apps.ocp4.example.com/config | jq"
Every 2.0s: curl -s https://config-app-appsec-api.apps.ocp4.example.com/config | jq
workstation: ...

{
  "application": {
    "description": "API that exposes its configuration",
    "name": "config-app"
  }
}
```

Wait until the controller application upgrades the deployment.

- 8. Change to the home directory to complete the exercise.

- 8.1. Change to the home directory.

```
[student@workstation appsec-api]$ cd
```

Finish

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

```
[student@workstation ~]$ lab finish appsec-api
```

Cluster and Node Maintenance with Kubernetes Cron Jobs

Objectives

- Automate regular cluster and application management tasks with Kubernetes cron jobs.

Maintenance Tasks

Cluster administrators can use scheduled tasks to automate maintenance tasks in the cluster. Other users can create scheduled tasks for regular application maintenance.

Maintenance tasks vary in the privileges that they require. Cluster maintenance tasks require privileged pods, whereas most applications might not require elevated privileges.

Kubernetes Batch API Resources

You can automate tasks in OpenShift by using standard Kubernetes jobs and cron jobs. The automated tasks can be configured to run once or on a regular schedule.

Job

Kubernetes jobs specify a task that is executed once.

Cron Job

Kubernetes cron jobs have a schedule to execute a task regularly.

When a cron job is due for execution, Kubernetes creates a job resource. Kubernetes creates these jobs from a template in the cron job definition. Other than this relationship, Kubernetes jobs and cron jobs are workload resource types, such as deployments or daemon sets.

Kubernetes Jobs

The job resource includes a pod template that describes the task to execute. You can use the `oc create job --dry-run=client` command to get the YAML representation of the Kubernetes job resource:

```
[user@host ~]$ oc create job --dry-run=client -o yaml test \
--image=registry.access.redhat.com/ubi8/ubi:8.6 \
-- curl https://example.com
```

A job contains a pod template, and this pod template must specify at least one container. You can add metadata such as labels or annotations to the job definition and pod template.

```
apiVersion: batch/v1
kind: Job
metadata:
  creationTimestamp: null
  name: test
spec: ❶
  template: ❷
    metadata:
```

```

    creationTimestamp: null
  spec: ❸
    containers: ❹
      - command: ❺
        - curl
        - https://example.com
      image: registry.access.redhat.com/ubi8/ubi:8.6 ❻
      name: test
      resources: {}
    restartPolicy: Never
  status: {}

```

- ❶ Job specification
- ❷ Pod template
- ❸ Pod specification
- ❹ Pod containers
- ❺ Command
- ❻ Container image

Kubernetes Cron Jobs

The cron job resource includes a job template that describes the task and a schedule. You can use the `oc create cronjob --dry-run=client` command to get the YAML representation of the Kubernetes cron job resource:

```
[user@host ~]$ oc create cronjob --dry-run=client -o yaml test \
--image=registry.access.redhat.com/ubi8/ubi:8.6 \
--schedule='0 0 * * *' \
-- curl https://example.com
```

In Kubernetes, cron job resources are similar to job resources. The `jobTemplate` key follows the same structure as a job. The `schedule` key describes when the task runs.

```

apiVersion: batch/v1
kind: CronJob
metadata:
  creationTimestamp: null
  name: test
spec: ❶
  jobTemplate: ❷
    metadata:
      creationTimestamp: null
      name: test
    spec: ❸
      template: ❹
        metadata:
          creationTimestamp: null
        spec: ❺
          containers:

```

```

- command: ⑥
- curl
- https://example.com
image: registry.access.redhat.com/ubi8/ubi:8.6 ⑦
name: test
resources: {}
restartPolicy: OnFailure
schedule: 0 0 * * * ⑧
status: {}

```

- ①** Cron job specification
- ②** Job template
- ③** Job specification
- ④** Pod template
- ⑤** Pod specification
- ⑥** Command
- ⑦** Container image
- ⑧** Cron job schedule specification

Linux Cron Jobs

The schedule specification for Kubernetes cron jobs is derived from the specification in Linux cron jobs. The `crontab` file specifies the scheduled tasks for the current user. The schedule specification has five fields to define the date and time when the job is executed. The `/etc/crontab` file comments include a syntax diagram:

```

# Example cron job definition:
#   ┌───────── minute (0 - 59)
#   |   ┌───────── hour (0 - 23)
#   |   |   ┌───────── day of month (1 - 31)
#   |   |   |   ┌───────── month (1 - 12) or jan,feb,mar,apr ...
#   |   |   |   |   ┌───────── day of week (0 - 7) or sun,mon,tue,wed,thu,fri,sat
#   |   |   |   |   |   (Sunday is 0 or 7)
#   |   |   |   |   |   m   h   dom mon dow   command
#   0 */2 * * * /path/to/task_executable arguments

```

Some examples of cron job specifications are as follows:

Schedule specification	Description
0 0 * * *	Run the specified task every day at midnight
0 0 * * 7	Run the specified task every Sunday at midnight
0 * * * *	Run the specified task every hour
0 */4 * * *	Run the specified task every four hours

**Note**

Refer to the `crontab(5)` manual page for more information about the cron job schedule specification.

Automate Maintenance Tasks with Cron Jobs

You can automate the maintenance tasks for applications that run inside the cluster, and also execute low-level commands inside privileged debug pods to apply cluster maintenance tasks.

Automating Application Maintenance Tasks

Regular maintenance tasks might need to run for applications that run in the cluster.

For example, consider creating periodic backups for an application. This application requires the following steps to create the backup:

- Activate maintenance mode.
- Create a compressed database backup.
- Deactivate maintenance mode.
- Copy the database backup to an external location.

The following cron job definition shows a possible implementation of these steps:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: wordpress-backup
spec:
  schedule: 0 2 * * 7 ①
  jobTemplate: ②
    spec:
      template: ③
        spec: ④
          dnsPolicy: ClusterFirst
          restartPolicy: Never
          containers: ⑤
            - name: wp-cli
              image: registry.io/wp-maintenance/wp-cli:2.7 ⑥
              resources: {}
              command: ⑦
                - bash
                - -xc
              args: ⑧
                - >
                  wp maintenance-mode activate ;
                  wp db export | gzip > database.sql.gz ;
                  wp maintenance-mode deactivate ;
                  rclone copy database.sql.gz s3://bucket/backups/ ;
                  rm -v database.sql.gz ;
```

① Schedule for every Sunday at 2 AM

② The Kubernetes job template

- ③ The Kubernetes pod template
- ④ The Kubernetes pod specification
- ⑤ The pod container configuration
- ⑥ The container image that runs the maintenance task
- ⑦ The command to execute inside the pod
- ⑧ Maintenance commands to execute

**Note**

The > symbol uses the YAML *folded style*, which converts all newlines to spaces when parsing. Each command is separated with a semicolon (;), because the string in the args key is passed as a single argument to the bash -xc command.

This combination of the command and args keys has the same effect as executing the commands in a single line inside the container:

```
[user@host ~]$ bash -xc 'wp maintenance-mode activate ; wp db export | gzip > database.sql.gz ; wp maintenance-mode deactivate ; rclone copy database.sql.gz s3://bucket/backups/ ; rm -v database.sql.gz ;'
```

For more information about the YAML folded style, refer to <https://yaml.org/spec/1.2.2/#folded-style>

Automating Cluster Maintenance Tasks

Cluster maintenance might require executing complex scripts in privileged pods. You can create a shell script with the commands to execute the maintenance task, and mount the script in the pod by using a configuration map.

For example, when images are updated, clusters might accumulate unused images. These images might occupy much space. Executing the `crlctl rmi --prune` command on all nodes of the cluster frees this space.

The following configuration map contains a shell script that cleans images in all cluster nodes by executing a debug pod and running the `crlctl` command with the `chroot` command to access the root file system of the node:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: maintenance
  app: crlctl
data:
  maintenance.sh: |
    #!/bin/bash
    NODES=$(oc get nodes -o=name) ①
    for NODE in ${NODES} ②
    do
      echo ${NODE}
      oc debug ${NODE} -- \ ③
```

```
chroot /host \
/bin/bash -xc 'crictl images ; crictl rmi --prune' ④
echo $?
done
```

- ①** List the nodes in the cluster.
- ②** Iterate over the nodes.
- ③** Run a debug pod on the node.
- ④** Prune the images.

This task can be scheduled regularly by using a cron job. The `quay.io/openshift/origin-cli:4.14` container provides the `oc` command that runs the debug pod. The pod mounts the configuration map and executes the maintenance script.

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: image-pruner
spec:
  schedule: 0 * * * *
  jobTemplate:
    spec:
      template:
        spec:
          dnsPolicy: ClusterFirst
          restartPolicy: Never
          containers:
            - name: image-pruner
              image: quay.io/openshift/origin-cli:4.14
              resources: {}
              command:
                - /opt/maintenance.sh ①
              volumeMounts: ②
                - name: scripts
                  mountPath: /opt
              volumes: ③
                - name: scripts
                  configMap:
                    name: maintenance
                    defaultMode: 0555
```

- ①** Path to the script
- ②③** Mounting the configuration map as a volume

Cluster maintenance tasks might require elevated privileges. Administrators can assign service accounts to any workload, including Kubernetes jobs and cron jobs.

You can create a service account with the required privileges, and specify the service account with the `serviceAccountName` key in the pod definition. You can also use the `oc set serviceaccount` command to change the service account of an existing workload.



References

Kubernetes Job

<https://kubernetes.io/docs/concepts/workloads/controllers/job/>

Kubernetes Cron Job

<https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>

How to Delete Exited Containers and Dangling Images with crictl?

<https://access.redhat.com/solutions/5610941>

► Guided Exercise

Cluster and Node Maintenance with Kubernetes Cron Jobs

Automate periodic cluster node cleaning for a development environment.

Outcomes

- Manually delete unused images from the nodes.
- Automate the image pruning by using a cron job.

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 start appsec-prune
```

Instructions

► 1. Log in to the OpenShift cluster and switch to the `appsec-prune` project.

- 1.1. Log in to the cluster as the `admin` user.

```
[student@workstation ~]$ oc login -u admin -p redhatocp \
  https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 1.2. Create the `appsec-prune` project.

```
[student@workstation ~]$ oc new-project appsec-prune
Now using project "appsec-prune" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 1.3. Change to the `~/D0280/labs/appsec-prune` directory.

```
[student@workstation ~]$ cd ~/D0280/labs/appsec-prune
[student@workstation appsec-prune]$
```

► 2. Clean up the unused container images in the node.

- 2.1. List the deployments and pods in the `prune-apps` namespace. Each deployment has a pod that uses a different image.

```
[student@workstation appsec-prune]$ oc get deployments -n prune-apps -o wide
NAME      ... IMAGES ...
nginx-ubi7 ... registry.ocp4.example.com:8443/ubi7/nginx-118:latest ...
nginx-ubi8 ... registry.ocp4.example.com:8443/ubi8/nginx-118:latest ...
nginx-ubi9 ... registry.ocp4.example.com:8443/ubi9/nginx-120:latest ...

[student@workstation appsec-prune]$ oc get pods -n prune-apps
NAME                      READY   STATUS    RESTARTS   AGE
pod/nginx-ubi7-594f548665-qvfq6  1/1     Running   0          5m
pod/nginx-ubi8-855f6959b-jvs6h  1/1     Running   0          5m
pod/nginx-ubi9-dd4c566d7-7vrrv  1/1     Running   0          5m
```

- 2.2. List the container images in the node. The node has three `httpd` images and three `nginx` images.

```
[student@workstation appsec-prune]$ oc debug node/master01 -- \
  chroot /host crictl images | egrep '^IMAGE|httpd|nginx'
...output omitted...
Starting pod/master01-debug ...
To use host binaries, run `chroot /host`
IMAGE                                     TAG      IMAGE ID ...
registry.ocp4.example.com:8443/rhscl/httpd-24-rhel7 latest  c19a96fc0b019 ...
registry.ocp4.example.com:8443/ubi8/httpd-24      latest  e54df115d5f0c ...
registry.ocp4.example.com:8443/ubi9/httpd-24      latest  4afe283d911ab ...
registry.ocp4.example.com:8443/ubi7/nginx-118    latest  3adc6d109b363 ...
registry.ocp4.example.com:8443/ubi8/nginx-118    latest  90f91167f6d1d ...
registry.ocp4.example.com:8443/ubi9/nginx-120    latest  0227435f34784 ...

Removing debug pod ...
...output omitted...
```

- 2.3. Remove the unused images in the node. Only the `httpd` container images are deleted, because no other container uses them.

```
[student@workstation appsec-prune]$ oc debug node/master01 -- \
  chroot /host crictl rmi --prune
...output omitted...
Starting pod/master01-debug ...
To use host binaries, run `chroot /host`
E1213 00:43:40.788951 166213 remote_image.go:266] "RemoveImage from image service failed" err="rpc error: code = Unknown desc = Image used by 5027ebb4...: image is in use by a container" image="c464e04f..." ①
Deleted: registry.ocp4.example.com:8443/rhscl/httpd-24-rhel7:latest
Deleted: registry.ocp4.example.com:8443/ubi8/httpd-24:latest
Deleted: registry.ocp4.example.com:8443/ubi9/httpd-24:latest

Removing debug pod ...
...output omitted...
```

① You can ignore the error that a container is using the image.

- 2.4. Delete the deployments in the `prune-apps` namespace to remove the pods that use the `nginx` images.

```
[student@workstation appsec-prune]$ oc delete deployment nginx-ubi{7,8,9} \
-n prune-apps
deployment.apps "nginx-ubi7" deleted
deployment.apps "nginx-ubi8" deleted
deployment.apps "nginx-ubi9" deleted
```

**Note**

The cron job removes the unused container images in a later step.

- 3. Create a cron job to automate the image pruning process.

- 3.1. Edit the `~/D0280/labs/appsec-prune/configmap-prune.yaml` file to match the following specification:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: maintenance
  labels:
    ge: appsec-prune
    app: crictl
data:
  maintenance.sh: |
    #!/bin/bash -eu
    NODES=$(oc get nodes -o=name)
    for NODE in ${NODES}
    do
      echo ${NODE}
      oc debug ${NODE} -- \
        chroot /host \
        /bin/bash -euxc 'crictl images ; crictl rmi --prune'
    done
```

**Note**

The `~/D0280/solutions/appsec-prune/configmap-prune.yaml` file contains the correct configuration and can be used for comparison.

- 3.2. Create the configuration map:

```
[student@workstation appsec-prune]$ oc apply -f configmap-prune.yaml
configmap/maintenance created
```

- 3.3. Edit the `~/D0280/labs/appsec-prune/cronjob-prune.yaml` file to match the following specification:

```

apiVersion: batch/v1
kind: CronJob
metadata:
  name: image-pruner
  labels:
    ge: appsec-prune
    app: crictl
spec:
  schedule: '*/4 * * * *'
  jobTemplate:
    spec:
      template:
        spec:
          dnsPolicy: ClusterFirst
          restartPolicy: Never
          containers:
            - name: crictl
              image: registry.ocp4.example.com:8443/openshift/origin-cli:4.14 ①
              resources: {}
              command:
                - /opt/maintenance.sh
              volumeMounts:
                - name: scripts
                  mountPath: /opt
              volumes:
                - name: scripts
                  configMap:
                    name: maintenance
                    defaultMode: 0555

```

- ① The `registry.ocp4.example.com:8443/openshift/origin-cli:4.14` container image is a copy of the official `quay.io/openshift/origin-cli:4.14` image that contains the `oc` command.



Note

The `~/D0280/solutions/appsec-prune/cronjob-prune.yaml` file contains the correct configuration and can be used for comparison.

3.4. Apply the changes to the image pruner resource.

```
[student@workstation appsec-prune]$ oc apply -f cronjob-prune.yaml
cronjob.batch/image-pruner created
```



Note

A warning indicates that the pod would violate several policies. The pod fails when the cron job is executed, because it lacks permissions to execute the maintenance task. A fix for this issue is implemented in a later step.

- 3.5. Wait until the cron job is scheduled, and get the name of the associated job. The job completion status is 0/1, and the pod has an error status. Press **Ctrl+C** to exit the **watch** command.

```
[student@workstation appsec-prune]$ watch oc get cronjobs,jobs,pods
Every 2.0s: oc get cronjobs,jobs,pods      workstation: Mon Feb 13 13:00:47 2024

NAME          SCHEDULE    SUSPEND   ACTIVE   LAST SCHEDULE AGE
cronjob.batch/image-pruner  */4 * * * *  False      1        53s   6m

NAME          COMPLETIONS DURATION AGE
job.batch/image-pruner-27883800  0/1       30s     30s

NAME          READY  STATUS    RESTARTS AGE
pod/image-pruner-27950092-g76lb  0/1   Error     0        15s
```

- 3.6. Get the logs of the pod. A permission error is displayed.

```
[student@workstation appsec-prune]$ oc logs pod/image-pruner-27950092-g76lb
Error from server (Forbidden): nodes is forbidden: User
"system:serviceaccount:appsec-prune:default" cannot list resource "nodes" in API
group "" at the cluster scope
```

- 3.7. Delete the failed cron job. This action deletes the failed job and pod resources.

```
[student@workstation appsec-prune]$ oc delete cronjob/image-pruner
cronjob.batch "image-pruner" deleted
```



Note

Recommended alternatives for image pruning are covered in the DO380 - Red Hat OpenShift Administration III: Scaling Deployments in the Enterprise course.

- 4. Set the appropriate permissions to run the image pruner cron job.

- 4.1. Create a service account to use with the cron job.

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

- 4.2. Add the privileged SCC to the image-pruner service account.

```
[student@workstation ~]$ oc adm policy add-scc-to-user privileged \
-z image-pruner
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:privileged added:
"image-pruner"
```

- 4.3. Add the cluster-admin role to the image-pruner service account of the namespace.

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

- 4.4. Edit the ~/DO280/labs/appsec-prune/cronjob-prune.yaml file to match the following addition to the file:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: image-pruner
  labels:
    ge: appsec-prune
    app: crictl
spec:
  schedule: '*/4 * * * *'
  jobTemplate:
    spec:
      template:
        spec:
          serviceAccountName: image-pruner ①
          dnsPolicy: ClusterFirst
          restartPolicy: Never
          containers:
            - name: crictl
              image: registry.ocp4.example.com:8443/openshift/origin-cli:4.14
              resources: {}
              command:
                - /opt/maintenance.sh
              volumeMounts:
                - name: scripts
                  mountPath: /opt
              volumes:
                - name: scripts
              configMap:
                name: maintenance
                defaultMode: 0555
```

- ① Add the serviceAccountName attribute to replace the default service account with the `image-pruner` service account

- 4.5. Create the cron job resource again.

```
[student@workstation appsec-prune]$ oc apply -f cronjob-prune.yaml
cronjob.batch/image-pruner created
```

- 4.6. Wait until the new job and the pod are created. Press `Ctrl+C` to exit the `watch` command when the job and the pod are marked as completed.

```
[student@workstation appsec-prune]$ watch oc get cronjobs,jobs,pods
Every 2.0s: oc get cronjobs,jobs,pods      workstation: Mon Feb 13 11:58:44 2024
```

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST	SCHEDULE	AGE
cronjob.batch/image-pruner	*/4 * * * *	False	0	30s		2m
NAME	COMPLETIONS	DURATION	AGE			
job.batch/image-pruner-27883660	1/1	9s	30s			

NAME	READY	STATUS	RESTARTS	AGE
pod/image-pruner-27883660-2ghvv	0/1	Completed	0	30s

4.7. Get the logs of the pod that executed the maintenance task.

```
[student@workstation appsec-prune]$ oc logs pod/image-pruner-27883660-2ghvv | tail
...output omitted...
+ crictl rmi --prune
E0106 18:08:31.686489 374926 remote_image.go:266] "RemoveImage from image service failed" err="rpc error: code = Unknown desc = Image used by 0c9ab998...: image is in use by a container" image="c464e04f..." ❶
Deleted: registry.ocp4.example.com:8443/ubi7/nginx-118:latest
Deleted: registry.ocp4.example.com:8443/ubi8/nginx-118:latest
Deleted: registry.ocp4.example.com:8443/ubi9/nginx-120:latest

Removing debug pod ...
...output omitted...
```

❶ You can ignore the error that a container is using the image.

► 5. Clean up resources.

5.1. Change to the student user home directory.

```
[student@workstation appsec-prune]$ cd
[student@workstation ~]$
```

5.2. Ensure that you are working on the appsec-prune project.

```
[student@workstation ~]$ oc project
Using project "appsec-prune" on server "https://api.ocp4.example.com:6443".
```

5.3. Remove the cron job resource and the configuration map.

```
[student@workstation ~]$ oc delete cronjob/image-pruner configmap/maintenance
cronjob.batch "image-pruner" deleted
configmap "maintenance" deleted
```

5.4. Remove the security constraint from the service account.

```
[student@workstation ~]$ oc adm policy remove-scc-from-user \
-z image-pruner privileged
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:privileged removed:
"image-pruner"
```

5.5. Remove the role from the service account.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-user \
cluster-admin -z image-pruner
clusterrole.rbac.authorization.k8s.io/cluster-admin removed: "image-pruner"
```

5.6. Delete the appsec-prune project.

```
[student@workstation ~]$ oc delete project appsec-prune prune-apps
project.project.openshift.io "appsec-prune" deleted
project.project.openshift.io "prune-apps" deleted
```

Finish

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

```
[student@workstation ~]$ lab finish appsec-prune
```

► Lab

Application Security

Deploy an application that requires additional operating system privileges to run.

Deploy an application that requires access to the Kubernetes APIs to perform cluster maintenance tasks.

Outcomes

- Deploy a cluster maintenance application that must be executed regularly.
- Grant application access to Kubernetes APIs.
- Run an application with a security context constraint (SCC).

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 start appsec-review
```

In this exercise, you deploy two applications:

- A legacy payroll application that must run as the fixed 0 UID to open the TCP 80 port.
- A project cleaner deletes projects with the `appsec-review-cleaner` label and that are longer than 10 seconds. This short expiration time is deliberate for the lab purposes.

You must deploy the project cleaner application to delete obsolete projects every minute.

The `lab start` command copies the required files for the exercise to the `lab` directory:

- A deployment manifest with the payroll application.
- A pod manifest that contains a project cleaner application. You can use this pod to test the project cleaner application and copy the pod specification into the cron job to complete the exercise.
- A manifest with the `project-cleaner` cluster role that grants the application access to find and delete namespaces.
- A cron job template file that you can edit to create cron jobs.
- A script that generates projects to verify that the project cleaner application works.

Instructions

1. Log in to your OpenShift cluster as the `developer` user with the `developer` password and create the `appsec-review` project.
2. Change to the `~/D0280/labs/appsec-review` directory and deploy the payroll application in the `payroll-app.yaml` file. Verify that the application cannot run.

3. As the `admin` user, look for an SCC that allows the workload in the `payroll-app.yaml` deployment to run.
4. Create the `payroll-sa` service account and assign to it the SCC that the application requires. Then, assign the `payroll-sa` service account to the `payroll-api` deployment.
5. Verify that the payroll API is accessible by running the `curl` command from the `payroll-api` deployment. Use the `http://localhost/payments/status` URL to verify that the API is working.
6. Create the `project-cleaner-sa` service account and assign it to the `project-cleaner.yaml` pod manifest to configure the application permissions.
7. Create the `project-cleaner` role in the `cluster-role.yaml` file and assign it to the `project-cleaner-sa` service account.
8. Edit the `cron-job.yaml` file to create the `appsec-review-cleaner` cron job by using the `project-cleaner.yaml` pod manifest as the job template. Create the cron job and configure it to run every minute. You can use the solution file in the `~/D0280/solutions/appsec-review/cron-job.yaml` path.
9. Optionally, verify that the project cleaner executed correctly. Use the `generate-projects.sh` script from the lab directory to generate projects for deletion. Wait for the next job execution and print the logs from that job's pod.

**Note**

The logs might not be in the last pod, but in the previous one.

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 grade appsec-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Application Security

Deploy an application that requires additional operating system privileges to run.

Deploy an application that requires access to the Kubernetes APIs to perform cluster maintenance tasks.

Outcomes

- Deploy a cluster maintenance application that must be executed regularly.
- Grant application access to Kubernetes APIs.
- Run an application with a security context constraint (SCC).

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 start appsec-review
```

In this exercise, you deploy two applications:

- A legacy payroll application that must run as the fixed 0 UID to open the TCP 80 port.
- A project cleaner deletes projects with the `appsec-review-cleaner` label and that are longer than 10 seconds. This short expiration time is deliberate for the lab purposes.

You must deploy the project cleaner application to delete obsolete projects every minute.

The `lab start` command copies the required files for the exercise to the lab directory:

- A deployment manifest with the payroll application.
- A pod manifest that contains a project cleaner application. You can use this pod to test the project cleaner application and copy the pod specification into the cron job to complete the exercise.
- A manifest with the `project-cleaner` cluster role that grants the application access to find and delete namespaces.
- A cron job template file that you can edit to create cron jobs.
- A script that generates projects to verify that the project cleaner application works.

Instructions

1. Log in to your OpenShift cluster as the `developer` user with the `developer` password and create the `appsec-review` project.

- 1.1. Log in 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 appsec-review project.

```
[student@workstation ~]$ oc new-project appsec-review
Now using project "appsec-review" on server ...
...output omitted...
```

2. Change to the ~/D0280/labs/appsec-review directory and deploy the payroll application in the payroll-app.yaml file. Verify that the application cannot run.

- 2.1. Change to the ~/D0280/labs/appsec-review directory to access the lab files.

```
[student@workstation ~]$ cd ~/D0280/labs/appsec-review
```

- 2.2. Run the oc apply command to create the payroll deployment.

```
[student@workstation appsec-review]$ oc apply -f payroll-app.yaml
deployment.apps/payroll-api created
```

- 2.3. Verify that the application fails to run by reading the deployment logs.

```
[student@workstation appsec-review]$ oc logs deployment/payroll-api
[2023-03-13 08:13:30 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2023-03-13 08:13:30 +0000] [1] [ERROR] Retrying in 1 second.
[2023-03-13 08:13:31 +0000] [1] [ERROR] Retrying in 1 second.
[2023-03-13 08:13:32 +0000] [1] [ERROR] Retrying in 1 second.
[2023-03-13 08:13:33 +0000] [1] [ERROR] Retrying in 1 second.
[2023-03-13 08:13:34 +0000] [1] [ERROR] Retrying in 1 second.
[2023-03-13 08:13:35 +0000] [1] [ERROR] Can't connect to ('', 80)
```

The container in the pod runs as root to listen on port 80.

3. As the admin user, look for an SCC that allows the workload in the payroll-app.yaml deployment to run.

- 3.1. Log in as the admin user with the redhatocp password.

```
[student@workstation appsec-review]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 3.2. Run the oc adm policy scc-subject-review command to get an SCC that allows the application to run.

```
[student@workstation appsec-review]$ oc adm policy scc-subject-review \
-f payroll-app.yaml
RESOURCE          ALLOWED BY
Deployment/payroll-api  anyuid
```

4. Create the payroll-sa service account and assign to it the SCC that the application requires. Then, assign the payroll-sa service account to the payroll-api deployment.

- 4.1. Run the oc create command to create the payroll-sa service account.

```
[student@workstation appsec-review]$ oc create sa payroll-sa
serviceaccount/payroll-sa created
```

- 4.2. Assign the anyuid SCC to the payroll-sa service account.

```
[student@workstation appsec-review]$ oc adm policy \
add-scc-to-user anyuid -z payroll-sa
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "payroll-sa"
```

- 4.3. Use the oc set serviceaccount command to add the payroll-sa service account to the payroll-api deployment.

```
[student@workstation appsec-review]$ oc set serviceaccount deployment \
payroll-api payroll-sa
deployment.apps/payroll-api serviceaccount updated
```

5. Verify that the payroll API is accessible by running the curl command from the payroll-api deployment. Use the http://localhost/payments/status URL to verify that the API is working.

- 5.1. Use the oc exec command with the payroll-api deployment to run the curl command. Provide the -sS option to hide progress output and show errors.

```
[student@workstation appsec-review]$ oc exec deployment/payroll-api \
-- curl -sS http://localhost/payments/status
[{"id":240,"status":"Paid","userId":1003},
 {"id":241,"status":"Pending","userId":1003}]
```

6. Create the project-cleaner-sa service account and assign it to the project-cleaner.yaml pod manifest to configure the application permissions.

- 6.1. Create the project-cleaner-sa service account.

```
[student@workstation appsec-review]$ oc create sa project-cleaner-sa
serviceaccount/project-cleaner-sa created
```

- 6.2. Edit the project-cleaner.yaml pod manifest file to use the project-cleaner-sa service account.

```

apiVersion: v1
kind: Pod
metadata:
  name: project-cleaner
  namespace: appsec-review
spec:
  restartPolicy: Never
  serviceAccountName: project-cleaner-sa
  containers:
    - name: project-cleaner
...output omitted...

```

7. Create the `project-cleaner` role in the `cluster-role.yaml` file and assign it to the `project-cleaner-sa` service account.
 - 7.1. Create the `project-cleaner` cluster role by applying the `cluster-role.yaml` manifest file.

```
[student@workstation appsec-review]$ oc apply -f cluster-role.yaml
clusterrole.rbac.authorization.k8s.io/project-cleaner created
```

- 7.2. Use the `oc adm policy add-clusterrole-to-user` command to add the `project-cleaner` role to the `project-cleaner-sa` service account.
8. Edit the `cron-job.yaml` file to create the `appsec-review-cleaner` cron job by using the `project-cleaner.yaml` pod manifest as the job template. Create the cron job and configure it to run every minute. You can use the solution file in the `~/DO280/solutions/appsec-review/cron-job.yaml` path.
 - 8.1. Edit the `cron-job.yaml` file to replace the `CHANGE_ME` string with the `*/1 * * * *` schedule to execute the job every minute.

```

apiVersion: batch/v1
kind: CronJob
metadata:
  name: appsec-review-cleaner
  namespace: appsec-review
spec:
  schedule: "*/1 * * * *"
  concurrencyPolicy: Forbid
  jobTemplate:
...output omitted...

```

- 8.2. Replace the `CHANGE_ME` label in the `jobTemplate` definition with the `spec` definition from the `project-cleaner.yaml` pod manifest. Although the long image name might show across two lines, you must add it as one line.

```

apiVersion: batch/v1
kind: CronJob
metadata:
  name: appsec-review-cleaner
  namespace: appsec-review
spec:
  schedule: "*/1 * * * *"
  concurrencyPolicy: Forbid
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          serviceAccountName: project-cleaner-sa
          containers:
            - name: project-cleaner
              image: registry.ocp4.example.com:8443/redhattraining/do280-project-
cleaner:v1.1
              imagePullPolicy: Always
              env:
                - name: "PROJECT_TAG"
                  value: "appsec-review-cleaner"
                - name: "EXPIRATION_SECONDS"
                  value: "10"

```

8.3. Create the cron job.

```
[student@workstation appsec-review]$ oc apply -f cron-job.yaml
cronjob.batch/appsec-review-cleaner created
```

9. Optionally, verify that the project cleaner executed correctly. Use the `generate-projects.sh` script from the lab directory to generate projects for deletion. Wait for the next job execution and print the logs from that job's pod.



Note

The logs might not be in the last pod, but in the previous one.

- 9.1. Run the `generate-projects.sh` script to create test projects that the project cleaner will delete the next time that it runs.

```
[student@workstation appsec-review]$ ./generate-projects.sh
obsolete-appsec-review-1 created at 15:29:14
obsolete-appsec-review-2 created at 15:29:15
obsolete-appsec-review-3 created at 15:29:16
namespace/obsolete-appsec-review-1 labeled
namespace/obsolete-appsec-review-2 labeled
namespace/obsolete-appsec-review-3 labeled
Last appsec-review-cleaner label applied at 15:29:20
...output omitted...
```

- 9.2. List the pods in the `appsec-review` project until you see a pod with the `Completed` status that is later than the last label that the script applied.

```
[student@workstation appsec-review]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
appsec-review-cleaner-27909204-g49gr   0/1     Completed   0          2m37s
appsec-review-cleaner-27909205-q2f2t   0/1     Completed   0          97s
appsec-review-cleaner-27909206-xcswb   0/1     Completed   0          37s
```

- 9.3. Print the logs from the last completed job, to verify that it deleted the obsolete projects.

```
[student@workstation appsec-review]$ oc logs pod/appsec-review-cleaner-27909206-xcswb
...output omitted...
Namespace 'obsolete-appsec-review-1' deleted
Namespace 'obsolete-appsec-review-2' deleted
Namespace 'obsolete-appsec-review-3' deleted
```

- 9.4. Change to the home directory to prepare for the next exercise.

```
[student@workstation appsec-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 grade appsec-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- Security context constraints (SCCs) limit the access from a running pod in OpenShift to the host environment.
- An application can assign an SCC to the application service account to use it.
- With the Kubernetes APIs, a user or an application can query and modify the cluster state.
- To give an application access to the Kubernetes APIs, you can create roles or cluster roles that describe the application requirements, and assign those roles to the application service account.
- You can automate cluster and application management tasks by creating Kubernetes cron jobs that run periodic management jobs.

Chapter 9

OpenShift Updates

Goal

Update an OpenShift cluster and minimize disruption to deployed applications.

Objectives

- Describe the cluster update process.
- Identify applications that use deprecated Kubernetes APIs.
- Update OLM-managed operators by using the web console and CLI.

Sections

- The Cluster Update Process (and Quiz)
- Detect Deprecated Kubernetes API Usage (and Quiz)
- Update Operators with the OLM (and Quiz)
- OpenShift Updates (Quiz)

The Cluster Update Process

Objectives

- 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 upgrade path to update your cluster and the underlying operating system. With this new distribution system, OpenShift clusters can perform Over-the-Air updates (OTA).

This software distribution system for OTA manages the controller manifests, cluster roles, and any other resources to update a cluster to a particular version. With this feature, a cluster can run the 4.14.x version seamlessly. With OTA, a cluster can 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://console.redhat.com/openshift>, and hosts cluster images at <https://quay.io>. You use a single interface to manage the lifecycle of all your OpenShift clusters. With OTA, you can update faster by skipping intermediate versions. For example, you can update from 4.14.1 to 4.14.3, and thus bypass 4.14.2.



Important

Starting with OpenShift 4.10, the OTA system requires a persistent connection to the internet. For more information about how to update disconnected clusters, consult the *Updating a Restricted Network Cluster* chapter in the references section.

Name	Status	Type	Created	Version	Provider (Region)
[REDACTED]	Ready	ROSA	08 Feb 2024	4.12.30	AWS (us-east-2)
[REDACTED]	Ready	ROSA	23 Jan 2024	4.12.30	AWS (us-east-2)
[REDACTED]	Ready	OCP	30 days left	4.6.62	OpenStack

Figure 9.1: Managing clusters at cloud.redhat.com

The service defines *upgrade paths* that correspond to cluster eligibility for certain updates. Upgrade paths belong to update channels. Consider a channel 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 the following parts: the tier (release candidate, fast, stable, and extended update support), the major version (4), and the minor version (.12). Example channel names include: `candidate-4.14`, `fast-4.14`, `stable-4.14`, and `eus-4.14`. Each channel delivers patches for a given cluster version.

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.



Important

Red Hat does not support the updates that are listed only in the *candidate* channel.

The Fast Channel

The *fast* channel delivers updates as soon as Red Hat declares the given version as a general availability release. Red Hat supports the updates that are released in this channel, and it is best suited to development and QA environments.



Note

Customers can help to improve OpenShift by joining the Red Hat connected customers program. If you join this program, then your cluster is registered to the *fast* channel.

The Stable Channel

Red Hat support and site reliability engineering (SRE) teams monitor operational clusters with the updates from the *fast* channel. If operational clusters pass additional testing and validation, then updates in the *fast* channel are enabled in the *stable* channel. Red Hat supports the updates that are released in this channel, and it is best suited to production environments.

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 any unforeseen problems in OpenShift clusters that testing did not reveal.

The Extended Update Support Channel

Starting with OpenShift Container Platform 4.8, Red Hat denotes all even-numbered minor releases (for example, 4.8, 4.10, 4.12, and 4.14) as Extended Update Support (EUS) releases.

EUS releases have no difference between `stable-4.x` and `eus-4.x` channels (where *x* denotes the even-numbered minor release) until OpenShift Container Platform moves to the EUS phase. You can switch to the EUS channel as soon as it becomes available.

Support Status for Update Channels

Red Hat offers support for all released updates in the *fast*, *stable*, and *eus* update channels. Red Hat supports the released updates in the *candidate* channel only if they are also listed in the *fast* or *stable* channels.

Update channel	Support status
candidate-4.x	Supported if the update is also listed in the <i>fast</i> or <i>stable</i> channels.
fast-4.x	Supported
stable-4.x	Supported
eus-4.x	Supported



Note

The x in the channel name denotes the minor version.

Upgrade Paths

You can apply each of the upgrade channels to a Red Hat OpenShift Container Platform version 4.14 cluster in different environments. The following paragraphs describe an example scenario where the 4.14.3 version has a defect.

Stable channel

When using the `stable-4.14` channel, you can upgrade your cluster from 4.14.0 to 4.14.1 or to 4.14.2. If an issue is discovered in the 4.14.3 release, then you cannot upgrade to that version. When a patch becomes available in the 4.14.4 release, you can update your cluster to that version.

This channel is suited to production environments, because the Red Hat SRE teams and support services test the releases in that channel.

Fast channel

The `fast-4.14` channel can deliver 4.14.1 and 4.14.2 updates but not 4.14.3. Red Hat also supports this channel, and you can apply it to development, QA, or production environments.

Administrators must specifically choose a different minor version channel, such as `fast-4.14`, to upgrade to a new release in a new minor version when it becomes available.

Candidate channel

You can use the `candidate-4.14` channel to install the latest features of OpenShift. With this channel, you can upgrade to all *z-stream* releases, such as 4.14.1, 4.14.2, and 4.14.3.

You use this channel to access the latest features of the product as they get released. This channel is suited to development and pre-production environments.

EUS channel

When switching to the `eus-4.14` channel, the `stable-4.14` channel does not receive *z-stream* updates until the next EUS version becomes available.

**Note**

Starting with OpenShift Container Platform 4.8, Red Hat denotes all even-numbered minor releases as Extended Update Support (EUS) releases.

The following graphic describes the update graphs for the *stable* and *candidate* channels:

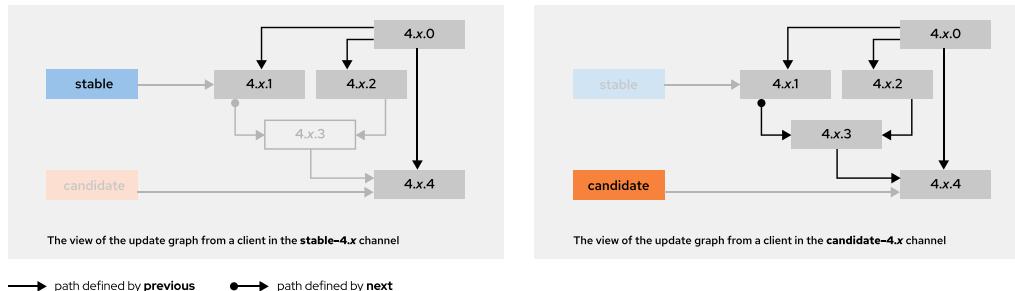


Figure 9.2: Update graphs for stable and candidate channels

Red Hat provides support for the General Availability (GA) updates that are released in the *stable* and *fast* channels. Red Hat does not support updates that are listed only in the *candidate* channel.

To ensure the stability of the cluster and the proper level of support, switch only from a *stable* channel to a *fast* channel. Although it is possible to switch from a *stable* channel or a *fast* channel to a *candidate* channel, it is not recommended. The *candidate* channel is best suited to testing feature acceptance and to assist in qualifying the next version of OpenShift Container Platform.

**Note**

The release of updates for patch and security 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 `eus-4.14`, `stable-4.14`, `fast-4.14`, or `candidate-4.14` by using the web console or the OpenShift CLI client:

Web console

Navigate to the Administration > Cluster Settings page on the details tab, and then click the pencil icon.

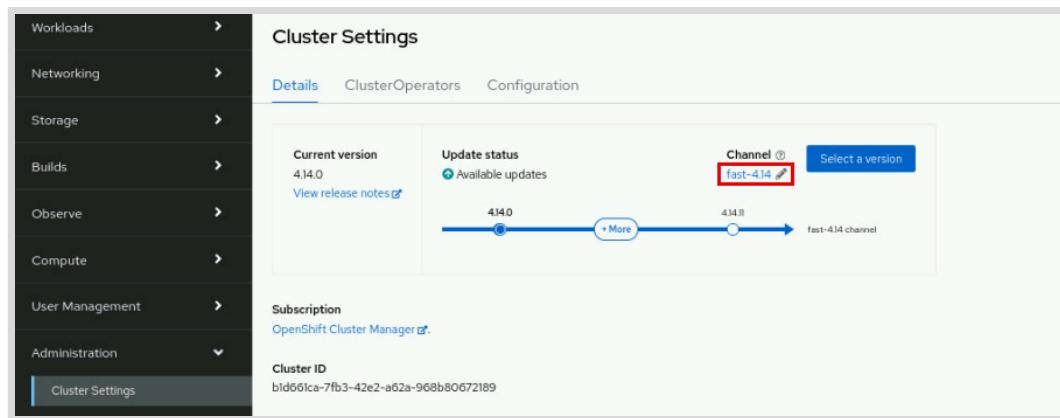


Figure 9.3: Current update channel in the web console

A window displays options to select an update channel.

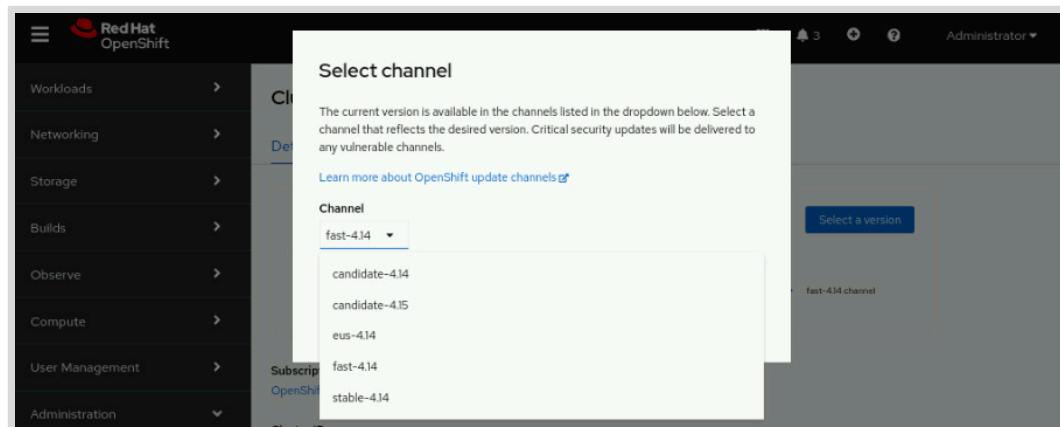


Figure 9.4: Changing the update channel in the web console

Command line

Execute the following command to switch to another update channel by using the oc client. You can also switch to another update channel, such as `stable-4.14`, to update to the next minor version of OpenShift Container Platform.

```
[user@host ~]$ oc patch clusterversion version --type="merge" \
--patch '{"spec":{"channel":"fast-4.14"}}'
clusterversion.config.openshift.io/version patched
```

Pausing the Machine Health Check Resource

During the upgrade process, nodes in the cluster might become temporarily unavailable. In the case of worker nodes, the machine health check might identify such nodes as unhealthy and reboot them. To avoid rebooting such nodes, pause all the machine health check resources before updating the cluster.



Note

The prerequisite to pause the machine health check resources is not required on single-node installations.

Run the following command to list all the available machine health check resources.

```
[user@host ~]$ oc get machinehealthcheck -n openshift-machine-api
NAME                                     MAXUNHEALTHY EXPECTEDMACHINES CURRENTHEALTHY
machine-api-termination-handler  100%
```

Add the `cluster.x-k8s.io/paused` annotation to the machine health check resource to pause it before updating the cluster.

```
[user@host ~]$ oc annotate machinehealthcheck -n openshift-machine-api \
  machine-api-termination-handler cluster.x-k8s.io/paused=""
machinehealthcheck.machine.openshift.io/machine-api-termination-handler annotated
```

Remove the annotation after the cluster is updated.

```
[user@host ~]$ oc annotate machinehealthcheck -n openshift-machine-api \
  machine-api-termination-handler cluster.x-k8s.io/paused-
machinehealthcheck.machine.openshift.io/machine-api-termination-handler annotated
```

Over-the-air Updates

OTA follows a client-server approach. Red Hat hosts the cluster images and the update infrastructure. OTA generates all possible update paths for your cluster. OTA also 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.

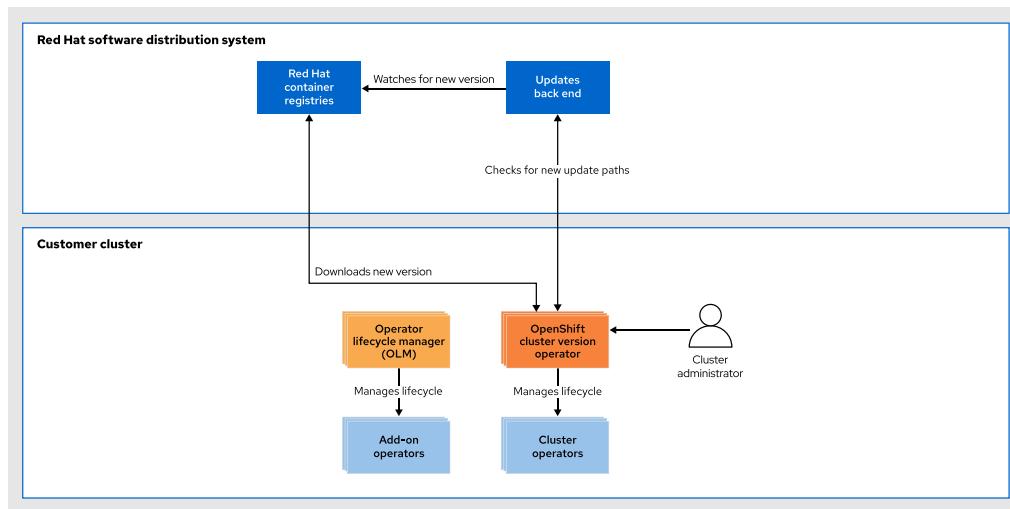


Figure 9.5: OpenShift Container Platform updates architecture

With telemetry, Red Hat can determine the update path. The cluster uses a Prometheus-based Telemeter component 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 *Data Collection* and *Telemeter Sample Metrics* documents 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 vendor (ISV) operators.

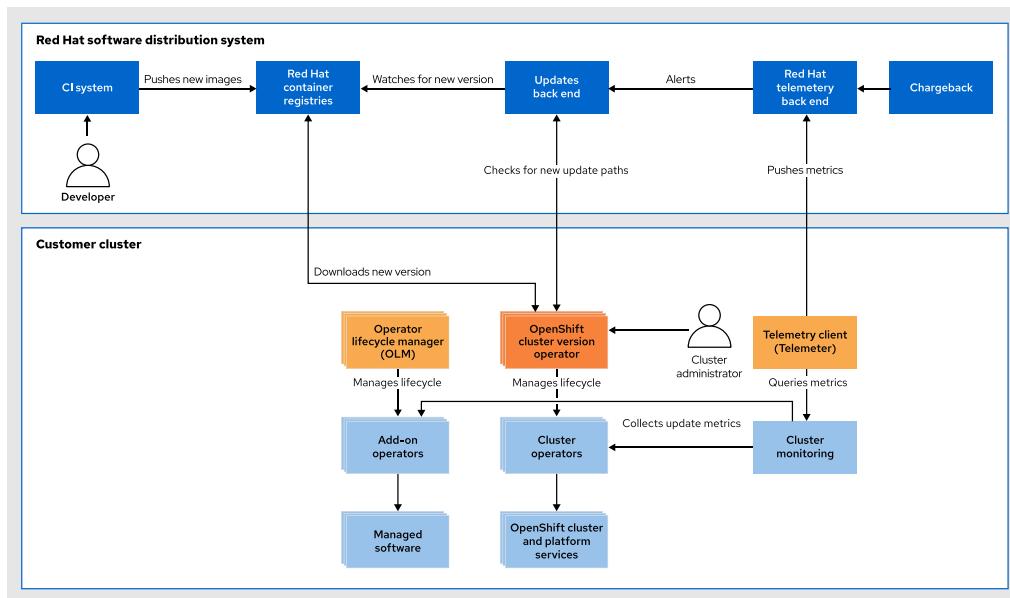


Figure 9.6: Managing cluster updates by using telemetry

The Update Process

The following components are involved in the cluster 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

The OLM orchestrates updates to any operators that are running in the cluster.

Updating the Cluster

You can update the cluster via the web console or from the command line. The **Administration** > **Cluster Settings** page displays an update status of *Available updates* when a new update is available. From this page, click **Select a version**, and then select the version and the cluster update option that you want to install:

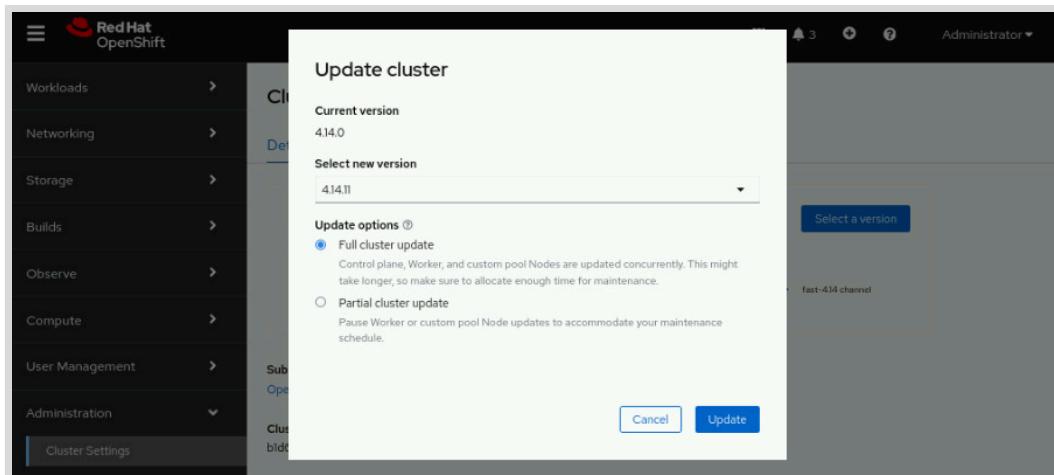


Figure 9.7: Update the cluster by using the web console

**Important**

Rolling back your cluster to an earlier version is not supported. **If your update is failing to complete, contact Red Hat support.**

The update process also updates the underlying operating system when updates are available. The updates use 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.

Update the Cluster by Using the Command Line

The following steps describe the procedure for updating a cluster as a cluster administrator by using the command-line interface:

- Be sure to update all operators that are installed through the OLM to the 4.14 version before updating the OpenShift cluster.
- Retrieve the cluster version and review the current update channel information. If you are running the cluster in production, then ensure that the channel reads `stable`.

```
[user@host ~]$ oc get clusterversion
NAME      VERSION      AVAILABLE      PROGRESSING      SINCE      STATUS
version   4.14.0       True          False           2d         Cluster version is 4.14.0

[user@host ~]$ oc get clusterversion -o jsonpath='{.items[0].spec.channel}{"\n"}'
stable-4.14
```

- View the available updates and note the version number of the update to apply.

```
[user@host ~]$ oc adm upgrade
Cluster version is 4.14.0

Upstream is unset, so the cluster will use an appropriate default.
```

Chapter 9 | OpenShift Updates

```
Channel: stable-4.14 (available channels: candidate-4.14, candidate-4.15,  
eus-4.14, fast-4.14, stable-4.14)
```

Recommended updates:

VERSION	IMAGE
4.14.10	quay.io/openshift-release-dev/ocp-release@sha256:...
...output omitted...	

- 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@host ~]$ 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@host ~]$ oc adm upgrade --to=VERSION
```

- 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@host ~]$ oc get clusterversion  
NAME      VERSION  AVAILABLE  PROGRESSING  SINCE    STATUS  
version   4.14.0   True       True        1m       Working towards 4.14.10 ...  
  
[user@host ~]$ oc get clusteroperators  
NAME                  VERSION  AVAILABLE  PROGRESSING  DEGRADED  ...  
authentication        4.14.0   True       False       False     ...  
baremetal             4.14.10 False      True       False     ...  
cloud-controller-manager  4.14.10 True       False       True     ...  
...output omitted...
```

- Use the following command to review the cluster version history and 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 that were applied to the cluster. This list 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, then updates in the history have a **Completed** state. Otherwise, the update has a **Partial** state if it failed or did not complete.

```
[user@host ~]$ oc describe clusterversion  
...output omitted...  
History:  
Completion Time: 2024-02-10T04:38:12Z  
Image: quay.io/openshift-release-dev/ocp-release@sha256:...  
Started Time: 2024-02-10T03:35:05Z  
State: Partial  
Verified: true  
Version: 4.14.10
```

```
Completion Time: 2024-02-10T12:39:02Z
Image: quay.io/openshift-release-dev/ocp-release@sha256:...
Started Time: 2024-02-10T12:23:14Z
State: Completed
Verified: false
Version: 4.14.10
```

**Important**

When an update is failing to complete, the Cluster Version Operator (CVO) reports the status of any blocking components and attempts to reconcile the update.

Rolling back your cluster to a previous version is not supported. **If your update is failing to complete, contact Red Hat support.**

- After the process completes, you can confirm that the cluster is updated to the new version.

```
[user@host ~]$ oc get clusterversion
NAME      VERSION  AVAILABLE  PROGRESSING  SINCE    STATUS
version   4.14.10  True       False        30m      Cluster version is 4.14.10
```



References

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 Red Hat OpenShift Container Platform 4.14 *Updating Clusters* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/updating_clusters/index#updating-restricted-network-cluster

For more information about updating operators that are installed through the Operator Lifecycle Manager, refer to the *Upgrading Installed Operators* section in the *Administrator Tasks* chapter in the Red Hat OpenShift Container Platform 4.14 *Working with Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-upgrading-operators

For more information about performing an EUS-to-EUS update, refer to the *Preparing to Perform an EUS-to-EUS Update* chapter in the Red Hat OpenShift Container Platform 4.14 *Updating Clusters* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/updating_clusters/index#updating-eus-to-eus-upgrade_eus-to-eus-upgrade

For more information about the OpenShift Container Platform upgrade paths, visit the following page in the customer portal:
<https://access.redhat.com/solutions/4583231>

For more information about the OpenShift Container Platform update graph, visit the following page in the customer portal:
https://access.redhat.com/labs/ocpupgrade/graph/update_path

For more information about OpenShift Extended Update Support (EUS), visit the following page in the customer portal:
<https://access.redhat.com/support/policy/updates/openshift-eus>

For more information about the OpenShift Container Platform lifecycle policy, visit the following page in the customer portal:
<https://access.redhat.com/support/policy/updates/openshift>

OpenShift 4 Data Collection

<https://github.com/openshift/cluster-monitoring-operator/blob/master/Documentation/data-collection.md>

OpenShift 4 Telemeter Sample Metrics

<https://github.com/openshift/cluster-monitoring-operator/blob/master/Documentation/sample-metrics.md>

► Quiz

The Cluster Update Process

Choose the correct answers to the following questions:

- ▶ 1. **Which component 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)

- ▶ 2. **Which component manages the updates of operators that are not cluster operators?**
 - a. Operator Lifecycle Manager (OLM)
 - b. Telemetry client (Telemeter)
 - c. Cluster Version Operator (CVO)

- ▶ 3. **Which two commands can retrieve the currently running cluster version? (Choose two.)**
 - a. oc get updatechannels
 - b. oc adm upgrade
 - c. oc get clusterchannel
 - d. oc get clusterversion
 - e. oc get clusterupgrades

- ▶ 4. **Which two channels are classified as general availability? (Choose two.)**
 - a. candidate-4.14
 - b. fast-4.14
 - c. stable-4.14
 - d. eus-4.14

- ▶ 5. **Which statement is true regarding the OTA feature?**
 - a. The *stable* channel is classified as General Availability (GA), whereas the *fast* channel is classified as a Release Candidate (RC).
 - b. When using the *stable* channel, you cannot skip intermediary versions. For example, when updating from 4.14.8 to 4.14.10, OpenShift must install the 4.14.9 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. Red Hat supports rolling back a failed update only when it was performed on z-stream versions of the same minor version (for example, from 4.14.2 to 4.14.3, but not from 4.12.3 to 4.14.1).

► Solution

The Cluster Update Process

Choose the correct answers to the following questions:

- ▶ 1. Which component 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)
- ▶ 2. Which component manages the updates of operators that are not cluster operators?
 - a. Operator Lifecycle Manager (OLM)
 - b. Telemetry client (Telemeter)
 - c. Cluster Version Operator (CVO)
- ▶ 3. Which two commands can retrieve the currently running cluster version? (Choose two.)
 - a. `oc get updatechannels`
 - b. `oc adm upgrade`
 - c. `oc get clusterchannel`
 - d. `oc get clusterversions`
 - e. `oc get clusterupgrades`
- ▶ 4. Which two channels are classified as general availability? (Choose two.)
 - a. `candidate-4.14`
 - b. `fast-4.14`
 - c. `stable-4.14`
 - d. `eus-4.14`
- ▶ 5. Which statement is true regarding the OTA feature?
 - a. The `stable` channel is classified as General Availability (GA), whereas the `fast` channel is classified as a Release Candidate (RC).
 - b. When using the `stable` channel, you cannot skip intermediary versions. For example, when updating from 4.14.8 to 4.14.10, OpenShift must install the 4.14.9 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. Red Hat supports rolling back a failed update only when it was performed on z-stream versions of the same minor version (for example, from 4.14.2 to 4.14.3, but not from 4.12.3 to 4.14.1).

Detect Deprecated Kubernetes API Usage

Objectives

- Identify applications that use deprecated Kubernetes APIs.

OpenShift Versions

Kubernetes is an open source container orchestration engine for automating the deployment, scaling, and management of containerized applications. The OpenShift Container Platform foundation is based on Kubernetes and therefore shares the underlying technology. The following table lists the OpenShift version and the Kubernetes version that it is based on:

OpenShift version	Kubernetes version
4.12	1.25
4.13	1.26
4.14	1.27

Kubernetes API Deprecation Policy

The Kubernetes API versions are categorized based on feature maturity (experimental, pre-release, and stable).

API version	Category	Description
v1alpha1	Alpha	Experimental features
v1beta1	Beta	Pre-release features
v1	Stable	Stable features, generally available

Use the following command to view the current version of a resource:

```
[user@host ~]$ oc api-resources | egrep '^NAME|cronjobs'
NAME      SHORTNAMES   APIVERSION NAMESPACED KIND
cronjobs  cj           batch/v1    true      CronJob
```

When a *stable* version of a feature is released, the *beta* versions are marked as deprecated and are removed after three Kubernetes releases. If a request uses a deprecated API version, then the API server returns a deprecation warning that includes the name of the current version of the cluster.

```
[user@host ~]$ egrep 'kind|apiVersion' cronjob-beta.yaml
kind: CronJob
apiVersion: batch/v1beta1

[user@host ~]$ oc create -f cronjob-beta.yaml
Warning: batch/v1beta1 CronJob is deprecated in v1.21+, unavailable in v1.25+;
use batch/v1 CronJob
cronjob.batch/hello created
```

If a request uses an API version that Kubernetes removed, then the API server returns an error, because that API version is not supported in the cluster.

```
[user@host ~]$ egrep 'kind|apiVersion' cronjob-alpha.yaml
apiVersion: batch/v1alpha1
kind: CronJob

[user@host ~]$ oc create -f cronjob-alpha.yaml
error: resource mapping not found for name: "hello" namespace: "" from "cronjob-alpha.yaml": no matches for kind "CronJob" in version "batch/v1beta1"
ensure CRDs are installed first
```

Deprecated and Removed Features in Kubernetes

The Kubernetes 1.27 release stopped serving some API versions that were marked as deprecated in previous releases. The following table contains a short list of the deprecated and removed API versions.

Resource	Removed API Group	Current API Group
CSIStorageCapacity	storage.k8s.io/v1beta1	storage.k8s.io/v1



Note

For more information about the API versions that are deprecated and removed in Kubernetes, consult *Kubernetes Deprecated API Migration Guide* in the references section.

Identifying Deprecated APIs

You can identify from the API request count whether a workload uses a deprecated API version. The API request count output contains four columns. A value in the REMOVEDINRELEASE column indicates that the API version is deprecated and specifies the Kubernetes version that will remove it.

```
[user@host ~]$ oc get apirequestcounts | awk '{if(NF==4){print $0}}'
NAME
      REMOVEDINRELEASE   REQUESTSINCURRENT HOUR   REQUESTSINLAST24H
...output omitted...
cronjobs.v1beta1.batch
      1.25           15                  44
horizontalpodautoscalers.v2beta2.autoscaling
      1.26            6                  30
```

```
podsecuritypolicies.v1beta1.policy
1.25
...output omitted...
```

28

77

A blank REMOVEDINRELEASE column indicates that the current API version will be kept in future releases. Even if an API has a blank value in the REMOVEDINRELEASE column, it might be deprecated; although such an API is not directly scheduled for removal in the next release, it might still be removed in the future.



Note

You can use a JSONPath filter to retrieve the results. The FILTER variable is written on a single line.

```
[user@host ~]$ FILTER='{range .items[?(@.status.removedInRelease!="")]}{.status.removedInRelease}"\t"{.status.requestCount}"\t".metadata.name"\n"}{end}'
[user@host ~]$ oc get apirequestcounts -o jsonpath="${FILTER}" | \
column -t -N "RemovedInRelease,RequestCount,Name"
RemovedInRelease RequestCount Name
1.25           44      cronjobs.v1beta1.batch
...output omitted...
```

If the command does not retrieve any information, then it indicates that none of the installed APIs are deprecated.

You can use a JSONPath filter for a list of actions for that resource and who did them.

```
[user@host ~]$ FILTER='{range .status.currentHour..byUser[*]}.{.byVerb[*].verb}\
{}{.username}{.userAgent}"\n"}{end}'
[user@host ~]$ TYPE=apirequestcount.apiserver.openshift.io/cronjobs.v1.batch
[user@host ~]$ echo ${TYPE} ; oc get ${TYPE} -o jsonpath="${FILTER}" | \
column -t -s ',' -N "Verbs,Username,UserAgent"

apirequestcount.apiserver.openshift.io/cronjobs.v1.batch
Verbs      Username      UserAgent
get update system:serviceaccount:kube-system:cronj...  kube-controller-manager/
v1...
watch      system:kube-controller-manager          kube-controller-manager/
v1...
...output omitted...
```

Deprecated and Removed Features in OpenShift

Red Hat OpenShift Container Platform (RHOC) is a set of modular components and services that are built on top of a Kubernetes container infrastructure.

Some features that were available in previous OpenShift releases are deprecated or removed. A deprecated feature is not recommended for new deployments, because a future release will remove it. The following table contains a short list of the deprecated and removed features in OpenShift.

OpenShift 4.12	OpenShift 4.13	OpenShift 4.14	Feature
General Availability	General Availability	Deprecated	Operator lifecycle and development deprecated
Deprecated	Deprecated	Deprecated	CoreDNS wildcard queries for the <code>cluster.local</code> domain
Deprecated	Deprecated	Deprecated	Persistent storage that uses FlexVolume
Not Available	General Availability	Removed	<code>--include-local-oci-catalogs</code> parameter for <code>oc-mirror</code>
General Availability	General Availability	Deprecated	<code>DeploymentConfig</code> objects

**Note**

For more information about the deprecated and removed API versions in Kubernetes, consult the *OpenShift Container Platform 4.14 release notes* in the references section.

Deprecated API Alerts in OpenShift

OpenShift includes two alerts that are triggered when a workload uses a deprecated API version:

APIRemovedInNextReleaseInUse

This alert is triggered for APIs that OpenShift Container Platform will remove in the next release.

APIRemovedInNextEUSReleaseInUse

This alert is triggered for APIs that OpenShift Container Platform Extended Update Support (EUS) will remove in the next release.

The alert describes the situation with context to identify the affected workload.

The screenshot shows the Red Hat OpenShift web interface. On the left, there's a sidebar with navigation links like Operators, Workloads, Networking, Storage, Builds, Observe (with sub-links Alerting, Metrics, Dashboards, Targets), Compute, and User Management. The 'Alerting' link under 'Observe' is currently selected. The main content area is titled 'Alerting rule details' for an alert named 'APIRemovedInNextReleaseInUse'. It shows the alert's name, source (Platform), severity (Info, set to fire at 1h), and a detailed description explaining it's for a deprecated API being removed in the next version. Below the description is a 'Summary' section. On the right, there's a code block showing the Prometheus query used to generate the alert.

```

group by (group, version, resource, removed_release)
/apiserver_requested_deprecated_apis[removed_releases="1.28"] * on (group, version, resource) group_left ()
sum by (group, version, resource)
(rate(apiserver_request_total{system_client!="cluster-policy-controller",system_client!="kube-controller-manager"}[4h])) > 0

```

Figure 9.8: Deprecated API alert

You can extract the alerts in JSON format from the Prometheus stateful set, and then filter the result to retrieve the deprecated API alerts.

```
[user@host ~]$ oc exec -it statefulset/prometheus-k8s -c prometheus \
-n openshift-monitoring -- \
curl -fsSL 'http://localhost:9090/api/v1/alerts' | jq . > alerts.json

[user@host ~]$ jq '[.data.alerts[] | 
select(.labels.alertname=="APIRemovedInNextReleaseInUse" or
.labels.alertname=="APIRemovedInNextEUSReleaseInUse")]' < alerts.json
[{"labels": {"alertname": "APIRemovedInNextReleaseInUse", ...output omitted...}, "state": "firing", ...output omitted...}, {"labels": {"alertname": "APIRemovedInNextEUSReleaseInUse", ...output omitted...}, "state": "firing", ...output omitted...}]
]
```



Note

If the output of the `jq` command is an empty JSON array [], then the alerts were not reported.

Explicit Acknowledgment Before Cluster Updates

OpenShift Container Platform 4.14 uses Kubernetes 1.27, which removed deprecated v1beta1 APIs.

OpenShift Container Platform requires an administrator to provide a manual acknowledgment before the cluster can be upgraded from version 4.13 to 4.14. This requirement helps to prevent issues after upgrading to OpenShift Container Platform 4.14, where workloads, tools, or other components that run on or interact with the cluster still use removed APIs.

Administrators must evaluate their cluster for workloads that use removed APIs, and migrate the affected components to the appropriate new API version. After migration, the administrator can provide an acknowledgment.

```
[user@host ~]$ oc patch configmap admin-acks -n openshift-config --type=merge \
--patch '{"data":{"ack-4.13-kube-1.27-api-removals-in-4.14":"true"}}'
configmap/admin-acks patched
```



References

For more information about the removed features in OpenShift, refer to the *Deprecated and Removed Features* section in the Red Hat OpenShift Container Platform 4.14 release notes at

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/release_notes/index#ocp-4-14-deprecated-removed-features

For more information about what version of the Kubernetes API is included with each OpenShift 4.x release, visit the following page in the customer portal:
<https://access.redhat.com/solutions/4870701>

For more information about the Kubernetes API deprecations and removals, visit the following page in the customer portal:
<https://access.redhat.com/articles/6955985>

For more information about the deprecated APIs in OpenShift Container Platform 4.14, visit the following page in the customer portal:
<https://access.redhat.com/articles/6955381>

For more information about how to get fired alerts on OpenShift by using the command-line, visit the following page in the customer portal:
<https://access.redhat.com/solutions/4250221>

What's New in Red Hat OpenShift 4.14

<https://www.redhat.com/en/whats-new-red-hat-openshift>

Preparing to Update to OpenShift Container Platform 4.14

https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/updating_clusters/index#rhel-micro-architecture-update-requirements

Kubernetes Deprecation Policy

<https://kubernetes.io/docs/reference/using-api/deprecation-policy/>

Kubernetes Deprecated API Migration Guide

<https://kubernetes.io/docs/reference/using-api/deprecation-guide/>

Kubernetes Removals and Deprecations in 1.27

<https://kubernetes.io/blog/2023/03/17/upcoming-changes-in-kubernetes-v1-27/>

Kubernetes 1.27 release announcement

<https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/>

► Quiz

Detect Deprecated Kubernetes API Usage

Choose the correct answers to the following questions:

- ▶ 1. **Red Hat OpenShift Container Platform 4.14 is based on which version of Kubernetes?**
 - a. Kubernetes 1.24
 - b. Kubernetes 1.27
 - c. Kubernetes 1.26
 - d. OpenShift Container Platform is not based on Kubernetes.
- ▶ 2. **What is the feature maturity status for Kubernetes resources with the v1beta1 API version?**
 - a. Experimental
 - b. Pre-release
 - c. Stable
- ▶ 3. **Which command can the cluster administrator use to identify deprecated API resources?**
 - a. oc get apirequestcounts
 - b. oc get deprecatedapis -n openshift-config
 - c. oc get apis --deprecated
 - d. oc get configmap deprecated-apis -n openshift-config
- ▶ 4. **Which two alerts identify the use of deprecated API versions in the OpenShift cluster? (Choose two.)**
 - a. APIRemovedInNextReleaseInUse
 - b. APIRequestCounts
 - c. APIRemovedInNextEUSReleaseInUse
 - d. DeprecatedAPIRequestCountsInUse
- ▶ 5. **True or False: OpenShift Container Platform requires administrators to provide a manual acknowledgement before applying an update that removes deprecated API versions.**
 - a. True
 - b. False

► Solution

Detect Deprecated Kubernetes API Usage

Choose the correct answers to the following questions:

- ▶ 1. **Red Hat OpenShift Container Platform 4.14 is based on which version of Kubernetes?**
 - a. Kubernetes 1.24
 - b. Kubernetes 1.27
 - c. Kubernetes 1.26
 - d. OpenShift Container Platform is not based on Kubernetes.

- ▶ 2. **What is the feature maturity status for Kubernetes resources with the v1beta1 API version?**
 - a. Experimental
 - b. Pre-release
 - c. Stable

- ▶ 3. **Which command can the cluster administrator use to identify deprecated API resources?**
 - a. `oc get apirequestcounts`
 - b. `oc get deprecatedapis -n openshift-config`
 - c. `oc get apis --deprecated`
 - d. `oc get configmap deprecated-apis -n openshift-config`

- ▶ 4. **Which two alerts identify the use of deprecated API versions in the OpenShift cluster? (Choose two.)**
 - a. `APIRemovedInNextReleaseInUse`
 - b. `APIRequestCounts`
 - c. `APIRemovedInNextEUSRReleaseInUse`
 - d. `DeprecatedAPIRequestCountsInUse`

- ▶ 5. **True or False: OpenShift Container Platform requires administrators to provide a manual acknowledgement before applying an update that removes deprecated API versions.**
 - a. True
 - b. False

Update Operators with the OLM

Objectives

- Update OLM-managed operators by using the web console and CLI.

Operator Updates

For operators that are installed in an OpenShift cluster, operator providers can release new versions. These new versions can contain bug fixes and new features. The Operator Lifecycle Manager (OLM) can update these operators.

However, new operator versions can introduce bugs and incompatibilities.

Cluster administrators should define operator update policies to ensure that bug fixes and new functions are adopted, with the cluster continuing to operate correctly.

OpenShift provides features to help to implement such policies.

- For each installed operator, you can decide whether the OLM automatically applies updates, or whether the updates require administrator approval.
- Operator providers can create multiple channels for an operator. The provider can follow different policies to push updates to each channel, so that each channel contains different versions of the operator. When installing an operator, you choose the channel to follow for updates.
- You can create custom catalogs, and decide which versions of operators to include in the catalog. For example, in a multicluster environment, you configure operators to update automatically, but add only tested versions to the catalog.

Providers can publish operators by other means than the OLM and operator catalogs. For example, a provider can publish operators as Helm charts or YAML resource files. The OLM does not manage operators that are installed by other means.

Operator Update Channels

Each operator provider can create multiple channels for an operator.

For example, a provider can create stable and preview channels for an operator. The provider publishes each new version of the operator to the preview channel. You can use the preview channel to test new features and to validate that the new versions fix bugs. If the provider receives feedback for preview versions of the operator and finds no serious issues with the latest version, then the provider publishes the version to the stable channel. You can use the stable channel for environments with higher reliability requirements, and trade off slower adoption of new features for improved stability.

Additionally, operators might have new features that introduce significant changes or incompatibilities with earlier versions. Operator providers might adopt a versioning scheme for the operator that separates major updates from minor updates, depending on the adoption cost of the new version. In this scenario, providers can create channels for different major versions of the operator.

For example, a provider creates an operator that installs an application. The provider creates **version-1** and **version-2** channels, to correspond to different major versions of the application. Users of the operator can stay on the **version-1** channel in the production environment, and test and design an update process to adopt the **version-2** channel in a staging environment.

When you install an operator, determine the most suitable channel for your requirements. Clusters with varying reliability requirements might use different channels.

You can edit an operator subscription to switch channels. Switching channels does not cause any operator update, unless switching channel makes a later version available and the operator is configured for automatic updates. Switching channels might cause unwanted results; always refer to the operator documentation to learn about possible issues.

Automatic and Manual Updates

When you install an operator, you can decide whether the OLM automatically applies updates, or whether the OLM requires an administrator to approve the update. On the operator installation wizard, you can choose between automatic or manual approval. When you create a subscription by using the `oc` command, the resource specification contains an `installPlanApproval` property that requires an `Automatic` or `Manual` value.

If the publishing policies of an operator suit your requirements, then you can configure automatic approvals. Click **Operators > Installed Operators** on the web console, or examine cluster service versions with the `oc` command, to review the version of installed operators.

If you install an operator and configure manual approvals, then you must approve updates before the OLM updates the operator.

The **Installed Operators** page in the web console displays available upgrades.

Name	Managed Namespaces	Status
DevWorkspace Operator	All Namespaces	Succeeded Up to date
MetalLB Operator	All Namespaces	Succeeded Up to date
Web Terminal	All Namespaces	Succeeded Upgrade available

Figure 9.9: The Installed Operators page with an available upgrade

The subscription resources and the install plan resources contain information about upgrades. You can use the `oc` command to examine those resources to find available upgrades.

```
[user@host ~]$ oc get sub -n openshift-operators web-terminal -o yaml
...output omitted...
spec:
  channel: fast
  installPlanApproval: Manual
  name: web-terminal
  source: do280-catalog-redhat
  sourceNamespace: openshift-marketplace
  startingCSV: web-terminal.v1.5.1
status:
...output omitted...
  conditions:
...output omitted...
  - lastTransitionTime: "2022-11-24T13:46:21Z"
    reason: RequiresApproval
    status: "True"
    type: InstallPlanPending
  currentCSV: web-terminal.v1.6.0 ①
  installPlanGeneration: 2
  installPlanRef:
    apiVersion: operators.coreos.com/v1alpha1
    kind: InstallPlan
    name: install-72vnw ②
    namespace: openshift-operators
    resourceVersion: "194989"
    uid: 8dc979fe-936f-475a-8977-36d210c4da98
  installedCSV: web-terminal.v1.5.1 ③
...output omitted...
  state: UpgradePending
```

- ① The `currentCSV` key shows the latest available version in the channel.
- ② The `installPlanRef` section contains a reference to the install plan resource.
- ③ The `installedCSV` key shows the current version.

The OLM also creates an install plan resource when the operator channel contains a later version of an operator.

```
[user@host ~]$ oc get installplan -n openshift-operators install-72vnw -o yaml
apiVersion: operators.coreos.com/v1alpha1
kind: InstallPlan
...output omitted...
spec:
  approval: Manual ①
  approved: false ②
  clusterServiceVersionNames:
  - web-terminal.v1.6.0 ③
  generation: 2
status:
```

```
...output omitted...
phase: RequiresApproval
...output omitted...
```

- ➊ The `approval` key indicates whether updates must be approved.
- ➋ The `approved` key shows whether the update is approved.
- ➌ The `clusterServiceVersionNames` shows the updated version.

To install the update, edit the specification of the install plan to change the `approved` key value to `true`.

```
[user@host ~]$ oc patch installplan install-72vnw --type merge \
  --patch '{"spec":{"approved":true}}'
installplan.operators.coreos.com/install-72vnw patched
```

You can also use the web console to approve an update. In the **Installed Operators**, click **Upgrade available**, and then click **Preview InstallPlan** to view the install plan. Review the install plan, and then click **Approve** to update the operator.

The screenshot shows the OpenShift web console interface for reviewing an install plan. At the top, it says "Project: openshift-operators". Below that, it shows an "InstallPlans" section with a single entry: "install-q9kt5" which is marked as "RequiresApproval". The "Components" tab is selected. A large blue box contains the heading "Review manual InstallPlan" and a brief description: "Review the manual install plan for operators `web-terminal.v1.6.0`. Once approved, the following resources will be created in order to satisfy the requirements for the components specified in the plan. Click the resource name to view the resource in detail." Below this box are two buttons: "Approve" and "Deny".

Below the "Components" tab, there is a table titled "web-terminal.v1.6.0" listing the resources to be created:

Name	Kind	Status	API version
CSV web-terminal.v1.6.0	ClusterServiceVersion	Unknown	operators.coreos.com/v1alpha1
SA web-terminal-controller	ServiceAccount	Unknown	core/v1
R web-terminal.v1.6.0-web-terminal-controller-596ff5ccc	Role	Unknown	rbac.authorization.k8s.io/v1
RB web-terminal.v1.6.0-web-terminal-controller-596ff5ccc	RoleBinding	Unknown	rbac.authorization.k8s.io/v1

Figure 9.10: Reviewing an install plan

Operator Updates and Cluster Updates

Operators might be incompatible with later versions of OpenShift. For example, an operator that uses an API that is removed from later versions of OpenShift does not work correctly when the cluster is updated. Operators can define a list of compatible OpenShift versions.

When updating a cluster, you might need to update operators if the installed version of the operator is not compatible with the updated OpenShift version. Before you update a cluster, review and install any operator updates that are needed for compatibility fixes. If no compatible updates are available, then you must update the cluster by uninstalling incompatible operators.

Uninstalling Operators

You can uninstall operators by using the web console or the `oc` command.

In the console, click **Operators > Installed operators** and locate the operator. Click the vertical ellipsis (\vdots) menu, and then click **Uninstall Operator**.

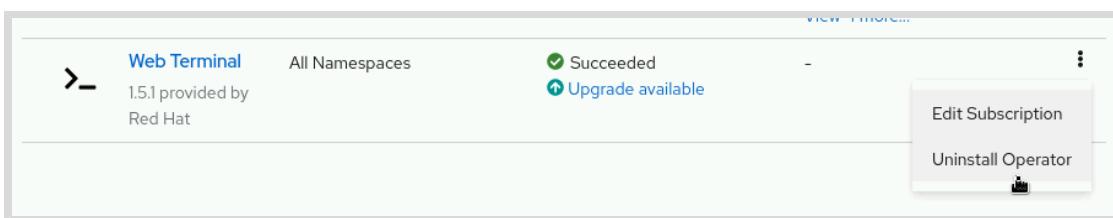


Figure 9.11: The uninstall operator button

After confirming the operation by clicking **Uninstall**, the OLM uninstalls the operator.

Alternatively, delete the subscription and cluster service versions by using the `oc` command.



Important

Uninstalling an operator can leave operator resources on the cluster. Always review the operator documentation to learn about cleanup processes that you must follow to completely remove an operator.



References

Refer to the *Upgrading Installed Operators* section in the *Administrator Tasks* chapter in the Red Hat OpenShift Container Platform 4.14 *Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/updating_clusters/index#updating-eus-to-eus-upgrade_eus-to-eus-upgrade

Refer to the *Deleting Operators from a Cluster* section in the *Administrator Tasks* chapter in the Red Hat OpenShift Container Platform 4.14 *Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-deleting-operators-from-a-cluster

For more information about creating custom catalogs with controlled operator versions, refer to the *Managing Custom Catalogs* section in the *Administrator Tasks* chapter in the Red Hat OpenShift Container Platform 4.14 *Operators* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/operators/index#olm-deleting-operators-from-a-cluster

► Quiz

Update Operators with the OLM

Choose the correct answers to the following questions:

- ▶ **1. Which component manages the updates of operators that are not cluster operators?**
 - a. Operator Lifecycle Manager (OLM)
 - b. Cluster Version Operator (CVO)
 - c. Telemetry client (Telemeter)
- ▶ **2. In which two ways can you configure operator updates? (Choose two.)**
 - a. With automatic updates, the OLM updates an operator as soon as the configured channel has a later version of the operator.
 - b. With automatic updates, the OLM switches the update channel automatically to the channel with the latest version of the operator, and updates to this version.
 - c. With manual updates, the OLM does not monitor channels, and you apply updates manually.
 - d. With manual updates, the OLM updates an operator when the configured channel has a later version of the operator, and an administrator approves the update.
- ▶ **3. In which two ways can you approve updates of an operator? (Choose two.)**
 - a. Update the subscription resource with the intended version.
 - b. Use the web console to review and approve the install plan resource.
 - c. Modify the install plan resource by using the Kubernetes API to approve the update.
 - d. Update the CVO resource specification with the intended version.
- ▶ **4. Which statement is true about update channels?**
 - a. The OLM restricts changes to the update channel to a set of supported changes.
 - b. You can change to any update channel.
 - c. The OLM can change the update channel automatically.
 - d. You cannot change the update channel of an installed operator. You must uninstall the operator to use a different update channel.
- ▶ **5. In which two ways can you uninstall an operator? (Choose two.)**
 - a. Delete the subscription and cluster service versions by using the Kubernetes API.
 - b. Change the `install` field of the subscription resource to the `false` value.
 - c. Use the web console to uninstall the operator.
 - d. Remove the operator from the CVO resource specification.

► Solution

Update Operators with the OLM

Choose the correct answers to the following questions:

► 1. Which component manages the updates of operators that are not cluster operators?

- a. Operator Lifecycle Manager (OLM)
- b. Cluster Version Operator (CVO)
- c. Telemetry client (Telemeter)

► 2. In which two ways can you configure operator updates? (Choose two.)

- a. With automatic updates, the OLM updates an operator as soon as the configured channel has a later version of the operator.
- b. With automatic updates, the OLM switches the update channel automatically to the channel with the latest version of the operator, and updates to this version.
- c. With manual updates, the OLM does not monitor channels, and you apply updates manually.
- d. With manual updates, the OLM updates an operator when the configured channel has a later version of the operator, and an administrator approves the update.

► 3. In which two ways can you approve updates of an operator? (Choose two.)

- a. Update the subscription resource with the intended version.
- b. Use the web console to review and approve the install plan resource.
- c. Modify the install plan resource by using the Kubernetes API to approve the update.
- d. Update the CVO resource specification with the intended version.

► 4. Which statement is true about update channels?

- a. The OLM restricts changes to the update channel to a set of supported changes.
- b. You can change to any update channel.
- c. The OLM can change the update channel automatically.
- d. You cannot change the update channel of an installed operator. You must uninstall the operator to use a different update channel.

► 5. In which two ways can you uninstall an operator? (Choose two.)

- a. Delete the subscription and cluster service versions by using the Kubernetes API.
- b. Change the `install` field of the subscription resource to the `false` value.
- c. Use the web console to uninstall the operator.
- d. Remove the operator from the CVO resource specification.

► Quiz

OpenShift Updates

Choose the correct answers to the following questions:

- ▶ **1. Which component retrieves the updated cluster images from Quay.io?**
 - a. Cluster Version Operator (CVO)
 - b. Operator Lifecycle Manager (OLM)
 - c. Telemetry client (Telemeter)
 - d. Cluster Monitoring (Prometheus)
- ▶ **2. Which component manages the updates of operators that are not cluster operators?**
 - a. Telemetry client (Telemeter)
 - b. Operator Lifecycle Manager (OLM)
 - c. Cluster Version Operator (CVO)
- ▶ **3. Which statement is true about deprecated APIs?**
 - a. Beta versions of features are maintained indefinitely.
 - b. When a stable version of a feature is released, the beta versions are marked as deprecated and are removed in the next Kubernetes release.
 - c. When a stable version of a feature is released, the beta versions are marked as deprecated. When a deprecated API can no longer be maintained, the API is removed.
 - d. When a stable version of a feature is released, the beta versions are marked as deprecated and are removed after three Kubernetes releases.
- ▶ **4. In which three ways can you discover usage of deprecated APIs? (Choose three.)**
 - a. You can disable deprecated APIs, so that usage of deprecated APIs fails.
 - b. APIRequestCount objects count API requests. Review the request count for deprecated APIs.
 - c. OpenShift monitoring includes alerts that notify administrators when the cluster receives a request that uses a deprecated API.
 - d. OpenShift annotates workloads that use deprecated APIs.
 - e. If a request uses a deprecated API version, then the API server returns a deprecation warning.
 - f. Cluster updates are not possible if deprecated APIs are in use.

► **5. Which resource do you edit to approve an operator update?**

- a. PackageManifest
- b. Subscription
- c. InstallPlan
- d. ClusterServiceVersion

► Solution

OpenShift Updates

Choose the correct answers to the following questions:

- ▶ **1. Which component retrieves the updated cluster images from Quay.io?**
 - a. Cluster Version Operator (CVO)
 - b. Operator Lifecycle Manager (OLM)
 - c. Telemetry client (Telemeter)
 - d. Cluster Monitoring (Prometheus)
- ▶ **2. Which component manages the updates of operators that are not cluster operators?**
 - a. Telemetry client (Telemeter)
 - b. Operator Lifecycle Manager (OLM)
 - c. Cluster Version Operator (CVO)
- ▶ **3. Which statement is true about deprecated APIs?**
 - a. Beta versions of features are maintained indefinitely.
 - b. When a stable version of a feature is released, the beta versions are marked as deprecated and are removed in the next Kubernetes release.
 - c. When a stable version of a feature is released, the beta versions are marked as deprecated. When a deprecated API can no longer be maintained, the API is removed.
 - d. When a stable version of a feature is released, the beta versions are marked as deprecated and are removed after three Kubernetes releases.
- ▶ **4. In which three ways can you discover usage of deprecated APIs? (Choose three.)**
 - a. You can disable deprecated APIs, so that usage of deprecated APIs fails.
 - b. APIRequestCount objects count API requests. Review the request count for deprecated APIs.
 - c. OpenShift monitoring includes alerts that notify administrators when the cluster receives a request that uses a deprecated API.
 - d. OpenShift annotates workloads that use deprecated APIs.
 - e. If a request uses a deprecated API version, then the API server returns a deprecation warning.
 - f. Cluster updates are not possible if deprecated APIs are in use.

► **5. Which resource do you edit to approve an operator update?**

- a. PackageManifest
- b. Subscription
- c. **InstallPlan**
- d. ClusterServiceVersion

Summary

- A major benefit of OpenShift 4 architectural changes is that you can update your clusters Over-the-Air (OTA).
- Red Hat provides a software distribution system that ensures the best path for updating your OpenShift 4 cluster and the underlying operating system.
- Red Hat maintains several distribution channels:
 - The *fast* channel delivers updates as soon as they are available.
 - The *stable* channel delivers updates that passed additional testing and validation in operational clusters.
 - The *candidate* channel delivers updates for testing feature acceptance in the next version of OpenShift Container Platform.
 - The *eus* channel (which is available only for Extended Updated Support releases) extends the maintenance phase.
- Red Hat does not support reverting your cluster to an earlier version.
- The Kubernetes API versions are categorized based on feature maturity.
- When a stable version of an API is released, the beta versions are marked as deprecated and are removed after three Kubernetes releases.
- Requests to a deprecated API display warnings and trigger alerts. You can track deprecated API usage by using `APIRequestCount` objects.
- The Operator Lifecycle Manager (OLM) can update operators that are installed in an OpenShift cluster.
 - For each installed operator, you can decide whether the OLM automatically applies updates, or whether the updates require administrator approval.
- Operator providers can create multiple channels for an operator with different release policies.

Chapter 10

Comprehensive Review

Goal

Review tasks from *Red Hat OpenShift Administration II: Configuring a Production Cluster*.

Sections

- Comprehensive Review

Lab

- Cluster Self-service Setup
- Secure Applications
- Deploy Packaged Applications

Comprehensive Review

Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills that you learned in *Red Hat OpenShift Administration II: Configuring a Production Cluster*.

Reviewing Red Hat OpenShift Administration II: Configuring a Production Cluster

Before beginning the comprehensive review for this course, you should be comfortable with the topics covered in each chapter. Do not hesitate to ask the instructor for extra guidance or clarification on these topics.

Chapter 1, Declarative Resource Management

Deploy and update applications from resource manifests that are parameterized for different target environments.

- Deploy and update applications from resource manifests that are stored as YAML files.
- Deploy and update applications from resource manifests that are augmented by Kustomize.

Chapter 2, Deploy Packaged Applications

Deploy and update applications from resource manifests that are packaged for sharing and distribution.

- Deploy an application and its dependencies from resource manifests that are stored in an OpenShift template.
- Deploy and update applications from resource manifests that are packaged as Helm charts.

Chapter 3, 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.

Chapter 4, Network Security

Protect network traffic between applications inside and outside the cluster.

- Allow and protect network connections to applications inside an OpenShift cluster.
- Restrict network traffic between projects and pods.
- Configure and use automatic service certificates.

Chapter 5, Expose non-HTTP/SNI Applications

Expose applications to external access without using an ingress controller.

- Expose applications to external access by using load balancer services.
- Expose applications to external access by using a secondary network.

Chapter 6, Enable Developer Self-Service

Configure clusters for safe self-service by developers from multiple teams, and disallow self-service if operations staff must provision projects.

- Configure compute resource quotas and Kubernetes resource count quotas per project and cluster-wide.
- Configure default and maximum compute resource requirements for pods per project.
- Configure default quotas, limit ranges, role bindings, and other restrictions for new projects, and the allowed users to self-provision new projects.

Chapter 7, Manage Kubernetes Operators

Install and update operators that the Operator Lifecycle Manager and the Cluster Version Operator manage.

- Explain the operator pattern and different approaches for installing and updating Kubernetes operators.
- Install and update operators by using the web console and the Operator Lifecycle Manager.
- Install and update operators by using the Operator Lifecycle Manager APIs.

Chapter 8, Application Security

Run applications that require elevated or special privileges from the host operating system or Kubernetes.

- Create service accounts and apply permissions, and manage security context constraints.
- Run an application that requires access to the Kubernetes API of the application's cluster.
- Automate regular cluster and application management tasks by using Kubernetes cron jobs.

Chapter 9, OpenShift Updates

Update an OpenShift cluster and minimize disruption to deployed applications.

- Describe the cluster update process.
- Identify applications that use deprecated Kubernetes APIs.
- Update OLM-managed operators by using the web console and CLI.

▶ Lab

Cluster Self-service Setup

Configure a cluster with default settings for self-service projects.

Outcomes

- Create a project template that sets quotas, ranges, and network policies.
- Restrict access to the `self-provisioner`s cluster role.
- Create groups and assign users to groups.
- Use role-based access control (RBAC) to grant permissions 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 start compreview-review
```

The `lab` command copies the exercise files to the `~/DO280` directory and creates the following users:

- `do280-support`
- `do280-platform`
- `do280-presenter`
- `do280-attendee`

The goal, as the cluster administrator, is to configure a dedicated cluster to host workshops on different topics.

Each workshop requires a project, so that workshops are isolated from each other.

You must set up the cluster so that when the presenter creates a workshop project, the project gets a base configuration.

The presenter must be mostly self-sufficient to administer a workshop with little help from the workshop support team.

The workshop support team must deploy applications that administer workshops and that enhance the workshop experience. You set up a project and the applications for this purpose on a second lab.

Specifications

Use the following values to access the OpenShift cluster:

Item	Value
Dev user/password	developer/developer

Item	Value
Admin user/password	admin/redhatocp
API URL	https://api.ocp4.example.com:6443

The following workshop groups are required:

- Create the groups with the specified users in the following table:

Group	User
platform	do280-platform
presenters	do280-presenter
workshop-support	do280-support

The `lab start` command creates the users with the `redhat` password.

- The `platform` group administers the cluster.
- The `presenters` group consists of the people who deliver the workshops.
- The `workshop-support` group maintains the needed applications to support the workshops and the workshop presenters.
- Ensure that only users from the following groups can create projects:

Group
platform
presenters
workshop-support

- An attendee must not be able to create projects. Because this exercise requires steps that restart the Kubernetes API server, this configuration must persist across API server restarts.
- The `workshop-support` group requires the following roles in the cluster:
 - The `admin` role to administer projects
 - A custom role that is provided in the `groups-role.yaml` file. You must create this custom role to enable support members to create workshop groups and to add workshop attendees.
- The `platform` group must be able to administer the cluster without restrictions.
- The `workshop-support` group must perform the following tasks for the workshop project:
 - Create a workshop-specific attendees group.
 - Assign the `edit` role to the attendees group.
 - Add users to the attendees group.
- Each workshop must be hosted in an independent project.
- All the resources that the cluster creates with a new workshop project must use `workshop` as the name for grading purposes.

- Each workshop must enforce the following maximum constraints:
 - The project uses up to 2 CPUs.
 - The project uses up to 1 Gi of RAM.
 - The project requests up to 1.5 CPUs.
 - The project requests up to 750 Mi of RAM.
 - Each workshop must enforce constraints to prevent an attendee's workload from consuming all the allocated resources for the workshop:
 - A workload uses up to 750m CPUs.
 - A workload uses up to 750 Mi.
 - Each workshop must have a resource specification for workloads:
 - A default limit of 500m CPUs.
 - A default limit of 500 Mi of RAM.
 - A default request of 0.1 CPUs.
 - A default request of 250 Mi of RAM.
- You can use the templates that are provided in the `quota.yaml`, `limitrange.yaml`, and `networkpolicy.yaml` files.
- Each workshop project must have this additional default configuration:
 - A local binding for the presenter user to the `admin` cluster role with the `workshop` name
 - The `workshop=project_name` label to help to identify the workshop workload
 - Must accept traffic only from within the same workshop by using the `workshop=project_name` label or from the ingress controller.
 - Use the `registry.ocp4.example.com:8443/redhattraining/hello-world-inginx:v1.0` image, which listens on the 8080 port, to simulate a workshop workload.
 - As the `do280-presenter` user, you must create a workshop with the `do280` name.
 - As the `do280-support` user, you must create the `do280-attendees` group with the `do280-attendee` user, and assign the `edit` role to the `do280-attendees` group.

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 grade compreview-review
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Cluster Self-service Setup

Configure a cluster with default settings for self-service projects.

Outcomes

- Create a project template that sets quotas, ranges, and network policies.
- Restrict access to the **self-provisioner**s cluster role.
- Create groups and assign users to groups.
- Use role-based access control (RBAC) to grant permissions 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 start compreview-review
```

The **lab** command copies the exercise files to the **~/D0280** directory and creates the following users:

- **do280-support**
- **do280-platform**
- **do280-presenter**
- **do280-attendee**

The goal, as the cluster administrator, is to configure a dedicated cluster to host workshops on different topics.

Each workshop requires a project, so that workshops are isolated from each other.

You must set up the cluster so that when the presenter creates a workshop project, the project gets a base configuration.

The presenter must be mostly self-sufficient to administer a workshop with little help from the workshop support team.

The workshop support team must deploy applications that administer workshops and that enhance the workshop experience. You set up a project and the applications for this purpose on a second lab.

1. Change to the **~/D0280/labs/comprevew-review** directory and log in to the cluster as the **admin** user.
 - 1.1. Change to the **lab** directory.

```
[student@workstation ~]$ cd ~/D0280/labs/comprevew-review
```

- 1.2. Open a terminal window and log in as the `admin` user with the `redhatocp` password.

```
[student@workstation compreview-review]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

2. Create the following groups and add a user as specified in the following table.

Group	User
workshop-support	do280-support
presenters	do280-presenter
platform	do280-platform

- 2.1. Create the `workshop-support` group.

```
[student@workstation compreview-review]$ oc adm groups new workshop-support
group.user.openshift.io/workshop-support created
```

- 2.2. Add the `do280-support` user to the `workshop-support` group.

```
[student@workstation compreview-review]$ oc adm groups add-users \
workshop-support do280-support
group.user.openshift.io/workshop-support added: "do280-support"
```

- 2.3. Create the `presenters` group.

```
[student@workstation compreview-review]$ oc adm groups new presenters
group.user.openshift.io/presenters created
```

- 2.4. Add the `do280-presenter` user to the `presenters` group.

```
[student@workstation compreview-review]$ oc adm groups add-users \
presenters do280-presenter
group.user.openshift.io/presenters added: "do280-presenter"
```

- 2.5. Create the `platform` group.

```
[student@workstation compreview-review]$ oc adm groups new platform
group.user.openshift.io/platform created
```

- 2.6. Add the `do280-platform` user to the `platform` group.

```
[student@workstation compreview-review]$ oc adm groups add-users \
platform do280-platform
group.user.openshift.io/platform added: "do280-platform"
```

- 2.7. Use the `oc get groups` command to verify that the group configuration is correct.

```
[student@workstation compreview-review]$ oc get groups
NAME                      USERS
...output omitted...
platform                  do280-platform
presenters                do280-presenter
workshop-support          do280-support
```

3. Grant to the workshop-support group the admin and the custom manage-groups cluster roles. You must create the manage-groups custom cluster role from the groups-role.yaml file.

- 3.1. Grant the admin cluster role to the workshop-support group.

```
[student@workstation compreview-review]$ oc adm policy \
add-cluster-role-to-group admin workshop-support
clusterrole.rbac.authorization.k8s.io/admin added: "workshop-support"
```

- 3.2. Run the oc create command to create the manage-groups cluster role in the groups-role.yaml file.

```
[student@workstation compreview-review]$ oc create -f groups-role.yaml
clusterrole.rbac.authorization.k8s.io/manage-groups created
```

- 3.3. Grant the manage-groups cluster role to the workshop-support group.

```
[student@workstation compreview-review]$ oc adm policy \
add-cluster-role-to-group manage-groups workshop-support
clusterrole.rbac.authorization.k8s.io/manage-groups added: "workshop-support"
```

4. Create a cluster role binding to assign the cluster-admin cluster role to the platform group.

```
[student@workstation compreview-review]$ oc adm policy \
add-cluster-role-to-group cluster-admin platform
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "platform"
```



Note

When you execute the `oc adm policy add-cluster-role-to-group cluster-admin platform` command to add the cluster role to the new group, a naming collision occurs with an existing object with this name. Consequently, the system creates an object and appends to the name -x, which is an iterating numeral that starts with -0.

To view the new role binding, use the `oc get clusterrolebinding | grep ^cluster-admin` command to list all cluster role bindings that begin with `cluster-admin`. Then, run `oc describe` on the listed item with the highest -x value to view the details for your new binding.

5. Allow only the platform, workshop-support and presenters groups to create projects, by editing the self-provisioner cluster role. Enforce that only users from

these groups can create projects. Also, make this change permanent by setting the `rbac.authorization.kubernetes.io/autoupdate` annotation with the `false` value.

- 5.1. Use the `oc edit` command to edit the `self-provisioners` cluster role binding.

```
[student@workstation compreview-review]$ oc edit clusterrolebinding \
  self-provisioners
```

Replace the subject of the role binding for the `system:authenticated:oauth` group with the `platform`, `workshop-support`, and `presenters` groups.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "false"
  creationTimestamp: "2023-01-24T23:31:00Z"
  name: self-provisioners
  resourceVersion: "250330"
  uid: a6053896-f68f-41ff-9bb3-5da579a701bc
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: self-provisioner
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: platform
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: workshop-support
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: presenters
```

- 5.2. As the `do280-attendee` user, verify that you cannot create a project.

Log in as the `do280-attendee` user with the `redhat` password.

```
[student@workstation compreview-review]$ oc login -u do280-attendee -p redhat
Login successful.
```

You don't have any projects. Contact your system administrator to request a project.

Use the `oc new-project` command to try to create a `template-test` project.

```
[student@workstation compreview-review]$ oc new-project template-test
Error from server (Forbidden): You may not request a new project via this API.
```

6. As the `admin` user, create a `template-test` namespace to design the project template.

- 6.1. Log in as the `admin` user with the `redhatocp` password.

```
[student@workstation compreview-review]$ oc login -u admin -p redhatocp
Login successful.
...output omitted...
```

- 6.2. Use the `oc new-project` command to create the `template-test` project.

```
[student@workstation compreview-review]$ oc new-project template-test
Now using project "template-test" on server...
...output omitted...
```

7. Create a template resource quota with the following specification.

Quota	Value
limits.cpu	2
limits.memory	1Gi
requests.cpu	1500m
requests.memory	750Mi

- 7.1. Edit the `quota.yaml` file and replace the `CHANGE_ME` label to match the following definition.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: workshop
  namespace: template-test
spec:
  hard:
    limits.cpu: 2
    limits.memory: 1Gi
    requests.cpu: 1500m
    requests.memory: 750Mi
```

- 7.2. Use the `oc create` command to create the quota in the `template-test` project.

```
[student@workstation compreview-review]$ oc create -f quota.yaml
resourcequota/workshop created
```

8. Create the `workshop` limit range with the following specification.

Limit type	Value
max.cpu	750m
max.mem	750Mi
default.cpu	500m
default.memory	500Mi
defaultRequest.cpu	100m
defaultRequest.memory	250Mi

- 8.1. Edit the `limitrange.yaml` file and replace the `CHANGE_ME` label to match the following definition.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: workshop
  namespace: template-test
spec:
  limits:
    - max:
        cpu: 750m
        memory: 750Mi
      default:
        cpu: 500m
        memory: 500Mi
      defaultRequest:
        cpu: 100m
        memory: 250Mi
      type: Container
```

- 8.2. Use the `oc create` command to create the limit range in the `template-test` project.

```
[student@workstation compreview-review]$ oc create -f limitrange.yaml
limitrange/workshop created
```

9. Create a network policy to accept traffic from within the `workshop` project or from outside the cluster. To identify the `workshop` project traffic, label the `template-test` namespace with the `workshop=template-test` label.
- 9.1. Use the `oc create deployment` command to create a deployment without resource specifications.

```
[student@workstation compreview-review]$ oc create deployment test-workload \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0
deployment.apps/test-workload created
```

- 9.2. Get the IP address of one of the NGINX pods.

```
[student@workstation compreview-review]$ oc get pod -o wide
NAME                  READY   STATUS    ...   IP      ...
test-workload-56bf7dc6fc-mshn9   1/1     Running   ...  10.8.0.138  ...
```

- 9.3. Use the `oc debug` command to run the `curl` command from a pod in the default project.

Use the `curl` command from the default namespace to query the NGINX server that runs in the test workload.

```
[student@workstation compreview-review]$ oc debug --to-namespace="default" \
-- curl -s http://10.8.0.138:8080
Starting pod/image-debug ...
<html>
<body>
  <h1>Hello, world from nginx!</h1>
</body>
</html>

Removing debug pod ...
```

- 9.4. Use the `oc label` command to add the label to the `template-test` namespace.

```
[student@workstation compreview-review]$ oc label ns template-test \
workshop=template-test
namespace/template-test labeled
```

- 9.5. Edit the network policy from the `networkpolicy.yaml` file. Replace the `CHANGE_ME` labels according to the following specification.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: workshop
  namespace: template-test
spec:
  podSelector: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              workshop: template-test
    - namespaceSelector:
            matchLabels:
              policy-group.network.openshift.io/ingress: ""
```

- 9.6. Run the `oc create` command to create the policy in the `template-test` project.

```
[student@workstation compreview-review]$ oc create -f networkpolicy.yaml
networkpolicy.networking.k8s.io/workshop created
```

- 9.7. Verify that you cannot connect to the `workshop` pod from the `default` project.

```
[student@workstation compreview-review]$ oc debug --to-namespace="default" \
-- curl -sS --connect-timeout 5 http://10.8.0.138:8080
Starting pod/image-debug ...
curl: (28) Connection timed out after 5000 milliseconds

Removing debug pod ...
```

9.8. Verify that you can connect to the workshop pod from the workshop project.

```
[student@workstation compreview-review]$ oc debug \
--to-namespace="template-test" \
-- curl -sS http://10.8.0.138:8080
Warning: would violate PodSecurity "restricted:latest": ...output omitted...
Starting pod/image-debug ...
<html>
<body>
<h1>Hello, world from nginx!</h1>
</body>
</html>

Removing debug pod ...
```

10. Create the workshop project template by using the previously created template resources.

10.1. Run the `oc adm create-bootstrap-project-template` command to create the `project-template.yaml` file to use as the template for new projects.

```
[student@workstation compreview-review]$ oc adm \
create-bootstrap-project-template \
-o yaml > project-template.yaml
```

10.2. Use the `oc get` command to create a YAML list with the following resources:

- `resourcequota/workshop`
- `limitrange/workshop`
- `networkpolicy/workshop`

Redirect the output to append to the `project-template.yaml` file.

```
[student@workstation compreview-review]$ oc get resourcequota/workshop \
limitrange/workshop \
networkpolicy/workshop \
-o yaml >> project-template.yaml
```

10.3. Edit the `project-template.yaml` file to perform the following operations:

- Cut the contents of the `items` stanza and paste them immediately before the `parameters` stanza. Keep the original indentation, because every YAML item of the list must appear at the beginning of the line.
- Remove any left-over content after the `parameters` block.
- Remove the following keys from the limit range and quota definitions:

- creationTimestamp
 - resourceVersion
 - uid
 - status
 - generation
- Replace the `template-test` text with the `${PROJECT_NAME}` text.
 - Add the `workshop=${PROJECT_NAME}` label.
 - Rename the `admin` role binding with the `workshop` name.

Use the search-and-replace editor function to replace the `template-test` string with the `${PROJECT_NAME}` template parameter. Optionally, you can use the `sed` command if it is available.

The solution file is in the `~/DO280/solutions/comprevew-review/project-template.yaml` path.

```
[student@workstation compreview-review]$ sed -i \
's/template-test/${PROJECT_NAME}/g' project-template.yaml
```

Then, move the resource list to the `objects` key after line 31. The `project-template.yaml` file has the following expected content.

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: project-request
objects:
- apiVersion: project.openshift.io/v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    name: ${PROJECT_NAME}
    labels:
      workshop: ${PROJECT_NAME}
  spec: {}
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: workshop
    namespace: ${PROJECT_NAME}
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: admin
  subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: User
      name: ${PROJECT_ADMIN_USER}
- apiVersion: v1
  kind: ResourceQuota
```

```
metadata:
  annotations:
    name: workshop
    namespace: ${PROJECT_NAME}
spec:
  hard:
    limits.cpu: "2"
    limits.memory: 1Gi
    requests.cpu: 1500m
    requests.memory: 750Mi
- apiVersion: v1
  kind: LimitRange
  metadata:
    annotations:
      name: workshop
      namespace: ${PROJECT_NAME}
  spec:
    limits:
      - default:
          cpu: 500m
          memory: 500Mi
        defaultRequest:
          cpu: 100m
          memory: 250Mi
      max:
        cpu: 750m
        memory: 750Mi
        type: Container
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    annotations:
      name: workshop
      namespace: ${PROJECT_NAME}
  spec:
    ingress:
      - from:
          - namespaceSelector:
              matchLabels:
                workshop: ${PROJECT_NAME}
          - namespaceSelector:
              matchLabels:
                policy-group.network.openshift.io/ingress: ""
    podSelector: {}
    policyTypes:
      - Ingress
parameters:
- name: PROJECT_NAME
- name: PROJECT_DISPLAYNAME
- name: PROJECT_DESCRIPTION
- name: PROJECT_ADMIN_USER
- name: PROJECT_REQUESTING_USER
```

- 10.4. Create the project template in the `project-template.yaml` file by using the `oc create` command in the `openshift-config` namespace.

```
[student@workstation compreview-review]$ oc create -f project-template.yaml \
-n openshift-config
template.template.openshift.io/project-request created
```

- 10.5. Use the `oc edit` command to change the cluster project configuration.

```
[student@workstation compreview-review]$ oc edit \
projects.config.openshift.io cluster
```

Edit the resource to match the following content:

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
...output omitted...
name: cluster
...output omitted...
spec:
  projectRequestTemplate:
    name: project-request
```

To edit the file, you use the default `vi` editor.

- 10.6. Use the `watch` command to view the API server pods.

```
[student@workstation compreview-review]$ watch oc get \
pod -n openshift-apiserver
```

Wait until new pods are created. Press `Ctrl+C` to exit the `watch` command.

11. As the `do280-presenter`, create the `do280` workshop project.

- 11.1. Log in as the `do280-presenter` user with the `redhat` password.

```
[student@workstation compreview-review]$ oc login -u do280-presenter -p redhat
Login successful.
...output omitted...
```

- 11.2. Use the `oc new-project` command to create the `do280` project.

```
[student@workstation compreview-review]$ oc new-project do280
Now using project "do280" on server ...
...output omitted...
```

- 11.3. Verify that the `oc new-project` command creates the following resources from the template:

- Quota
- Limit range
- Network policy

```
[student@workstation compreview-review]$ oc get resourcequota/workshop \
  limitrange/workshop \
  networkpolicy/workshop
NAME          AGE   REQUEST           LIMIT
resourcequota/workshop  95s   requests.cpu: 0/1500m ... limits.cpu: 0/2 ...
NAME          CREATED AT
limitrange/workshop  2023-03-03T10:37:28Z
NAME          POD-SELECTOR   AGE
networkpolicy.networking.k8s.io/workshop <none>    95s
```

11.4. Verify that the do280 project definition has the workshop=do280 label.

```
[student@workstation compreview-review]$ oc get project do280 -o yaml
apiVersion: project.openshift.io/v1
kind: Project
metadata:
...output omitted...
labels:
  workshop: do280
...output omitted...
name: do280
resourceVersion: "1293438"
...output omitted...
```

12. As the do280-support user, create the do280-attendees group. Then, assign the edit role to the do280-attendees group, and add the do280-attendee user to the group.

12.1. Log in as the do280-support user with the redhat password.

```
[student@workstation compreview-review]$ oc login -u do280-support -p redhat
Login successful.
...output omitted...
```

12.2. Create the do280-attendees group.

```
[student@workstation compreview-review]$ oc adm groups new do280-attendees
group.user.openshift.io/do280-attendees created
```

12.3. Assign the edit role to the do280-attendees group in the do280-workshop project.

Add the edit role to the do280-attendees group in the do280 project.

```
[student@workstation compreview-review]$ oc adm policy \
  add-role-to-group edit do280-attendees -n do280
clusterrole.rbac.authorization.k8s.io/edit added: "do280-attendees"
```

12.4. As the do280-attendee user, verify that you cannot access the do280 project.

Log in as the do280-attendee user with the redhat password.

```
[student@workstation compreview-review]$ oc login -u do280-attendee -p redhat
Login successful.
You don't have any projects. . .
```

- 12.5. As the do280-support user, add the do280-attendee user to the do280-attendees group.

Log in as the do280-support user with the redhat password.

```
[student@workstation compreview-review]$ oc login -u do280-support -p redhat
Login successful.
...output omitted...
```

Use the oc adm groups command to add the do280-attendee user to the workshop-do280-attendees group.

```
[student@workstation compreview-review]$ oc adm groups add-users \
do280-attendees do280-attendee
group.user.openshift.io/do280-attendees added: "do280-attendee"
```

- 12.6. As the do280-attendee user, verify that you can create workloads in the do280 project.

Log in as the do280-attendee user with the redhat password.

```
[student@workstation compreview-review]$ oc login -u do280-attendee -p redhat
Login successful.
You have one project on this server: "do280"
Using project "do280".
```

Use the oc create deployment command to create a deployment without resource specifications.

```
[student@workstation compreview-review]$ oc create deployment \
attendee-workload \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:v1.0
deployment.apps/attendee-workload created
```

13. Change to the home directory to prepare for the next exercise.

```
[student@workstation appsec-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 grade compreview-review
```

Finish

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

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

▶ Lab

Secure Applications

Configure a project that requires custom settings.

Secure applications by encrypting and restricting network traffic.

Automate cluster maintenance tasks.

Outcomes

- Create a project quota.
- Create a limit range.
- Use role-based access control to grant permissions to service accounts and groups.
- Encrypt the traffic end-to-end with TLS by using a signed certificate.
- Restrict cluster internal traffic to pods by using network policies.
- Grant application access to Kubernetes APIs.
- Configure a cluster maintenance application to run periodically.

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 start compreview-apps
```

The `lab` command copies the exercise files into the `~/D0280/labs/comprevew-apps` directory and creates the `workshop-support` group with the `do280-support` user. The `lab` command also restores the project template configuration from the previous exercise.

The goal, as a cluster administrator, is to prepare the `workshop-support` namespace for the support team. Create a namespace instead of a project to avoid using the project template. The project template applies a default configuration for workshop projects, and does not apply the configuration to the `workshop-support` namespace. Then, as a support team member, you configure and deploy the applications that maintain the cluster and support the workshop experience.

You must set up an application that automatically deletes completed workshops, and set up a social media API that attendees from all workshops use.

Specifications

- Create the `workshop-support` namespace with the `category`: `support` label.
- Grant to the `workshop-support` group the `admin` role in the cluster.
- Workloads from the `workshop-support` namespace must enforce the following constraints:

- The project uses up to 4 CPUs.
 - The project uses up to 4 Gi of RAM.
 - The project requests up to 3.5 CPUs.
 - The project requests up to 3 Gi of RAM.
- Define the default resource specification for workloads:
 - A default limit of 300m CPUs.
 - A default limit of 400 Mi of RAM.
 - A default request of 100m CPUs.
 - A default request of 250 Mi of RAM.
 - Any quota or limit range must have the `workshop-support` name for grading purposes.
 - As the `do280-support` user, deploy the `project-cleaner` application from the `project-cleaner/example-pod.yaml` file to the `workshop-support` namespace by using a `project-cleaner` cron job that runs every minute.

The project cleaner deletes projects with the `workshop` label that exist for more than 10 seconds. This short expiration time is deliberate for this lab.

- You must create a `project-cleaner-sa` service account to use in the project cleaner application.
- The role that the project cleaner needs is defined in the `project-cleaner/cluster-role.yaml` file.
- Deploy the `beeper-db` database in the `beeper-api/beeper-db.yaml` file to the `workshop-support` namespace.
- Deploy the `beeper-api` application in the `beeper-api/deployment.yaml` file to the `workshop-support` namespace.
- You must configure this application to use TLS end-to-end by using the following specification:
 - Use the `beeper-api.pem` certificate and the `beeper-api.key` in the `certs` directory.
 - Configure the `/etc/pki/beeper-api/` path as the mount point for the certificate and key.
 - Set the `TLS_ENABLED` environment variable to the `true` value.
- Update the startup, readiness, and liveness probes to use TLS.
- Create a passthrough route with the `beeper-api.apps.ocp4.example.com` hostname, and use the `beeper-api.apps.ocp4.example.com/api/beeps` URL with the CA certificate to verify the beeper API is accessible from outside the cluster.
- The database pods, which are pods in the `workshop-support` namespace with the `app=beeper-db` label, must accept only TCP traffic from the `beeper-api` pods in the `workshop-support` namespace on the 5432 port. You can use the `category=support` label to identify the pods that belong to the `workshop-support` namespace.
- Configure the cluster network so that the `workshop-support` namespace accepts only external ingress traffic to pods that listen on the 8080 port, and blocks traffic from other projects.

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 grade compreview-apps
```

Finish

As the student user on the workstation machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-apps
```

► Solution

Secure Applications

Configure a project that requires custom settings.

Secure applications by encrypting and restricting network traffic.

Automate cluster maintenance tasks.

Outcomes

- Create a project quota.
- Create a limit range.
- Use role-based access control to grant permissions to service accounts and groups.
- Encrypt the traffic end-to-end with TLS by using a signed certificate.
- Restrict cluster internal traffic to pods by using network policies.
- Grant application access to Kubernetes APIs.
- Configure a cluster maintenance application to run periodically.

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 start comprevew-apps
```

The `lab` command copies the exercise files into the `~/D0280/labs/comprevew-apps` directory and creates the `workshop-support` group with the `do280-support` user. The `lab` command also restores the project template configuration from the previous exercise.

The goal, as a cluster administrator, is to prepare the `workshop-support` namespace for the support team. Create a namespace instead of a project to avoid using the project template. The project template applies a default configuration for workshop projects, and does not apply the configuration to the `workshop-support` namespace. Then, as a support team member, you configure and deploy the applications that maintain the cluster and support the workshop experience.

You must set up an application that automatically deletes completed workshops, and set up a social media API that attendees from all workshops use.

1. Change to the `~/D0280/labs/comprevew-apps` directory and log in to the cluster as the `admin` user.
 - 1.1. Open a terminal window and change to the `lab` directory.

```
[student@workstation ~]$ cd ~/D0280/labs/comprevew-apps
```

- 1.2. Log in as the `admin` user with the `redhatocp` password.

```
[student@workstation comprevie-apps]$ oc login -u admin -p redhatocp \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

2. Create and prepare the `workshop-support` namespace with the following actions:

- Add the `category=support` label.
- Grant the `admin` cluster role to the `workshop-support` group.

- 2.1. Create the `workshop-support` namespace.

```
[student@workstation comprevie-apps]$ oc create namespace workshop-support
namespace/workshop-support created
```

- 2.2. Use the `oc label` command to add the `category=support` label to the `workshop-support` namespace.

```
[student@workstation beeper-api]$ oc label namespace \
workshop-support category=support
namespace/workshop-support labeled
```

- 2.3. Change to the `workshop-support` namespace by using the `oc project` command.

```
[student@workstation beeper-api]$ oc project workshop-support
Now using project "workshop-support" on server...
```

- 2.4. Create a cluster role binding to assign the `admin` cluster role to the `workshop-support` group.

```
[student@workstation comprevie-apps]$ oc adm policy \
add-cluster-role-to-group admin workshop-support
clusterrole.rbac.authorization.k8s.io/admin added: "workshop-support"
```

3. Create the resource quota for the `workshop-support` namespace with the following specification.

Quota	Value
limits.cpu	4
limits.memory	4Gi
requests.cpu	3500m
requests.memory	3Gi

- 3.1. Run the `oc create quota` command to create the quota.

```
[student@workstation compreview-apps]$ oc create quota workshop-support \
--hard=limits.cpu=4,limits.memory=4Gi,requests.cpu=3500m,requests.memory=3Gi
resourcequota/workshop-support created
```

4. Create the workshop limit range with the following specification.

Limit type	Value
default.cpu	300m
default.memory	400Mi
defaultRequest.cpu	100m
defaultRequest.memory	250Mi

- 4.1. Edit the `limitrange.yaml` file and replace the `CHANGE_ME` label to match the following definition.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: workshop-support
  namespace: workshop-support
spec:
  limits:
    - default:
        cpu: 300m
        memory: 400Mi
      defaultRequest:
        cpu: 100m
        memory: 250Mi
      type: Container
```

- 4.2. Use the `oc apply` command to create the limit range in the `workshop-support` project.

```
[student@workstation compreview-apps]$ oc apply -f limitrange.yaml
limitrange/workshop-support created
```

5. Create the `project-cleaner-sa` service account in the `workshop-support` namespace. Then, assign the role from the `project-cleaner/cluster-role.yaml` file to the `project-cleaner-sa` service account.

- 5.1. Create the `project-cleaner-sa` service account.

```
[student@workstation compreview-apps]$ oc create sa project-cleaner-sa
serviceaccount/project-cleaner-sa created
```

- 5.2. Change to the `~/D0280/labs/compreview-apps/project-cleaner` directory to access the application files.

```
[student@workstation compreview-apps]$ cd \
~/DO280/labs/compreview-apps/project-cleaner
```

- 5.3. Create the project-cleaner cluster role by applying the cluster-role.yaml manifest file.

```
[student@workstation project-cleaner]$ oc apply -f cluster-role.yaml
clusterrole.rbac.authorization.k8s.io/project-cleaner created
```

- 5.4. Use the oc adm policy add-cluster-role-to-user command to add the project-cleaner role to the project-cleaner-sa service account.

```
[student@workstation project-cleaner]$ oc adm policy add-cluster-role-to-user \
project-cleaner -z project-cleaner-sa
clusterrole.rbac.authorization.k8s.io/project-cleaner added: "project-cleaner-sa"
```

6. As the do280-support user, create the project-cleaner cron job by editing the cron-job.yaml file and by using the example-pod.yaml pod manifest as the job template. Configure the cron job to run every minute.

- 6.1. Log in as the do280-support user with the redhat password.

```
[student@workstation project-cleaner]$ oc login -u do280-support -p redhat
Login successful.
...output omitted...
```

- 6.2. Edit the cron-job.yaml file:

- Replace the CHANGE_ME label with the "*/1 * * * *" schedule to execute the job every minute.
- Replace the CHANGE_ME label in the jobTemplate definition with the spec definition from the example-pod.yaml pod manifest.
- Replace the CHANGE_ME label in the serviceAccountName key with the project-cleaner-sa service account.

Although the long image name might show across two lines, you must add it as one line.

A solution file is in the ~/DO280/solutions/compreview-apps/project-cleaner/cron-job.yaml path.

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: project-cleaner
  namespace: workshop-support
spec:
  schedule: "*/1 * * * *"
  concurrencyPolicy: Forbid
  jobTemplate:
```

```

spec:
  template:
    spec:
      restartPolicy: Never
      serviceAccountName: project-cleaner-sa
      containers:
        - name: project-cleaner
          image: registry.ocp4.example.com:8443/redhattraining/do280-project-
cleaner:v1.1
          imagePullPolicy: Always
          env:
            - name: "PROJECT_TAG"
              value: "workshop"
            - name: "EXPIRATION_SECONDS"
              value: "10"
          resources:
            limits:
              cpu: 100m
              memory: 200Mi

```

6.3. Create the cron job.

```
[student@workstation project-cleaner]$ oc apply -f cron-job.yaml
cronjob.batch/project-cleaner created
```

6.4. Verify that the project cleaner application is deployed correctly, by creating a clean-test project.

```
[student@workstation project-cleaner]$ oc new-project clean-test
Now using project "clean-test" on server...
...output omitted...
```

Change to the workshop-support namespace.

```
[student@workstation project-cleaner]$ oc project workshop-support
Now using project "workshop-support" on server...
```

Wait for a successful job run. Then, get the pod name from the last job run.

```
[student@workstation project-cleaner]$ oc get jobs,pods
NAME                                COMPLETIONS   DURATION   AGE
job.batch/project-cleaner-27949859  1/1          7s         2m40s
job.batch/project-cleaner-27949860  1/1          7s         100s
job.batch/project-cleaner-27949861  1/1          6s         40s

NAME                               READY   STATUS    RESTARTS   AGE
pod/project-cleaner-27949859-f98vj 0/1    Completed  0          2m40s
pod/project-cleaner-27949860-j8td5  0/1    Completed  0          100s
pod/project-cleaner-27949861-p262t  0/1    Completed  0          40s
```

Read the logs of the pod that completed the job.

```
[student@workstation project-cleaner]$ oc logs \
pod/project-cleaner-27949861-p262t
Listing namespaces with label workshop:
- namespace: clean-test, created 55.327453 seconds ago...
Deleting namespaces: clean-test
Namespace 'clean-test' deleted
```

**Note**

You might see deleted projects from other exercises in the course.

- 6.5. Verify that the cron job deletes the `clean-test` project, by using the `oc get project` command.

```
[student@workstation project-cleaner]$ oc get project clean-test
Error from server (NotFound): namespaces "clean-test" not found
```

7. Create the beeper database by applying the `beeper-api/beeper-db.yaml` file.

- 7.1. Change to the `~/D0280/labs/comprevew-apps/beeper-api` directory to access the application files.

```
[student@workstation project-cleaner]$ cd ~/D0280/labs/comprevew-apps/beeper-api
```

- 7.2. Use the `oc apply` command to create the database in the `workshop-support` namespace.

```
[student@workstation beeper-api]$ oc apply -f beeper-db.yaml
secret/beeper-db created
service/beeper-db created
persistentvolumeclaim/beeper-db created
deployment.apps/beeper-db created
```

- 7.3. Verify that the database pod is running by using the `oc get pod` command to get the pods with the `app=beeper-db` label.

```
[student@workstation beeper-api]$ oc get pod -l app=beeper-db
NAME                  READY   STATUS    RESTARTS   AGE
beeper-db-688756744f-rgxpg   1/1     Running   0          3m51s
```

8. Configure TLS on the `beeper-api` deployment by using a signed certificate by a corporate CA to accept TLS connections from outside the cluster.

You have the CA certificate and the signed certificate for the `beeper-api.apps.ocp4.example.com` domain in the `beeper-api/certs` directory of the lab.

Use the following settings in the deployment to configure TLS:

- Set the path for the certificate and key to `/etc/pki/beeper-api/`.
- Set the `TLS_ENABLED` environment variable to the `true` value.
- Update the startup, readiness, and liveness probes to use TLS.

- 8.1. Create the beeper-api-cert secret by using the beeper-api.pem certificate and the beeper-api.key key from the lab directory.

```
[student@workstation beeper-api]$ oc create secret tls beeper-api-cert \
--cert certs/beeper-api.pem --key certs/beeper-api.key
secret/beeper-api-cert created
```

- 8.2. Edit the beeper-api deployment in the deployment.yaml file to mount the beeper-api-cert secret on the /etc/pki/beeper-api/ path.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: beeper-api
  namespace: workshop-support
spec:
  ...output omitted...
  spec:
    containers:
      - name: beeper-api
    ...output omitted...
    env:
      - name: TLS_ENABLED
        value: "false"
    volumeMounts:
      - name: beeper-api-cert
        mountPath: /etc/pki/beeper-api/
  volumes:
    - name: beeper-api-cert
      secret:
        defaultMode: 420
        secretName: beeper-api-cert
```

- 8.3. Edit the beeper-api deployment in the deployment.yaml file on lines 32, 37, 42, and 47 to configure TLS for the application and the startup, readiness, and liveness probes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: beeper-api
  namespace: workshop-support
spec:
  ...output omitted...
  spec:
    containers:
      - name: beeper-api
    ...output omitted...
    ports:
      - containerPort: 8080
    readinessProbe:
      httpGet:
        port: 8080
```

```

    path: /readyz
    scheme: HTTPS
  livenessProbe:
    httpGet:
      port: 8080
      path: /livez
      scheme: HTTPS
  startupProbe:
    httpGet:
      path: /readyz
      port: 8080
      scheme: HTTPS
    failureThreshold: 30
    periodSeconds: 3
  env:
    - name: TLS_ENABLED
      value: "true"
...output omitted...

```

- 8.4. Use the `oc apply` command to create the `beeper-api` deployment.

```
[student@workstation beeper-api]$ oc apply -f deployment.yaml
deployment.apps/beeper-api created
```

- 8.5. Edit the `service.yaml` file to configure the `beeper-api` service to listen on the standard HTTPS 443 port and to forward connections to pods with the `app: beeper-api` label on port 8080.

```

apiVersion: v1
kind: Service
metadata:
  name: beeper-api
  namespace: workshop-support
spec:
  selector:
    app: beeper-api
  ports:
    - port: 443
      targetPort: 8080
      name: https

```

- 8.6. Use the `oc apply` command to create the `beeper-api` service.

```
[student@workstation beeper-api]$ oc apply -f service.yaml
service/beeper-api created
```

9. Expose the beeper API to outer cluster access by using the FQDN in the signed certificate by the corporate CA.

- 9.1. Create a passthrough route for the `beeper-api` service by using the `beeper-api.apps.ocp4.example.com` hostname.

```
[student@workstation beeper-api]$ oc create route \
  passthrough beeper-api-https \
  --service beeper-api \
  --hostname beeper-api.apps.ocp4.example.com
route.route.openshift.io/beeper-api-https created
```

- 9.2. Use the curl command to the https://beeper-api.apps.ocp4.example.com/api/beeps URL to verify that the beeper API is accessible from outside the cluster. Add the --cacert option to accept the certs/ca.pem CA.

```
[student@workstation beeper-api]$ curl -s --cacert \
  certs/ca.pem https://beeper-api.apps.ocp4.example.com/api/beeps; echo
[]
```

10. Optionally, open a web browser and verify that you can access the API by navigating to the https://beeper-api.apps.ocp4.example.com/swagger-ui.html URL. When you see the warning about the security risk, click Advanced... and then click Accept the Risk and Continue.
11. Configure network policies to allow only TCP ingress traffic on port 5432 to database pods from the beeper-api pods.
 - 11.1. Verify that you can access the beeper-db service from the workshop-support namespace by testing TCP connectivity to the database service. Use the oc debug command to create a pod with the nc command with the -z option to test TCP access.

```
[student@workstation beeper-api]$ oc debug --to-namespace="workshop-support" -- \
  nc -z -v beeper-db.workshop-support.svc.cluster.local 5432
Starting pod/image-debug ...
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connected to 172.30.219.94:5432.
Ncat: 0 bytes sent, 0 bytes received in 0.02 seconds.

Removing debug pod ...
```

- 11.2. Create an entry in the database by using the following curl command.

```
[student@workstation beeper-api]$ curl -s --cacert certs/ca.pem -X 'POST' \
  'https://beeper-api.apps.ocp4.example.com/api/ beep' \
  -H 'Content-Type: application/json' \
  -d '{ "username": "user1", "content": "first message" }'
```

- 11.3. Edit the db-networkpolicy.yaml file so that only pods with the app: beeper-api label can connect to database pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: database-policy
  namespace: workshop-support
spec:
```

```

podSelector:
  matchLabels:
    app: beeper-db
ingress:
  - from:
    - namespaceSelector:
      matchLabels:
        category: support
    podSelector:
      matchLabels:
        app: beeper-api
  ports:
    - protocol: TCP
      port: 5432

```

- 11.4. Create the network policy.

```
[student@workstation beeper-api]$ oc apply -f db-networkpolicy.yaml
networkpolicy.networking.k8s.io/beeper-api-ingresspolicy created
```

- 11.5. Verify that you cannot connect to the database, by running the previous nc command.

```
[student@workstation beeper-api]$ oc debug --to-namespace="workshop-support" -- \
  nc -z -v beeper-db.workshop-support.svc.cluster.local 5432
Starting pod/image-debug ...
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connection timed out.

Removing debug pod ...
```

- 11.6. Verify that the API pods have access to the database pods, by running the curl command to query the API by using the external route.

```
[student@workstation beeper-api]$ curl -s --cacert \
  certs/ca.pem https://beeper-api.apps.ocp4.example.com/api/beeps; echo
[{"id":1,"username":"user1","content":"first message","votes":0}]
```

12. Configure network policies in the workshop-support namespace to accept only ingress connections from the OpenShift router pods to port 8080.

- 12.1. Verify that you can access the API service from the workshop-support namespace by testing TCP connectivity. Use the oc debug command to create a pod with the nc command with the -Z option to test TCP access.

```
[student@workstation beeper-api]$ oc debug --to-namespace="workshop-support" -- \
  nc -z -v beeper-api.workshop-support.svc.cluster.local 443
Starting pod/image-debug ...
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connected to 172.30.32.28:443.
Ncat: 0 bytes sent, 0 bytes received in 0.02 seconds.

Removing debug pod ...
```

- 12.2. Edit the `beeper-api-ingresspolicy.yaml` file to accept ingress connections from router pods by adding a namespace selector with the `policy-group.network.openshift.io/ingress` label.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: beeper-api-ingresspolicy
  namespace: workshop-support
spec:
  podSelector: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              policy-group.network.openshift.io/ingress: ""
      ports:
        - protocol: TCP
          port: 8080
```

- 12.3. Create the network policy.

```
[student@workstation beeper-api]$ oc apply -f beeper-api-ingresspolicy.yaml
networkpolicy.networking.k8s.io/beeper-api created
```

- 12.4. Verify that you cannot access the API service from the `workshop-support` namespace. Use the `oc debug` command to create a pod with the `nc` command with the `-z` option to test TCP access.

```
[student@workstation beeper-api]$ oc debug --to-namespace="workshop-support" -- \
  nc -z -v beeper-api.workshop-support.svc.cluster.local 443
Starting pod/image-debug ...
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connection timed out.

Removing debug pod ...
```

- 12.5. Verify that the API pods are accessible from outside the cluster by running the `curl` command to query the API external route.

```
[student@workstation beeper-api]$ curl -s --cacert \
  certs/ca.pem https://beeper-api.apps.ocp4.example.com/livez; echo
{"status": "UP"}
```

13. Change to the home directory.

```
[student@workstation appsec-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 grade compreview-apps
```

Finish

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

```
[student@workstation ~]$ lab finish compreview-apps
```

▶ Lab

Deploy Packaged Applications

Deploy a Helm chart

Deploy an application with Kustomize

Expose a service using MetalLB

Outcomes

- Deploy an application from a chart.
- Deploy an application from a Kustomize layer.
- Configure an application to connect to the MySQL database.

Before You Begin

If you did not previously reset your `workstation` and `server` machines, then save any work that you want to keep from earlier exercises on those machines, and reset them now.

Use the `lab` command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and prepares the environment for the exercise.

```
[student@workstation ~]$ lab start compreview-package
```

Specifications

Deploy an application that uses a database by using a Helm chart and Kustomization files. Access the application by using a route.

- Use the `developer` user with the `developer` password for this exercise.
- Use a `comprevue-package` project for all the resources.
- Deploy a MySQL database by using the `mysql-persistent` Helm chart in the `http://helm.ocp4.example.com/charts` repository. Use the latest version in the repository, and the default resource names that the chart generates.
- Use Kustomize in the `/home/student/D0280/labs/comprevue-package/roster/` path to deploy the application. Add a new Kustomize `production` overlay that adds probes to the application.

The `/home/student/D0280/solutions/comprevue-package/roster/overlays/production/` directory contains the solution `kustomization.yaml` file and the `patch-roster-prod.yaml` file.

- Deploy the overlay.
- Verify that the application creates a route, and that the application is available through the route by using the TLS/SSL protocol (HTTPS).

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 grade compreview-package
```

Finish

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

```
[student@workstation ~]$ lab finish compreview-package
```

► Solution

Deploy Packaged Applications

Deploy a Helm chart

Deploy an application with Kustomize

Expose a service using MetalLB

Outcomes

- Deploy an application from a chart.
- Deploy an application from a Kustomize layer.
- Configure an application to connect to the MySQL database.

Before You Begin

If you did not previously reset your **workstation** and **server** machines, then save any work that you want to keep from earlier exercises on those machines, and reset them now.

Use the `lab` command to prepare your system for this exercise.

This command ensures that the cluster API is reachable and prepares the environment for the exercise.

```
[student@workstation ~]$ lab start comprevue-package
```

1. Add the classroom Helm repository at `http://helm.ocp4.example.com/charts` and examine its contents.
 - 1.1. Use the `helm repo list` command to list the repositories that are configured for the student user.

```
[student@workstation ~]$ helm repo list
Error: no repositories to show
```

If the `do280-repo` repository is present, then continue to the next step. Otherwise, add the repository.

```
[student@workstation ~]$ helm repo add \
do280-repo http://helm.ocp4.example.com/charts
"do280-repo" has been added to your repositories
```

- 1.2. Use the `helm search` command to list all the charts in the repository.

```
[student@workstation ~]$ helm search repo
NAME          CHART VERSION APP VERSION DESCRIPTION
do280-repo/mysql-persistent 0.0.2      0.0.2      This content is...
...output omitted...
```

The `mysql-persistent` chart is in the classroom repository. This chart is a copy of a chart from the <https://github.com/openshift-helm-charts/charts/> repository.

2. Create a `comprevue-package` project.

2.1. Log in to the cluster as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
```

2.2. Create the `comprevue-package` project.

```
[student@workstation ~]$ oc new-project comprevue-package
Now using project "comprevue-package" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

3. Deploy the `do280-repo/mysql-persistent` chart.

3.1. Use the `helm install` command to create a release of the `do280-repo/mysql-persistent` chart.

```
[student@workstation ~]$ helm install roster-database do280-repo/mysql-persistent
...output omitted...
```

3.2. Use the `watch` command to verify that the pods are running. Wait for the `mysql-1-deploy` pod to show a `Completed` status.

```
[student@workstation ~]$ watch oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql-1-7w5bn 1/1     Running   0          150m
mysql-1-deploy 0/1     Completed  0          150m
```

Press `Ctrl+C` to exit the `watch` command.

4. Examine the provided Kustomize configuration and the deployed chart, and verify that the production overlay generates a deployment, service, route, configuration map, and a secret. Verify that the `patch-roster-prod.yaml` patch file applies the liveness and readiness probes to the `roster` deployment.

4.1. Change to the `/home/student/D0280/labs/comprevue-package/` directory.

```
[student@workstation ~]$ cd D0280/labs/comprevue-package/
```

4.2. Use the `tree` command to examine the directory structure.

```
[student@workstation compreview-package]$ tree
.
└── roster
    ├── base
    │   ├── configmap.yaml
    │   ├── kustomization.yaml
    │   ├── roster-deployment.yaml
    │   ├── roster-route.yaml
    │   ├── roster-service.yaml
    │   └── secret.yaml
    └── overlays
        └── production
            ├── kustomization.yaml
            └── patch-roster-prod.yaml

4 directories, 8 files
```

The Kustomization configuration includes a deployment, a route, and a service.

4.3. Examine the roster/base/roster-deployment.yaml file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
spec:
  replicas: 1
  selector:
    matchLabels:
      app: roster
  template:
...output omitted...
  spec:
    containers:
    - image: registry.ocp4.example.com:8443/redhattraining/do280-roster:v1
      name: do280-roster
      envFrom:
      - configMapRef:
          name: roster
      - secretRef:
          name: roster
...output omitted...
```

The deployment does not set any configuration to access the database. The deployment extracts environment variables from a `roster` configuration map and a `roster` secret.

4.4. Use the oc kustomize command to verify that the production overlay generates a deployment, service, route, configuration map, and a secret, and configures the liveness and readiness probes to the roster deployment.

```
[student@workstation compreview-package]$ oc kustomize roster/overlays/production/
apiVersion: v1
data:
```

```

...output omitted...
kind: ConfigMap
...output omitted...
---
apiVersion: v1
kind: Secret
...output omitted...
---
apiVersion: v1
kind: Service
...output omitted...
---
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
spec:
...output omitted...
  template:
...output omitted...
    spec:
      containers:
...output omitted...
livenessProbe:
  initialDelaySeconds: 20
  periodSeconds: 30
  tcpSocket:
    port: 9090
  timeoutSeconds: 3
  name: roster
ports:
- containerPort: 9090
  protocol: TCP
readinessProbe:
  initialDelaySeconds: 3
  periodSeconds: 10
  tcpSocket:
    port: 9090
  timeoutSeconds: 3
---
apiVersion: route.openshift.io/v1
kind: Route
...output omitted...

```

5. Deploy the Kustomize files.

- 5.1. Use the `oc apply -k` command to deploy the production overlay.

```
[student@workstation comprevieew-package]$ oc apply -k roster/overlays/production/
configmap/roster created
secret/roster created
service/roster created
deployment.apps/roster created
route.route.openshift.io/roster unchanged
```

- 5.2. Use the `watch` command to verify that the pods are running. Wait for the `roster` pod to show the Running status.

```
[student@workstation compreview-package]$ watch oc get pods
NAME          READY   STATUS    RESTARTS   AGE
...output omitted...
roster-5d4888dc6f-4rp4n   1/1     Running   0          166m
```

Press `Ctrl+C` to exit the `watch` command.

- 5.3. Use the `oc get route` command to obtain the application URL.

```
[student@workstation compreview-package]$ oc get route
NAME      HOST/PORT   ...
roster   roster-compreview-package.apps.ocp4.example.com ...
```

- 5.4. Open a web browser and navigate to `https://roster-compreview-package.apps.ocp4.example.com`. Use the TLS/SSL protocol (HTTPS). The application is displayed.

- 5.5. Change to the home directory.

```
[student@workstation compreview-package]$ 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 grade compreview-package
```

Finish

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish compreview-package
```