



# Red Hat

## Training and Certification

### Student Workbook (ROLE)

OCP 4.14 DO180

### Red Hat OpenShift Administration I: Operating a Production Cluster

Edition 1







## 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](https://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**

## **I: Operating a Production Cluster**



**OCP 4.14 DO180**

**Red Hat OpenShift Administration I: Operating a Production Cluster**

**Edition 1 20240111**

**Publication date 20240111**

Authors: Guy Bianco IV, Christopher Caillouet, Alex Corcoles, Natalie Lind, Maria Ordóñez, Randy Thomas, Hervé Quatremain, Manna Kong  
Course Architect: Fernando Lozano  
DevOps Engineers: Benjamin Marco, Zach Guterman  
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) [mailto:[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: **Sourabh Mishra, Ted Singdal**

<b>Document Conventions</b>	<b>ix</b>
Admonitions .....	xi
Inclusive Language .....	x
<b>Introduction</b>	<b>xi</b>
Red Hat OpenShift Administration I: Operating a Production Cluster .....	xi
Orientation to the Classroom Environment .....	xii
Performing Lab Exercises .....	xviii
<b>1. Introduction to Kubernetes and OpenShift</b>	<b>1</b>
Containers and Kubernetes .....	2
Quiz: Containers and Kubernetes .....	6
Red Hat OpenShift Components and Editions .....	8
Quiz: Red Hat OpenShift Components and Editions .....	12
Navigate the OpenShift Web Console .....	14
Guided Exercise: Navigate the OpenShift Web Console .....	19
Monitor an OpenShift Cluster .....	33
Guided Exercise: Monitor an OpenShift Cluster .....	39
Quiz: Introduction to Kubernetes and OpenShift .....	50
Lab: Introduction to Kubernetes and OpenShift .....	54
Summary .....	61
<b>2. Kubernetes and OpenShift Command-line Interfaces and APIs</b>	<b>63</b>
The Kubernetes and OpenShift Command-line Interfaces .....	64
Guided Exercise: The Kubernetes and OpenShift Command-line Interfaces .....	75
Inspect Kubernetes Resources .....	83
Guided Exercise: Inspect Kubernetes Resources .....	94
Assess the Health of an OpenShift Cluster .....	105
Guided Exercise: Assess the Health of an OpenShift Cluster .....	119
Lab: Kubernetes and OpenShift Command-line Interfaces and APIs .....	129
Quiz: Kubernetes and OpenShift Command-line Interfaces and APIs .....	138
Summary .....	142
<b>3. Run Applications as Containers and Pods</b>	<b>143</b>
Create Linux Containers and Kubernetes Pods .....	144
Guided Exercise: Create Linux Containers and Kubernetes Pods .....	155
Find and Inspect Container Images .....	164
Guided Exercise: Find and Inspect Container Images .....	177
Troubleshoot Containers and Pods .....	189
Guided Exercise: Troubleshoot Containers and Pods .....	197
Lab: Run Applications as Containers and Pods .....	203
Quiz: Run Applications as Containers and Pods .....	211
Summary .....	215
<b>4. Deploy Managed and Networked Applications on Kubernetes</b>	<b>217</b>
Deploy Applications from Images and Templates .....	218
Guided Exercise: Deploy Applications from Images and Templates .....	231
Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API .....	237
Guided Exercise: Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API .....	241
Kubernetes Pod and Service Networks .....	246
Guided Exercise: Kubernetes Pod and Service Networks .....	256
Scale and Expose Applications to External Access .....	264
Guided Exercise: Scale and Expose Applications to External Access .....	271
Lab: Deploy Managed and Networked Applications on Kubernetes .....	278
Summary .....	286

<b>5. Manage Storage for Application Configuration and Data</b>	<b>287</b>
Externalize the Configuration of Applications .....	288
Guided Exercise: Externalize the Configuration of Applications .....	300
Provision Persistent Data Volumes .....	305
Guided Exercise: Provision Persistent Data Volumes .....	314
Select a Storage Class for an Application .....	321
Guided Exercise: Select a Storage Class for an Application .....	327
Manage Non-shared Storage with Stateful Sets .....	335
Guided Exercise: Manage Non-shared Storage with Stateful Sets .....	341
Lab: Manage Storage for Application Configuration and Data .....	348
Summary .....	361
<b>6. Configure Applications for Reliability</b>	<b>363</b>
Application High Availability with Kubernetes .....	364
Guided Exercise: Application High Availability with Kubernetes .....	366
Application Health Probes .....	372
Guided Exercise: Application Health Probes .....	376
Reserve Compute Capacity for Applications .....	382
Guided Exercise: Reserve Compute Capacity for Applications .....	385
Limit Compute Capacity for Applications .....	390
Guided Exercise: Limit Compute Capacity for Applications .....	397
Application Autoscaling .....	401
Guided Exercise: Application Autoscaling .....	404
Lab: Configure Applications for Reliability .....	411
Quiz: Configure Applications for Reliability .....	419
Summary .....	421
<b>7. Manage Application Updates</b>	<b>423</b>
Container Image Identity and Tags .....	424
Guided Exercise: Container Image Identity and Tags .....	430
Update Application Image and Settings .....	439
Guided Exercise: Update Application Image and Settings .....	448
Reproducible Deployments with OpenShift Image Streams .....	455
Guided Exercise: Reproducible Deployments with OpenShift Image Streams .....	463
Automatic Image Updates with OpenShift Image Change Triggers .....	466
Guided Exercise: Automatic Image Updates with OpenShift Image Change Triggers .....	472
Lab: Manage Application Updates .....	477
Summary .....	482
<b>8. Comprehensive Review</b>	<b>483</b>
Comprehensive Review .....	484
Lab: Deploy Web Applications .....	487
Lab: Troubleshoot and Scale Applications .....	495

# 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 I: Operating a Production Cluster

This course prepares OpenShift Cluster Administrators to perform day-to-day management of Kubernetes workloads and collaborate with Developers, DevOps Engineers, System Administrators, and SREs to ensure the availability of application workloads. DO180 focuses on managing typical end-user applications. These applications are accessible from a web or mobile UI and represent the majority of cloud native and containerized workloads. Management of applications also include deployment of their dependencies such as databases, messaging, and authentication systems. This course is based on Red Hat® OpenShift® Container Platform 4.14.

### Course Objectives

- Managing OpenShift clusters from the command-line interface and from the web console.
- Deploying applications on OpenShift from container images, templates, and Kubernetes manifests.
- Troubleshooting network connectivity between applications inside and outside an OpenShift cluster.
- Connecting Kubernetes workloads to storage for application data.
- Configuring Kubernetes workloads for high availability and reliability.
- Managing updates to container images, settings, and Kubernetes manifests of an application.

### Audience

- System Administrators interested in the ongoing management of OpenShift clusters and containerized applications.
- Site Reliability Engineers interested in the ongoing maintenance and troubleshooting of containerized applications on Kubernetes.
- System and Software Architects interested in understanding the features and functionality of an OpenShift cluster.

### Prerequisites

- Containers, Kubernetes and Red Hat OpenShift Technical Overview (DO080) at <https://www.redhat.com/en/services/training/do080-deploying-containerized-applications-technical-overview> or equivalent knowledge of Linux containers.
- Significant Linux experience is not needed for this course. Basic skills operating a Bash shell are sufficient.

# Orientation to the Classroom Environment

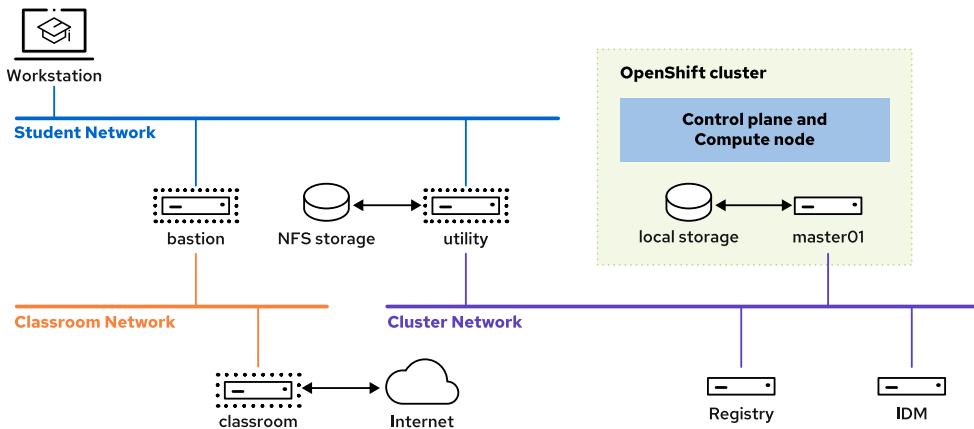
---

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) 4.12 single-node (SNO) bare metal 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**, which has the **student** 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.

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	<a href="https://console-openshift-console.apps.ocp4.example.com">https://console-openshift-console.apps.ocp4.example.com</a>
API	<a href="https://api.ocp4.example.com:6443">https://api.ocp4.example.com:6443</a>

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 DO180 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**

<b>Classroom Registry Repository</b>	<b>Public Source Repository</b>
redhattraining/docker-nginx	<code>docker.io/library/nginx</code> <code>quay.io/redhattraining/docker-nginx</code>
redhattraining/bitnami-mysql	<code>docker.io/bitnami/mysql</code> <code>quay.io/redhattraining/bitnami-mysql</code>
redhattraining/do180-dbinit	<code>quay.io/redhattraining/do180-dbinit</code>
redhattraining/do180-httpd-app	<code>quay.io/redhattraining/do180-httpd-app</code>
redhattraining/do180-roster	<code>quay.io/redhattraining/do180-roster</code>
redhattraining/famous-quotes	<code>quay.io/redhattraining/famous-quotes</code>
redhattraining/hello-world-nginx	<code>quay.io/redhattraining/hello-world-nginx</code>
redhattraining/httpd-noimage	<code>quay.io/redhattraining/httpd-noimage</code>
redhattraining/long-load	<code>quay.io/redhattraining/long-load</code>
redhattraining/loadtest	<code>quay.io/redhattraining/loadtest</code>
redhattraining/mysql-app	<code>quay.io/redhattraining/mysql-app</code>
redhattraining/php-ssl	<code>quay.io/redhattraining/php-ssl</code>
redhattraining/php-webapp	<code>quay.io/redhattraining/php-webapp</code>
redhattraining/php-webapp-mysql	<code>quay.io/redhattraining/php-webapp-mysql</code>
redhattraining/versioned-hello	<code>quay.io/redhattraining/versioned-hello</code>
redhattraining/webphp	<code>quay.io/redhattraining/webphp</code>

Classroom Registry Repository	Public Source Repository
rhel8/mysql-80	registry.redhat.io/rhel8/mysql-80
rhel9/mysql-80	registry.redhat.io/rhel9/mysql-80
ubi8/httpd-24	registry.access.redhat.com/ubi8/httpd-24
ubi8/ubi	registry.access.redhat.com/ubi8/ubi
ubi9/ubi	registry.access.redhat.com/ubi9/ubi

## 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) [<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 a lab environment. At the top, there are tabs for 'Table of Contents', 'Course' (which is selected), 'Lab Environment', and other icons. Below the tabs, a section titled 'Lab Controls' contains instructions for creating and deleting the lab environment. It says to click 'CREATE' to build all virtual machines and 'DELETE' to remove them. A note states that once created, the environment can be stopped and restarted. Below this is a table of virtual machines:

		ACTION -	OPEN CONSOLE
bastion	active		
classroom	active		
servera	building		
serverb	building		
workstation	active		

At the bottom of the table, there are buttons for 'DELETE', 'STOP', and an information icon. The 'STOP' button is highlighted in red.

Figure 0.2: An example course Lab Environment management page

### Machine States

Virtual machine state	Description
building	The virtual machine is being created.

Virtual machine state	Description
active	The virtual machine is running and available. If it just started, it still might be starting services.
stopped	The virtual machine is shut down. On starting, the virtual machine boots into the same state 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 virtual machines that are needed for this classroom.
CREATING	The ROLE classroom virtual machines are being created. Creation can take several minutes to complete.
DELETE	Delete the ROLE classroom. Destroys all virtual machines in the classroom. <b>All saved work on those systems' disks is lost.</b>
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 <b>workstation</b> virtual machine only, and from there, use <b>ssh</b> to connect to the other virtual machines.
ACTION > Start	Start (power on) the virtual machine.
ACTION > Shutdown	Gracefully shut down the virtual machine, preserving disk contents.
ACTION > Power Off	Forcefully shut down the virtual machine, while still preserving disk contents. This action is equivalent to removing the power from a physical machine.
ACTION > Reset	Forcefully shut down the virtual machine and reset associated storage to its initial state. <b>All saved work on that system's disks is lost.</b>

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

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

If you want to return the classroom environment to its original state at the start of the course, then click **DELETE** to remove the entire classroom environment. After the lab is deleted, then 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 DO180 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/D0180`.

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/D0180/labs/updates-rollout` workspace.
- Copies the files for the activity to the `/home/student/D0180/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

# Introduction to Kubernetes and OpenShift

### Goal

Identify the main Kubernetes cluster services and OpenShift platform services and monitor them by using the web console.

### Objectives

- Describe the main characteristics of containers and Kubernetes.
- Describe the relationship between OpenShift, Kubernetes, and other Open Source projects, and list key features of Red Hat OpenShift products and editions.
- Navigate the OpenShift web console to identify running applications and cluster services.
- Navigate the Events, Compute, and Observe panels of the OpenShift web console to assess the overall state of a cluster.

### Sections

- Containers and Kubernetes (and Matching Quiz)
- Red Hat OpenShift Components and Editions (and Matching Quiz)
- Navigate the OpenShift Web Console (and Guided Exercise)
- Monitor an OpenShift Cluster (and Guided Exercise)

### Lab

- Introduction to Kubernetes and OpenShift

# Containers and Kubernetes

## Objectives

- Describe the main characteristics of containers and Kubernetes.

## Introduction to Containers and Kubernetes

Red Hat OpenShift Container Platform (RHOC) is a complete platform to run applications in clusters of servers. OpenShift is based on existing technologies such as *containers* and Kubernetes. Containers provide a way to package applications and their dependencies that can be executed on any system with a container runtime.

Kubernetes provides many features to build clusters of servers that run containerized applications. However, Kubernetes does not intend to provide a complete solution, but rather provides extension points so system administrators can complete Kubernetes. OpenShift builds on the extension points of Kubernetes to provide a complete platform.

## Containers Overview

A *container* is a process that runs an application independently from other containers on the host. Containers are created from a *container image*, which includes all the runtime dependencies of the application. Containers can then be executed on any host, without requiring the installation of any dependency on the host, and without conflicts between the dependencies of different containers.

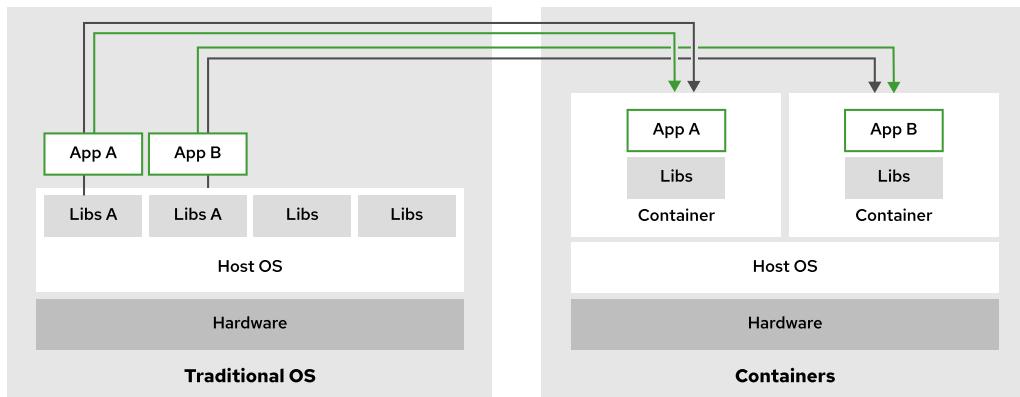


Figure 1.1: Applications in containers versus on the host operating system

Containers use Linux kernel features, such as namespaces and control groups (cgroups). For example, containers use cgroups for resource management, such as CPU time allocation and system memory. Namespaces isolate a container's processes and resources from other containers and the host system.

The Open Container Initiative (OCI) maintains standards about containers, container images, and container runtimes. Because most container engine implementations are designed to conform to the OCI specifications, applications that are packaged according to the specification can run on any conforming platform.

## Kubernetes Overview

Kubernetes is a platform that simplifies the deployment, management, and scaling of containerized applications.

You can run containers on any Linux system by using tools such as Podman. Further work is required to create containerized services that, for example, can run across a cluster for redundancy.

Kubernetes creates a cluster that runs applications across multiple nodes. If a node fails, then Kubernetes can restart an application in a different node.

Kubernetes uses a declarative configuration model. Kubernetes administrators write a definition of the workloads to execute in the cluster, and Kubernetes ensures that the running workloads match the definition. For example, an administrator defines several workloads. Each workload defines the amount of required memory. Kubernetes then creates the necessary containers in different nodes, to meet the memory requirements.

Kubernetes defines workloads in terms of *pods*. A pod is one or multiple containers that run in the same cluster node. Pods with multiple containers are useful in certain situations, when two containers must run in the same cluster node to share some resource. For example, a job is a workload that runs a task in a pod until completion.

## Kubernetes Features

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

### **Service discovery and load balancing**

Distributing applications over multiple nodes can complicate communication between applications.

Kubernetes automatically configures networking and provides a DNS service for pods. With these features, pods can communicate with services from other pods transparently across nodes by using only hostnames instead of IP addresses. Multiple pods can back a service for performance and reliability. For example, Kubernetes can evenly split incoming requests to an NGINX web server by considering the availability of the NGINX pods.

### **Horizontal scaling**

Kubernetes can monitor the load on a service, and create or delete pods to adapt to the load. With some configurations, Kubernetes can also provision nodes dynamically to accommodate cluster load.

### **Self-healing**

If applications declare health check procedures, then Kubernetes can monitor, restart, and reschedule failing or unavailable applications. Self-healing protects applications both from internal failure (the application stops unexpectedly) or external failure (the node that runs the application becomes unavailable).

### **Automated rollout**

Kubernetes can gradually roll out updates to your application's containers. If something goes wrong during the rollout, then Kubernetes can roll back to the previous version of the deployment. Kubernetes routes requests to the rolled out version of the application, and deletes pods from the previous version when the rollout completes.

### **Secrets and configuration management**

You can manage the configuration settings and secrets of your applications without requiring changes to containers. Application secrets can be usernames, passwords, and service endpoints, or any configuration setting that must be kept private.



### Note

Kubernetes does not encrypt secrets.

## Operators

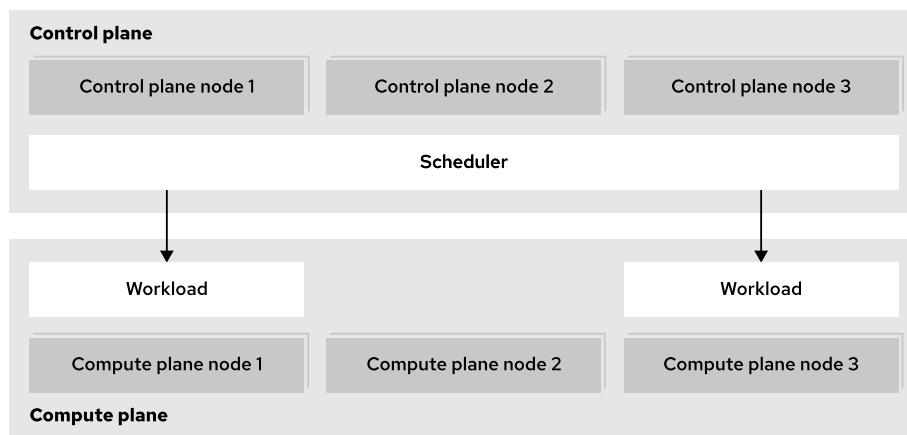
Operators are packaged Kubernetes applications that can manage Kubernetes workloads in a declarative manner. For example, an operator can define a database server resource. If the user creates a database resource, then the operator creates the necessary workloads to deploy the database, configure replication, and back up automatically.

## Kubernetes Architectural Concepts

Kubernetes uses several servers (also called *nodes*) to ensure the resilience and scalability of the applications that it manages. These nodes can be physical or virtual machines. The nodes come in two types, each of which provides a different aspect of the cluster operations.

*Control plane* nodes provide global cluster coordination for scheduling the deployed workloads. These nodes also manage the state of the cluster configuration in response to cluster events and requests.

*Compute plane* nodes run the user workloads.



**Figure 1.2: Kubernetes components**

Although a server can act as both a control plane node and a compute node, the two roles are usually separated for increased stability, security, and manageability. Kubernetes clusters can range from small single-node clusters, to large clusters with up to 5,000 nodes.

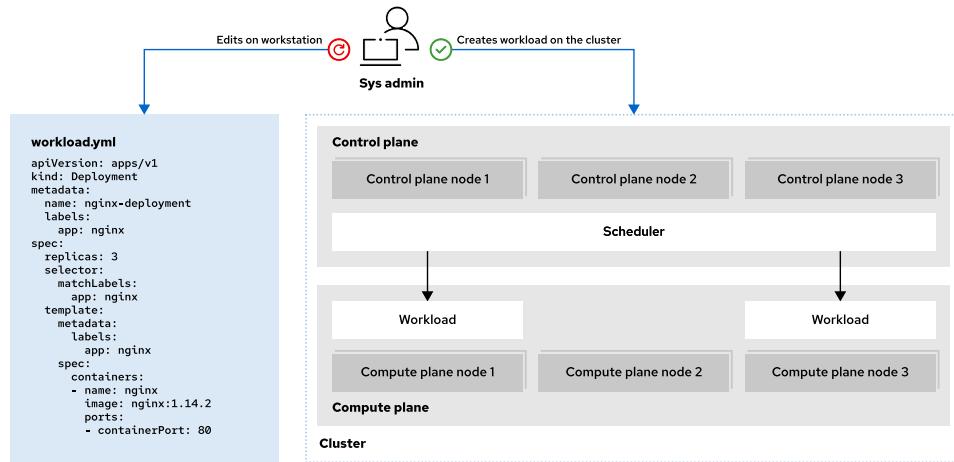
## The Kubernetes API and Configuration Model

Kubernetes provides a model to define the workloads to run on a cluster.

Administrators define workloads in terms of resources.

All resource types work in the same way, with a uniform API. Tools such as the `kubectl` command can manage resources of all types, even custom resource types.

Kubernetes can import and export resources as text files. By working with resource definitions in text formats, administrators can describe their workloads instead of performing the right sequence of operations to create them. This approach is called *declarative resource management*.



**Figure 1.3: Declarative resource management**

Declarative resource management can reduce the work to create workloads and improve maintainability. Additionally, when using text files to describe resources, administrators can use generic tools such as version control systems to gain further benefits such as change tracking.

To support declarative resource management, Kubernetes uses *controllers* that continuously track the state of the cluster and perform the necessary steps to keep the cluster in the intended state. This process means that changes to resources often require some time to be effective. However, Kubernetes can automatically apply complex changes. For example, if you increase the RAM requirements of an application that cannot be satisfied on the current node, then Kubernetes can move the application to a node with sufficient RAM. Kubernetes redirects traffic to the new instance only when the movement is complete.

Kubernetes also supports *namespaces*. Administrators can create namespaces, and most resources must be created inside a namespace. Besides helping organize large amounts of resources, namespaces provide the foundations for features such as resource access. Administrators can define permissions for namespaces, to allow specific users to view or modify resources.



## References

### Containers

<https://kubernetes.io/docs/concepts/containers/>

### Kubernetes Concepts Overview

<https://kubernetes.io/docs/concepts/overview/>

### What Kubernetes Is Not

<https://kubernetes.io/docs/concepts/overview/#what-kubernetes-is-not>

### Considerations for Large Clusters

<https://kubernetes.io/docs/setup/best-practices/cluster-large/>

## ► Quiz

# Containers and Kubernetes

Match the following items to their counterparts in the table.

Application packaging format that is compatible with container runtimes

Automatic change in the number of replicas of an application to respond to load

Automatic network configuration for transparent cluster communication

Automatic restart of failing or unavailable applications

Configuration of applications externally from their container images

Management of instances of different versions of an application

Reconciliation of a description of the intended state of a cluster

Concept	Description
Service discovery and load balancing	
Horizontal scaling	
Self-healing	
Automated rollout	
Secrets and configuration management	
Declarative resource management	
Container images	

## ► Solution

# Containers and Kubernetes

Match the following items to their counterparts in the table.

Concept	Description
Service discovery and load balancing	Automatic network configuration for transparent cluster communication
Horizontal scaling	Automatic change in the number of replicas of an application to respond to load
Self-healing	Automatic restart of failing or unavailable applications
Automated rollout	Management of instances of different versions of an application
Secrets and configuration management	Configuration of applications externally from their container images
Declarative resource management	Reconciliation of a description of the intended state of a cluster
Container images	Application packaging format that is compatible with container runtimes

# Red Hat OpenShift Components and Editions

---

## Objectives

- Describe the relationship between OpenShift, Kubernetes, and other Open Source projects, and list key features of Red Hat OpenShift products and editions.

## Introduction to Red Hat OpenShift

Kubernetes provides many features to run container workloads on clusters. However, for some features, Kubernetes provides only the building blocks to implement them, because different environments might need different solutions. Kubernetes administrators can select an existing solution or implement their own solution to fit their specific requirements.

OpenShift uses the extensibility of Kubernetes to build a complete solution by adding the following features to a Kubernetes cluster:

### Integrated developer workflow

When running applications on Kubernetes, you need to build and store container images for the applications. OpenShift integrates a built-in container registry, CI/CD pipelines, and S2I, a tool to build artifacts from source repositories to container images.

### Observability

To achieve the intended reliability, performance, and availability of applications, cluster administrators might need additional tools to prevent and solve issues. OpenShift includes monitoring and logging services for both your applications and the cluster.

### Server management

Kubernetes requires an operating system to run on that must be installed, configured, and maintained.

OpenShift provides installation and update procedures for many scenarios.

Additionally, hosts in a cluster use Red Hat Enterprise Linux CoreOS (RHEL CoreOS) as the underlying operating system. RHEL CoreOS is an immutable operating system that is optimized for running containerized applications. RHEL CoreOS uses the same kernel and packages as Red Hat Enterprise Linux. OpenShift also provides features to manage RHEL CoreOS following the Kubernetes configuration model.

OpenShift also brings unified tools and a graphical web console to manage all the different capabilities, and additional enhancements such as improved security measures.

This diagram shows many of the projects that provide the various functions within each OpenShift cluster:

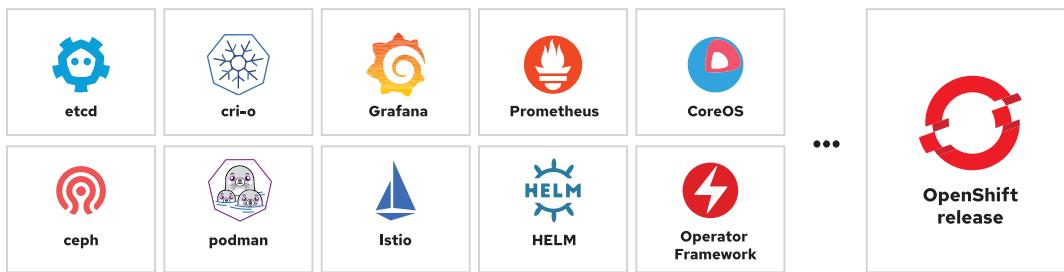


Figure 1.4: Open source projects in an OpenShift release

Because OpenShift features build on Kubernetes extensibility, administrators can often operate these features by using their existing Kubernetes knowledge. Besides Kubernetes knowledge, OpenShift administrators can use most existing products that are designed for Kubernetes.

## Red Hat OpenShift Editions

When you initially explore OpenShift, using Red Hat OpenShift Local is a viable approach that deploys a cluster on a local computer for testing and exploration. Red Hat also provides a Developer Sandbox where you get 30 days of free access to a shared OpenShift cluster. These options provide access to a cluster and support testing and exploration as you consider adopting OpenShift, but are not suitable environments for production deployments.

When you are ready to adopt Red Hat OpenShift for production workloads, various editions are available to suit any business requirement for the cluster deployment.

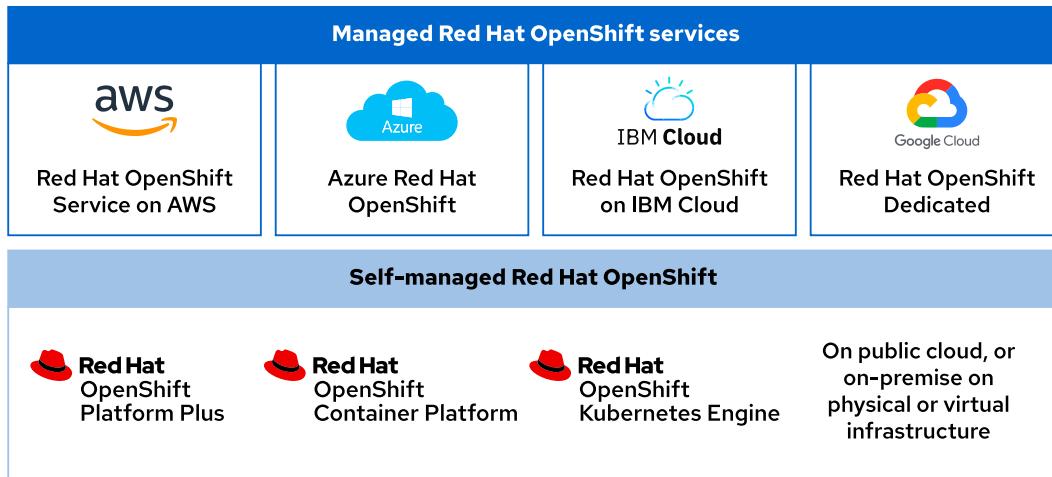


Figure 1.5: Product editions

Public cloud partners, such as Amazon Web Services, Microsoft Azure, IBM Cloud, and Google Cloud each provide quick access to an on-demand Red Hat OpenShift deployment. These managed deployments offer quick access to a cluster on infrastructure that you can rely on from a Red Hat trusted cloud provider.

You can also deploy a Red Hat OpenShift cluster by using the available installers on physical or virtual infrastructure, either on-premise or in a public cloud. These self-managed offerings are available in several forms.

The choice of deployment methods depends on many factors. When using the managed services, more responsibilities are delegated to Red Hat and the cloud provider. Because the installation process integrates with the cloud provider, the managed service creates and manages all necessary cloud resources.

When using the installers, you can still delegate hardware management to a cloud provider, or use your own. However, you manage the rest of the solution. With self-managed editions, you have greater control and flexibility, but you also take on greater responsibilities over the service.

For example, with managed services, the Red Hat Site Reliability Engineering teams update the clusters and remedy update issues (although you still participate in scheduling the updates). On self-managed editions, you update the clusters and remedy update issues. On the other hand, on self-managed editions you have complete control over aspects such as authentication, when managed editions might restrict some options.

Each managed edition documents the responsibilities for customers, Red Hat, and the cloud provider.

Additionally, to assist in managing clusters, the Red Hat Insights Advisor is available from the Red Hat Hybrid Cloud Console. The Insights Advisor helps administrators to identify and remediate cluster issues by analyzing data that the Insights Operator provides. The data from the operator is uploaded to the Red Hat Hybrid Cloud Console, where you further inspect the recommendations and their impact on the cluster.

The contents of this course apply to both managed services and self-managed editions.

Red Hat OpenShift Kubernetes Engine includes the latest version of the Kubernetes platform with the additional security hardening and enterprise stability that Red Hat is famous for delivering. This deployment runs on the Red Hat Enterprise Linux CoreOS immutable container operating system, by using Red Hat OpenShift Virtualization for virtual machine management, and provides an administrator console to aid in operational support.

Red Hat OpenShift Container Platform builds on the features of the OpenShift Kubernetes Engine to include additional cluster manageability, security, stability, and ease of application development for businesses. Additional features of this tier include a developer console, as well as log management, cost management, and metering information. This offering adds Red Hat OpenShift Serverless (Knative), Red Hat OpenShift Service Mesh (Istio), Red Hat OpenShift Pipelines (Tekton), and Red Hat OpenShift GitOps (Argo CD) to the deployment.

Red Hat OpenShift Platform Plus expands further on the offering to deliver the most valuable and robust available features. This offering includes Red Hat Advanced Cluster Management for Kubernetes, Red Hat Advanced Cluster Security for Kubernetes, and the Red Hat Quay private registry platform. For the most complete and full-featured container experience, Red Hat OpenShift Platform Plus bundles all the necessary tools for a complete development and administrative approach to containerized application platform management.

All OpenShift editions use the same code. Most content in this course applies to all OpenShift editions.



## References

### What Kubernetes Is Not

<https://kubernetes.io/docs/concepts/overview/#what-kubernetes-is-not>

For more information, refer to the *OpenShift Container Platform Overview* section in the *OpenShift Container Platform Architecture* chapter in the Red Hat OpenShift Container Platform 4.14 Architecture documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/architecture/index#architecture-platform-benefits\\_architecture](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/architecture/index#architecture-platform-benefits_architecture)

For more information, refer to the *About Red Hat OpenShift Local* section in the *Introducing Red Hat OpenShift Local* chapter in the Red Hat Red Hat OpenShift Local 2.30 Getting Started Guide documentation at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_openshift\\_local/2.30/html-single/getting\\_started\\_guide/index#about\\_gsg](https://access.redhat.com/documentation/en-us/red_hat_openshift_local/2.30/html-single/getting_started_guide/index#about_gsg)

For more information, refer to the *Supported Platforms for OpenShift Container Platform Clusters* section in the *Installation and Update* chapter in the Red Hat OpenShift Container Platform 4.14 Architecture documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/architecture/index#supported-platforms-for-openshift-clusters\\_architecture-installation](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/architecture/index#supported-platforms-for-openshift-clusters_architecture-installation)

### Red Hat OpenShift Cloud Services

<https://www.redhat.com/en/technologies/cloud-computing/openshift/openshift-cloud-services>

### Overview of Responsibilities for Red Hat OpenShift Service on AWS

<https://docs.aws.amazon.com/ROSA/latest/userguide/rosa-responsibilities.html>

### Red Hat OpenShift Kubernetes Engine

<https://www.redhat.com/en/technologies/cloud-computing/openshift/kubernetes-engine>

### Red Hat OpenShift Platform Plus

<https://www.redhat.com/en/technologies/cloud-computing/openshift/platform-plus>

## ► Quiz

# Red Hat OpenShift Components and Editions

Match the following items to their counterparts in the table.

A complete application platform that builds on the extensibility of Kubernetes

A system to deploy OpenShift to a local computer for testing and exploration

An application platform that is based on containers that does not provide features such as building applications

An extended offering that includes a container registry

An immutable operating system that is optimized for running containerized applications

OpenShift offerings where Red Hat and cloud providers assume responsibilities

OpenShift offerings where you have greater control, flexibility, and responsibilities

Concept	Description
Kubernetes	
Red Hat OpenShift Container Platform	
Red Hat OpenShift Platform Plus	
Red Hat OpenShift Local	
OpenShift managed services	
OpenShift self-managed offerings	
Red Hat Enterprise Linux CoreOS	

## ► Solution

# Red Hat OpenShift Components and Editions

Match the following items to their counterparts in the table.

Concept	Description
Kubernetes	An application platform that is based on containers that does not provide features such as building applications
Red Hat OpenShift Container Platform	A complete application platform that builds on the extensibility of Kubernetes
Red Hat OpenShift Platform Plus	An extended offering that includes a container registry
Red Hat OpenShift Local	A system to deploy OpenShift to a local computer for testing and exploration
OpenShift managed services	OpenShift offerings where Red Hat and cloud providers assume responsibilities
OpenShift self-managed offerings	OpenShift offerings where you have greater control, flexibility, and responsibilities
Red Hat Enterprise Linux CoreOS	An immutable operating system that is optimized for running containerized applications

# Navigate the OpenShift Web Console

---

## Objectives

- Navigate the OpenShift web console to identify running applications and cluster services.

## Overview of the Red Hat OpenShift Web Console

The Red Hat OpenShift Web Console provides a graphical user interface to perform many administrative tasks for managing a cluster. The web console uses the Kubernetes APIs and OpenShift extension APIs to deliver a robust graphical experience. The menus, tasks, and features within the web console are always available by using the CLI. The web console provides ease of access and management for the complex tasks that cluster administration requires.

Kubernetes provides a web-based dashboard, which is not deployed by default within a cluster. The Kubernetes dashboard provides minimal security permissions, and accepts only token-based authentication. This dashboard also requires a proxy setup that limits access to the web console from only the system terminal that creates the proxy. By contrast with the stated limitations of the Kubernetes web console, OpenShift includes a fuller-featured web console.

The OpenShift web console is not related to the Kubernetes dashboard, but is a separate tool for managing OpenShift clusters. Additionally, operators can extend the web console features and functions to include more menus, views, and forms to aid in cluster administration.

## Accessing the OpenShift Web Console

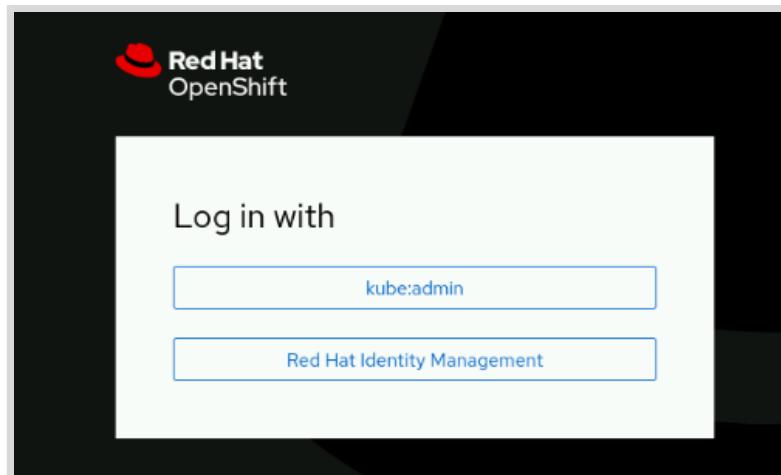
You access the web console by any modern web browser. The web console URL is generally configurable, and you can discover the address for your cluster web console by using the `oc` command-line interface (CLI). From a terminal, you must first authenticate to the cluster via the CLI by using the `oc login -u <USERNAME> -p <PASSWORD> <API_ENDPOINT>:<PORT>` command:

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

Then, you execute the `oc whoami --show-console` command to retrieve the web console URL:

```
[user@host ~]$ oc whoami --show-console  
https://console-openshift-console.apps.ocp4.example.com
```

Lastly, use a web browser to navigate to the URL, which displays the authentication page:



**Figure 1.6: The OpenShift authentication page**

Using the credentials for your cluster access brings you to the home page for the web console.

A screenshot of the Red Hat OpenShift web console home page. The left sidebar shows a navigation menu with items like Home, Operators, Workloads, Networking, Storage, Builds, Observe, and Compute. The main content area is titled "Overview" and includes sections for "Getting started resources", "Cluster", and "Explore new admin features". It also features links for "Set up your cluster", "Build with guided documentation", "Monitor your sample application", and "See what's new in OpenShift 4.12".

**Figure 1.7: The OpenShift home page**

## Web Console Perspectives

The OpenShift web console provides the **Administrator** and **Developer** console perspectives. The sidebar menu layout and the features that it displays differ between using the two console perspectives. The first item on the console sidebar menu is the perspective switcher, to switch between the **Administrator** and the **Developer** perspectives.

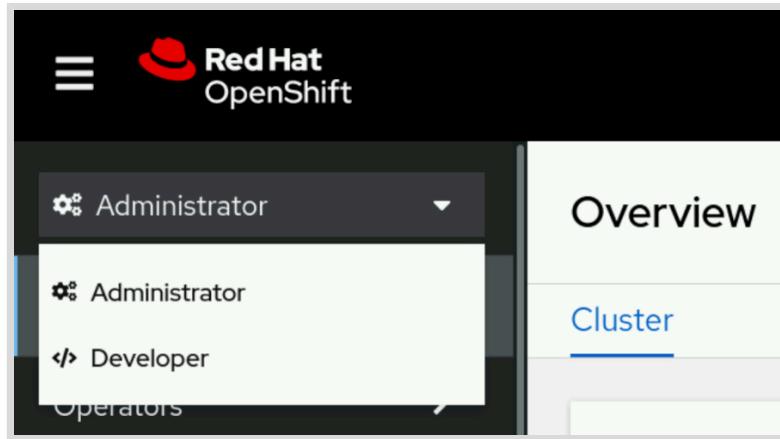


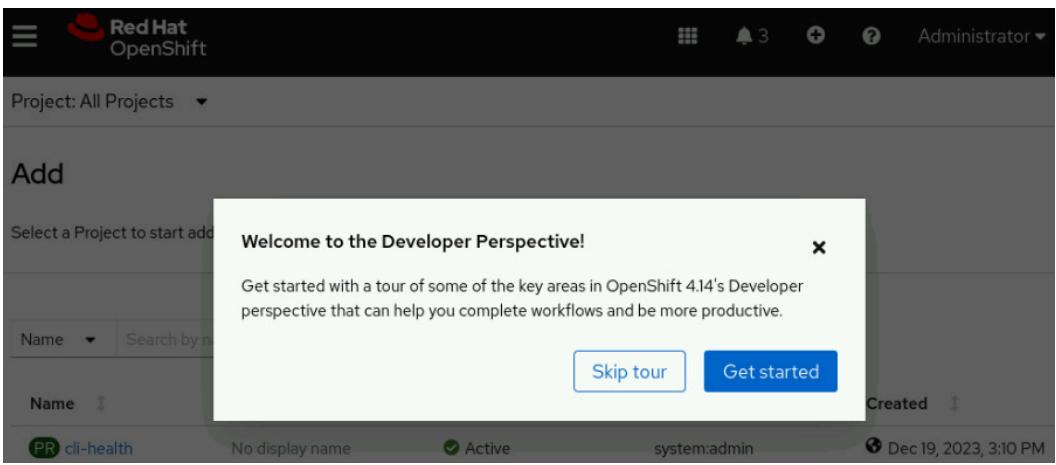
Figure 1.8: The OpenShift web console perspective switcher

Each perspective presents the user with different menu categories and pages that cater to the needs of the two separate personas. The **Administrator** perspective focuses on cluster configuration, deployments, and operations of the cluster and running workloads. The **Developer** perspective pages focus on creating and running applications.

## OpenShift Web Console Layout

From the web console home page, the primary navigation method is through the sidebar. The sidebar organizes cluster functions and administration into several major categories. By selecting a category from the sidebar, the category expands to reveal the various areas that each provide specific cluster information, configuration, or functionality.

 **Note**  
An initial login to the web console presents the option for a short informational tour. Click **Skip Tour** if you prefer to dismiss the tour option at this time.



The screenshot shows the Red Hat OpenShift web console interface. At the top, there's a dark header with the Red Hat OpenShift logo and a 'Administrator' dropdown. Below the header is a sidebar with a 'Project: All Projects' dropdown. The main content area has a title 'Add' and a sub-section 'Welcome to the Developer Perspective!'. This section contains a message: 'Get started with a tour of some of the key areas in OpenShift 4.14's Developer perspective that can help you complete workflows and be more productive.' It includes two buttons: 'Skip tour' and 'Get started'. At the bottom of the screen, there's a footer with project details: 'PR cli-health', 'No display name', 'Active', 'system:admin', and a timestamp 'Dec 19, 2023, 3:10 PM'.

By default, the console displays the **Home > Overview** page, which provides a quick glimpse of initial cluster configurations, documentation, and general cluster status. Navigate to **Home > Projects** to list all projects in the cluster that are available to the credentials in use.

You might initially peruse the **Operators > OperatorHub** page, which provides access to the collection of operators that are available for your cluster.

The screenshot shows the OpenShift OperatorHub interface. At the top, there is a search bar labeled "Filter by keyword...". Below the search bar, there are two operator cards:

- LVM Storage**: Provided by Red Hat. It has a red hat icon and the text "do180 Operator Catalog Cs".
- MetalLB Operator**: Provided by Community. It has a blue square icon with a white letter "A" and the text "do180 Operator Catalog Cs".

On the left side of the interface, there are filters for "Source" (do180 Operator Catalog Cs (2)) and "Provider" (Red Hat (1)). There are also buttons for "All Items" and a "2 items" indicator.

**Figure 1.10: The OpenShift OperatorHub**

By adding operators to the cluster, you can extend the features and functions that your OpenShift cluster provides. Use the search filter to find the available operators to enhance the cluster and to supply the OpenShift aspects that you require.

By clicking the link on the Operator Hub page, you can peruse the Developer Catalog.

This screenshot is identical to Figure 1.10, but the "Developer Catalog" link in the top navigation bar is highlighted with a red box.

Select any project, or use the search filter to find a specific project, to visit the Developer Catalog for that project, where shared applications, services, event sources, or source-to-image builders are available.

The screenshot shows the Red Hat OpenShift Developer Catalog interface. At the top, there's a header with the title 'Developer Catalog' and a sub-instruction: 'Add shared applications, services, event sources, or source-to-image builders to your Project from the developer catalog. Cluster administrators can customize the content made available in the catalog.' Below the header, there are several filters on the left: 'All items' (selected), 'CI/CD', 'Databases', 'Languages', 'Middleware', and 'Other'. There's also a 'Type' filter with options like 'Builder Images (12)', 'Devfiles (6)', 'Helm Charts (51)', and 'Operator Backed (2)'. In the center, there's a search bar with the placeholder 'Filter by keyword...' and a sorting dropdown set to 'A-Z'. On the right, it says '116 items'. The main area displays two card-like results: one for '.NET Builder Images' (provided by Red Hat) and another for '.NET Helm Charts'. Both cards show a brief description and a link to more information.

**Figure 1.12: The Developer Catalog**

After finding the preferred additions for a project, a cluster administrator can further customize the content that the catalog provides. By adding the necessary features to a project from this approach, developers can customize features to provide an ideal application deployment.

## Red Hat OpenShift Key Concepts

When you navigate the OpenShift web console, it is useful to know some introductory OpenShift, Kubernetes, and container terminology. The following list includes some basic concepts that can help you to navigate the OpenShift web console.

- **Pods:** The smallest unit of a Kubernetes-managed containerized application. A pod consists of one or more containers.
- **Deployments:** The operational unit that provides granular management of a running application.
- **Projects:** A Kubernetes namespace with additional annotations that provide multitenancy scoping for applications.
- **Routes:** Networking configuration to expose your applications and services to resources outside the cluster.
- **Operators:** Packaged Kubernetes applications that extend cluster functions.

These concepts are covered in more detail throughout the course. You can find these concepts throughout the web console as you explore the features of an OpenShift cluster from the graphical environment.



### References

For more information about the OpenShift web console, refer to Red Hat OpenShift Container Platform *Web Console* documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/web\\_console/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/web_console/index)

## ► Guided Exercise

# Navigate the OpenShift Web Console

Access an OpenShift cluster by using its web console and review pages to identify key OpenShift cluster services.

### Outcomes

- Explore the features and components of Red Hat OpenShift by using the web console.
- Create a sample application by using the Developer perspective in the web console.
- Switch to the Administrator perspective and examine the resources that are created for the sample application.
- Use the web console to describe the cluster nodes, networking, storage, and authentication.
- View the default cluster operators, pods, deployments, and services.

### 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 validated for the exercise.

```
[student@workstation ~]$ lab start intro-navigate
```

### Instructions

- 1. As the developer user, locate and then navigate to the Red Hat OpenShift web console.

- 1.1. Use the terminal to log in to the OpenShift cluster as the `developer` user with the `developer` password.

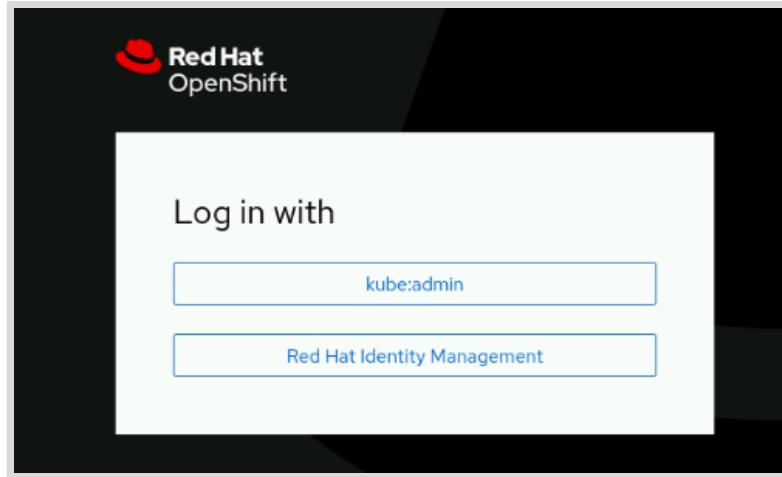
```
[student@workstation ~]$ oc login -u developer -p developer \
https://api.ocp4.example.com:6443

...output omitted...
```

- 1.2. Identify the URL for the OpenShift 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`. Either type the URL in a web browser, or right-click and select Open Link from the terminal.



- 2. Log in to the OpenShift web console as the developer user.
- 2.1. Click Red Hat Identity Management and log in as the developer user with the developer password.

A screenshot of the Red Hat OpenShift login form. The title is "Log in to your account". It has two input fields: "Username" containing "developer" and "Password" containing a masked password. Below the fields is a blue "Log in" button. At the bottom of the form, the text "Welcome to Red Hat OpenShift" is visible.



**Note**

Click Skip Tour to dismiss the option to view a short tour on the first visit.

The screenshot shows the Red Hat OpenShift web console. At the top, there's a navigation bar with the Red Hat logo and 'OpenShift'. Below it, a dropdown menu says 'Project: All Projects'. The main area has a title 'Add' and a sub-section 'Select a Project to start adding'. A prominent modal window titled 'Welcome to the Developer Perspective!' is displayed. It contains text: 'Get started with a tour of some of the key areas in OpenShift 4.x's Developer perspective that can help you complete workflows and be more productive.' Below the text are two buttons: 'Skip tour' (in a blue box) and 'Get started' (in a blue box). In the background, there's a table-like list of projects. One project is highlighted: 'cli-health' (PR), 'No display name', 'Active', 'system:admin', and 'Created Dec 19, 2023, 3:10 PM'. There are also sorting columns for 'Name' and 'Created'.

- 3. Use the **Developer** perspective of the web console to create your first project.

- 3.1. From the **Getting Started** section in the **+Add** page, click **Create a new project** to open the **Create Project** wizard.

The screenshot shows the Red Hat OpenShift web console. On the left, a sidebar menu is open under the 'Developer' tab. The 'Getting Started' section is highlighted in the sidebar. The main content area has a title 'Getting Started' with sub-sections: 'OpenShift helps you quickly develop, host, and scale applications. To get started, create a project for your application.', 'To learn more, visit the OpenShift documentation.', 'Download the command-line tools', and a 'Create a new project' button. Below this, there's another section titled 'Add' with the text 'Select a Project to start adding to it or [create a Project](#)'.

- 3.2. Create a project named **intro-navigate** by using the wizard. Use **intro-navigate** for the display name, and add a brief description of the project.

**Create Project**

An OpenShift project is an alternative representation of a Kubernetes namespace.

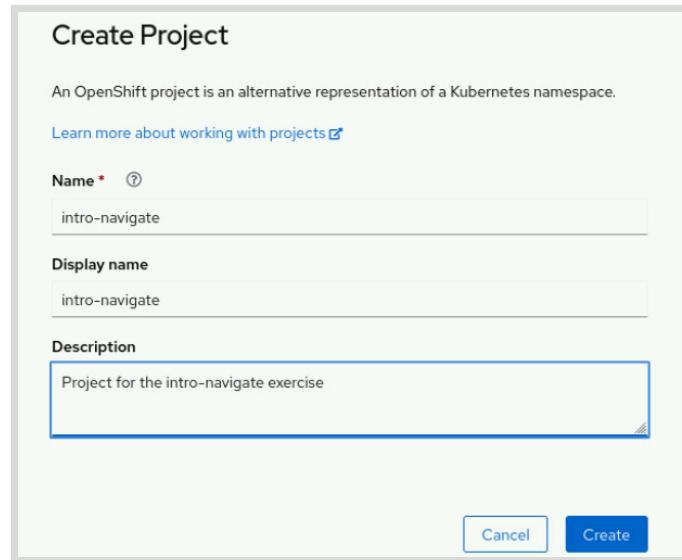
[Learn more about working with projects ↗](#)

**Name \*** ⓘ  
intro-navigate

**Display name**  
intro-navigate

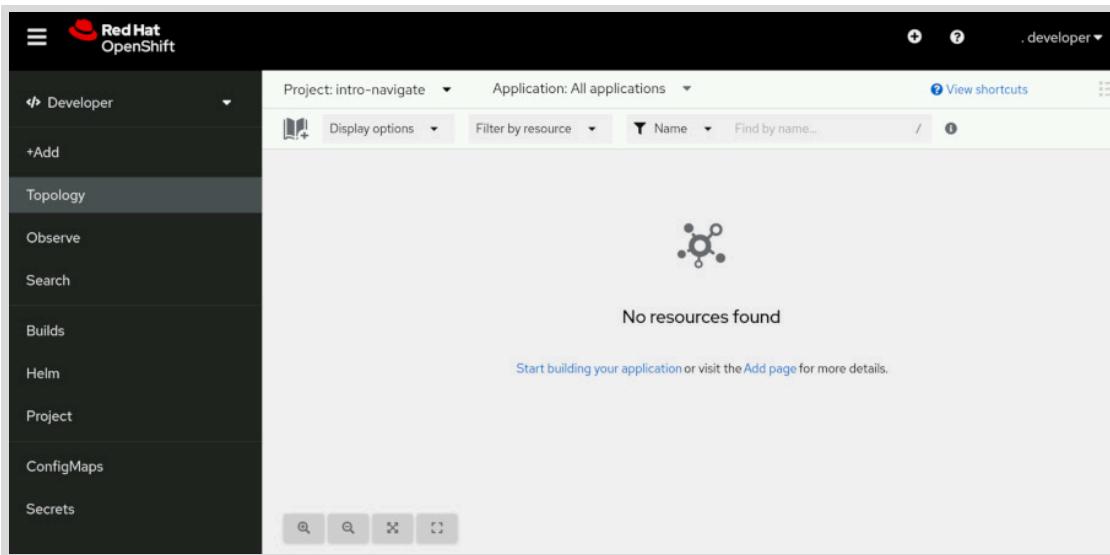
**Description**  
Project for the intro-navigate exercise

[Cancel](#) [Create](#)



3.3. Click **Create** to create the project.

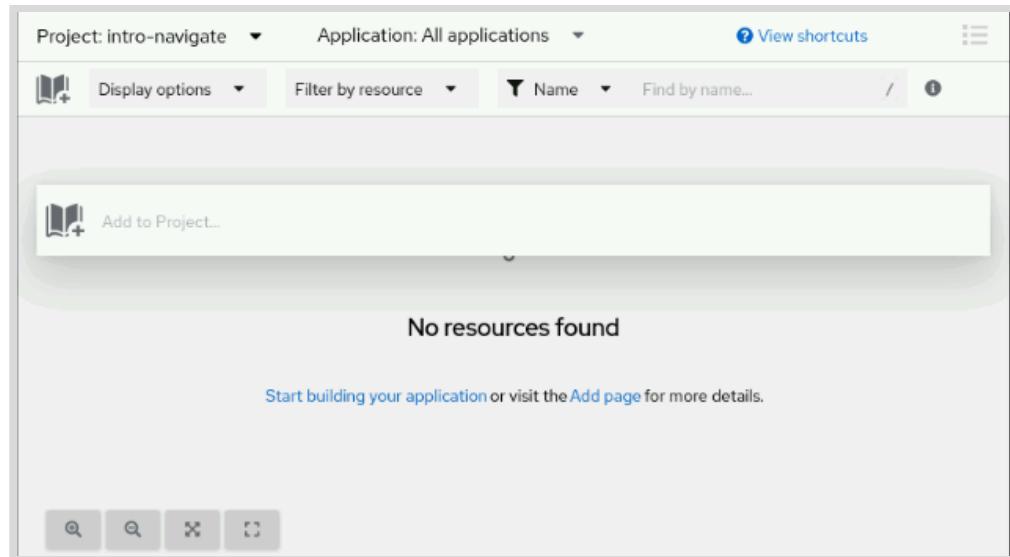
3.4. Click **Topology** to view the project.



The screenshot shows the Red Hat OpenShift developer interface. The left sidebar has a dark theme with white text and icons. The 'Topology' option is highlighted with a blue bar. The main content area has a light gray background. At the top, it says 'Project: intro-navigate' and 'Application: All applications'. Below that is a search bar with 'Display options', 'Filter by resource', 'Name', and 'Find by name...'. In the center, there's a small network graph icon with several nodes connected by lines. Below it, the text 'No resources found' is displayed. At the bottom, there's a message: 'Start building your application or visit the [Add page](#) for more details.' There are also some small navigation icons at the very bottom.

► 4. Deploy a sample application in the project.

4.1. Select the **Start building your application** link to browse the available sample applications.



- 4.2. Enter Apache into the search bar to see the available sample applications for deployment.

The screenshot shows the Apache search results. The left sidebar lists three items: 'Apache HTTP Server' (Templates, Red Hat, Inc.), 'Apache HTTP Server (httpd)' (Builder Images), and 'Httpd' (Builder Images). The right side panel is titled 'Httpd' and contains a 'Create' button. Below the button, there is a detailed description of the application: 'Build and serve static content via Apache HTTP Server (httpd) 2.4 on RHEL 7. For more information about using this builder image, including OpenShift considerations, see <https://github.com/scrlorg/httpd-container/blob/master/2.4/README.md>'.

- 4.3. Select the samples option from **Httpd** from the list of available applications, and then click **Create** at the bottom of the page.
- 4.4. Examine the default values for the sample application, and then select **Create** at the bottom of the page.

### Create Sample application

Name \*

A unique name given to the component that will be used to name associated resources.

Builder Image version \*

IST 2.4-el7

 Apache HTTP Server 2.4 (RHEL 7)

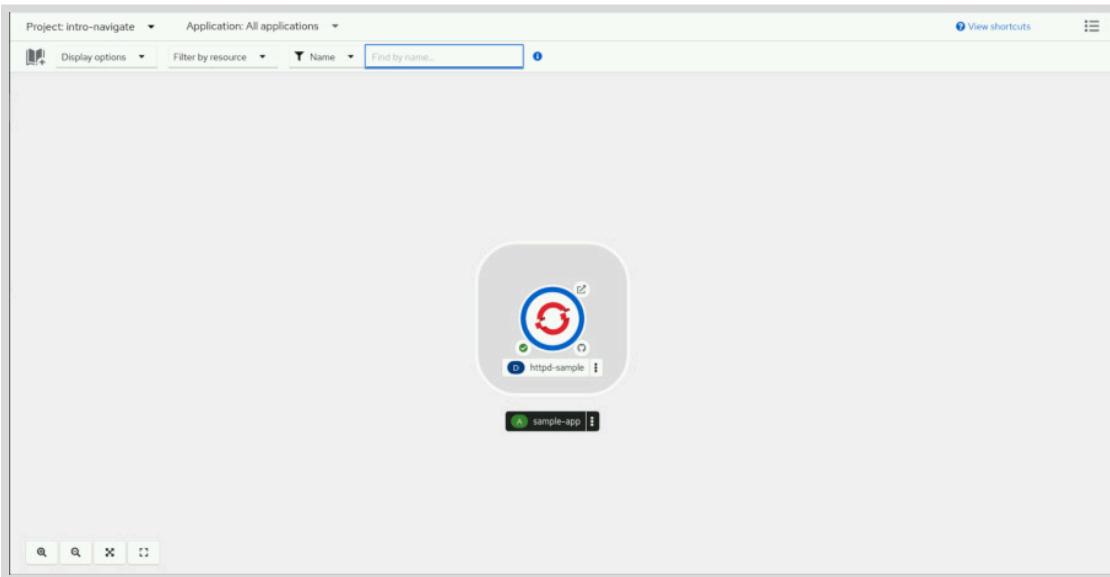
BUILDER HTTPD

Build and serve static content via Apache HTTP Server (httpd) 2.4 on RHEL 7. For more information about using this builder image, including OpenShift considerations, see <https://github.com/sclorg/httpd-container/blob/master/2.4/README.md>.

Sample repository: <https://github.com/sclorg/httpd-ex.git>

**Create** **Cancel**

The Topology page opens and displays the httpd-sample application deployment.



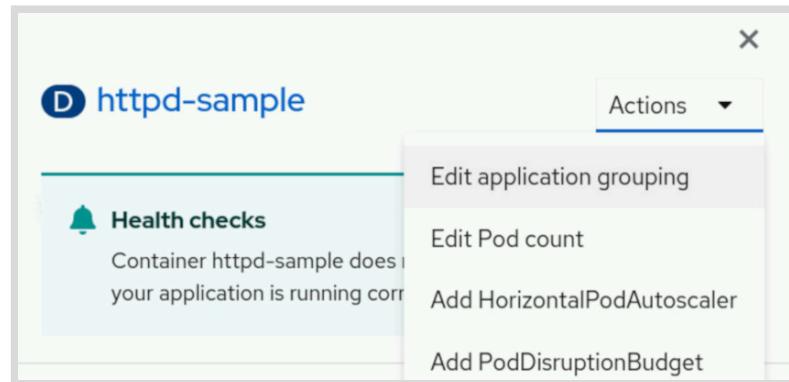
The screenshot shows the OpenShift Topology page. At the top, there are navigation dropdowns for 'Project: intro-navigate' and 'Application: All applications'. Below these are filter options: 'Display options', 'Filter by resource', 'Name', and a search bar 'Find by name...'. The main area displays a single application icon for 'httpd-sample'. This icon is a rounded square containing a blue circle with a white double-headed arrow, indicating it's a deployment. Below the icon, the text 'httpd-sample' is visible. At the bottom of the screen, there are standard browser navigation buttons: back, forward, search, and refresh.

► 5. View the deployment details for the httpd-sample application.

- 5.1. Select the httpd-sample deployment to view the details of the application.

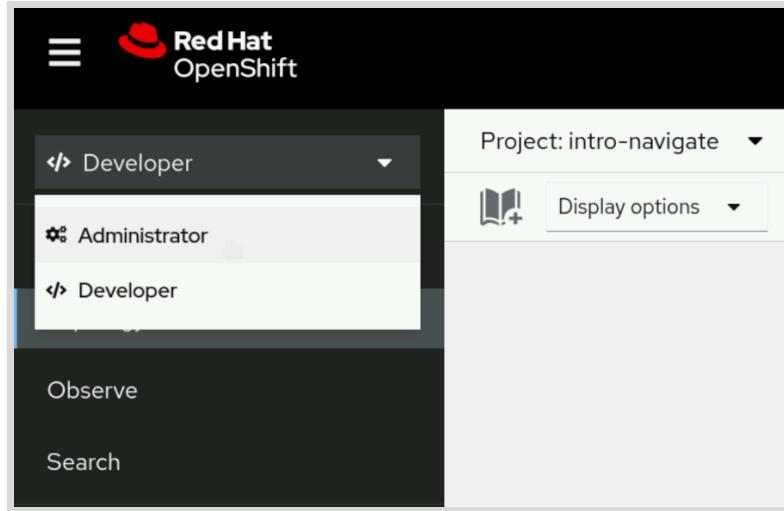
The screenshot shows the OpenShift web console interface. On the left, there's a sidebar with search and filter icons. The main area displays a deployment named "httpd-sample". The deployment icon is a blue circle with a white recycling symbol. Below it, a green button labeled "simple-app" with a gear icon is visible. To the right, the deployment details are shown under the "Details" tab. A "Health checks" section indicates that the container does not have health checks. The "Resources" tab is selected, showing a single pod named "httpd-sample-5958b65d47-sj6nb" which is "Running". There are "View logs" and "Edit" buttons next to the pod. An "Actions" dropdown menu is open, showing options like "Edit application grouping", "Edit Pod count", "Add HorizontalPodAutoscaler", and "Add PodDisruptionBudget".

- 5.2. Select the **Actions** list to view the available controls for the **httpd-sample** deployment.



- ▶ 6. Switch into the **Administrator** perspective and inspect the deployment.

- 6.1. From the OpenShift web console, locate the left panel. If you do not see the left panel, then click the main menu icon at the upper left of the web console. Click **Developer** and then click **Administrator** to change to the **Administrator** perspective. The web console changes to the new perspective and exposes additional information through the sidebar.



- 6.2. Select the **intro-navigate** project to open the Project Details page. This page includes a general overview of the project, such as the project status and resource usage details.

A screenshot of the OpenShift Project Details page for the "intro-navigate" project. The top navigation bar shows "Projects &gt; Project details" and the project name "intro-navigate" with an "Active" status. The "Actions" dropdown is visible. Below the header, there are tabs for "Overview" (selected), "Details", "YAML", "Workloads", and "RoleBindings". The "Overview" section contains a "Status" card showing "Active" with a green checkmark. It also displays a large question mark icon and the message "Alerts could not be loaded.". The "Utilization" section shows a timeline from 11:00 PM to 11:45 PM with a "1 hour" dropdown. The utilization chart is empty.

► 7. View the **httpd-sample** pods and deployment.

- 7.1. From the OpenShift web console menu, navigate to **Workloads > Pods** to view the **httpd-sample** pods.

Name	Status	Ready	Restarts	Owner	Memory	CPU	Created
httpd-sample-l-build	Completed	0/1	0	httpd-sample-l	-	-	Jan 3, 2024, 9:40 PM
httpd-sample-549d748586-wqjw9	Running	1/1	0	httpd-sample-549d748586-wqjw9	-	-	Jan 3, 2024, 9:41 PM

7.2. Navigate to Workloads > Deployments to view the list of deployments in the project.

Name	Status	Labels	Pod selector
httpd-sample	1 of 1 pods	app=httpd-sample	app=httpd-sample app.kubernetes.io/com...=httpd-sa... app.kubernetes.io/ins...=httpd-sa... app.kubernetes.io/na...=httpd-sam... app.kubernetes.io/part...=sample-a... app.openshift.io/runtim...=httpd app.openshift.io/runtim...=2.4-...

7.3. Click **httpd-sample** to view the deployment details.

Details	Metrics	YAML	ReplicaSets	Pods	Environment	Events
<b>Deployment details</b>						
1 Pod						
Name				Update strategy		
httpd-sample				RollingUpdate		
Namespace				Max unavailable		
NS intro-navigate				25% of 1 pod		

► 8. View the **httpd-sample** service and route.

- 8.1. Navigate to **Networking > Services** and click **httpd-sample** to view the details of the **httpd-sample** service.

The screenshot shows the OpenShift web console interface. On the left, a sidebar menu is open under the 'Networking' category, with 'Services' selected. The main area is titled 'Services' and shows a table with one row for the service 'httpd-sample'. The table includes columns for Name, Labels, Pod selector, and Location. The 'Labels' column contains several key-value pairs: app=httpd-sample, app.kubernetes.io/\*=httpd..., app.kubernetes.io/~httpd-s..., app.kubernetes.io/~httpd-sa..., app.kubernetes.io/\*=sample..., app.openshift.io/runti...=htt..., and app.openshift.io/runti...=2.4... . The 'Location' column shows two IP addresses: 172.30.16.158:8080 and 172.30.16.158:8443. A blue 'Create Service' button is located in the top right corner of the main content area.

- 8.2. Navigate to **Networking > Route** and click **httpd-sample** to view the details of the **httpd-sample** route.

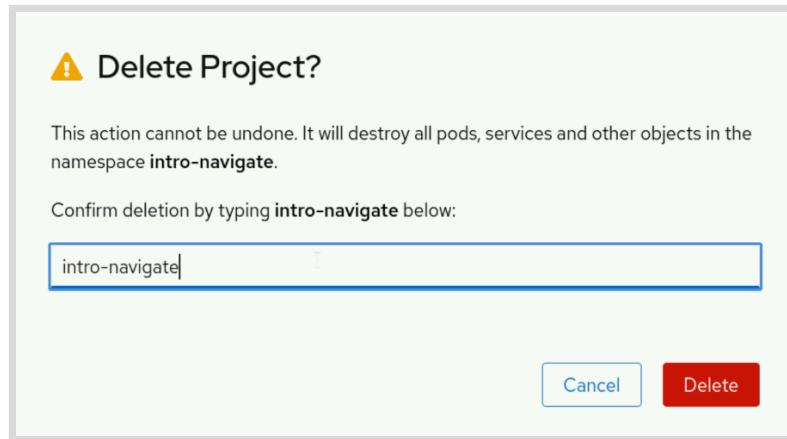
The screenshot shows the OpenShift web console interface. On the left, a sidebar menu is open under the 'Networking' category, with 'Routes' selected. The main area is titled 'Routes' and shows a table with one row for the route 'httpd-sample'. The table includes columns for Name, Status, Location, and Service. The 'Status' column shows 'Accepted'. The 'Location' column shows the URL <http://httpd-sample-intro-navigate.apps.ocp4.example.com>. The 'Service' column shows the service 'httpd-sample'. A blue 'Create Route' button is located in the top right corner of the main content area.

► **9.** Delete the project and log out of the web console.

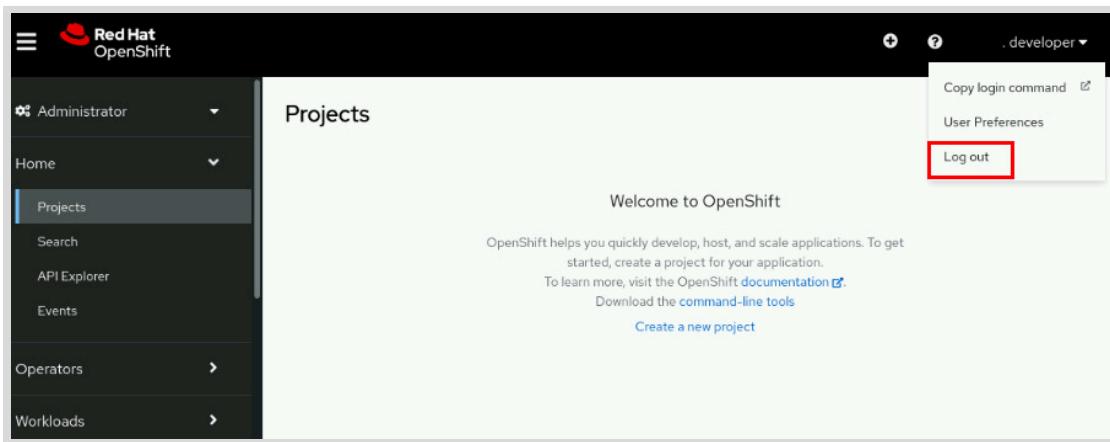
- 9.1. From the OpenShift web console menu, navigate to **Home > Projects**. Select **Delete Project** from the context menu for the **intro-navigate** project.

The screenshot shows the OpenShift web console interface. On the left, a sidebar menu is open under the 'Projects' category. The main area is titled 'Projects' and shows a table with one row for the project 'intro-navigate'. The table includes columns for Name, Display name, Status, Requester, and Created. The 'Status' column shows 'Active'. The 'Requester' column shows 'developer'. The 'Created' column shows 'Jan 3, 2024, 9:39 PM'. In the bottom right corner of the project row, there is a red box highlighting the 'Delete Project' button.

- 9.2. Enter the project name in the text field and then select **Delete**.



- 9.3. Log out of the web console. From the OpenShift web console right panel, click **developer** and then select **Log out** from the account menu.



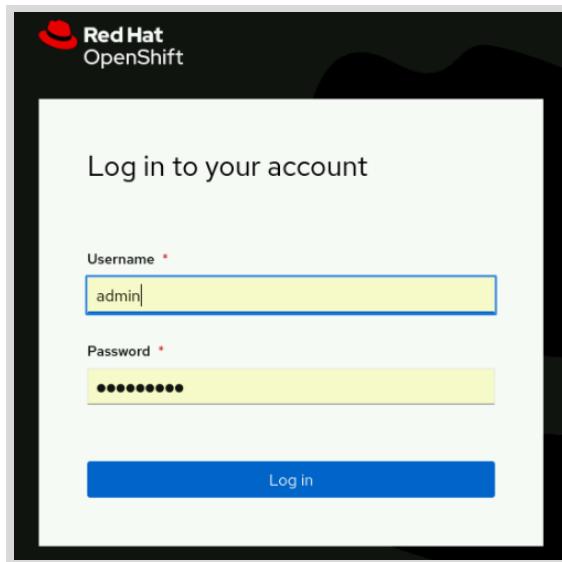
- 10. Log in to the OpenShift web console as the **admin** user to inspect additional cluster details.



#### Note

When you use a cluster administrator account, you can browse the cluster components, but do not alter or remove any components.

- 10.1. Log in to the web console. Select **Red Hat Identity Management** and then enter the **admin** username and the **redhatocp** password.



- 10.2. From the OpenShift web console menu, navigate to **Operators > Installed Operators**. Each operator provides a specific function for the cluster. Select an individual operator to display its details.

A screenshot of the OpenShift web console. The left sidebar shows a dark theme with the following navigation items: "Administrator", "Home", "Operators" (with a dropdown arrow), "OperatorHub", and "Installed Operators" (which is highlighted with a blue bar). The main content area is titled "Project: All Projects" and "Installed Operators". It contains a brief description: "Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and ClusterServiceVersion using the [Operator SDK](#)". Below this is a search bar with "Name" and "Search by name..." fields, and a small icon.

- 10.3. Navigate to **Workloads > Pods** to view the list of all pods in the cluster. The search bar at the top can narrow down the list of pods. Select an individual pod to display its details.

A screenshot of the OpenShift web console. The left sidebar shows a dark theme with the following navigation items: "Administrator", "Home", "Operators", "Workloads" (with a dropdown arrow), "Pods" (which is highlighted with a blue bar), "Deployments", "DeploymentConfigs", "StatefulSets", and "Secrets". The main content area is titled "Project: All Projects" and "Pods". It includes a "Filter" dropdown, a search bar with "Name" and "Search by name..." fields, and a table header with columns: Name, Names..., Status, Ready, Restarts, Owner, Memory, and CPU. Below the header is a table listing several pods. One pod is expanded to show its details: Name: 8f0613a2, Namespace: openshift-marketplace, Status: Completed, Ready: 0/1, Restarts: 0, Owner: 8f0613a2d1, Memory: -, CPU: -. Other pod names listed include d1af29fb, aae524, 5b9316cc, 5db08e8, a5c06519, 5008eb3, 16e63pr2, and vp.

- 10.4. Navigate to **Workloads > Deployments** to view the list of all deployments in the cluster. Select an individual deployment to display its details.

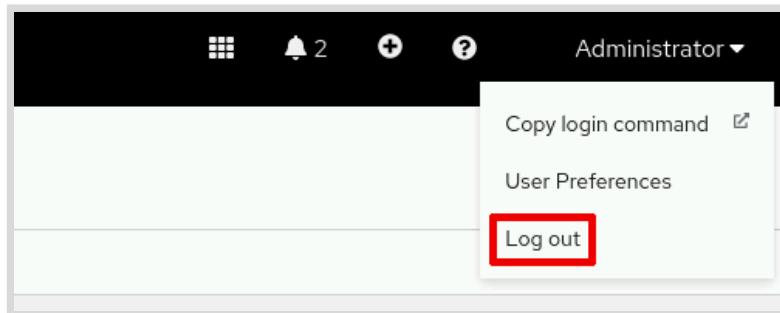
Name	Namespace	Status	Labels	Pod selector
apiserver	openshift-oauth-apiserver	1 of 1 pods	apiserver=true a...*openshift-oauth-ap... revision=1	Q apiserver=true, app=openshift-oauth-apiserver
apiserver	openshift-apiserver	1 of 1 pods	apiserver=true app=openshift-apiserver revision=1	Q apiserver=true, app=openshift-apiserver-a
authentication-operator	openshift-authentication-operator	1 of 1 pods	a...*authentication-ope... Q app=authentication-operator	

- 10.5. Navigate to **Networking > Services** to view the list of all services in the cluster. Select an individual service to display its details.

Name	Namespace	Labels	Pod selector	Location
alertmanager-main	openshift-monitoring	app.kubernetes.io/component=alertmanager app.kubernetes.io/instance=main app.kubernetes.io/name=alertmanager app.kubernetes.io/part-of=openshift-monitoring app.kubernetes.io/cluster=cluster app.kubernetes.io/alertmanager app.kubernetes.io/monitoring=true	Q app.kubernetes.io/component=alert-... 172.30.124.93:9094 172.30.124.93:9092 172.30.124.93:9097	

► 11. Log out of the web console.

- 11.1. Log out of the web console. From the OpenShift web console right panel, click **Administrator** and then select **Log out** from the account menu.



## 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 intro-navigate
```

# Monitor an OpenShift Cluster

## Objectives

- Navigate the Events, Compute, and Observe panels of the OpenShift web console to assess the overall state of a cluster.

## Overview of Nodes, Machines, and Machine Configurations

In Kubernetes, a *node* is any single system in the cluster where pods can run. These systems are any of the bare metal, virtual, or cloud computers that are members of the cluster. Nodes run the necessary services to communicate within the cluster, and receive control plane operational requests. When you deploy a pod, an available node is tasked with satisfying the request.

Whereas the *node* and *machine* terms are often interchangeable, Red Hat OpenShift Container Platform (RHOC) uses the *machine* term more specifically. In OpenShift, a *machine* is the resource that describes a cluster node. Using a machine resource is particularly valuable when using public cloud providers to provision infrastructure.

A `MachineConfig` resource defines the initial state and any changes to files, services, operating system updates, and critical OpenShift service versions for the `kubelet` and `cri-o` services. OpenShift relies on the Machine Config Operator (MCO) to maintain the operating systems and configuration of the cluster machines. The MCO is a cluster-level operator that ensures the correct configuration of each machine. This operator also performs routine administrative tasks, such as system updates. This operator uses the machine definitions in a `MachineConfig` resource to continually validate and remediate the state of cluster machines to the intended state. After a `MachineConfig` change, the MCO orchestrates the execution of the changes for all affected nodes.



### Note

The orchestration of `MachineConfig` changes through the MCO is prioritized alphabetically by zone, by using the `topology.kubernetes.io/zone` node label.

## Identifying Errors from Nodes

Administrators routinely view the logs and connect to the nodes in the cluster by using a terminal. This technique is necessary to manage a cluster and to remediate issues that arise. From the web console, navigate to **Compute > Nodes** to view the list of all nodes in the cluster.

Name	Status	Roles	Pods	Memory	CPU	Filesys...	Created
master01	Ready	control-plane, master, worker	87	12.2 GiB / 19.58 GiB	0.782 cores / 8 cores	21.48 GiB / 119.8 GiB	Feb 9, 2023, 4:22 PM

**Figure 1.41: Node list in the web console**

Click a node's name to navigate to the overview page for the node. On the node overview page, you can view the node logs or connect to the node by using the terminal.

```

1 Mar 27 19:03:13.348810 master01 ovs-vswitchd[1316]: ovs|01273|connmgr|INFO|br-ex<->unix#6115: 2 flow_mods
2 Mar 27 19:03:13.676462 master01 systemd[1]: run-runc-8f71953bec59d0f0f54ace37726f66226a7100d6a0a4377c3fd4
3 Mar 27 19:03:28.330295 master01 ovs-vswitchd[1316]: ovs|01274|connmgr|INFO|br-ex<->unix#6119: 2 flow_mods
4 Mar 27 19:03:32.462815 master01 kubenswrapper[3215]: I0327 19:03:32.462733 3215 kubelet_getters.go:182] "
5 Mar 27 19:03:32.462815 master01 kubenswrapper[3215]: I0327 19:03:32.462809 3215 kubelet_getters.go:182] "
6 Mar 27 19:03:32.462815 master01 kubenswrapper[3215]: I0327 19:03:32.462835 3215 kubelet_getters.go:182] "
7 Mar 27 19:03:32.462815 master01 kubenswrapper[3215]: I0327 19:03:32.462844 3215 kubelet_getters.go:182] "
8 Mar 27 19:03:43.333093 master01 ovs-vswitchd[1316]: ovs|01275|connmgr|INFO|br-ex<->unix#6127: 2 flow_mods
9 Mar 27 19:03:53.676128 master01 systemd[1]: run-runc-8f71953bec59d0f0f54ace37726f66226a7100d6a0a4377c3fd4
10 Mar 27 19:03:58.328757 master01 ovs-vswitchd[1316]: ovs|01276|connmgr|INFO|br-ex<->unix#6131: 2 flow_mods

```

**Figure 1.42: Logs page in the web console**

From the previous page in the web console, view the node logs and investigate the system information to aid troubleshooting and remediation for node issues.

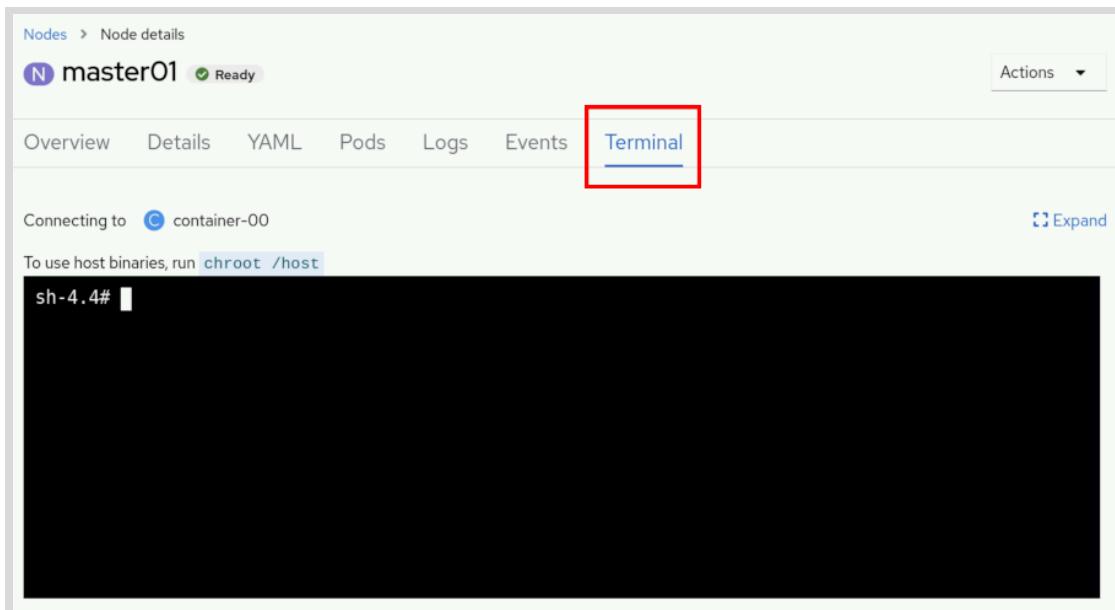


Figure 1.43: Terminal shell in the web console

The preceding page shows the web console terminal that is connected to the cluster node. From this tab, you can access the debug pod and use the commands from the host binaries to view the status of the node's services. An OpenShift node debug pod is an interface to a container that runs on the node.

Although making changes directly on the cluster node from the terminal is not recommended, it is common practice to connect to the cluster node for diagnostic investigation and remediation. From this terminal, you can use the same binaries that are available within the cluster node itself.

Additionally, the tabs on the node overview page show metrics, events, and the node's YAML definition file.

## Accessing Pod Logs

Administrators often peruse pod logs to assess the health of a deployed pod or to troubleshoot pod deployment issues. Navigate to the **Workloads > Pods** page to view the list of all pods in the cluster.

The screenshot shows the 'Pods' page under the 'Workloads' section. The sidebar shows 'Administrator' selected. The main area displays a table of pods with columns: Name, Names..., Status, Ready, Restarts, Owner, Memory, CPU. One pod is shown in detail: '8f0613a2d1' (Status: Completed, Ready: 0/1, Restarts: 0, Owner: af29fbbaee, Memory: -, CPU: -). A 'Create Pod' button is visible in the top right.

Figure 1.44: Pods page in the web console

You can filter and order pods by project and by other fields. To view the pod details page, click a pod name in the list.

The screenshot shows the 'Pods' section of the Red Hat OpenShift web interface. A specific pod named '8f0613a2d1af29fbaaee5245b9316cc5db08e8a5c065195008eb316e63pr2vp' is selected. The pod status is 'Completed'. Below the pod name, there are tabs for 'Details', 'Metrics', 'YAML', 'Environment', 'Logs', 'Events', and 'Terminal'. The 'Details' tab is currently active. Under the 'Pod details' heading, the following information is displayed:

Name	Status
8f0613a2d1af29fbaaee5245b9316cc5db08e8a5c065195008eb316e63pr2vp	Completed
Namespace	Restart policy
openshift-marketplace	Never restart
Labels	Active deadline seconds
	Not configured

**Figure 1.45: Pod details page**

The pod details page contains links to pod metrics, environment variables, logs, events, a terminal, and the pod's YAML definition. The pod logs are available on the **Pods > Logs** page and provide information about the pod status. The **Pods > Terminal** page opens a shell connection to the pod for inspection and issue remediation. Although it is not recommended to alter a running pod, the terminal is useful for diagnosing and remediating pod issues. To fix a pod, update the pod configuration to reflect the necessary changes, and redeploy the pod.

## Red Hat OpenShift Container Platform Metrics and Alerts

In an RHOCP cluster, HTTP service endpoints provide data metrics that are collected to provide information for monitoring cluster and application performance. These metrics are authored at the application level for each service by using the client libraries that are provided by Prometheus, an open source monitoring and alerting toolkit. Metrics data is available from the service `/metrics` endpoint. You can use the data for creating monitors to alert based on degradation of the service. Monitors are processes that continuously assess the value for a specific metric and provide alerts that are based on a predefined condition, to signal a degradation in the service or a performance issue. Authoring a `ServiceMonitor` resource defines how a specific service uses the metrics to define a monitor and the alerting values. The same approach is available for monitoring pods by defining a `PodMonitor` resource that uses the metrics that are gathered from the pod.

Depending on the monitor definitions, alerting is then available based on the metric that is polled and the defined success criteria. The monitor continuously compares the gathered metric, and creates an alert when the success criteria are no longer met. As an example, a web service monitor polls on the listening port, port 80, and alerts only if the response from that port becomes invalid.

From the web console, navigate to **Observe > Metrics** to visualize gathered metrics by using a Grafana-based data query utility. On this page, users can submit queries to build data graphs and dashboards, which administrators can view to gather valuable statistics for the cluster and applications.

For configured monitors, visit **Observe > Alerting** to view firing alerts, and filter on the alert severity to view those alerts that need remediation. Alerting data is a key component to help administrators to deliver cluster and application accessibility and functions.

## Kubernetes Events

Administrators are typically familiar with the contents of log files for services, whereas logs tend to be highly detailed and granular. *Events* provide a high-level abstraction to log files and to provide information about more significant changes. Events are useful in understanding the performance and behavior of the cluster, nodes, projects, or pods, at a glance. Events provide details to understand general performance and to highlight meaningful issues. Logs provide a deeper level of detail for remediating specific issues.

The Home > Events page shows the events for all projects or for a specific project. You can further filter and search events.

The screenshot shows the RHOCP Events console. The left sidebar has 'Events' selected. The main area is titled 'Events' with filters for 'Resources' (1), 'All types', and a search bar. It shows a single event: 'collect-profiles-28387320' (Resource: job-controller) with status 'Job completed'. The timestamp is 'Dec 22, 2023, 5:01'.

Figure 1.46: RHOCP Events console

## Red Hat OpenShift Container Platform API Explorer

Starting from version 4, RHOCP includes the API Explorer feature, for users to view the catalog of Kubernetes resource types that are available within the cluster. By navigating to Home > API Explorer, you can view and explore the details for resources. Such details include the description, schema, and other metadata for the resource. This feature is helpful for all users, and especially for new administrators.

The screenshot shows the RHOCP API Explorer. The left sidebar has 'API Explorer' selected. The main area is titled 'API Explorer' with filters for 'All groups', 'All versions', 'All scopes', and a 'Filter by kind...' dropdown. A table lists a single resource: 'Binding' (Kind), 'v1' (Group), 'v1' (Version), 'true' (Namespaced). The description is: 'Binding ties one object to another; for example, a pod is bound to a node by a scheduler. Deprecated in 1.7, please use the bindings subresource of pods instead.'

Figure 1.47: The RHOCP API Explorer



## References

For more information about Red Hat OpenShift Container Platform machines, refer to the *Overview of Machine Management* chapter in the Red Hat OpenShift Container Platform *Machine Management* documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/machine\\_management/index#overview-of-machine-management](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/machine_management/index#overview-of-machine-management)

### The API Explorer

<https://cloud.redhat.com/blog/openshift-4-2-the-api-explorer>

### Well-known Labels, Annotations and Taints, topology.kubernetes.io/zone

<https://kubernetes.io/docs/reference/labels-annotations-taints/#topologykubernetesiozone>

## ► Guided Exercise

# Monitor an OpenShift Cluster

Assess the overall status of an OpenShift cluster by using the web console, and identify projects and pods of core architectural components of Kubernetes and OpenShift.

### Outcomes

- Explore and show the monitoring features and components.
- Explore the Overview page to inspect the cluster status.
- Use a terminal connection to the `master01` node to view the `crio` and `kubelet` services.
- Explore the Monitoring page, alert rule configurations, and the `etcd` service dashboard.
- Explore the events page, and filter events by resource name, type, and message.

### 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 prepared for the exercise.

```
[student@workstation ~]$ lab start intro-monitor
```

### Instructions

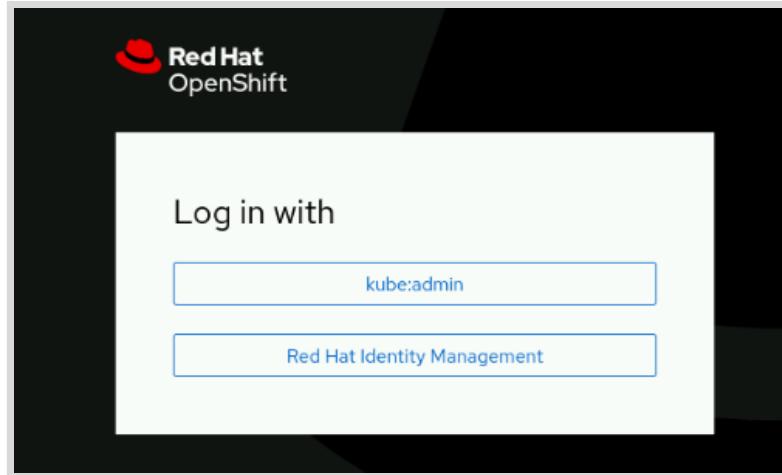
- 1. As the `developer` user, locate and then navigate to the Red Hat OpenShift web console.
- 1.1. Use the terminal to log in to the OpenShift cluster as the `developer` user with the `developer` password.

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

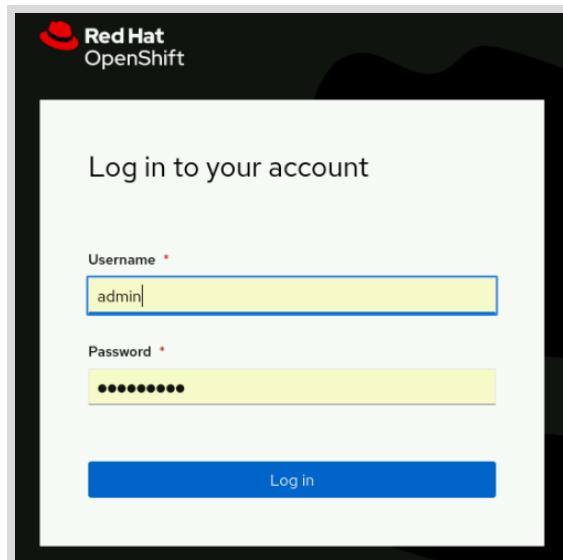
- 1.2. Identify the URL for the OpenShift web console.

```
[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`. Either type the URL in a web browser, or right-click and select `Open Link` from the terminal.



- 2. Log in to the OpenShift web console as the **admin** user.
- 2.1. Click **Red Hat Identity Management** and log in as the **admin** user with the **redhatocp** password.



- 3. View the cluster health and overall status.
- 3.1. Review the **Cluster Overview** page.  
If you do not see this page after a successful login, then locate the left panel from the OpenShift web console. If you do not see the left panel, then click the main menu icon at the upper left of the web console. Navigate to **Home > Overview** to view general cluster information.

The **Overview** section contains links to helpful documentation and an initial cluster configuration walkthrough.

- 3.2. Scroll down to view the **Status** section, which provides a summary of cluster performance and health.

Many of the headings are links to sections with more detailed cluster information.

- 3.3. Continue scrolling to view the **Cluster utilization** section, which contains metrics and graphs that show resource consumption.

- 3.4. Continue scrolling to view the **Details** section, including information such as the cluster API address, cluster ID, and Red Hat OpenShift version.

**Administrator**

**Home**

- Overview
- Projects
- Search
- API Explorer
- Events
- Operators

**Details**

Cluster API address: <https://api.ocp4.example.com:6443>

Cluster ID: 3b68e8bb8-6464-4218-9e4c-56c91577b204

OpenShift Cluster Manager

Infrastructure provider: None

OpenShift version: 4.14.0

Update channel: Not available

Control plane high availability: No (single control plane node)

[View settings](#)

- 3.5. Scroll to the **Cluster Inventory** section, which contains links to the **Nodes**, **Pods**, **StorageClasses**, and **PersistentVolumeClaim** pages.

**Administrator**

**Home**

- Overview
- Projects

**Cluster inventory**

1 Node

115 Pods

2 StorageClasses

1 PersistentVolumeClaim

- 3.6. The last part of the page contains the **Activity** section, which lists ongoing activities and recent events for the cluster.

**Administrator**

**Home**

- Overview
- Projects
- Search
- API Explorer
- Events

**Activity**

Ongoing

There are no ongoing activities.

Recent events

**Pause**

- ⚠️ ⟳ readyz=true
- ⟳ Received signal to terminate, becoming unready, but keeping serving
- 4:09 PM 🕒 Updated ConfigMap/kube-rbac-proxy -n openshift-machine-config-operator: cause by changes in data.config-file.yaml
- 4:06 PM 🕒 Updated ConfigMap/kube-rbac-proxy -n openshift-machine-config-operator: cause by changes in data.config-file.yaml

[View events](#)

- 4. Use the OpenShift web console to access the terminal of a cluster node. From the terminal, determine the status of the `kubelet` node agent service and the `CRI-O` container runtime interface service.

- 4.1. Navigate to **Compute > Nodes** to view the machine that provides the cluster resources.

**Storage**

**Builds**

**Observe**

**Compute**

**Nodes**

**Nodes**

**Filter**

Name:  Search by name...

Name	Status	Roles	Pods	Memory	CPU	Filesys...	Created
master0_1	Ready	control-plane, master, worker	89	11.22 GiB / 15.61 GiB	1.302 cores / 6 cores	24.18 GiB / 49.78 GiB	Dec 2023 PM

**Note**

The classroom cluster runs on a single node named `master01`, which serves as the control and data planes for the cluster, and is intended for training purposes. A production cluster uses multiple nodes to ensure stability and to provide a highly available architecture.

4.2. Click the `master01` link to view the details of the cluster node.

Nodes > Node details  
N master01 Ready Actions ▾

Overview Details YAML Pods Logs Events Terminal

Status

✓ Ready      Health checks  
Not configured

4.3. Click the **Terminal** tab to connect to a shell on the `master01` node.

Nodes > Node details  
N master01 Ready Actions ▾

Overview Details YAML Pods Logs Events **Terminal**

Connecting to C container-00 Expand

To use host binaries, run `chroot /host`

```
sh-4.4#
```

With the interactive shell on this page, you can run commands directly on the cluster node.

4.4. Run the `chroot /host` command to enable host binaries on the node.

Connecting to C container-00

To use host binaries, run `chroot /host`

```
sh-4.4# chroot /host
sh-5.1#
```

- 4.5. View the status of the `kubelet` node agent service by running the `systemctl status kubelet` command.

```
To use host binaries, run chroot /host
● kubelet.service - Kubernetes Kubelet
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; preset: disabled)
   Drop-In: /etc/systemd/system/kubelet.service.d
     └─01-kubens.conf, 10-mco-default-madv.conf, 20-logging.conf, 20-nodenet.conf
     Active: active (running) since Fri 2024-01-05 19:49:51 UTC; 1h 4min ago
       Main PID: 2842 (kubelet)
         Tasks: 23 (limit: 101688)
        Memory: 293.4M
          CPU: 8min 13.900s
        CGroup: /system.slice/kubelet.service
                  └─2842 /usr/bin/kubelet --config=/etc/kubernetes/kubelet.conf --bootstrap-kubeconfig=/etc/kubernetes/kub...
Jan 05 20:53:31 master01 kubenswrapper[2842]: I0105 20:53:31.250394  2842 operation_generator.go:718] "MountVolume"
[lines 1-13]
```

Press q to exit the command and to return to the terminal prompt.

- 4.6. View the status of the CRI-O container runtime interface service by running the `systemctl status cri-o` command.

```
To use host binaries, run chroot /host
● cri-o.service - Container Runtime Interface for OCI (CRI-O)
   Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; preset: disabled)
   Drop-In: /etc/systemd/system/crio.service.d
     └─01-kubens.conf, 10-mco-default-madv.conf, 10-mco-profile-unix-socket.conf, 20-nodenet.conf
     Active: active (running) since Fri 2024-01-05 19:49:39 UTC; 1h 13min ago
       Docs: https://github.com/cri-o/cri-o
       Main PID: 2649 (crio)
         Tasks: 44
        Memory: 4.1G
          CPU: 8min 52.066s
        CGroup: /system.slice/crio.service
                  ├─2649 /usr/bin/crio
                  └─88431 /usr/bin/runc --root /run/runc --systemd-cgroup exec --process /tmp/exec-process-1747750210 9084...
[lines 1-13]
```

Press q to exit the command and to return to the terminal prompt.

## ► 5. Inspect the cluster monitoring and alert rule configurations.

- 5.1. From the OpenShift web console menu, navigate to **Observe > Alerting** to view cluster alert information.

Alert Rule	Status	Severity	Time Since Last Firing	Platform
AL_HighOverallControlPlaneMemory	Firing	Warning	Since Dec 22, 2023, 12:08 AM	Platform
AL_InsightsDisabled	Firing	Info	Since Dec 22, 2023, 12:06 AM	Platform
AL_MultipleDefaultStorageClasses	Firing	Warning	~	Platform

- 5.2. Select the **Alerting rules** tab to view the various alert definitions.

Name	Severity	Alert state	Source
AR AlertmanagerClusterDown	Warning	-	Platform
AR AlertmanagerClusterFailedToSendAlerts	Warning	-	Platform

5.3. Filter the alerting rules by name and search for the **storageClasses** term.

Name	Severity	Alert state
AR MultipleDefaultStorageClasses	Warning	1 Firing

5.4. Select the Warning alert that is labeled **MultipleDefaultStorageClasses** to view the details of the alerting rule. Inspect the **Description** and **Expression** definition for the rule.

Name	Severity	Alert state
AR MultipleDefaultStorageClasses	Warning	1 Firing

The screenshot shows the 'Alerting rules > Alerting rule details' section. At the top, it displays the alert rule name 'MultipleDefaultStorageClasses' with a warning icon. Below this, the 'Alerting rule details' section is shown with the following fields:

Name	Source
MultipleDefaultStorageClasses	Platform

**Severity**: Warning

**Description**: Cluster storage operator monitors all storage classes configured in the cluster and checks there is not more than one default StorageClass configured.

**Summary**: More than one default StorageClass detected.

**Expression**: `max_over_time(default_storage_class_count[5m]) > 1`

► 6. Inspect cluster metrics and execute an example query.

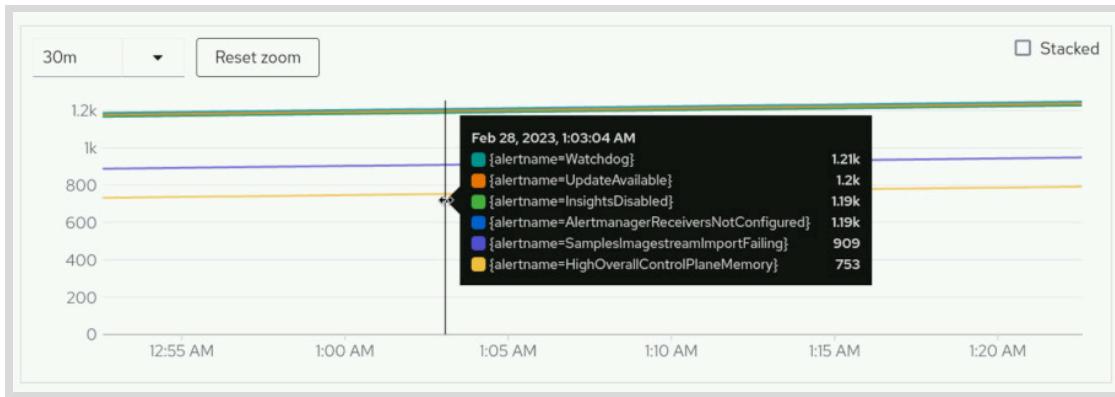
6.1. Navigate to **Observe > Metrics** to open the cluster metrics utility.

The screenshot shows the 'Metrics' page under the 'Observe' menu. The left sidebar includes options like Networking, Storage, Builds, Observe (with sub-options Alerting, Metrics, Dashboards, Targets), Compute, and a refresh dropdown. The main area is titled 'Metrics' with a graph icon and a message 'No query entered'. It includes a text input field for entering a query, a 'Hide graph' button, and a 'Insert example query' button.

6.2. Click **Insert example query** to populate the metrics graph with sample data.

The screenshot shows the same 'Metrics' page as above, but with a red box highlighting the 'Insert example query' button. The rest of the interface remains the same, showing the 'Metrics' title, the 'No query entered' message, and the input field.

6.3. From the graph, hover over any point on the timeline to view the detailed data points.



► 7. View the cluster events log from the web console.

7.1. Navigate to Home > Events to open the cluster events log.

The screenshot shows the OpenShift web console's Events page. The left sidebar is titled "Administrator" and includes links for Home, Overview, Projects, Search, API Explorer, and Events (which is selected). The main area has a header "Project: All Projects". Below it is a search bar with "Resources 1" and "All types". A "Streaming events..." button is present. The event list shows three entries:

- collect-profiles-28387320 (Generated from job-controller) - Job completed
- openshift-operator-lifecycle-manager (Generated from job-controller)
- Dec 22, 2023, 5:01



**Note**

The event log updates every 15 minutes and can require additional time to populate entries.

7.2. Scroll down to view a chronologically ordered stream that contains cluster events.

The screenshot shows a scrollable list of cluster events. At the top, there is a "Streaming events..." button and a "Showing 178" indicator. The list contains the following events:

- machine-config-operator (Generated from machine-config-operator)
  - Updated ConfigMap/kube-rbac-proxy -n openshift-machine-config-operator: cause by changes in data.config-file.yaml
- collect-profiles-28387320 (Generated from job-controller)
  - Job completed
- collect-profiles (Generated from cronjob-controller)
  - Deleted job collect-profiles-28387275
- collect-profiles (Generated from cronjob-controller)
  - Saw completed job: collect-profiles-28387320, status: Complete



**Note**

Select an event to open the **Details** page of the related resource.

- 8. Filter the events by resource name, type, or message.

- 8.1. From the **Resources** drop-down, use the search bar to filter for the **pod** term, and select the box labeled **Pod** to display events that relate to that resource.

The screenshot shows the 'Events' interface. At the top, there are three dropdown menus: 'Resources' (set to '1'), 'All types' (set to 'All types'), and a search bar labeled 'Filter Events by name or mess... /'. Below these is a search input field containing 'pod'. Underneath is a list of resource types with checkboxes: 'HPA' (HorizontalPodAutoscaler), 'Pod' (selected and checked), and 'PDB' (PodDisruptionBudget). The 'Pod' entry has a small green circular icon with a white 'P'.

- 8.2. Continue to refine the filter by selecting **Normal** from the types drop-down.

The screenshot shows the 'Events' interface. The 'Resources' dropdown is set to '1' and the 'Resource' dropdown is set to 'Pod'. The 'All types' dropdown is open, showing three options: 'All types' (selected and highlighted with a red border), 'Normal' (also highlighted with a red border), and 'Warning'. Below the dropdowns, there is a button labeled 'Streaming events' with a green circular icon containing two vertical bars.

- 8.3. Filter the results by using the **Message** text field. Enter the `started` container text to retrieve the matching events.

The screenshot shows the 'Events' page in the OpenShift web interface. At the top, there are filters for 'Resources' (set to 1), 'Normal', and a search bar containing 'started container'. Below the filters, there's a dropdown for 'Resource' set to 'Pod' and a 'X' button. A red box highlights the search bar. The main area displays three log entries:

- P collect-profiles-28387320-72rfq** (Generated from kubelet on master0) - Started container collect-profiles (Dec 22, 2023, 5:0)
- P collect-profiles-28387305-j25r6** (Generated from kubelet on master0) - Started container collect-profiles (Dec 22, 2023, 4:4)
- P collect-profiles-28387290-46fxs** (Generated from kubelet on master0) - Started container collect-profiles (Dec 22, 2023, 4:3)

On the left, there's a green circular icon with a play/pause symbol and the text 'Streaming events..'. On the right, it says 'Showing 12'.

## 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 intro-monitor
```

## ► Quiz

# Introduction to Kubernetes and OpenShift

Choose the correct answers to the following questions:

As the student user on the workstation machine, open a web browser and navigate to <https://console-openshift-console.apps.ocp4.example.com> to access the Red Hat OpenShift web console. Then, log in as the `admin` user with the `redhatocp` password.

Use the OpenShift web console to answer the following questions.

- ▶ **1. What is the installed version for the OpenShift cluster?**
  - a. 3.9.1
  - b. 4.3.2
  - c. 4.14.0
  - d. 5.4.3
  
- ▶ **2. Which three severity types are available for the alerts in the cluster? (Choose three.)**
  - a. Warning
  - b. Firing
  - c. Info
  - d. Urgent
  - e. Critical
  - f. Oops
  
- ▶ **3. Which three labels are on the thanos-querier route in the openshift-monitoring namespace? (Choose three.)**
  - a. app.kubernetes.io/component=query-layer
  - b. app.kubernetes.io/instance=thanos-querier
  - c. app.kubernetes.io/part-of=openshift-storage
  - d. app.kubernetes.io/version=0.30.2
  
- ▶ **4. Which two objects are listed as the StorageClasses objects for the cluster? (Choose two.)**
  - a. ceph-storage
  - b. nfs-storage
  - c. k8s-lvm-vg1
  - d. local-volume
  - e. lvms-vg1

► 5. When All Projects is selected at the top, which three operators are installed in the cluster? (Choose three.)

- a. MongoDB Operator
- b. MetalLB Operator
- c. Red Hat Fuse
- d. LVM Storage
- e. Package Server

## ► Solution

# Introduction to Kubernetes and OpenShift

Choose the correct answers to the following questions:

As the student user on the **workstation** machine, open a web browser and navigate to <https://console-openshift-console.apps.ocp4.example.com> to access the Red Hat OpenShift web console. Then, log in as the **admin** user with the **redhatocp** password.

Use the OpenShift web console to answer the following questions.

- ▶ **1. What is the installed version for the OpenShift cluster?**
  - a. 3.9.1
  - b. 4.3.2
  - c. 4.14.0
  - d. 5.4.3
  
- ▶ **2. Which three severity types are available for the alerts in the cluster? (Choose three.)**
  - a. Warning
  - b. Firing
  - c. Info
  - d. Urgent
  - e. Critical
  - f. Oops
  
- ▶ **3. Which three labels are on the thanos-querier route in the openshift-monitoring namespace? (Choose three.)**
  - a. app.kubernetes.io/component=query-layer
  - b. app.kubernetes.io/instance=thanos-querier
  - c. app.kubernetes.io/part-of=openshift-storage
  - d. app.kubernetes.io/version=0.30.2
  
- ▶ **4. Which two objects are listed as the StorageClasses objects for the cluster? (Choose two.)**
  - a. ceph-storage
  - b. nfs-storage
  - c. k8s-lvm-vg1
  - d. local-volume
  - e. lvms-vg1

► 5. When All Projects is selected at the top, which three operators are installed in the cluster? (Choose three.)

- a. MongoDB Operator
- b. MetalLB Operator
- c. Red Hat Fuse
- d. LVM Storage
- e. Package Server

## ► Lab

# Introduction to Kubernetes and OpenShift

Find essential information about your OpenShift cluster by navigating its web console.

## Outcomes

Navigate the Red Hat OpenShift Container Platform web console to find various information items and configuration details.

## 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 Red Hat OpenShift Container Platform is deployed and ready for the lab.

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

## Instructions

1. Log in to the Red Hat OpenShift Container Platform web console, with Red Hat Identity Management as the **admin** user with the **redhatocp** password, and review the answers for the preceding quiz.
2. View the cluster version on the Overview page for the cluster.
3. View the available alert severity types within the filters on the Alerting page.
4. View the labels for the **thanos-querier** route.
5. View the available storage classes in the cluster.
6. View the installed operators for the cluster.

## 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 intro-review
```

## ► Solution

# Introduction to Kubernetes and OpenShift

Find essential information about your OpenShift cluster by navigating its web console.

### Outcomes

Navigate the Red Hat OpenShift Container Platform web console to find various information items and configuration details.

### Before You Begin

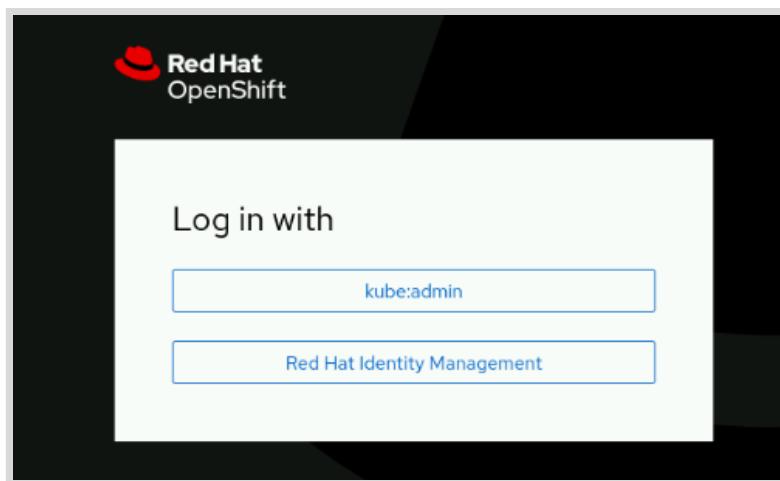
As the **student** user on the **workstation** machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the Red Hat OpenShift Container Platform is deployed and ready for the lab.

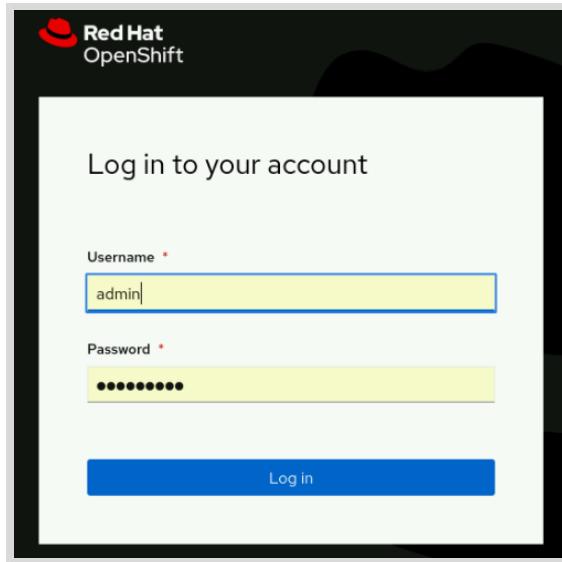
```
[student@workstation ~]$ lab start intro-review
```

### Instructions

1. Log in to the Red Hat OpenShift Container Platform web console, with Red Hat Identity Management as the **admin** user with the **redhatocp** password, and review the answers for the preceding quiz.
  - 1.1. Use a browser to view the login page at the <https://console-openshift-console.apps.ocp4.example.com> address.



- 1.2. Click Red Hat Identity Management, and supply the **admin** username and the **redhatocp** password, and then click **Log in** to access the home page.



**2.** View the cluster version on the Overview page for the cluster.

2.1. From the Home > Overview page, scroll down to view the cluster details.

A screenshot of the Red Hat OpenShift Cluster Overview page. The top navigation bar includes the Red Hat OpenShift logo, a menu icon, and user information. The main area is titled "Overview" and has a "Cluster" tab selected. Below this, there's a section titled "Getting started resources" with three items: "Set up your cluster", "Build with guided documentation", and "Explore new admin features". Each item has a brief description and a link to more information.

2.2. Locate the OpenShift version in the Details section.

**Details**

**Cluster API address**  
https://api.ocp4.example.com:6443

**Cluster ID**  
3b68e8b8-6464-4218-9e4c-56c91577b204

[OpenShift Cluster Manager](#)

**Infrastructure provider**  
None

**OpenShift version**  
4.14.0

**Update channel**  
Not available

**Control plane high availability**  
No (single control plane node)

3. View the available alert severity types within the filters on the Alerting page.

3.1. Navigate to the **Observe > Alerting** page.

**Alerting**

[Alerts](#)   [Silences](#)   [Alerting rules](#)

Name	Severity	State	Source
AL AlertmanagerReceiversNotConfigured	⚠ Warning	⚠ Pending Since Dec 22, 2023, 10:55 AM	Platform
AL InsightsDisabled	ⓘ Info	⚠ Pending Since Dec 22, 2023, 10:57 AM	Platform
AL TargetDown	⚠ Warning	⚠ Pending Since Dec 22, 2023, 10:55 AM	Platform
AL Watchdog	ⓘ None	⚠ Firing Since Dec 22, 2023, 10:54 AM	Platform

3.2. Click the Filter drop-down to view the available severity options.

The screenshot shows the 'Alerting' page in the Red Hat OpenShift web interface. The left sidebar has 'Alerts' selected. The main area has a filter bar with 'Name' and a search input. Below it, there's a 'Clear all filters' button and a 'Severity' dropdown menu. The 'Severity' dropdown is highlighted with a red box and contains the following options:

- Critical (0)
- Warning (2)
- Info (1)
- None (1)

To the right of the dropdown, the alert list is shown with columns: Severity, State, and Source. One alert is listed:

Severity	State	Source
None	Firing	Platform

A note below the alert says: "to ensure that the entire alerting This alert is always firing..."

**4.** View the labels for the thanos-querier route.

4.1. Navigate to the Networking > Routes page.

The screenshot shows the 'Routes' page in the Red Hat OpenShift web interface. The left sidebar has 'Networking' selected, and 'Routes' is highlighted. The main area has a 'Create Route' button and a table of routes. The table columns are: Name, Namespace, Status, Location, and Service. Two routes are listed:

Name	Namespace	Status	Location	Service
alertmanager-main	openshift-monitoring	Accepted	https://alertmanager-main-openshift-monitoring.apps.ocp4.example.com/api	alertmanager-main
canary	openshift-ingress-canary	Accepted	https://canary-openshift-ingress-canary.apps.ocp4.example.com	ingress-canary

4.2. Type the thanos keyword in the text search field.

The screenshot shows the 'Routes' page with a search filter applied. The search input field contains the word 'thanos' and is highlighted with a red box. The table below shows one route matching the search term:

Name	Namespace	Status	Location	Service
thanos-querier	openshift-monitoring	Accepted	https://thanos-querier-openshift-monitoring.apps.ocp4.example.com/api	thanos-querier

4.3. Select the thanos-querier route in the Name column.

Name	Namespace	Status	Location	Service	⋮
thanos-querier	openshift-monitoring	Accepted	https://thanos-querier-openshift-monitoring.apps.ocp4.example.com/api	thanos-querier	⋮

4.4. Scroll down on the thanos-querier route details page to view the labels.

Name	Namespace	Labels	Location	Status	Host	Path	Router canonical hostname
thanos-querier	openshift-monitoring	app.kubernetes.io/component=query-layer app.kubernetes.io/instance=thanos-querier app.kubernetes.io/managed-by=cluster-monitoring-operator app.kubernetes.io/name=thanos-query app.kubernetes.io/part-of=openshift-monitoring app.kubernetes.io/version=0.30.2	https://thanos-querier-openshift-monitoring.apps.ocp4.example.com/api	Accepted	thanos-querier-openshift-monitoring.apps.ocp4.example.com	/api	router-default.apps.ocp4.example.com

5. View the available storage classes in the cluster.

5.1. Navigate to the Storage > StorageClasses page.

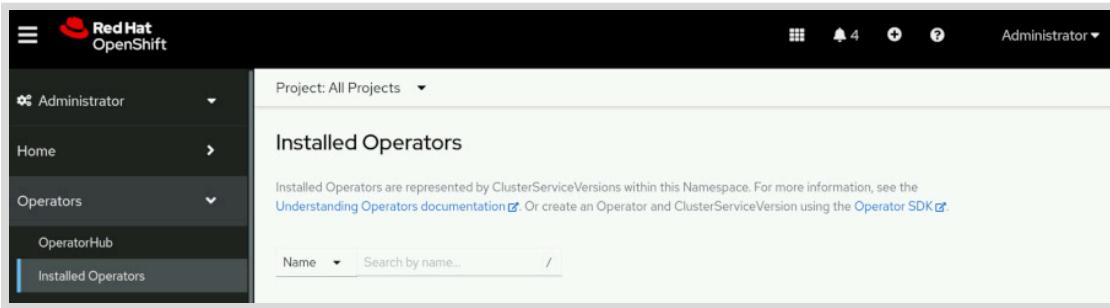
Name	Provisioner	Reclaim policy
lvm-vgl	topolvm.io	Delete
nfs-storage - Default	k8s-sigs.io/nfs-subdir-external-provisioner	Delete

5.2. View the available storage classes in the cluster.

Name	Provisioner	Reclaim policy
lvm-vgl	topolvm.io	Delete
nfs-storage - Default	k8s-sigs.io/nfs-subdir-external-provisioner	Delete

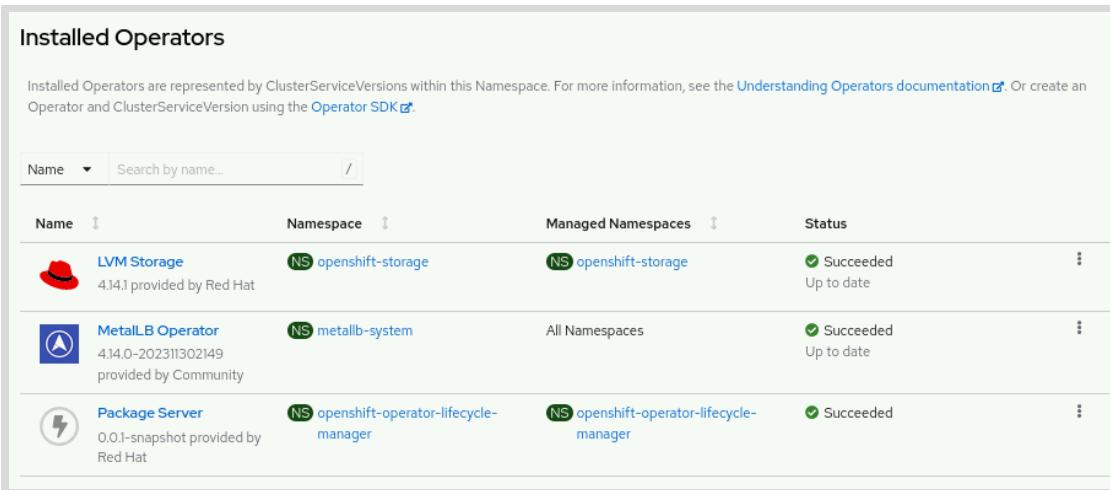
6. View the installed operators for the cluster.

6.1. Navigate to the Operators > Installed Operators page.



The screenshot shows the Red Hat OpenShift web interface. The top navigation bar includes the Red Hat logo, a user dropdown for 'Administrator', and a notification icon with the number '4'. The main content area is titled 'Installed Operators'. It displays a brief description: 'Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and ClusterServiceVersion using the [Operator SDK](#).'. Below this is a search bar with 'Name' and 'Search by name...' fields, and a '/' separator. The left sidebar has a navigation tree with 'Home', 'Operators' (selected), 'OperatorHub', and 'Installed Operators'.

6.2. View the list of installed operators in the cluster.



The screenshot shows the 'Installed Operators' list. The title is 'Installed Operators'. A note at the top says: 'Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and ClusterServiceVersion using the [Operator SDK](#).'. Below is a search bar with 'Name' and 'Search by name...' fields, and a '/' separator. The table has columns: 'Name', 'Namespace', 'Managed Namespaces', and 'Status'. Three operators are listed:

Name	Namespace	Managed Namespaces	Status
LVM Storage 4.14.1 provided by Red Hat	openshift-storage	openshift-storage	<span style="color: green;">✓ Succeeded</span> Up to date
MetalLB Operator 4.14.0-202311302149 provided by Community	metallb-system	All Namespaces	<span style="color: green;">✓ Succeeded</span> Up to date
Package Server 0.0.1-snapshot provided by Red Hat	openshift-operator-lifecycle- manager	openshift-operator-lifecycle- manager	<span style="color: green;">✓ Succeeded</span>

## 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 intro-review
```

# Summary

---

- A *container* is an encapsulated process that includes the required runtime dependencies for an application to run.
- Containerization addresses the application development challenges around code portability, to aid in consistently running an application from diverse environments.
- Containerization also aims to modularize applications to improve development and maintenance on the various components of the application.
- When running containers at scale, it becomes challenging to configure and deliver high availability applications and to set up networking without a container platform, such as Kubernetes.
- Pods are the smallest organizational unit for a containerized application in a Kubernetes cluster.
- Red Hat OpenShift Container Platform (RHOCP) adds enterprise-class functions to the Kubernetes container platform to deliver the wider business needs.
- Most administrative tasks that cluster administrators and developers perform are available through the RHOCP web console.
- Logs, metrics, alerts, terminal connections to the nodes and pods in the cluster, and many other features are available through the RHOCP web console.



## Chapter 2

# Kubernetes and OpenShift Command-line Interfaces and APIs

### Goal

Access an OpenShift cluster by using the command line and query its Kubernetes API resources to assess the health of a cluster.

### Objectives

- Access an OpenShift cluster by using the Kubernetes and OpenShift command-line interfaces.
- Query, format, and filter attributes of Kubernetes resources.
- Query the health of essential cluster services and components.

### Sections

- The Kubernetes and OpenShift Command-line Interfaces (and Guided Exercise)
- Inspect Kubernetes Resources (and Guided Exercise)
- Assess the Health of an OpenShift Cluster (and Guided Exercise)

### Lab

- Kubernetes and OpenShift Command-line Interfaces and APIs

# The Kubernetes and OpenShift Command-line Interfaces

---

## Objectives

- Access an OpenShift cluster by using the Kubernetes and OpenShift command-line interfaces.

## The Kubernetes and OpenShift Command-line Interfaces

You can manage an OpenShift cluster from the web console or by using the `kubectl` or `oc` command-line interfaces (CLI). The `kubectl` commands are native to Kubernetes, and are a thin wrapper over the Kubernetes API. The OpenShift `oc` commands are a superset of the `kubectl` commands, and add commands for the OpenShift-specific features. In this course, examples of both the `kubectl` and the `oc` commands are shown, to highlight the differences between the commands.

With the `oc` command, you can create applications and manage Red Hat OpenShift Container Platform (RHOC) projects from a terminal. The OpenShift CLI is ideal in the following situations:

- Working directly with project source code.
- Scripting OpenShift Container Platform operations.
- Managing projects that are restricted by bandwidth.
- When the web console is unavailable.
- Working with OpenShift resources, such as `routes` and `deployment configs`.

## Kubernetes Command-line Tool

The `oc` CLI installation also includes an installation of the `kubectl` CLI, which is the recommended method for installing the `kubectl` CLI for OpenShift users.

You can also install the `kubectl` CLI independently of the `oc` CLI. You must use a `kubectl` CLI version that is within one minor version difference of your cluster. For example, a v1.26 client can communicate with v1.25, v1.26, and v1.27 control planes. Using the latest compatible version of the `kubectl` CLI can help to avoid unforeseen issues.

To perform a manual installation of the `kubectl` binary for a Linux installation, you must first download the latest release by using the `curl` command.

```
[user@host ~]$ curl -LO "https://dl.k8s.io/release/$(curl -L \
-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

Then, you must download the `kubectl` checksum file and then validate the `kubectl` binary against the checksum file.

```
[user@host ~]$ curl -LO "https://dl.k8s.io/$(curl -L \
-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
```

```
[user@host ~]$ echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
kubectl: OK
```

If the check fails, then the `sha256sum` command exits with nonzero status, and prints a `kubectl: FAILED` message.

You can then install the `kubectl` CLI.

```
[user@host ~]$ sudo install -o root -g root -m 0755 kubectl \
/usr/local/bin/kubectl
```



### Note

If you do not have `root` access on the target system, you can still install the `kubectl` CLI to the `~/.local/bin` directory. For more information, refer to <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>.

Finally, use the `kubectl version` command to verify the installed version. This command prints the client and server versions. Use the `--client` option to view the client version only.

```
[user@host ~]$ kubectl version --client
```

Alternatively, a distribution that is based on Red Hat Enterprise Linux (RHEL) can install the `kubectl` CLI with the following command:

```
[user@host ~]$ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
[user@host ~]$ sudo yum install -y kubectl
```

To view a list of the available `kubectl` commands, use the `kubectl --help` command.

```
[user@host ~]$ kubectl --help
kubectl controls the Kubernetes cluster manager.

Find more information at:
https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):
  create      Create a resource from a file or from stdin
  expose     Take a replication controller, service, deployment or pod and
             expose it as a new Kubernetes Service
  run        Run a particular image on the cluster
  set        Set specific features on objects
```

```
Basic Commands (Intermediate):
...output omitted...
```

You can also use the `--help` option on any command to view detailed information about the command, including its purpose, examples, available subcommands, and options. For example, the following command provides information about the `kubectl create` command and its usage.

```
[user@host ~]$ kubectl create --help
Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

Examples:
# Create a pod using the data in pod.json
kubectl create -f ./pod.json

# Create a pod based on the JSON passed into stdin
cat pod.json | kubectl create -f -

# Edit the data in registry.yaml in JSON then create the resource using the
# edited data
kubectl create -f registry.yaml --edit -o json

Available Commands:
clusterrole           Create a cluster role
clusterrolebinding   Create a cluster role binding for a particular cluster
role
...output omitted...
```

Kubernetes uses many resource components to support applications. The `kubectl explain` command provides detailed information about the attributes of a given resource. For example, use the following command to learn more about the attributes of a pod resource.

```
[user@host ~]$ kubectl explain pod
KIND:     Pod
VERSION:  v1

DESCRIPTION:
Pod is a collection of containers that can run on a host. This resource is
created by clients and scheduled onto hosts.

FIELDS:
apiVersion <string>
APIVersion defines the versioned schema of this representation of an
object.
...output omitted...
```

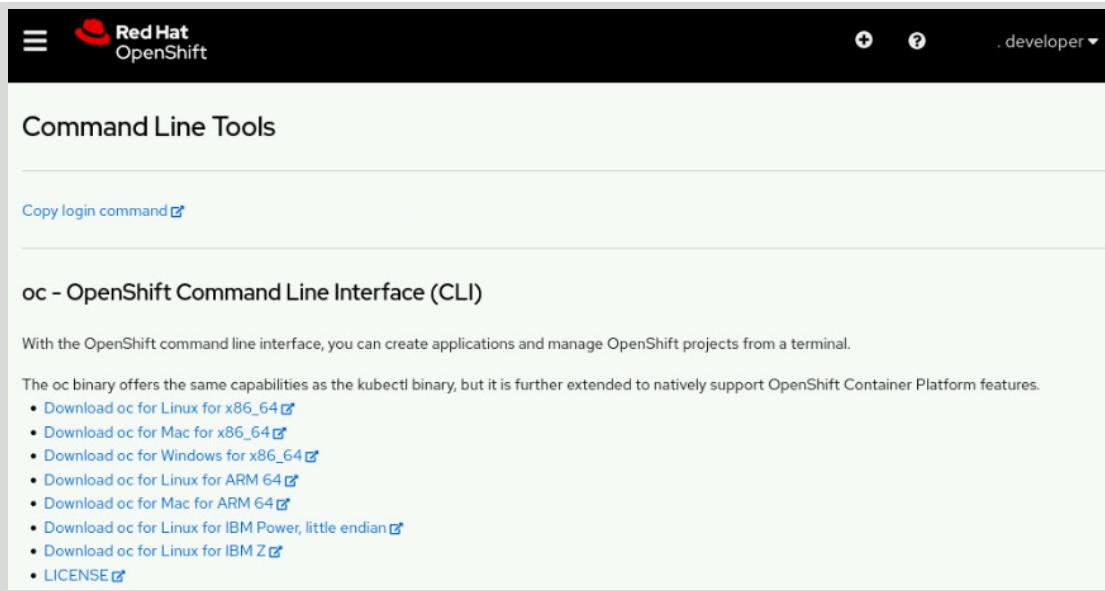
Refer to Kubernetes Documentation - Getting Started [<https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/>] for further information about the `kubectl` commands.

## The OpenShift Command-line Tool

The main method of interacting with an RHOCP cluster is by using the `oc` command.

You can download the `oc` CLI from the OpenShift web console to ensure that the CLI tools are compatible with the RHOCP cluster. From the OpenShift web console, navigate to **Help > Command line tools**. The **Help** menu is represented by a ? icon. The web console provides several installation options for the `oc` client, such as downloads for the following operating systems:

- x86\_64 Windows, Mac, and Linux systems
- ARM 64 Linux and Mac systems
- Linux for IBM Z, IBM Power, and little endian



The screenshot shows the Red Hat OpenShift web console interface. At the top, there's a navigation bar with a menu icon, the Red Hat logo, and the text "Red Hat OpenShift". To the right are icons for help, developer mode, and user authentication. Below the header, the page title is "Command Line Tools". Under this, there's a "Copy login command" button. The main content area is titled "oc - OpenShift Command Line Interface (CLI)". It contains a brief description: "With the OpenShift command line interface, you can create applications and manage OpenShift projects from a terminal." Below this, it states: "The oc binary offers the same capabilities as the kubectl binary, but it is further extended to natively support OpenShift Container Platform features." A list of download links follows:

- [Download oc for Linux for x86\\_64](#)
- [Download oc for Mac for x86\\_64](#)
- [Download oc for Windows for x86\\_64](#)
- [Download oc for Linux for ARM 64](#)
- [Download oc for Mac for ARM 64](#)
- [Download oc for Linux for IBM Power, little endian](#)
- [Download oc for Linux for IBM Z](#)
- [LICENSE](#)

The basic usage of the `oc` command is through its subcommands in the following syntax:

```
[user@host ~]$ oc command
```

Because the `oc` CLI is a superset of the `kubectl` CLI, the `version`, `--help`, and `explain` commands are the same for both CLIs. However, the `oc` CLI includes additional commands that are not included in the `kubectl` CLI, such as the `oc login` and `oc new-project` commands.

## Managing Resources at the Command Line

Developers who are familiar with Kubernetes can use the `kubectl` utility to manage a RHOCP cluster. This course uses the `oc` command-line utility, to take advantage of additional RHOCP features. The `oc` commands manage resources that are exclusive to RHOCP, such as projects, deployment configurations, routes, and image streams.

Before you can interact with your RHOCP cluster, you must authenticate your requests. Use the `oc login` command to authenticate your requests. The `oc login` command provides role-based authentication and authorization that protects the RHOCP cluster from unauthorized access. The syntax to log in is shown below:

```
[user@host ~]$ oc login cluster-url
```

For example, in this course, you can use the following command:

```
[user@host ~]$ oc login https://api.ocp4.example.com:6443
Username: developer
Password: developer
Login successful.
```

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

```
$ oc new-project <projectname>
```

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

After authenticating to the RHOCP cluster, you can create a project with the `oc new-project` command. Projects provide isolation between your application resources. Projects are Kubernetes namespaces with additional annotations that provide multitenancy scoping for applications.

```
[user@host ~]$ oc new-project myapp
```

Several essential commands can manage RHOCP and Kubernetes resources, as described here. Unless otherwise specified, the following commands are compatible with both the `oc` and `kubectl` CLIs.

Some commands require a user with cluster administrator access. The following list includes several useful `oc` commands for cluster administrators.

### **oc cluster-info**

The `cluster-info` command prints the address of the control plane and other cluster services. The `oc cluster-info dump` command expands the output to include helpful details for debugging cluster problems.

```
[user@host ~]$ oc cluster-info
Kubernetes control plane is running at https://api.ocp4.example.com:6443
...output omitted...
```

### **oc api-versions**

The structure of cluster resources has a corresponding API version, which the `oc api-versions` command displays. The command prints the supported API versions on the server, in the form of "group/version".

In the following example, the group is `admissionregistration.k8s.io` and the version is `v1`:

```
[user@host ~]$ oc api-versions
admissionregistration.k8s.io/v1
...output omitted...
```

### **oc get clusteroperator**

The cluster operators that Red Hat ships serve as the architectural foundation for RHOCP. RHOCP installs cluster operators by default. Use the `oc get clusteroperator` command to see a list of the cluster operators:

```
[user@host ~]$ oc get clusteroperator
NAME                  VERSION   AVAILABLE PROGRESSING DEGRADED SINCE ...
authentication        4.14.0    True      False       False    18d
baremetal             4.14.0    True      False       False    18d
...output omitted...
```

Other useful commands are available to both regular and administrator users:

### **oc get**

Use the `get` command to retrieve information about resources in the selected project. Generally, this command shows only the most important characteristics of the resources, and omits more detailed information.

The `oc get RESOURCE_TYPE` command displays a summary of all resources of the specified type.

For example, the following command returns the list of the pod resources in the current project:

```
[user@host ~]$ oc get pod
NAME                READY   STATUS    RESTARTS   AGE
quotes-api-6c9f758574-nk8kd   1/1     Running   0          39m
quotes-ui-d7d457674-rbk17    1/1     Running   0          67s
```

You can use the `oc get RESOURCE_TYPE RESOURCE_NAME` command to export a resource definition. Typical use cases include creating a backup or modifying a definition. The `-o yaml` option prints the object representation in YAML format. You can change to JSON format by providing a `-o json` option.

### **oc get all**

Use the `oc get all` command to retrieve a summary of the most important components of a cluster. This command iterates through the major resource types for the current project, and prints a summary of their information:

```
[user@host ~]$ oc get all
NAME          DOCKER REPO                               TAGS      UPDATED
is/nginx     172.30.1.1:5000/basic-kubernetes/nginx   latest    About an hour ago

NAME      REVISION  DESIRED  CURRENT  TRIGGERED BY
dc/nginx  1         1         1        config,image(nginx:latest)

NAME      DESIRED  CURRENT  READY    AGE
rc/nginx-1 1         1         1        1h

NAME      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
svc/nginx  172.30.72.75  <none>           80/TCP, 443/TCP  1h

NAME      READY  STATUS    RESTARTS  AGE
po/nginx-1-ypp8t  1/1    Running   0          1h
```

### **oc describe**

If the summaries from the `get` command are insufficient, then you can use the `oc describe RESOURCE_TYPE RESOURCE_NAME` command to retrieve additional information. Unlike the `get` command, you can use the `describe` command to iterate through all the different

resources by type. Although most major resources can be described, this function is not available across all resources. The following example demonstrates describing a pod resource:

```
[user@host ~]$ oc describe mysql-openshift-1-glgrp
Name:           mysql-openshift-1-glgrp
Namespace:      mysql-openshift
Priority:       0
Node:           cluster-worker-1/172.25.250.52
Start Time:     Fri, 15 Feb 2019 02:14:34 +0000
Labels:         app=mysql-openshift
                deployment=mysql-openshift-1
...output omitted...
Status:         Running
IP:             10.129.0.85
```

### oc explain

To learn about the fields of an API resource object, use the `oc explain` command. This command describes the purpose and the fields that are associated with each supported API resource. You can also use this command to print the documentation of a specific field of a resource. Fields are identified via a JSONPath identifier. The following example prints the documentation for the `.spec.containers.resources` field of the pod resource type:

```
[user@host ~]$ oc explain pods.spec.containers.resources
KIND:        Pod
VERSION:    v1

FIELD: resources <ResourceRequirements>

DESCRIPTION:
Compute Resources required by this container. Cannot be updated. More info:
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

ResourceRequirements describes the compute resource requirements.

FIELDS:
claims      <[]ResourceClaim>
Claims lists the names of resources, defined in spec.resourceClaims, that are used by this container.

This is an alpha field and requires enabling the DynamicResourceAllocation feature gate.

This field is immutable. It can only be set for containers.

limits <map[string]Quantity>
Limits describes the maximum amount of compute resources allowed. More info:
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

requests <map[string]Quantity>
Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is
```

```
explicitly specified, otherwise to an implementation-defined value. Requests  
cannot exceed Limits. More info:  
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
```

Add the `--recursive` flag to display all fields of a resource without descriptions. Information about each field is retrieved from the server in OpenAPI format.

#### **oc create**

Use the `create` command to create a RHOCP resource in the current project. This command creates resources from a resource definition. Typically, this command is paired with the `oc get RESOURCE_TYPE RESOURCE_NAME -o yaml` command for editing definitions. Developers commonly use the `-f` flag to indicate the file that contains the JSON or YAML representation of an RHOCP resource.

For example, to create resources from the `pod.yaml` file, use the following command:

```
[user@host ~]$ oc create -f pod.yaml  
pod/quotes-pod created
```

RHOCP resources in the YAML format are discussed later.

#### **oc status**

The `oc status` command provides a high-level overview of the current project. The command shows services, deployments, build configurations, and active deployments. Information about any misconfigured components is also shown. The `--suggest` option shows additional details for any identified issues.

#### **oc delete**

Use the `delete` command to delete an existing RHOCP resource from the current project. You must specify the resource type and the resource name.

For example, to delete the `quotes-ui` pod, use the following command:

```
[user@host ~]$ oc delete pod quotes-ui  
pod/quotes-ui deleted
```

A fundamental understanding of the RHOCP architecture is needed here, because deleting managed resources, such as pods, results in the automatic creation of new instances of those resources. When a project is deleted, it deletes all the resources and applications within it.

Each of these commands is executed in the current selected project. To execute commands in a different project, you must include the `--namespace` or `-n` options.

```
[user@host ~]$ oc get pods -n openshift-apiserver  
NAME          READY   STATUS    RESTARTS   AGE  
apiserver-68c9485699-ndqlc   2/2     Running   2          18d
```

Refer to the references for a complete list of `oc` commands.

## **Authentication with OAuth**

For users to interact with RHOCP, they must first authenticate to the cluster. The authentication layer identifies the user that is associated with requests to the RHOCP API. After authentication,

the authorization layer then uses information about the requesting user to determine whether the request is allowed.

A user in OpenShift is an entity that can make requests to the RHOCP API. An RHOCP User object represents an actor that can be granted permissions in the system by adding roles to the user or to the user's groups. Typically, this represents the account of a developer or an administrator.

Several types of users can exist.

### Regular users

Most interactive RHOCP users are represented by this user type. An RHOCP User object represents a regular user.

### System users

Infrastructure uses system users to interact with the API securely. Some system users are automatically created, including the cluster administrator, with access to everything. By default, unauthenticated requests use an anonymous system user.

### Service accounts

ServiceAccount objects represent service accounts. RHOCP creates service accounts automatically when a project is created. Project administrators can create additional service accounts to define access to the contents of each project.

Each user must authenticate to access a cluster. After authentication, policy determines what the user is authorized to do.



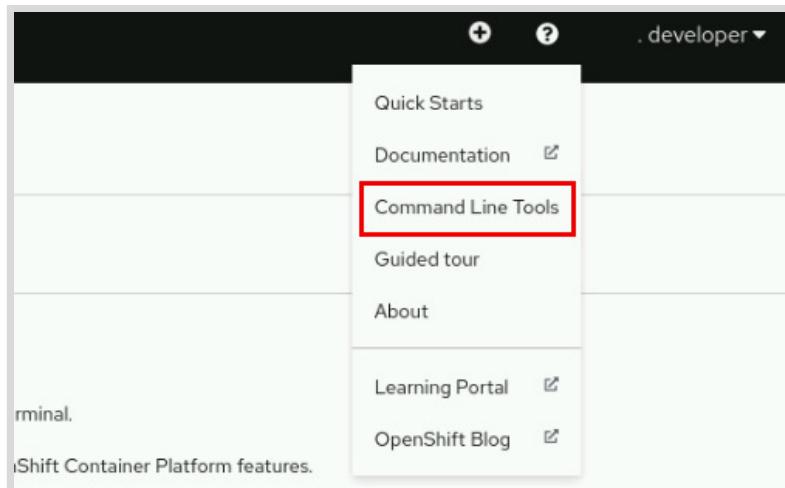
#### Note

Authentication and authorization are covered in greater detail in the "DO280: Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster" course.

The RHOCP control plane includes a built-in OAuth server. To authenticate themselves to the API, users obtain OAuth access tokens. Token authentication is the only guaranteed method to work with any OpenShift cluster, because enterprise Single Sign-On (SSO) might replace the login form of the web console.

When a person requests a new OAuth token, the OAuth server uses the configured identity provider to determine the identity of the person who makes the request. The OAuth server then determines the user that the identity maps to; creates an access token for that user; and then returns the token for use.

To retrieve an OAuth token by using the OpenShift web console, navigate to **Help > Command line tools**. The Help menu is represented by a ? icon.



On the **Command Line Tools** page, navigate to **Copy login Command**. The following page requires you to log in with your OpenShift user credentials. Next, navigate to **Display token**. Use the command under the **Log in with this token** label to log in to the OpenShift API.

**Your API token is**  
sha256~g5R--bbM\_sZqCQUXjMTGtld84sgkTG402p8xDmU1oJU

**Log in with this token**

```
oc login --token=sha256~g5R--bbM_sZqCQUXjMTGtld84sgkTG402p8xDmU1oJU --server=https://api.ocp4.example.com:6443
```

**Use this token directly against the API**

```
curl -H "Authorization: Bearer sha256~g5R--bbM_sZqCQUXjMTGtld84sgkTG402p8xDmU1oJU" "https://api.ocp4.example.com:6443/apis/user.openshift.io/v1/users/~"
```

[Request another token](#)

Copy the command from the web console and paste it on the command line. The copied command uses the `--token` and `--server` options, similar to the following example.

```
[user@host ~]$ oc login --token=sha256-BW...rA8 \
--server=https://api.ocp4.example.com:6443
```



## References

For more information, refer to the *Getting Started with the OpenShift CLI* 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#cli-getting-started](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#cli-getting-started)

For more information, refer to the *CLI Developer Commands* chapter in the Red Hat OpenShift Container Platform 4.14 *CLI Tools* documentation at Refer to the **OpenShift CLI Developer Command Reference**

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/cli\\_tools/index#cli-developer-commands](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#cli-developer-commands)

### Kubernetes Documentation – Install and Set Up kubectl on Linux

<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/authentication\\_and\\_authorization/index#understanding-authentication](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/authentication_and_authorization/index#understanding-authentication)

## ► Guided Exercise

# The Kubernetes and OpenShift Command-line Interfaces

Access an OpenShift cluster by using the command-line to get information about cluster services and nodes.

### Outcomes

- Use the OpenShift web console to locate the installation file for the `oc` OpenShift command-line interface.
- Get and use a token from the web console to access the cluster from the command line.
- Identify key differences between the `kubectl` and `oc` command-line tools.
- Identify the main components of OpenShift and Kubernetes.

### Before You Begin

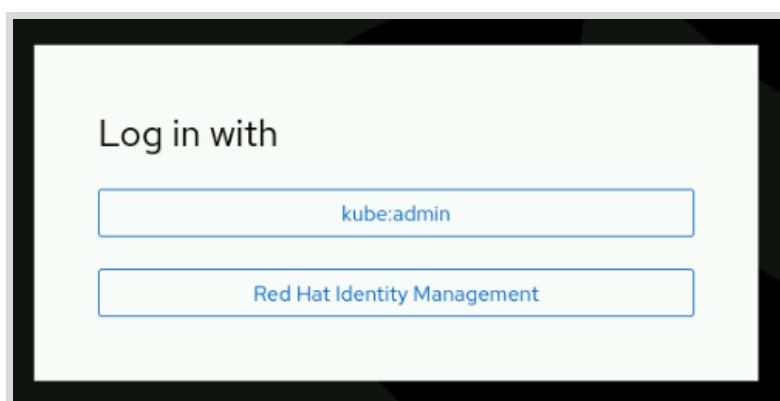
As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that all resources are available for this exercise.

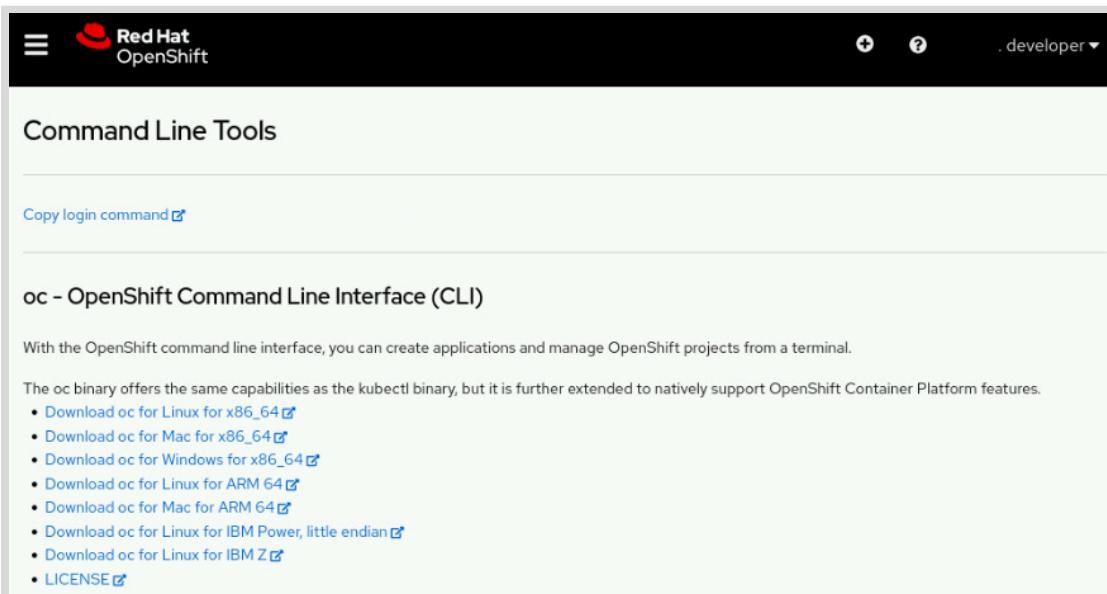
```
[student@workstation ~]$ lab start cli-interfaces
```

### Instructions

- 1. Log in to the OpenShift web console as the **developer** user. Locate the installation file for the `oc` OpenShift command-line interface (CLI).
- 1.1. Open a web browser and then navigate to `https://console-openshift-console.apps.ocp4.example.com`.
  - 1.2. Click **Red Hat Identity Management** and log in as the **developer** user with the **developer** password.



13. Locate the installation file for the `oc` CLI. From the OpenShift web console, select **Help > Command line tools**. The **Help** menu is represented by a ? icon.



The screenshot shows the Red Hat OpenShift web console with the title 'Command Line Tools'. Below the title is a link 'Copy login command'. Under the heading 'oc - OpenShift Command Line Interface (CLI)', it says: 'With the OpenShift command line interface, you can create applications and manage OpenShift projects from a terminal.' It also states: 'The oc binary offers the same capabilities as the kubectl binary, but it is further extended to natively support OpenShift Container Platform features.' A bulleted list of download links follows:

- [Download oc for Linux for x86\\_64](#)
- [Download oc for Mac for x86\\_64](#)
- [Download oc for Windows for x86\\_64](#)
- [Download oc for Linux for ARM 64](#)
- [Download oc for Mac for ARM 64](#)
- [Download oc for Linux for IBM Power, little endian](#)
- [Download oc for Linux for IBM Z](#)
- [LICENSE](#)

The `oc` binary is available for multiple operating systems and architectures. For each operating system and architecture, the `oc` binary also includes the `kubectl` binary.



### Note

You do not need to download or install the `oc` and `kubectl` binaries, which are already installed on the **workstation** machine.

- 2. Download an authorization token from the web console. Then, use the token and the `oc` command to log in to the OpenShift cluster.
- 2.1. From the **Command Line Tools** page, click the **Copy login command** link.
  - 2.2. The link opens a login page. Click **Red Hat Identity Management** and log in as the **developer** user with the **developer** password.
  - 2.3. A web page is displayed. Click the **Display token** link to show your API token and the login command.

**Your API token is**

sha256~nJg06D38lkgTfN9K2Ib7cTvBs3dvVKQfnrXrm5oBiU4

**Log in with this token**

```
oc login --token=sha256~nJg06D38lkgTfN9K2Ib7cTvBs3dvVKQfnrXrm5oBiU4  
--server=https://api.ocp4.example.com:6443
```

**Use this token directly against the API**

```
curl -H "Authorization: Bearer sha256~nJg06D38lkgTfN9K2Ib7cTvBs3dvVKQfnrXrm5oBiU4"  
"https://api.ocp4.example.com:6443/apis/user.openshift.io/v1/users/~"
```

[Request another token](#)

- 2.4. Copy the `oc login` command to your clipboard. Open a terminal window and then use the copied command to log in to the cluster with your token.

```
[student@workstation ~]$ oc login --token=sha256-fypX...ot6A \  
--server=https://api.ocp4.example.com:6443  
Logged into "https://api.ocp4.example.com:6443" as "developer" using the token  
provided.  
...output omitted...
```

- 3. Compare the available commands for the `kubectl` and `oc` commands.

- 3.1. Use the `help` command to list and review the available commands for the `kubectl` command.

```
[student@workstation ~]$ kubectl help  
kubectl controls the Kubernetes cluster manager.  
  
Find more information at: https://kubernetes.io/docs/reference/kubectl/  
  
Basic Commands (Beginner):  
  create          Create a resource from a file or from stdin  
  expose          Take a replication controller, service, deployment or pod and  
  expose it as a new Kubernetes service  
  run             Run a particular image on the cluster  
  set             Set specific features on objects  
  
Basic Commands (Intermediate):  
  explain         Get documentation for a resource  
  get             Display one or many resources  
  edit            Edit a resource on the server  
  delete          Delete resources by file names, stdin, resources and names, or  
  by resources and label selector  
  ...output omitted....
```

Notice that the `kubectl` command does not provide a `login` command.

- 3.2. Examine the available subcommands and options for the `kubectl create` command by using the `--help` option.

```
[student@workstation ~]$ kubectl create --help
Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

Examples:
# Create a pod using the data in pod.json
kubectl create -f ./pod.json
...output omitted...
Available Commands:
clusterrole          Create a cluster role
clusterrolebinding   Create a cluster role binding for a particular cluster
role
configmap            Create a config map from a local file, directory or
literal value
cronjob              Create a cron job with the specified name
deployment           Create a deployment with the specified name
...output omitted...
Options:
--allow-missing-template-keys=true:
If true, ignore any errors in templates when a field or map key is missing in the
template. Only applies to
golang and jsonpath output formats.

--dry-run='none':
Must be "none", "server", or "client". If client strategy, only print the object
that would be sent, without
sending it. If server strategy, submit server-side request without persisting the
resource.
...output omitted...
Usage:
kubectl create -f FILENAME [options]

Use "kubectl create <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all
commands).
```

You can use the `--help` option with any `kubectl` command. The `--help` option provides information about a command, including the available subcommands and options, and the command syntax.

- 3.3. List and review the available commands for the `oc` binary by using the `help` command.

```
[student@workstation ~]$ oc help
OpenShift Client

This client helps you develop, build, deploy, and run your applications on any
OpenShift or Kubernetes cluster. It also includes the administrative
commands for managing a cluster under the 'adm' subcommand.
```

```
Basic Commands:
  login           Log in to a server
  new-project     Request a new project
  new-app          Create a new application
  status           Show an overview of the current project
  project          Switch to another project
  projects         Display existing projects
  explain          Get documentation for a resource
...output omitted...
```

The `oc` command supports the same capabilities as the `kubectl` command. The `oc` command provides additional commands to natively support an OpenShift cluster. For example, the `new-project` command creates a project, which is a Kubernetes namespace, in the OpenShift cluster. The `new-app` command is unique to the `oc` command. It creates applications by using existing source code or prebuilt images.

- 3.4. Use the `--help` option with the `oc create` command to view the available subcommands and options.

```
[student@workstation ~]$ oc create --help
Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

Examples:
# Create a pod using the data in pod.json
oc create -f ./pod.json
...output omitted...

Available Commands:
  build           Create a new build
  clusterresourcequota Create a cluster resource quota
  clusterrole      Create a cluster role
  clusterrolebinding Create a cluster role binding for a particular cluster
  role
  configmap       Create a config map from a local file, directory or
literal value
  cronjob          Create a cron job with the specified name
  deployment        Create a deployment with the specified name
  deploymentconfig Create a deployment config with default options that uses
  a given image
...output omitted...
Options:
  --allow-missing-template-keys=true:
    If true, ignore any errors in templates when a field or map key is missing in the
    template. Only applies to
    golang and jsonpath output formats.

  --dry-run='none':
    Must be "none", "server", or "client". If client strategy, only print the object
    that would be sent, without
    sending it. If server strategy, submit server-side request without persisting the
    resource.
...output omitted...
```

Usage:

```
oc create -f FILENAME [options]
....output omitted....
```

The `oc create` command includes the same subcommands and options as the `kubectl create` command, and provides additional subcommands for OpenShift resources. For example, you can use the `oc create` command to create OpenShift resources such as a deployment configuration, a route, and an image stream.

- ▶ **4.** Identify the components and Kubernetes resources of an OpenShift cluster by using the terminal. Unless otherwise noted, all commands are available for the `oc` and `kubectl` commands.
  - 4.1. In a terminal, use the `oc login` command to log in to the cluster as the `admin` user with the `redhatocp` password. Regular cluster users, such as the `developer` user, cannot list resources at a cluster scope.

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

- 4.2. Identify the cluster version with the `version` command.

```
[student@workstation ~]$ oc version
Client Version: 4.14.0
Kustomize Version: v5.0.1
Server Version: 4.14.0
Kubernetes Version: v1.27.6+f67aeb3
```

- 4.3. Use the `cluster-info` command to identify the URL for the Kubernetes control plane.

```
[student@workstation ~]$ oc cluster-info
Kubernetes control plane is running at https://api.ocp4.example.com:6443

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

- 4.4. Identify the supported API versions by using the `api-versions` command.

```
[student@workstation ~]$ oc api-versions
admissionregistration.k8s.io/v1
apiextensions.k8s.io/v1
apiregistration.k8s.io/v1
apiserver.openshift.io/v1
apps.openshift.io/v1
apps/v1
...output omitted....
```

- 4.5. List cluster operators with the `get clusteroperator` command.

```
[student@workstation ~]$ oc get clusteroperator
NAME          VERSION  AVAILABLE  PROGRESSING  DEGRADED
SINCE ...
authentication  4.14.0   True       False        False      18d
baremetal       4.14.0   True       False        False      18d
cloud-controller-manager 4.14.0   True       False        False      18d
cloud-credential 4.14.0   True       False        False      18d
cluster-autoscaler 4.14.0   True       False        False      18d
config-operator 4.14.0   True       False        False      18d
console         4.14.0   True       False        False      18d
control-plane-machine-set 4.14.0   True       False        False      18d
csi-snapshot-controller 4.14.0   True       False        False      18d
dns             4.14.0   True       False        False      5h24m
etcd            4.14.0   True       False        False      18d
image-registry 4.14.0   True       False        False      18d
ingress         4.14.0   True       False        False      18d
...output omitted...
```

- 4.6. Use the `get` command to list pods in the `openshift-api` project. Specify the project with the `-n` option.

```
[student@workstation ~]$ oc get pods -n openshift-apiserver
NAME           READY  STATUS    RESTARTS  AGE
apiserver-68c9485699-ndqlc  2/2    Running   6          18d
```

- 4.7. Use the `oc status` command to retrieve the status of resources in the `openshift-authentication` project.

```
[student@workstation ~]$ oc status -n openshift-authentication
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+,
unavailable in v4.10000+
In project openshift-authentication on server https://api.ocp4.example.com:6443

https://oauth-openshift.apps.ocp4.example.com (passthrough) to pod port 6443 (svc/
oauth-openshift)
  deployment/oauth-openshift deploys quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:64e6...de42
    deployment #7 running for 2 weeks - 1 pod
    deployment #6 deployed 2 weeks ago
    deployment #4 deployed 2 weeks ago
    deployment #5 deployed 2 weeks ago
    deployment #3 deployed 2 weeks ago
    deployment #2 deployed 2 weeks ago
    deployment #1 deployed 2 weeks ago
...output omitted...
```

- 4.8. Use the `explain` command to list the description and available fields for `services` resources.

```
[student@workstation ~]$ oc explain services
KIND:     Service
VERSION:  v1
```

**DESCRIPTION:**

Service is a named abstraction of software service (for example, mysql) consisting of local port (for example 3306) that the proxy listens on, and the selector that determines which pods will answer requests sent through the proxy.

**FIELDS:**

`apiVersion <string>`

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values.

*...output omitted...*

- 4.9. Use the `get` command to list cluster nodes.

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

NAME	STATUS	ROLES	AGE	VERSION
master01	Ready	control-plane, master, worker	18d	v1.27.6+f67aeb3

A single node exists in the cluster.

## 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 cli-interfaces
```

# Inspect Kubernetes Resources

---

## Objectives

- Query, format, and filter attributes of Kubernetes resources.

## Kubernetes and OpenShift Resources

Kubernetes uses API resource objects to represent the intended state of everything in the cluster. All administrative tasks require creating, viewing, and changing the API resources. Use the `oc api-resources` command to view the Kubernetes resources.



### Note

The `oc` commands in the examples are identical to the equivalent `kubectl` commands.

```
[user@host ~]$ oc api-resources
NAME           SHORTNAMES   APIVERSION   NAMESPACED   KIND
bindings       v1          true          Binding
componentstatuses   cs          v1          false        ComponentStatus
configmaps     cm          v1          true          ConfigMap
endpoints      ep          v1          true          Endpoints
...output omitted...
daemonsets     ds          apps/v1      true          DaemonSet
deployments    deploy      apps/v1      true          Deployment
replicasets    rs          apps/v1      true          ReplicaSet
statefulsets   sts         apps/v1      true          StatefulSet
...output omitted...
```

The `SHORTNAME` for a component helps to minimize typing long CLI commands. For example, you can use `oc get cm` instead of `oc get configmaps`.

The `APIVERSION` column divides the objects into API groups. The column uses the <API-Group>/<API-Version> format. The API-Group object is blank for Kubernetes core resource objects.

Many Kubernetes resources exist within the context of a Kubernetes namespace. Kubernetes namespaces and OpenShift projects are broadly similar. A 1:1 relationship always exists between a namespace and an OpenShift project.

The `KIND` is the formal Kubernetes resource schema type.

The `oc api-resources` command can further filter the output with options that operate on the data.

## The `api-resources` Command Options

Option Example	Description
<code>--namespaced=true</code>	If false, return non-namespaced resources, otherwise return namespaced resources
<code>--api-group apps</code>	Limit to resources in the specified API group. Use <code>--api-group=''</code> to show core resources.
<code>--sort-by name</code>	If non-empty, sort list of resources using specified field. The field can be either 'name' or 'kind'.

For example, use the following `oc api-resources` command to see all the namespaced resources in the `apps` API group, sorted by name.

```
[user@host ~]$ oc api-resources --namespaced=true --api-group apps --sort-by name
NAME           SHORTNAMES   APIVERSION   NAMESPACED   KIND
controllerrevisions   apps/v1       true         ControllerRevision
daemonsets        ds          apps/v1       true         DaemonSet
deployments        deploy      apps/v1       true         Deployment
replicasets        rs          apps/v1       true         ReplicaSet
statefulsets       sts         apps/v1       true         StatefulSet
```

Each resource contains fields that identify the resource or that describe the intended configuration of the resource. Use the `oc explain` command to get information about valid fields for an object. For example, execute the `oc explain pod` command to get information about possible pod object fields.

```
[user@host ~]$ oc explain pod
KIND:     Pod
VERSION:  v1

DESCRIPTION:
Pod is a collection of containers that can run on a host. This resource is
created by clients and scheduled onto hosts.

FIELDS:
apiVersion <string>
    APIVersion defines the versioned schema of this representation of an
    ...
kind <string>
    Kind is a string value representing the REST resource this object
    ...
metadata <ObjectMeta>
    Standard object's metadata. More info:
    ...
spec <PodSpec>
    Specification of the desired behavior of the pod. More info:
    ...output omitted...
```

Every Kubernetes resource contains the `kind`, `apiVersion`, `spec`, and `status` fields. However, when you create an object definition, you do not need to provide the `status` field. Instead, Kubernetes generates the `status` field, and it lists information such as runtime status and readiness. The `status` field is useful for troubleshooting an error or for verifying the current state of a resource.

You can use the YAML path to a field and dot-notation to get information about a particular field. For example, the following `oc explain` command shows details for the `pod.spec` fields.

```
[user@host ~]$ oc explain pod.spec
KIND:     Pod
VERSION:  v1

FIELD: spec <PodSpec>

DESCRIPTION:
  Specification of the desired behavior of the pod. More info:
  https://git.k8s.io/community/contributors/devel/sig-architecture/api-
  conventions.md#spec-and-status

  PodSpec is a description of a pod.

FIELDS:
  activeDeadlineSeconds <integer>
    Optional duration in seconds the pod may be active on the node relative to
    ...output omitted...
```

The following Kubernetes main resource types can be created and configured by using a YAML or a JSON manifest file, or by using OpenShift management tools:

#### **Pods (pod)**

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

#### **Services (svc)**

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

#### **ReplicaSet (rs)**

Ensure that a specified number of pod replicas are running at any given time.

#### **Persistent Volumes (pv)**

Define storage areas for Kubernetes pods to use.

#### **Persistent Volume Claims (pvc)**

Represent a request for storage by a pod. PVCs link a PV to a pod so that its containers can use the provisioned storage, usually by mounting the storage into the container's file system.

#### **ConfigMaps (cm) and Secrets**

Contain a set of keys and values that other resources can use. ConfigMaps and Secrets centralize configuration values that several resources use. Secrets differ from ConfigMaps in that the values of Secrets are always encoded (not encrypted), and their access is restricted to fewer authorized users.

#### **Deployment (deploy)**

A representation of a set of containers that are included in a pod, and the deployment strategies to use. A `deployment` object contains the configuration to apply to all containers

of each pod replica, such as the base image, tags, storage definitions, and the commands to execute when the containers start. Although Kubernetes replicas can be created stand-alone in OpenShift, they are usually created by higher-level resources such as deployment controllers.

Red Hat OpenShift Container Platform (RHOC) adds the following main resource types to Kubernetes:

### **BuildConfig (bc)**

Defines a process to execute in the OpenShift project. The OpenShift Source-to-Image (S2I) feature uses a BuildConfig to build a container image from application source code that is stored in a Git repository. A bc works together with a dc to provide an extensible continuous integration and continuous delivery workflows.

### **DeploymentConfig (dc)**

OpenShift 4.5 introduced the Deployment resource concept to replace the DeploymentConfig default configuration for pods. Both concepts represent a set of containers that are included in a pod, and the deployment strategies to use.

The Deployment object serves as the improved version of the DeploymentConfig object. Some functional replacements between both objects are as follows:

- Deployment objects no longer support automatic rollback or lifecycle hooks.
- Every change in the pod template that Deployment objects use triggers a new rollout automatically.
- The deployment process of a Deployment object can be paused at any time without affecting the deployer process.
- A Deployment object can have as many active replica sets as the user wants, and can scale down previous replicas. In contrast, the DeploymentConfig object can have only two active replication sets at a time.

Although Deployment objects are the default replacement of the deprecated DeploymentConfig objects in OpenShift 4.12 and later versions, you can still use DeploymentConfig objects if you need a specific feature that they provide, but they are not recommended for new installations. In this case, you must specify the type of object when creating a new application by including the `--as-deployment-config` flag.

### **Routes**

Represent a DNS hostname that the OpenShift router recognizes as an ingress point for applications and microservices.

## **Structure of Resources**

Almost every Kubernetes object includes two nested object fields that govern the object's configuration: the object spec and the object status. The spec object describes the intended state of the resource, and the status object describes the current state. You specify the spec section of the resource when you create the object. Kubernetes controllers continuously update the status of the object throughout the existence of the object. The Kubernetes control plane continuously and actively manages every object's actual state to match the desired state you supplied.

The status field uses a collection of condition resource objects with the following fields.

## Condition Resource Fields

Field	Example	Description
Type	ContainersReady	The type of the condition
Status	False	The state of the condition
Reason	RequirementsNotMet	An optional field to provide extra information
Message	2/3 containers are running	An optional textual description for the condition
LastTransitionTime	2023-03-07T18:05:28Z	The last time that conditions were changed

For example, in Kubernetes, a `Deployment` object can represent an application that is running on your cluster. When you create a `Deployment` object, you might configure the deployment spec object to specify that you want three replicas of the application to be running. Kubernetes reads the deployment spec object and starts three instances of your chosen application, and updates the `status` field to match your spec object. If any of those instances fails, then Kubernetes responds to the difference between the spec and status objects by making a correction, in this case to start a replacement instance.

Other common fields provide base information in addition to the `spec` and `status` fields of a Kubernetes object.

## API Resource Fields

Field	Description
<code>apiVersion</code>	Identifier of the object schema version.
<code>kind</code>	Schema identifier.
<code>metadata.name</code>	Creates a label with a <code>name</code> key that other resources in Kubernetes can use to find it.
<code>metadata.namespace</code>	The namespace, or the RHOCP project where the resource is.
<code>metadata.labels</code>	Key-value pairs that can connect identifying metadata with Kubernetes objects.

Resources in Kubernetes consist of multiple objects. These objects define the intended state of a resource. When you create or modify an object, you make a persistent record of the intended state. Kubernetes reads the object and modifies the current state accordingly.

All RHOCP and Kubernetes objects can be represented as a JSON or YAML structure. Consider the following pod object in the YAML format:

```

apiVersion: v1 1
kind: Pod 2
metadata: 3
  name: wildfly 4

```

```

namespace: my_app ⑤
labels:
  name: wildfly ⑥
spec: ⑦
  containers:
    - resources:
        limits:
          cpu: 0.5
    image: quay.io/example/todojee:v1 ⑧
    name: wildfly ⑨
    ports:
      - containerPort: 8080 ⑩
        name: wildfly
    env: ⑪
      - name: MYSQL_DATABASE
        value: items
      - name: MYSQL_USER
        value: user1
      - name: MYSQL_PASSWORD
        value: mypa55
...object omitted...
status: ⑫
  conditions:
    - lastProbeTime: null
      lastTransitionTime: "2023-08-19T12:59:22Z"
      status: "True"
      type: PodScheduled

```

- ① Identifier of the object schema version.
- ② Schema identifier. In this example, the object conforms to the pod schema.
- ③ Metadata for a given resource, such as annotations, labels, name, and namespace.
- ④ A unique name for a pod in Kubernetes that enables administrators to run commands on it.
- ⑤ The namespace, or the RHOCUP project that the resource resides in.
- ⑥ Creates a label with a `name` key that other resources in Kubernetes, usually a service, can use to find it.
- ⑦ Defines the pod object configuration, or the intended state of the resource.
- ⑧ Defines the container image name.
- ⑨ Name of the container inside a pod. Container names are important for `oc` commands when a pod contains multiple containers.
- ⑩ A container-dependent attribute to identify the port that the container uses.
- ⑪ Defines a collection of environment variables.
- ⑫ Current state of the object. Kubernetes provides this field, which lists information such as runtime status, readiness, and container images.

Labels are key-value pairs that you define in the `.metadata.labels` object path, for example:

```

kind: Pod
apiVersion: v1
metadata:
  name: example-pod
  labels:
    app: example-pod
    group: developers
...object omitted...

```

The preceding example contains the `app=example-pod` and `group=developers` labels. Developers often use labels to target a set of objects by using the `-l` or the `--selector` option. For example, the following `oc get` command lists pods that contain the `group=developers` label:

```
[user@host ~]$ oc get pod --selector group=developers
NAME           READY   STATUS    RESTARTS   AGE
example-pod-6c9f758574-7fhg   1/1     Running   5          11d
```

## Command Outputs

The `kubectl` and `oc` CLI commands provide many output formatting options. By default, many commands display a small subset of the most useful fields for the given resource type in a tabular output. Many commands support a `-o wide` option that shows additional fields.

### Tabular Fields

<code>oc get pods</code>	<code>oc get pods -o wide</code>	Example value
NAME	NAME	example-pod
READY	READY	1/1
STATUS	STATUS	Running
RESTARTS	RESTARTS	5
AGE	AGE	11d
	IP	10.8.0.60
	NODE	master01
	NOMINATED NODE	<none>
	READINESS GATES	<none>

To view all the fields that are associated with a resource, the `describe` subcommand shows a detailed description of the selected resource and related resources. You can select a single object by name, or all objects of that type, or provide a name prefix, or a label selector.

For example, the following command first looks for an exact match on the `TYPE` object and the `NAME-PREFIX` object. If no such resource exists, then the command outputs details for every resource of that type with a name with a `NAME_PREFIX` prefix.

```
[user@host ~]$ oc describe TYPE NAME-PREFIX
```

The `describe` subcommand provides detailed human-readable output. However, the format of the `describe` output might change between versions, and thus is not recommended for script development. Any scripts that rely on the output of the `describe` subcommand might break after a version update.

Kubernetes provides YAML and JSON-formatted output options that are suitable for parsing or scripting.

## YAML Output

The `-o yaml` option provides a YAML-formatted output that is parseable and still human-readable.

```
[user@host ~]$ oc get pods -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Pod
  metadata:
    annotations:
...object omitted...
```



### Note

The reference documentation provides a more detailed introduction to YAML.

You can use any tool that can process YAML documents to filter the YAML output for your chosen field. For example, you can use the `yq` tool at <https://mikefarah.gitbook.io/yq/> to process YAML and JSON files.

The `yq` processor uses a dot notation to separate field names in a query. The following example pipes the YAML output to the `yq` command to parse the `podIP` field.

```
[user@host ~]$ oc get pods -o yaml | yq r - 'items[0].status.podIP'
10.8.0.60
```

The `[0]` in the example specifies the first index in the `items` array.



### Note

The lab environment includes version 3.3.0 of the `yq` command, which these examples use. Later versions of the `yq` command introduce incompatibilities with earlier versions. The content in this course might not work with other versions.



### Note

Another tool named `yq` is at <https://kislyuk.github.io/yq/>. The two `yq` tools are not compatible; commands that are designed for one of them do not work with the other.

## JSON Output

Kubernetes uses the JSON format internally to process resource objects. Use the `-o json` option to view a resource in the JSON format.

```
[user@host ~]$ oc get pods -o json
{
  "apiVersion": "v1",
  "items": [
    {
      "apiVersion": "v1",
      "kind": "Pod",
      "metadata": {
        "annotations": {
```

You can use other tools to process JSON documents, such as the `jq` tool at <https://jqlang.github.io/jq/>. Similar to the `yq` processor, use the `jq` processor and dot notation on the fields to query specific information from the JSON-formatted output.

```
[user@host ~]$ oc get pods -o json | jq '.items[0].status.podIP'
"10.8.0.60"
```

Alternatively, the example might have used `.items[].status.podIP` for the query string. The empty brackets instruct the `jq` tool to query all items.

## Custom Output

Kubernetes provides a custom output format that combines the convenience of extracting data via `jq` styled queries with a tabular output format. Use the `-o custom-columns` option with comma-separated `<column name> : <jq query string>` pairs.

```
[user@host ~]$ oc get pods \
-o custom-columns=PodName:".metadata.name", \
ContainerName:"spec.containers[].name", \
Phase:"status.phase", \
IP:"status.podIP", \
Ports:"spec.containers[].ports[].containerPort"
PodName          ContainerName   Phase       IP           Ports
myapp-77fb5cd997-xplhz  myapp        Running   10.8.0.60  <none>
```

Kubernetes also supports the use of JSONPath expressions. JSONPath is a query language for JSON. JSONPath expressions refer to a JSON data structure; they filter and extract formatted fields from JSON objects.

In the following example, the JSONPath expression uses the `range` operator to iterate over the list of pods to extract the name of the pod, its IP address, and the assigned ports.

```
[user@host ~]$ oc get pods \
-o jsonpath='{range .items[]}{\"Pod Name: \"}{.metadata.name}
{\"IP: \"}{.status.podIP}
{\"Ports: \"}{.spec.containers[].ports[].containerPort}{\"\\n\"}{end}'
Pod Name: myapp-77fb5cd997-xplhz
IP: 10.8.0.60
Ports:
```

You can customize the format of the output with Go templates, which the Go programming language uses. Use the `-o go-template` option followed by a Go template, where Go expressions are inside double braces, `{}{}`.

```
[user@host ~]$ oc get pods \
-o go-template='{{range .items}}{{.metadata.name}}\n{{end}}'
myapp-77fb5cd997-xplhz
```



## References

For more information, refer to the *OpenShift CLI (oc)* 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#cli-using-cli\\_cli-developer-commands](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#cli-using-cli_cli-developer-commands)

For more information about custom columns, refer to the *oc get* section 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#cli-using-cli\\_cli-developer-commands](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/cli_tools/index#cli-using-cli_cli-developer-commands)

For more detailed introduction to YAML, see the *YAML in a Nutshell* chapter in the Red Hat Enterprise Linux Atomic Host 7 *Getting Started with Kubernetes* documentation at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html-single/getting\\_started\\_with\\_kubernetes/index#yaml\\_in\\_a\\_nutshell](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_kubernetes/index#yaml_in_a_nutshell)

Labels and selector details are available in the *Working with Kubernetes Objects* section of the

### Kubernetes Documentation - Labels and Selectors

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

Kubernetes object details are available in the *Understanding Kubernetes Objects* section of the

### Kubernetes Documentation - Understanding Kubernetes Objects

<https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>

Command output details are available in the *Get* section of the

### Kubernetes Documentation - Getting Started

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

For more information on JSONPath operators and syntax, see the

### Kubernetes Documentation - JSONPath Support

<https://kubernetes.io/docs/reference/kubectl/jsonpath/>  
documentation.

For more information about Go templates and syntax, see the

### Go Templates

<https://pkg.go.dev/text/template>  
documentation.

## ► Guided Exercise

# Inspect Kubernetes Resources

Verify the state of an OpenShift cluster by querying its recognized resource types, their schemas, and extracting information from Kubernetes resources that are related to OpenShift cluster services.

### Outcomes

- List and explain the supported API resources for a cluster.
- Identify resources from specific API groups.
- Format command outputs in the YAML and JSON formats.
- Use filters to parse command outputs.
- Use JSONPath and custom columns to extract information from resources.

### Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command ensures that the cluster is accessible and that all resources are available for this exercise. It also creates a `myapp` application in the `cli-resources` project.

```
[student@workstation ~]$ lab start cli-resources
```

### Instructions

- 1. Log in to the OpenShift cluster as the developer user with the developer password. Select the `cli-resources` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `cli-resources` project as the active project.

```
[student@workstation ~]$ oc project cli-resources
...output omitted...
```

- 2. List the available cluster resource types with the `api-resources` command. Then, use filters to list namespaced and non-namespaced resources.

- 2.1. List the available resource types with the `api-resources` command.

```
[student@workstation ~]$ oc api-resources
NAME           SHORTNAMES   APIVERSION  NAMESPACED   KIND
bindings       v1          true        Binding
componentstatuses   cs         v1          false       ComponentStatus
configmaps      cm         v1          true        ConfigMap
endpoints       ep         v1          true        Endpoints
events          ev         v1          true        Event
limitranges     limits     v1          true        LimitRange
namespaces      ns         v1          false       Namespace
nodes           no         v1          false       Node
persistentvolumeclaims pvc       v1          true        PersistentVolumeClaim
persistentvolumes    pv        v1          false       PersistentVolume
pods            po         v1          true        Pod
...output omitted...
controllerrevisions   ds        apps/v1    true        ControllerRevision
daemonsets      ds         apps/v1    true        DaemonSet
...output omitted...
cronjobs        cj        batch/v1   true        CronJob
jobs            jobs      batch/v1   true        Job
...output omitted...
```

The `api-resources` command prints the supported API resources, including resource names, available shortnames, and the API versions.

You can use the `APIVERSIONS` field to determine which API group provides the resource. The field lists the group followed by the API version of the resource. For example, the `jobs` resource type is provided by the `batch` API group, and `v1` is the API version of the resource.

- 2.2. Use the `--namespaced` option to limit the output of the `api-resources` command to namespaced resources.

Then, determine the number of available namespaced resources. Use the `-o name` option to list the resource names, and then pipe the output to the `wc -l` command.

```
[student@workstation ~]$ oc api-resources --namespaced
NAME           SHORTNAMES   APIVERSION  NAMESPACED   KIND
bindings       v1          true        Binding
configmaps      cm         v1          true        ConfigMap
endpoints       ep         v1          true        Endpoints
events          ev         v1          true        Event
limitranges     limits     v1          true        LimitRange
persistentvolumeclaims pvc       v1          true        PersistentVolumeClaim
pods            po         v1          true        Pod
podtemplates    templates  v1          true        PodTemplate
replicationcontrollers rc        v1          true        ReplicationController
resourcequotas  quota      v1          true        ResourceQuota
secrets         secrets    v1          true        Secret
...output omitted...
[student@workstation ~]$ oc api-resources --namespaced -o name | wc -l
113
```

The cluster has 113 namespaced cluster resource types, such as the pods, deployments, and services resources.

- 2.3. Limit the output of the `api-resources` command to non-namespaced resources.

Then, determine the number of available non-namespaced resources. To list the resource names, use the `-o name` option and then pipe the output to the `wc -l` command.

```
[student@workstation ~]$ oc api-resources --namespaced=false
NAME           SHORTNAMES   APIVERSION   ...
componentstatuses   cs          v1          ...
namespaces        ns          v1          ...
nodes            no          v1          ...
persistentvolumes  pv          v1          ...
mutatingwebhookconfigurations admissionregistration.k8s.io/v1 ...
validatingwebhookconfigurations admissionregistration.k8s.io/v1 ...
customresourcedefinitions      crd,crds    apiextensions.k8s.io/v1 ...
...output omitted...
[student@workstation ~]$ oc api-resources --namespaced=false -o name | wc -l
118
```

The cluster has 118 non-namespaced cluster resource types, such as the nodes, images, and project resources.

- 3. Identify and explain the available cluster resource types that the core API group provides. Then, describe a resource from the core API group in the `cli-resources` project.

- 3.1. Filter the output of the `api-resources` command to show only resources from the core API group. Use the `--api-group` option and set '' as the value.

```
[student@workstation ~]$ oc api-resources --api-group ''
NAME           SHORTNAMES   APIVERSION   NAMESPACED   KIND
bindings       v1          true         Binding
componentstatuses   cs          v1          false        ComponentStatus
configmaps     cm          v1          true         ConfigMap
endpoints      ep          v1          true         Endpoints
events         ev          v1          true         Event
limitranges    limits      v1          true         LimitRange
namespaces     ns          v1          false        Namespace
nodes          no          v1          false        Node
persistentvolumeclaims  pvc         v1          true         PersistentVolumeClaim
persistentvolumes  pv          v1          false        PersistentVolume
pods           po          v1          true         Pod
podtemplates   v1          v1          true         PodTemplate
replicationcontrollers  rc          v1          true         ReplicationController
resourcequotas quota       v1          true         ResourceQuota
secrets        v1          v1          true         Secret
serviceaccounts sa          v1          true         ServiceAccount
services        svc         v1          true         Service
```

The core API group provides several resource types, such as nodes, events, and pods.

- 3.2. Use the `explain` command to list a description and the available fields for the pods resource type.

```
[student@workstation ~]$ oc explain pods
KIND:     Pod
VERSION:  v1

DESCRIPTION:
Pod is a collection of containers that can run on a host. This resource is
created by clients and scheduled onto hosts.

FIELDS:
apiVersion <string>
  APIVersion defines the versioned schema of this representation of an
  object. Servers should convert recognized schemas to the latest internal
  value, and may reject unrecognized values. More info:
...output omitted...
```

- 3.3. List all pods in the `cli-resources` project.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
myapp-54fcfdcd9d7-2h5vx   1/1     Running   0          4m25s
```

A single pod exists in the `cli-resources` project. The pod name might differ in your output.

- 3.4. Use the `describe` command to view the configuration and events for the pod. Specify the pod name from the previous step.

```
[student@workstation ~]$ oc describe pod myapp-54fcfdcd9d7-2h5vx
Name:           myapp-54fcfdcd9d7-2h5vx
Namespace:      cli-resources
...output omitted...
Status:         Running
IP:             10.8.0.127
Controlled By: ReplicaSet/myapp-54fcfdcd9d7
Containers:
  myapp:
    Container ID:  cri-o://e0da...669d
    Image:        registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
    Image ID:     registry.ocp4.example.com:8443/ubi8/
    httpd-24@sha256:91ad...fd83
    ...output omitted...
    Limits:
      cpu:        500m
      memory:    128Mi
    Requests:
      cpu:        500m
      memory:    128Mi
    Environment: <none>
    ...output omitted...
```

Events:				
Type	Reason	Age	From	Message
Normal	Scheduled	10m	default-scheduler	Successfully assigned cli-resources/myapp-54fcfdcd9d7-2h5vx to master01 ...output omitted...

- 3.5. Retrieve the details of the pod in a structured format. Use the `get` command and specify the output as the YAML format. Compare the results of the `describe` command versus the `get` command.

```
[student@workstation ~]$ oc get pod myapp-54fcfdcd9d7-2h5vx -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
  ...output omitted...
  labels:
    app: myapp
    pod-template-hash: 54fcfdcd9d7
  name: myapp-54fcfdcd9d7-2h5vx
  namespace: cli-resources
  ...output omitted...
spec:
  containers:
    - image: registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
      imagePullPolicy: Always
      name: myapp
      resources:
        limits:
          cpu: 500m
          memory: 128Mi
        requests:
          cpu: 500m
          memory: 128Mi
  ...output omitted...
```

Using a structured format with the `get` command provides more details about a resource than the `describe` command.

- 4. Identify and explain the available cluster resource types that the Kubernetes `apps` API group provides. Then, describe a resource from the `apps` API group in the `cli-resources` project.

- 4.1. List the resource types that the `apps` API group provides.

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
controllerrevisions		apps/v1	true	ControllerRevision
daemonsets	ds	apps/v1	true	DaemonSet
deployments	deploy	apps/v1	true	Deployment
replicasets	rs	apps/v1	true	ReplicaSet
statefulsets	sts	apps/v1	true	StatefulSet

- 4.2. Use the `explain` command to list a description and fields for the `deployments` resource type.

```
[student@workstation ~]$ oc explain deployments
GROUP:     apps
KIND:      Deployment
VERSION:   apps/v1

DESCRIPTION:
Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
apiVersion <string>
  APIVersion defines the versioned schema of this representation of an
  object. Servers should convert recognized schemas to the latest internal
  value, and may reject unrecognized values. More info:
...output omitted...
```

- 4.3. Use the `get` command to identify any deployment resources in the `cli-resources` project.

```
[student@workstation ~]$ oc get deploy
NAME      READY    UP-TO-DATE   AVAILABLE   AGE
myapp    1/1       1           1           25m
```

- 4.4. The `myapp` deployment exists in the `cli-resources` project. Use the `get` command and the `-o wide` option to identify the container name and the container image in the deployment.

```
[student@workstation ~]$ oc get deploy myapp -o wide
NAME ... CONTAINERS IMAGES                                     SELECTOR
myapp ... myapp      registry.ocp4.example.com:8443/ubi8/httpd-24:1-215 app=myapp
```

The `myapp` deployment uses the `registry.ocp4.example.com:8443/ubi8/httpd-24:1-215` container image for the `myapp` container.

- 4.5. Describe the `myapp` deployment to view more details about the resource.

```
[student@workstation ~]$ oc describe deployment myapp
Name:                  myapp
Namespace:             cli-resources
CreationTimestamp:     Wed, 01 Mar 2023 18:41:39 -0500
Labels:                my-app
Annotations:           deployment.kubernetes.io/revision: 1
Selector:              app=myapp
Replicas:              1 desired | 1 updated | 1 total | 1 available | 0
                      unavailable
StrategyType:          RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=myapp
  Containers:
```

```

myapp:
  Image:      registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
  Port:       8080
  Host Port: 8080
  Limits:
    cpu:        500m
    memory:     128Mi
  Environment: <none>
  Mounts:      <none>
  Volumes:     <none>
Conditions:
  Type      Status  Reason
  ----      ----   -----
  Available  True    MinimumReplicasAvailable
  Progressing True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  myapp-54fcfdcd9d7 (1/1 replicas created)
Events:
  Type      Reason          Age      From           Message
  ----      ----   -----   ----
  Normal   ScalingReplicaSet 30m     deployment-controller  Scaled up replica set
  myapp-54fcfdcd9d7 to 1

```

- 5. Identify and explain the available cluster resource types that the OpenShift configuration API group provides. Then, describe a resource from the OpenShift configuration API group.

5.1. List the resource types that the OpenShift configuration API group provides.

```

[student@workstation ~]$ oc api-resources --api-group config.openshift.io
NAME           SHORTNAMES   APIVERSION   NAMESPACED   KIND
apiservers
authentications
  Authentication
builds
clusteroperators   co
  ClusterOperator
clusterversions
  ClusterVersion
consoles
dnses
featuregates
imagecontentpolicies
  ImageContentPolicy
imagedigestmirrorsets idms
  ImageDigestMirrorSet
images
imagetagmirrorsets   itms
  ImageTagMirrorSet
infrastructures
  Infrastructure
ingresses
networks
nodes
oauths

```

operatorhub	config.openshift.io/v1	false	OperatorHub
projects	config.openshift.io/v1	false	Project
proxies	config.openshift.io/v1	false	Proxy
schedulers	config.openshift.io/v1	false	Scheduler

The `config.openshift.io` API group provides multiple, non-namespaced resource types.

- 5.2. Use the `explain` command to list a description and fields for the `projects` resource type.

```
[student@workstation ~]$ oc explain projects
GROUP:    project.openshift.io
KIND:     Project
VERSION:  v1

DESCRIPTION:
Projects are the unit of isolation and collaboration in OpenShift. A
project has one or more members, a quota on the resources that the project
may consume, and the security controls on the resources in the project.
Within a project, members may have different roles - project administrators
can set membership, editors can create and manage the resources, and
viewers can see but not access running containers. In a normal cluster
project administrators are not able to alter their quotas - that is
restricted to cluster administrators.

Listing or watching projects will return only projects the user has the
reader role on.
...output omitted...
```

- 5.3. Describe the `cli-resources` project.

```
[student@workstation ~]$ oc describe project cli-resources
Name:   cli-resources
Created: 10 minutes ago
Labels:  kubernetes.io/metadata.name=cli-resources
        pod-security.kubernetes.io/audit=restricted
        pod-security.kubernetes.io/audit-version=v1.24
        pod-security.kubernetes.io/warn=restricted
        pod-security.kubernetes.io/warn-version=v1.24
Annotations: openshift.io/description=
            openshift.io/display-name=
            openshift.io/requester=system:admin
            openshift.io/sa.scc.mcs=s0:c26,c25
            openshift.io/sa.scc.supplemental-groups=1000710000/10000
            openshift.io/sa.scc.uid-range=1000710000/10000
Display Name: <none>
Description: <none>
Status:   Active
Node Selector: <none>
Quota:   <none>
Resource limits: <none>
```

- 5.4. Retrieve more details of the `cli-resources` project. Use the `get` command, and format the output to use JSON.

```
[student@workstation ~]$ oc get project cli-resources -o json
{
  "apiVersion": "project.openshift.io/v1",
  "kind": "Project",
  "metadata": {
    ...output omitted...
    "labels": {
      "kubernetes.io/metadata.name": "cli-resources",
      "pod-security.kubernetes.io/audit": "restricted",
      "pod-security.kubernetes.io/audit-version": "v1.24",
      "pod-security.kubernetes.io/warn": "restricted",
      "pod-security.kubernetes.io/warn-version": "v1.24"
    },
    "name": "cli-resources",
    "resourceVersion": "705313",
    "uid": "53cbbe45-31ea-4b41-93a9-4ba5c2c4c1f3"
  },
  ...output omitted...
  "status": {
    "phase": "Active"
  }
}
```

The `get` command provides additional details, such as the `kind` and `apiVersion` attributes, of the project resource.

- 6. Verify the cluster status by inspecting cluster services. Format command outputs by using filters.

- 6.1. Retrieve the list of pods for the Etcd operator. The Etcd operator is available in the `openshift-etcd` namespace. Specify the namespace with the `--namespace` or `-n` option.

```
[student@workstation ~]$ oc get pods -n openshift-etcd
Error from server (Forbidden): pods is forbidden: User "developer" cannot list
resource "pods" in API group "" in the namespace "openshift-etcd"
```

The `developer` user cannot access resources in the `openshift-etcd` namespace. Regular cluster users, such as the `developer` user, cannot query resources in the `openshift-` namespaces.

Log in as the `admin` user with the `redhatocp` password. Then, retrieve the list of pods in the `openshift-etcd` namespace.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
Login successful
...output omitted...
[student@workstation ~]$ oc get pods -n openshift-etcd
NAME          READY   STATUS    RESTARTS   AGE
etcd-master01  4/4     Running   36         25d
installer-3-master01 0/1     Completed  0          25d
```

- 6.2. Retrieve the image of the `etcd-master01` pod in the `openshift-etcd` namespace. Use filters to limit the output to the `.spec.containers` attribute of the pod to get the first element. Compare the outputs of the JSONPath, the `jq` filters, and the Go template format.

```
[student@workstation ~]$ oc get pods etcd-master01 -n openshift-etcd \
-o=jsonpath='{.spec.containers[0].image}'{"\n"}  
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:24d9...2020
```

```
[student@workstation ~]$ oc get pods -n openshift-etcd etcd-master01 \
-o json | jq .spec.containers[0].image  
"quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:24d9...2020"
```

```
[student@workstation ~]$ oc get pods -n openshift-etcd etcd-master01 \
-o go-template='{{(index .spec.containers 0).image}}'  
quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:24d9b9d9d7fadacbc505c849a1e4b390b2f0fc452ad851b7cce21e8cf
```

- 6.3. Retrieve the condition status of the `prometheus-k8s-0` pod in the `openshift-monitoring` namespace. Configure the output to use the YAML format, and then filter the output with the `yq` filter.

```
[student@workstation ~]$ oc get pods -n openshift-monitoring prometheus-k8s-0 \
-o yaml | yq r - 'status.conditions'  
- lastProbeTime: null  
lastTransitionTime: "2023-12-12T18:07:17Z"  
status: "True"  
type: Initialized  
- lastProbeTime: null  
lastTransitionTime: "2023-12-15T18:07:45Z"  
status: "True"  
type: Ready  
- lastProbeTime: null  
lastTransitionTime: "2023-12-15T18:07:45Z"  
status: "True"  
type: ContainersReady  
- lastProbeTime: null  
lastTransitionTime: "2023-12-12T22:39:52Z"  
status: "True"  
type: PodScheduled
```

The `r -` option tells the `yq` command to read the standard input (STDIN) for the YAML output of the `get` command.

- 6.4. Use the `get` command to retrieve detailed information for the pods in the `openshift-storage` namespace. Use the YAML format and custom columns to filter the output according to the following table:

Column title	Object
PodName	metadata.name
ContainerName	spec.containers[].name
Phase	status.phase
IP	status.podIP
Ports	spec.containers[].ports[].containerPort

```
[student@workstation ~]$ oc get pods -n openshift-storage \
-o custom-columns=PodName:".metadata.name",\
ContainerName:"spec.containers[].name",\
Phase:"status.phase",\
IP:"status.podIP",\
Ports:"spec.containers[].ports[].containerPort"
PodName          ContainerName      Phase   IP        Ports
lvms-operator-7fcf897cb-... manager    Running 10.8.0.97  9443
topolvm-controller-.... topolvm-controller Running 10.8.0.98  9808
topolvm-node-9spzf       lvmd       Running 10.8.0.100 <none>
vg-manager-z8g5k         vg-manager  Running 10.8.0.101 <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 cli-resources
```

# Assess the Health of an OpenShift Cluster

---

## Objectives

- Query the health of essential cluster services and components.

## Query Operator Conditions

Operators are important components of Red Hat OpenShift Container Platform (RHOCP). Operators automate the required tasks to maintain a healthy RHOCP cluster that would otherwise require human intervention. Operators are the preferred method of packaging, deploying, and managing services on the control plane.

Operators integrate with Kubernetes APIs and CLI tools such as `kubectl` and `oc` commands. Operators provide the means of monitoring applications, performing health checks, managing over-the-air (OTA) updates, and ensuring that applications remain in your specified state.

Because CRI-O and the Kubelet run on every node, almost every other cluster function can be managed on the control plane by using Operators. Components that are added to the control plane by using operators include critical networking and credential services.

Operators in RHOCP are managed by two different systems, depending on the purpose of the operator.

### Cluster Version Operator (CVO)

Cluster operators perform cluster functions. These operators are installed by default, and the CVO manages them.

Cluster operators use a Kubernetes `kind` value of `clusteroperators`, and thus can be queried via `oc` or `kubectl` commands. As a user with the `cluster-admin` role, use the `oc get clusteroperators` command to list all the cluster operators.

[user@host ~]\$ oc get clusteroperators						
NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE	
<b>MESSAGE</b>						
authentication	4.14.0	True	False	False	3d1h	
baremetal	4.14.0	True	False	False	38d	
cloud-controller-manager	4.14.0	True	False	False	38d	
cloud-credential	4.14.0	True	False	False	38d	
cluster-autoscaler	4.14.0	True	False	False	38d	
config-operator	4.14.0	True	False	False	38d	
console	4.14.0	True	False	False	38d	
<i>...output omitted...</i>						

For more details about a cluster operator, use the `describe clusteroperators operator-name` command to view the field values that are associated with the operator, including the current status of the operator. The `describe` command provides a human-readable output format for a resource. As such, the output format might change with an RHOCP version update.

For an output format that is less likely to change with a version update, use one of the `-o` output options of the `get` command. For example, use the following `oc get clusteroperators dns -o yaml` command for the YAML-formatted output details for the dns operator.

```
[user@host ~]$ oc get clusteroperators dns -o yaml
apiVersion: config.openshift.io/v1
kind: ClusterOperator
metadata:
  annotations:
    ...output omitted...
  status:
    conditions:
      - lastTransitionTime: "2023-03-20T13:55:21Z"
        message: DNS "default" is available.
        reason: AsExpected
        status: "True"
        type: Available
    ...output omitted...
    relatedObjects:
      - group: ""
        name: openshift-dns-operator
        resource: namespaces
    ...output omitted...
    versions:
      - name: operator
        version: 4.14.0
    ...output omitted...
```

### Operator Lifecycle Manager (OLM) Operators

Optional add-on operators that the OLM manages can be made accessible for users to run in their applications.

As a user with the `cluster-admin` role, use the `get operators` command to list all the add-on operators.

```
[user@host~]$ oc get operators
NAME                                AGE
lvms-operator.openshift-storage     34d
metallb-operator.metallb-system     34d
```

You can likewise use the `describe` and `get` commands to query details about the fields that are associated with the add-on operators.

Operators use one or more pods to provide cluster services. You can find the namespaces for these pods under the `relatedObjects` section of the detailed output for the operator. As a user with a `cluster-admin` role, use the `-n namespace` option on the `get pod` command to view the pods. For example, use the following `get pods` command to retrieve the list of pods in the `openshift-dns-operator` namespace.

```
[user@host~]$ oc get pods -n openshift-dns-operator
NAME          READY   STATUS    RESTARTS   AGE
dns-operator-64688bfdd4-8zklh   2/2     Running   38        38d
```

Use the `-o yaml` or `-o json` output formats to view or analyze more details about the pods. The resource conditions, which are found in the status for the resource, track the current state of the resource object. The following example uses the `jq` processor to extract the `status` values from the JSON output details for the dns pod.

```
[user@host-]$ oc get pod -n openshift-dns-operator \
dns-operator-64688bfdd4-8zklh -o json | jq .status
{
  "conditions": [
    {
      "lastProbeTime": null,
      "lastTransitionTime": "2023-02-09T21:24:50Z",
      "status": "True",
      "type": "Initialized"
    },
    ...
  ...output omitted...
```

In addition to listing the pods of a namespace, you can also use the `--show-labels` option of the `get` command to print the labels used by the pods. The following example retrieves the pods and their labels in the `openshift-etcd` namespace.

```
[user@host-]$ oc get pods -n openshift-etcd --show-labels
NAME           READY   STATUS    RESTARTS   AGE   LABELS
etcd-master01  4/4     Running   68         35d
  app=etcd,etcd=true,k8s-app=etcd,revision=3
installer-3-master01  0/1     Completed   0         35d   app=installer
```

## Examining Cluster Metrics

Another way to gauge the health of an RHOC cluster is to examine the compute resource usage of cluster nodes and pods. The `oc adm top` command provides this information. For example, to list the total memory and CPU usage of all pods in the cluster, you can use the `--sum` option with the command to print the sum of the resource usage.

```
[user@host-]$ oc adm top pods -A --sum
NAMESPACE          NAME                CPU(cores)   MEMORY(bytes)
metallb-system    controller-....-ddr8v  0m          57Mi
metallb-system    metallb-....-n2zsv   0m          48Mi
...output omitted...
openshift-storage topolvm-node-9spzf  0m          68Mi
openshift-storage vg-manager-z8g5k    0m          23Mi
-----           -----
                           428m        10933Mi
```

The `-A` option shows pods from all namespaces. Use the `-n namespace` option to filter the results to show the pods in a single namespace. Use the `--containers` option to display the resource usage of containers within a pod. For example, use the following command to list the resource usage of the containers in the `etcd-master01` pod in the `openshift-etcd` namespace.

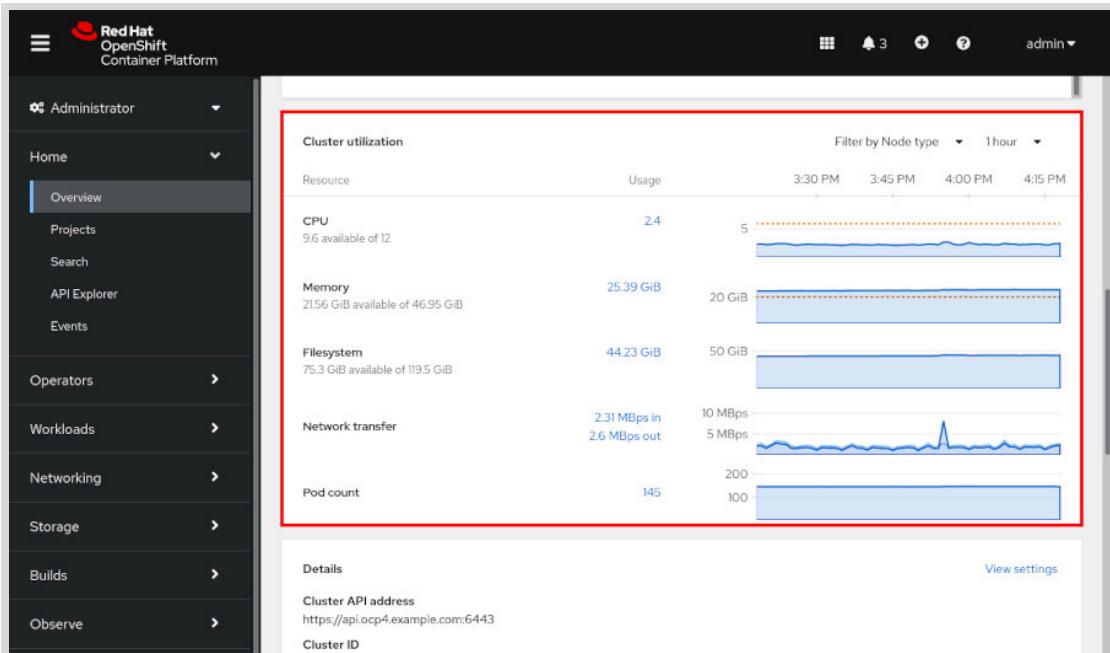
```
[user@host~]$ oc adm top pods etcd-master01 -n openshift-etcd --containers
POD                  NAME          CPU(cores)   MEMORY(bytes)
etcd-master01        POD           0m          0Mi
etcd-master01        etcd          71m         933Mi
etcd-master01        etcd-metrics  6m          32Mi
etcd-master01        etcd-readyz  4m          66Mi
etcd-master01        etcdctl       0m          0Mi
```

## Viewing Cluster Metrics

The OpenShift web console incorporates graphs to visualize cluster and resource analytics. Cluster administrators and users with either the `view` or the `cluster-monitoring-view` cluster role can access the **Home > Overview** page. The Overview page displays a collection of cluster-wide metrics, and provides a high-level view of the overall health of the cluster.

The Overview page displays the following metrics:

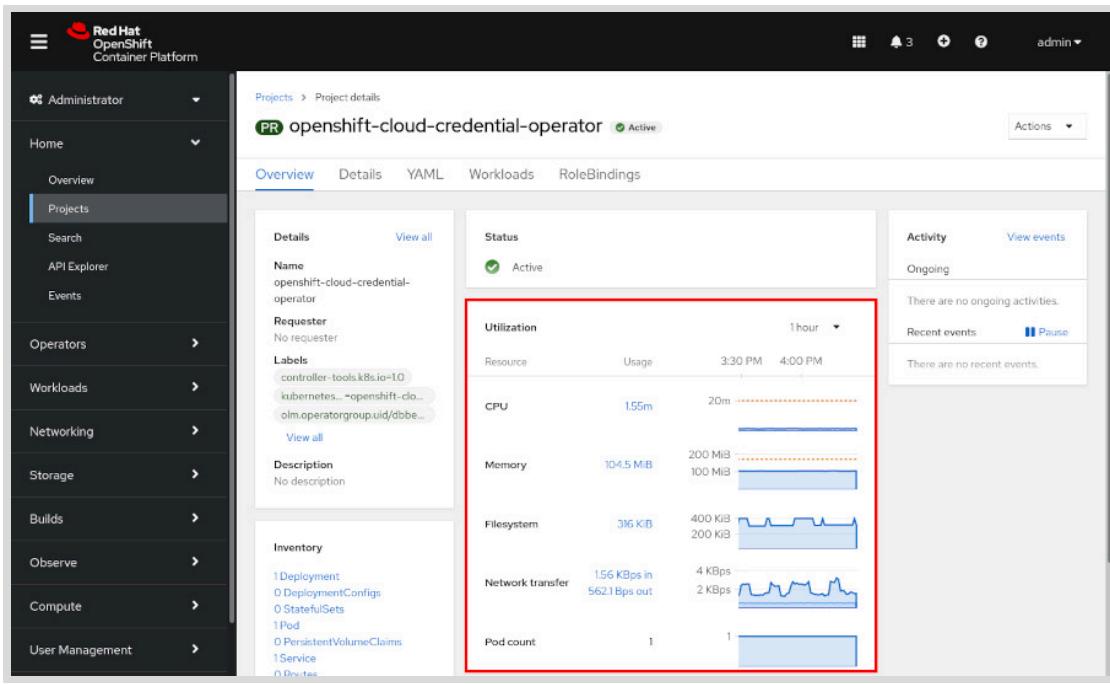
- Current cluster capacity based on CPU, memory, storage, and network usage
- A time-series graph of total CPU, memory, and disk usage
- The ability to display the top consumers of CPU, memory, and storage



For any of the listed resources in the **Cluster Utilization** section, administrators can click the link for current resource usage. The link displays a window with a breakdown of top consumers for that resource. Top consumers can be sorted by project, by pod, or by node. The list of top consumers can be useful for identifying problematic pods or nodes. For example, a pod with an unexpected memory leak might appear at the top of the list.

## Viewing Project Metrics

The **Project Details** page displays metrics that provide an overview of the resources that are used within the scope of a specific project. The **Utilization** section displays usage information about resources, such as CPU and memory, along with the ability to display the top consumers for each resource.

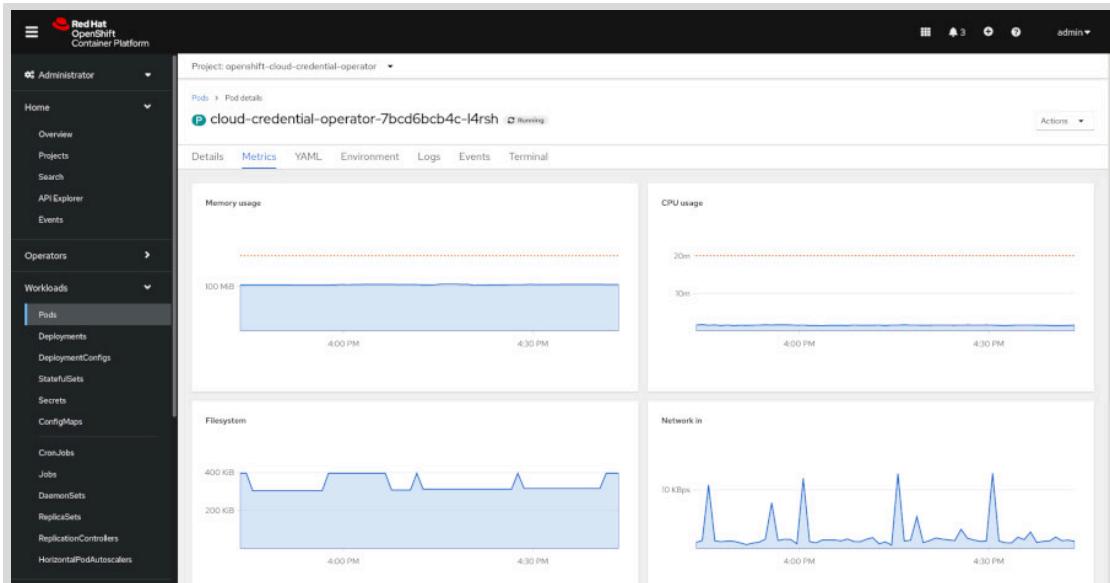


All metrics are pulled from Prometheus. Click any graph to navigate to the **Metrics** page. View the executed query, and inspect the data further.

If a resource quota is created for the project, then the current project request and limits appear on the **Project Details** page.

## Viewing Resource Metrics

When troubleshooting, it is often useful to view metrics at a smaller granularity than for the entire cluster or project. The **Pod Details** page displays time-series graphs of the CPU, memory, and file system usage for a specific pod. A sudden change in these critical metrics, such as a CPU spike caused by high load, is visible on this page.

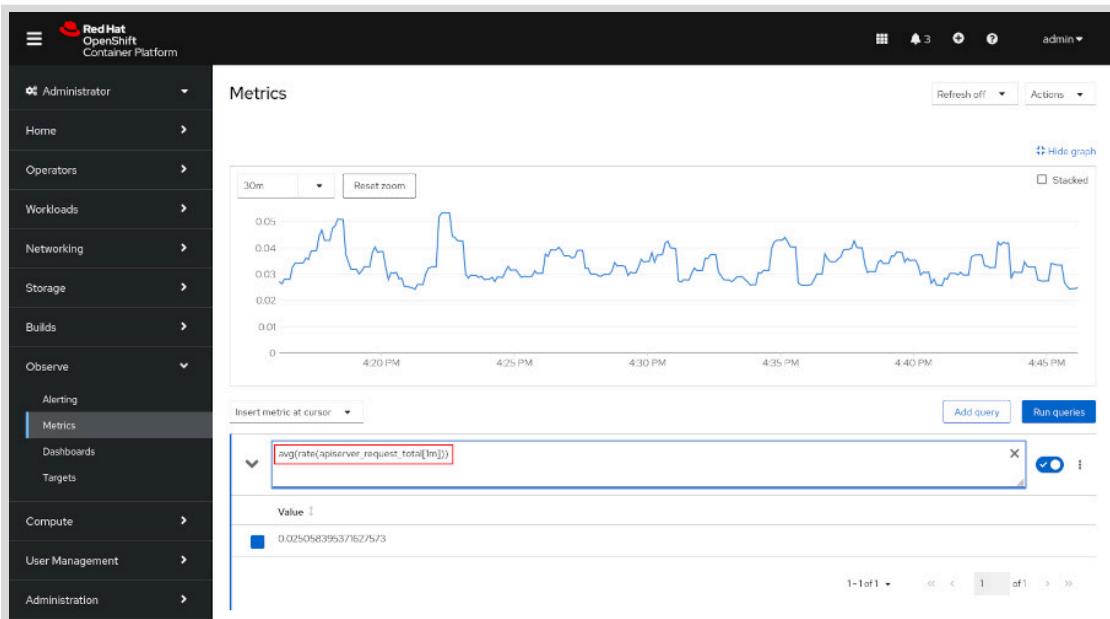


**Figure 2.9: Time-series graphs showing various metrics for a pod**

## Performing Prometheus Queries in the Web Console

The Prometheus UI is a feature-rich tool for visualizing metrics and configuring alerts. The OpenShift web console provides an interface for executing Prometheus queries directly from the web console.

To perform a query, navigate to **Observe > Metrics**, enter a Prometheus Query Language expression in the text field, and click **Run Queries**. The results of the query are displayed as a time-series graph:



**Figure 2.10: Using a Prometheus query to display a time-series graph**



### Note

The Prometheus Query Language is not discussed in detail in this course. Refer to the references section for a link to the official documentation.

## Query Cluster Events and Alerts

Some developers consider OpenShift logs to be too low-level, thus making troubleshooting difficult. Fortunately, RHOCP provides a high-level logging and auditing facility called events. Kubernetes generates event objects in response to state changes in cluster objects, such as nodes, pods, and containers. Events signal significant actions, such as starting a container or destroying a pod.

To read events, use the `get events` command. The command lists the events for the current RHOCP project (namespace). You can display the events for a different project by adding the `-n namespace` option to the command. To list the events for all the projects, use the `-A` (or `--all-namespaces`) option.



### Note

To sort the events by time, add the `--sort-by .metadata.creationTimestamp` option to the `oc get events` command.

The following `get events` command prints events in the `openshift-kube-controller-manager` namespace.

```
[user@host~]$ oc get events -n openshift-kube-controller-manager
LAST SEEN  TYPE      REASON          OBJECT
MESSAGE
12m        Normal    CreatedSCCRanges   pod/kube-controller-manager-master01
created SCC...
```

You can use the `describe pod pod-name` command to further narrow the results to a single pod. For example, to retrieve only the events that relate to a `mysql` pod, you can refer to the `Events` field from the output of the `oc describe pod mysql` command:

```
[user@host~]$ oc describe pod mysql
...output omitted...
Events:
FirstSeen  LastSeen  Count  From           Reason          Message
Wed, 10 ... Wed, 10 ... 1    {scheduler } scheduled  Successfully as...
...output omitted...
```

## Kubernetes Alerts

RHOCP includes a monitoring stack, which is based on the Prometheus open source project. The monitoring stack is configured to monitor the core RHOCP cluster components, by default. You can optionally configure the monitoring stack also to monitor user projects.

The components of the monitoring stack are installed in the `openshift-monitoring` namespace. The Prometheus Operator in the `openshift-monitoring` namespace creates, configures, and manages platform Prometheus and Alertmanager instances. An Alertmanager pod in the `openshift-monitoring` namespace receives alerts from Prometheus. Alertmanager can also send alerts to external notification systems.

Use the following `get all` command to display a list of all resources, their status, and their types in the `openshift-monitoring` namespace.

```
[user@host~]$ oc get all -n openshift-monitoring --show-kind
NAME                           READY  STATUS  RESTARTS AGE
pod/alertmanager-main-0         6/6    Running  85       34d
pod/cluster-monitoring-operator-56b769b58f-dtmqj 2/2    Running  34       35d
pod/kube-state-metrics-75455b796c-8q28d     3/3    Running  51       35d
...output omitted...
```

The `alertmanager-main-0` pod is the Alertmanager for the cluster. The following `logs` command shows the logs of the `alertmanager-main-0` pod, which displays the received messages from Prometheus.

```
[user@host~]$ oc logs alertmanager-main-0 -n openshift-monitoring
ts=2023-03-16T14:21:50.479Z caller=main.go:231 level=info msg="Starting
Alertmanager" version="(version=0.24.0, branch=rhaos-4.14-rhel-8,
revision=519ccb87494d2830821a0da0a657af69d852c93b)"
ts=2023-03-16T14:21:50.479Z caller=main.go:232 level=info
build_context="(go=g01.19.4, user=root@232132c11c68, date=20230105-00:26:49)"
ts=2023-03-16T14:21:50.527Z caller=coordinator.go:113 level=info
component=configuration msg="Loading configuration file" file=/etc/alertmanager/
config_out/alertmanager.env.yaml
...output omitted...
```

## Check Node Status

RHOCP clusters can have several components, including at least one control plane and at least one compute node. The two components can occupy a single node. The following `oc` command, or the matching `kubectl` command, can display the overall health of all cluster nodes.

```
[user@host~]$ oc cluster-info
```

The `oc cluster-info` output is high-level, and can verify that the cluster nodes are running. For a more detailed view into the cluster nodes, use the `get nodes` command.

```
[user@host~]$ oc get nodes
NAME      STATUS    ROLES          AGE      VERSION
master01   Ready     control-plane,master,worker   35d      v1.27.6+f67aeb3
```

The example shows a single `master01` node with multiple roles. The `STATUS` value of `Ready` means that this node is healthy and can accept new pods. A `STATUS` value of `NotReady` means that a condition triggered the `NotReady` status and the node is not accepting new pods.

As with any other RHOCP resource, you can drill down into further details of the node resource with the `describe node node-name` command. For parsable output of the same information, use the `-o json` or the `-o yaml` output options with the `get node node-name` command. For more information about using and parsing these output formats, see *Inspect Kubernetes Resources*.

The output of the `get nodes node-name` command with the `-o json` or `-o yaml` option is long. The following examples use the `-jsonpath` option or the `jq` processor to parse the `get node node-name` command output.

```
[user@host~]$ oc get node master01 -o jsonpath=\
*'{"Allocatable:\n"}{.status.allocatable}{`\n\n`}
>{"Capacity:\n"}{.status.capacity}{`\n`}
Allocatable:
{"cpu":"7500m", "ephemeral-storage":"114396791822", "hugepages-1Gi":"0",
"hugepages-2Mi":"0", "memory":"19380692Ki", "pods":"250"}

Capacity:
{"cpu":"8", "ephemeral-storage":"125293548Ki", "hugepages-1Gi":"0",
"hugepages-2Mi":"0", "memory":"20531668Ki", "pods":"250"}
```

The JSONPath expression in the previous command extracts the allocatable and capacity measures for the `master01` node. These measures help to understand the available resources on a node.

View the `status` object of a node to understand the current health of the node.

```
[user@host-]$ oc get node master01 -o json | jq '.status.conditions'
[
  {
    "lastHeartbeatTime": "2023-03-22T16:34:57Z",
    "lastTransitionTime": "2023-02-23T20:35:15Z",
    "message": "kubelet has sufficient memory available",
    "reason": "KubeletHasSufficientMemory",
    "status": "False",
    "type": "MemoryPressure" ①
  },
  {
    "lastHeartbeatTime": "2023-03-22T16:34:57Z",
    "lastTransitionTime": "2023-02-23T20:35:15Z",
    "message": "kubelet has no disk pressure",
    "reason": "KubeletHasNoDiskPressure",
    "status": "False",
    "type": "DiskPressure" ②
  },
  {
    "lastHeartbeatTime": "2023-03-22T16:34:57Z",
    "lastTransitionTime": "2023-02-23T20:35:15Z",
    "message": "kubelet has sufficient PID available",
    "reason": "KubeletHasSufficientPID",
    "status": "False",
    "type": "PIDPressure" ③
  },
  {
    "lastHeartbeatTime": "2023-03-22T16:34:57Z",
    "lastTransitionTime": "2023-02-23T20:35:15Z",
    "message": "kubelet is posting ready status",
    "reason": "KubeletReady",
    "status": "True",
    "type": "Ready" ④
  }
]
```

- ①** If the status of the `MemoryPressure` condition is true, then the node is low on memory.
- ②** If the status of the `DiskPressure` condition is true, then the disk capacity of the node is low.
- ③** If the status of the `PIDPressure` condition is true, then too many processes are running on the node.
- ④** If the status of the `Ready` condition is false, then the node is not healthy and is not accepting pods.

More conditions indicate other potential problems with a node.

## Possible Node Conditions

Condition	Description
OutOfDisk	If true, then the node has insufficient free space on the node for adding new pods.
NetworkUnavailable	If true, then the network for the node is not correctly configured.
NotReady	If true, then one of the underlying components, such as the container runtime or network, is experiencing issues or is not yet configured.
SchedulingDisabled	Pods cannot be scheduled for placement on the node.

To gain deeper insight into a given node, you can view the logs of processes that run on the node. A cluster administrator can use the `oc adm node-logs` command to view node logs. Node logs might contain sensitive output, and thus are limited to privileged node administrators. Use `oc adm node-logs node-name` to filter the logs to a single node.

The `oc adm node-logs` command has other options to further filter the results.

### Filters for `oc adm node-logs`

Option Example	Description
<code>--role master</code>	Use the <code>--role</code> option to filter the output to nodes with a specified role.
<code>-u kubelet</code>	The <code>-u</code> option filters the output to a specified unit.
<code>--path=cron</code>	The <code>--path</code> option filters the output to a specific process under the <code>/var/logs</code> directory.
<code>--tail 1</code>	Use <code>--tail x</code> to limit output to the last <code>x</code> log entries.

Use `oc adm node-logs --help` for a complete list of command options.

For example, to retrieve the most recent log entry for the `crio` service on the `master01` node, you can use the following command.

```
[user@host~]$ oc adm node-logs master01 -u crio --tail 1
-- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Fri 2023-03-17 15:11:43 UTC.
--
Mar 17 06:16:09.519642 master01 crio[2987]: time="2023-03-17 06:16:09.519474755Z"
  level=info msg="Image status:
&ImageStatusResponse{Image:&Image{Id:6ef8...79ce,RepoTags:[],RepoDigests:
...output omitted...
```

When you create a pod with the CLI, the `oc` or `kubectl` command is sent to the `apiserver` service, which then validates the command. The `scheduler` service reads the YAML or JSON pod definition, and then assigns pods to compute nodes. Each compute node runs a `kubelet` service that converts the pod manifest to one or more containers in the CRI-O container runtime.

Each compute node must have an active `kubelet` service and an active `crio` service. To verify the health of these services, first start a debug session on the node by using the `debug` command.

```
[user@host~]$ oc debug node/node-name
```

Replace the `node-name` value with the name of your node.



### Note

The `debug` command is covered in greater detail in a later section.

Within the debug session, change to the `/host` root directory so that you can run binaries in the host's executable path.

```
sh-4.4# chroot /host
```

Then, use the `systemctl is-active` calls to confirm that the services are active.

```
sh-4.4# for SERVICES in kubelet crio; do echo ---- $SERVICES ---- ;  
systemctl is-active $SERVICES ; echo ""; done  
---- kubelet ----  
active  
  
---- crio ----  
active
```

For more details about the status of a service, use the `systemctl status` command.

```
sh-4.4# systemctl status kubelet  
● kubelet.service - Kubernetes Kubelet  
  Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset:  
disabled)  
  Drop-In: /etc/systemd/system/kubelet.service.d  
          └─01-kubens.conf, 10-mco-default-madv.conf, 20-logging.conf, 20-  
nodenet.conf  
    Active: active (running) since Thu 2023-03-23 14:39:11 UTC; 1h 26min ago  
      Main PID: 3215 (kubelet)  
        Tasks: 28 (limit: 127707)  
        Memory: 391.7M  
          CPU: 14min 34.568s  
...output omitted...
```

## Check Pod Status

With RHOCP, you can view logs in running containers and pods to ease troubleshooting. When a container starts, RHOCP redirects the container's standard output and standard error to a disk in the container's ephemeral storage. With this redirect, you can view the container logs by using `logs` commands, even after the container stops. However, the pod hosting the container must still exist.

In RHOCP, the following command returns the output for a container within a pod:

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

Replace *pod-name* with the name of the target pod, and replace *container-name* with the name of the target container. The `-c container-name` argument is optional, if the pod has only one container. You must use the `-c container-name` argument to connect to a specific container in a multicontainer pod. Otherwise, the command defaults to the only running container and returns the output.

When debugging images and setup problems, it is useful to get an exact copy of a running pod configuration, and then troubleshoot it with a shell. If a pod is failing or does not include a shell, then the `rsh` and `exec` commands might not work. To resolve this issue, the `debug` command creates a copy of the specified pod and starts a shell in that pod.

By default, the `debug` command starts a shell inside the first container of the referenced pod. The debug pod is a copy of your source pod, with some additional modifications. For example, the pod labels are removed. The executed command is also changed to the `'/bin/sh'` command for Linux containers, or the `'cmd.exe'` executable for Windows containers. Additionally, readiness and liveness probes are disabled.

A common problem for containers in pods is security policies that prohibit a container from running as a root user. You can use the `debug` command to test running a pod as a non-root user by using the `--as-user` option. You can also run a non-root pod as the root user with the `--as-root` option.

With the `debug` command, you can invoke other types of objects besides pods. For example, you can use any controller resource that creates a pod, such as a deployment, a build, or a job. The `debug` command also works with nodes, and with resources that can create pods, such as image stream tags. You can also use the `--image=IMAGE` option of the `debug` command to start a shell session by using a specified image.

If you do not include a resource type and name, then the `debug` command starts a shell session into a pod by using the OpenShift tools image.

```
[user@host~]$ oc debug
```

The next example tests running a job pod as a non-root user.

```
[user@host~]$ oc debug job/test --as-user=1000000
```

The following example creates a debug session for a node.

```
[user@host~]$ oc debug node/master01
Starting pod/master01-debug-wtn9r ...
To use host binaries, run chroot /host
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-5.1#
```

The debug pod is deleted when the remote command completes, or when the user interrupts the shell.

## Collect Information for Support Requests

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support. It is recommended that you provide the following information:

- Data gathered by using the `oc adm must-gather` command as a cluster administrator
- The unique cluster ID

The `oc adm must-gather` command collects resource definitions and service logs from your cluster that are most likely needed for debugging issues. This command creates a pod in a temporary namespace on your cluster, and the pod then gathers and downloads debugging information. By default, the `oc adm must-gather` command uses the default plug-in image, and writes into the `./must-gather.local` directory on your local system. To write to a specific local directory, you can also use the `--dest-dir` option, such as in the following example:

```
[user@host~]$ oc adm must-gather --dest-dir /home/student/must-gather
```

Then, create a compressed archive file from the `must-gather` directory. For example, on a Linux-based system, you can run the following command:

```
[user@host~]$ tar cvaf mustgather.tar must-gather/
```

Replace `must-gather/` with the actual directory path.

Then, attach the compressed archive file to your support case in the Red Hat Customer Portal.

Similar to the `oc adm must-gather` command, the `oc adm inspect` command gathers information on a specified resource. For example, the following command collects debugging data for the `openshift-apiserver` and `kube-apiserver` cluster operators.

```
[user@host~]$ oc adm inspect clusteroperator/openshift-apiserver \
clusteroperator/kube-apiserver
```

The `oc adm inspect` command can also use the `--dest-dir` option to specify a local directory to write the gathered information. The command shows all logs by default. Use the `--since` option to filter the results to logs that are later than a relative duration, such as 5s, 2m, or 3h.

```
[user@host~]$ oc adm inspect clusteroperator/openshift-apiserver --since 10m
```



## References

For more information, refer to the *Control Plane Architecture* chapter in the Red Hat OpenShift Container Platform 4.14 *Architecture* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/architecture/index#control-plane](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/architecture/index#control-plane)

For more information, refer to the Red Hat OpenShift Container Platform 4.14 *Monitoring* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/monitoring/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/monitoring/index)

For more information, refer to the Red Hat OpenShift Container Platform 4.14 *Nodes* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#working-with-nodes](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#working-with-nodes)

### Querying Prometheus

<https://prometheus.io/docs/prometheus/latest/querying/basics/>

For more information, refer to the *Troubleshooting Kubernetes* chapter in the Red Hat Enterprise Linux Atomic Host 7 *Getting Started with Kubernetes* documentation at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html-single/getting\\_started\\_with\\_kubernetes/index#troubleshooting\\_kubernetes](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_kubernetes/index#troubleshooting_kubernetes)

For more information about gathering diagnostic data about your cluster, refer to the *Gathering data about your cluster* chapter in the Red Hat OpenShift Container Platform 4.14 *Support* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/support/index#gathering-cluster-data](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/support/index#gathering-cluster-data)

## ► Guided Exercise

# Assess the Health of an OpenShift Cluster

Verify the health of an OpenShift cluster by querying the status of its cluster operators, nodes, pods, and systemd services. Also verify cluster events and alerts.

## Outcomes

- View the status and get information about cluster operators.
- Retrieve information about cluster pods and nodes.
- Retrieve the status of a node's systemd services.
- View cluster events and alerts.
- Retrieve debugging information for the cluster.

## 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 cli-health
```

## Instructions

► 1. Retrieve the status and view information about cluster operators.

- 1.1. 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...
```

- 1.2. List the operators that users installed in the OpenShift cluster.

```
[student@workstation ~]$ oc get operators
NAME                                AGE
lvm-operator.openshift-storage      27d
metallb-operator.metallb-system    27d
```

- 1.3. List the cluster operators that are installed by default in the OpenShift cluster.

```
[student@workstation ~]$ oc get clusteroperators
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      ...
cloud-credential 4.14.0   True       False        False      ...
cluster-autoscaler 4.14.0   True       False        False      ...
config-operator 4.14.0   True       False        False      ...
console         4.14.0   True       False        False      ...
control-plane-machine-set 4.14.0   True       False        False      ...
csi-snapshot-controller 4.14.0   True       False        False      ...
dns             4.14.0   True       False        False      ...
etcd            4.14.0   True       False        False      ...
...output omitted...
```

- 1.4. Use the `describe` command to view detailed information about the `openshift-apiserver` cluster operator, such as related objects, events, and version.

```
[student@workstation ~]$ oc describe clusteroperators openshift-apiserver
Name:           openshift-apiserver
Namespace:
Labels:         <none>
Annotations:    exclude.release.openshift.io/internal-openshift-hosted: true
                 include.release.openshift.io/self-managed-high-availability: true
                 include.release.openshift.io/single-node-developer: true
API Version:   config.openshift.io/v1
Kind:          ClusterOperator
Metadata:
...output omitted...
Spec:
Status:
  Conditions:
    Last Transition Time: 2023-02-09T22:41:08Z
    Message:           All is well
    Reason:            AsExpected
    Status:            False
    Type:              Degraded
...output omitted...
  Extension:        <nil>
Related Objects:
  Group:           operator.openshift.io
  Name:            cluster
  Resource:        openshiftapiservers
  Group:
  Name:            openshift-config
  Resource:        namespaces
  Group:
  Name:            openshift-config-managed
  Resource:        namespaces
  Group:
  Name:            openshift-apiserver-operator
  Resource:        namespaces
  Group:
```

```
Name:      openshift-apiserver
Resource:  namespaces
...output omitted...
Versions:
  Name:      operator
  Version:   4.14.0
  Name:      openshift-apiserver
  Version:   4.14.0
Events:    <none>
```

The `Related Objects` attribute includes information about the name, resource type, and groups for objects that are related to the operator.

15. List the pods in the `openshift-apiserver-operator` namespace. Then, view the detailed status of an `openshift-apiserver-operator` pod by using the JSON format and the `jq` command. Your pod names might differ.

```
[student@workstation ~]$ oc get pods -n openshift-apiserver-operator
NAME                  READY   STATUS    RESTARTS   AGE
openshift-apiserver-operator-7ddc8958fb-7m2kr   1/1     Running   11          27d
```

```
[student@workstation ~]$ oc get pod -n openshift-apiserver-operator \
  openshift-apiserver-operator-7ddc8958fb-7m2kr \
  -o json | jq .status
{
  "conditions": [
    ...output omitted...
    {
      "lastProbeTime": null,
      "lastTransitionTime": "2023-03-08T15:41:34Z",
      "status": "True",
      "type": "Ready"
    },
    ...output omitted...
  ],
  "containerStatuses": [
    {
      ...
    }
  ],
  "hostIP": "192.168.50.10",
  "phase": "Running",
```

```

    "podIP": "10.8.0.5",
    ...output omitted...
}

```

► 2. Retrieve the status, resource consumption, and events of cluster pods.

- 2.1. List the memory and CPU usage of all pods in the cluster. Use the `--sum` option to print the sum of the resource usage. The resource usage on your system probably differs.

```
[student@workstation ~]$ oc adm top pods -A --sum
NAMESPACE          NAME                           CPU(cores)
MEMORY(bytes)metallb-system controller-5f6fdf8c4f-ddr8v      0m
39Mi
metallb-system     metallb-operator-controller-manager-... 1m   38Mi
metallb-system     metallb-operator-webhook-server-...   1m   18Mi
metallb-system     speaker-2dds4                   10m  94Mi
...output omitted...
505m               8982Mi
```

- 2.2. List the pods and their labels in the `openshift-etcd` namespace.

```
[student@workstation ~]$ oc get pods -n openshift-etcd --show-labels
NAME        READY  STATUS    RESTARTS AGE   LABELS
etcd-master01 4/4    Running   40       27d   app=etcd,etcd=true,k8s-app=etcd,revision=3
installer-3-master01 0/1    Completed  0       27d   app=installer
```

- 2.3. List the resource usage of the containers in the `etcd-master01` pod in the `openshift-etcd` namespace. The resource usage on your system probably differs.

```
[student@workstation ~]$ oc adm top pods etcd-master01 \
-n openshift-etcd --containers
POD          NAME           CPU(cores)  MEMORY(bytes)
etcd-master01 POD            0m         0Mi
etcd-master01 etcd           57m        1096Mi
etcd-master01 etcd-metrics   7m         20Mi
etcd-master01 etcd-readyz   4m         40Mi
etcd-master01 etcdctl        0m         0Mi
```

- 2.4. Display a list of all resources, their status, and their types in the `openshift-monitoring` namespace.

```
[student@workstation ~]$ oc get all -n openshift-monitoring --show-kind
NAME          READY  STATUS    ... 
pod/alertmanager-main-0  6/6    Running  ...
pod/cluster-monitoring-operator-56b769b58f-dtmqj  2/2    Running  ...
pod/kube-state-metrics-75455b796c-8q28d  3/3    Running  ...
...output omitted...
NAME          TYPE    CLUSTER-IP ...
service/alertmanager-main  ClusterIP  172.30.85.183 ...
service/alertmanager-operated  ClusterIP  None    ...
```

```
service/cluster-monitoring-operator           ClusterIP  None   ...
service/kube-state-metrics                  ClusterIP  None   ...
...output omitted...
```

- 2.5. View the logs of the `alertmanager-main-0` pod in the `openshift-monitoring` namespace. The logs might differ on your system.

```
[student@workstation ~]$ oc logs alertmanager-main-0 -n openshift-monitoring
...output omitted...
ts=2023-03-09T14:57:11.850Z caller=coordinator.go:113 level=info
  component=configuration msg="Loading configuration file" file=/etc/alertmanager/
config_out/alertmanager.env.yaml
ts=2023-03-09T14:57:11.850Z caller=coordinator.go:126 level=info
  component=configuration msg="Completed loading of configuration file" file=/etc/
alertmanager/config_out/alertmanager.env.yaml
```

- 2.6. Retrieve the events for the `openshift-kube-controller-manager` namespace.

```
[student@workstation ~]$ oc get events -n openshift-kube-controller-manager
LAST SEEN    TYPE      REASON          OBJECT
175m        Normal    CreatedSCCRanges pod/kube-controller-manager-master01...
11m         Normal    CreatedSCCRanges pod/kube-controller-manager-master01...
```

► 3. Retrieve information about cluster nodes.

- 3.1. View the status of the nodes in the cluster.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES          AGE     VERSION
master01  Ready     control-plane,master,worker  27d    v1.27.6+f67aeb3
```

- 3.2. Retrieve the resource consumption of the `master01` node. The resource usage on your system probably differs.

```
[student@workstation ~]$ oc adm top node
NAME      CPU(cores)   CPU%    MEMORY(bytes)  MEMORY%
master01  781m        10%    11455Mi       60%
```

- 3.3. Use a JSONPath filter to determine the capacity and allocatable CPU for the `master01` node. The values might differ on your system.

```
[student@workstation ~]$ oc get node master01 -o jsonpath=\
'Allocatable: {.status.allocatable.cpu}{"\n"}' \
'Capacity: {.status.capacity.cpu}{"\n"}'
Allocatable: 7500m
Capacity: 8
```

- 3.4. Determine the number of allocatable pods for the node.

```
[student@workstation ~]$ oc get node master01 -o jsonpath=\
' {.status.allocatable.pods}{ "\n"}'
250
```

- 3.5. Use the `describe` command to view the events, resource requests, and resource limits for the node. The output might differ on your system.

```
[student@workstation ~]$ oc describe node master01
...output omitted...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
Resource          Requests      Limits
-----          -----
cpu                3158m (42%)   980m (13%)
memory            12667Mi (66%)  1250Mi (6%)
ephemeral-storage  0 (0%)       0 (0%)
hugepages-1Gi     0 (0%)       0 (0%)
hugepages-2Mi     0 (0%)       0 (0%)
Events:
Type    Reason           Age           From        Message
----    -----           ---           ----        -----
Normal  Starting         106m          kubelet    Starting
kubelet.
Normal  NodeHasSufficientMemory  106m (x9 over 106m)  kubelet  Node master01
status is now: NodeHasSufficientMemory
Normal  NodeHasNoDiskPressure   106m (x7 over 106m)  kubelet  Node master01
status is now: NodeHasNoDiskPressure
Normal  NodeHasSufficientPID    106m (x7 over 106m)  kubelet  Node master01
status is now: NodeHasSufficientPID
...output omitted...
```

► 4. Retrieve the logs and status of the systemd services on the `master01` node.

- 4.1. Display the logs of the node. Filter the logs to show the most recent log for the `crio` service. The logs might differ on your system.

```
[student@workstation ~]$ oc adm node-logs master01 -u crio --tail 1
-- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Thu 2023-03-09 16:57:00 UTC.
--
Mar 09 02:39:29.158989 master01 crio[3201]: time="2023-03-09 02:39:29.158737393Z"
  level=info msg="Image status: &ImageStatusResponse
...output omitted...
```

- 4.2. Display the two most recent logs of the `kubelet` service on the node. The logs might differ on your system.

```
[student@workstation ~]$ oc adm node-logs master01 -u kubelet --tail 2
-- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Thu 2023-03-09 16:59:16 UTC.
--
Mar 09 02:40:57.466711 master01 systemd[1]: Stopped Kubernetes Kubelet.
Mar 09 02:40:57.466835 master01 systemd[1]: kubelet.service: Consumed 1h 27min
8.069s CPU time
-- Logs begin at Thu 2023-02-09 21:19:09 UTC, end at Thu 2023-03-09 16:59:16 UTC.
--
Mar 09 16:58:52.133046 master01 kubenswrapper[3195]: I0309 16:58:52.132866      3195
kubelet_getters.go:182] "Pod status updated" pod="openshift-etcd/etcdb-master01"
status=Running
Mar 09 16:58:52.133046 master01 kubenswrapper[3195]: I0309 16:58:52.132882      3195
kubelet_getters.go:182] "Pod status updated" pod="openshift-kube-apiserver/kube-
apiserver-master01" status=Running
```

- 4.3. Create a debug session for the node. Then, use the `chroot /host` command to access the host binaries.

```
[student@workstation ~]$ oc debug node/master01
Starting pod/master01-debug-khltm ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-5.1#
```

- 4.4. Verify the status of the `kubelet` service.

```
sh-5.1# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet
  Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; preset: disabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
            └─01-kubens.conf, 10-mco-default-madv.conf, 20-logging.conf, 20-
nodenet.conf
    Active: active (running) since Thu 2023-03-09 14:54:51 UTC; 2h 8min ago
      Main PID: 3195 (kubelet)
        Tasks: 28 (limit: 127707)
       Memory: 540.7M
          CPU: 18min 32.117s
...output omitted...
```

Press `Ctrl+C` to quit the command.

- 4.5. Confirm that the `crio` service is active.

```
sh-5.1# systemctl is-active crio
active
```

- 4.6. Exit the debug pod.

```
sh-5.1# exit
exit
sh-4.4# exit
exit

Removing debug pod ...
```

▶ 5. Retrieve debugging information for the cluster.

- 5.1. Retrieve debugging information of the cluster by using the `oc adm must-gather` command. Specify the `/home/student/must-gather` directory as the destination directory. This command might take several minutes to complete.

```
[student@workstation ~]$ oc adm must-gather --dest-dir /home/student/must-gather
[must-gather      ] OUT Using must-gather plug-in image: quay.io/openshift-
release-dev/ocp-v4.0-art-dev@sha256:07d3...e94c
...output omitted...
Reprinting Cluster State:
When opening a support case, bugzilla, or issue please include the following
summary data along with any other requested information:
ClusterID: 94ff22c1-88a0-44cf-90f6-0b7b8b545434
ClusterVersion: Stable at "4.14.0"
ClusterOperators:
All healthy and stable
```

- 5.2. Verify that the debugging information exists in the destination directory. List the last five kubelet service logs, and confirm that an error occurred with proxying the data from the 192.168.50.10 IP address. Replace `quay-io...` with the generated directory name.

```
[student@workstation ~]$ ls ~/must-gather
event-filter.html
quay-io-openshift-release-dev-ocp-v4-0-art-dev-sha256-07d3...e94c
timestamp
[student@workstation ~]$ tail -5 \
~/must-gather/quay-io.../host_service_logs/masters/kubelet_service.log
...output omitted...
Mar 09 01:12:09.680445 master01 kubenswrapper[3275]: I1206 01:12:09.680399 3275
  logs.go:323] "Finished parsing log file" path="/var/log/pods/openshift-service-
ca_service-ca-5d96446959-69jq8_c9800778-c955-4b89-9bce-9f043237c986/service-ca-
controller/9.log"
Mar 09 01:12:12.771111 master01 kubenswrapper[3275]: E1206 01:12:12.770971
  3275 upgradeaware.go:426] Error proxying data from client to backend: readfrom
  tcp 192.168.50.10:44410->192.168.50.10:10010: write tcp 192.168.50.10:44410-
>192.168.50.10:10010: write: broken pipe
```

- 5.3. Generate debugging information for the `openshift-apiserver` cluster operator. Specify the `/home/student/inspect` directory as the destination directory. Limit the debugging information to the last five minutes.

```
[student@workstation ~]$ oc adm inspect clusteroperator/openshift-apiserver \
--dest-dir /home/student/inspect --since 5m
Gathering data for ns/openshift-config...
Gathering data for ns/openshift-config-managed...
Gathering data for ns/openshift-kube-apiserver-operator...
Gathering data for ns/openshift-kube-apiserver...
Gathering data for ns/openshift-etcd-operator...
Wrote inspect data to /home/student/inspect.
...output omitted...
```

- 5.4. Verify that the debugging information exists in the destination directory, and review the `cluster.yaml` file from the `~/inspect/cluster-scoped-resources/operator.openshift.io/openshiftapiservers` directory.

```
[student@workstation ~]$ ls inspect/
cluster-scoped-resources
event-filter.html
namespaces
timestamp
[student@workstation ~]$ cat \
~/inspect/cluster-scoped-resources/operator.openshift.io/\
openshiftapiservers/cluster.yaml
apiVersion: operator.openshift.io/v1
kind: OpenShiftAPIServer
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2023-12-12T16:03:42Z"
  generation: 3
  managedFields:
    - apiVersion: operator.openshift.io/v1
      fieldsType: FieldsV1
      fieldsV1:
        f:metadata:
          f:annotations:
            .: {}
...output omitted...
```

- 5.5. Delete the debugging information from your system.

```
[student@workstation ~]$ rm -rf must-gather inspect
```

## 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 cli-health
```



## ► Lab

# Kubernetes and OpenShift Command-line Interfaces and APIs

Find detailed information about your OpenShift cluster and assess its health by querying its Kubernetes resources.

### Outcomes

- Use the command line to retrieve information about the cluster resources.
- Identify cluster operators and API resources.
- List the available namespaced resources.
- Identify the resources that belong to the core API group.
- List the resource types that the `oauth.openshift.io` API group provides.
- List the resource usage of containers in a pod.
- Use the JSONPath filter to get the number of allocatable pods and compute resources for a node.
- List the memory and CPU usage of all pods in the cluster.
- Use jq filters to retrieve the `conditions` status of a pod.
- View cluster events and alerts.

### 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 cli-review
```

### Instructions

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your `workstation` machine.

Log in to the OpenShift cluster as the `developer` user with the `developer` password. Use the `cli-review` project for your work.

1. Log in to the OpenShift cluster and create the `cli-review` project.
2. Use the `oc` command to list the following information for the cluster:
  - Retrieve the cluster version.
  - Identify the supported API versions.

- Identify the fields for the `pod.spec.securityContext` object.
- 3.** From the terminal, log in to the OpenShift cluster as the `admin` user with the `redhatocp` password. Then, use the command line to identify the following cluster resources:
- List the cluster operators.
  - Identify the available namespaced resources.
  - Identify the resources that belong to the core API group.
  - List the resource types that the `oauth.openshift.io` API group provides.
  - List the events in the `openshift-kube-controller-manager` namespace.
- 4.** Identify the following information about the cluster services and its nodes:
- Retrieve the `conditions` status of the `etcd-master01` pod in the `openshift-etcd` namespace by using `jq` filters to limit the output.
  - List the compute resource usage of the containers in the `etcd-master01` pod in the `openshift-etcd` namespace.
  - Get the number of allocatable pods for the `master01` node by using a JSONPath filter.
  - List the memory and CPU usage of all pods in the cluster.
  - Retrieve the compute resource consumption of the `master01` node.
  - Retrieve the capacity and allocatable CPU for the `master01` node by using a JSONPath filter.
- 5.** Retrieve debugging information for the cluster. Specify the `/home/student/D0180/labs/cli-review/debugging` directory as the destination directory.

Then, generate debugging information for the `kube-apiserver` cluster operator. Specify the `/home/student/D0180/labs/cli-review/inspect` directory as the destination directory. Limit the debugging information to the last five minutes.

## 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 cli-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 cli-review
```

## ► Solution

# Kubernetes and OpenShift Command-line Interfaces and APIs

Find detailed information about your OpenShift cluster and assess its health by querying its Kubernetes resources.

### Outcomes

- Use the command line to retrieve information about the cluster resources.
- Identify cluster operators and API resources.
- List the available namespaced resources.
- Identify the resources that belong to the core API group.
- List the resource types that the `oauth.openshift.io` API group provides.
- List the resource usage of containers in a pod.
- Use the JSONPath filter to get the number of allocatable pods and compute resources for a node.
- List the memory and CPU usage of all pods in the cluster.
- Use `jq` filters to retrieve the `conditions` status of a pod.
- View cluster events and alerts.

### 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 cli-review
```

### Instructions

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your `workstation` machine.

Log in to the OpenShift cluster as the `developer` user with the `developer` password. Use the `cli-review` project for your work.

1. Log in to the OpenShift cluster and create the `cli-review` project.

- 1.1. Log in to the OpenShift cluster.

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

1.2. Create the `cli-review` project.

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

2. Use the `oc` command to list the following information for the cluster:

- Retrieve the cluster version.
- Identify the supported API versions.
- Identify the fields for the `pod.spec.securityContext` object.

2.1. Identify the cluster version.

```
[student@workstation ~]$ oc version
Client Version: 4.14.0
Kustomize Version: v5.0.1
Kubernetes Version: v1.27.6+f67aeb3
```

2.2. Identify the supported API versions.

```
[student@workstation ~]$ oc api-versions
admissionregistration.k8s.io/v1
apiextensions.k8s.io/v1
apiregistration.k8s.io/v1
apiserver.openshift.io/v1
apps.openshift.io/v1
apps/v1
...output omitted...
```

2.3. Identify the fields for the `pod.spec.securityContext` object.

```
[student@workstation ~]$ oc explain pod.spec.securityContext
KIND:     Pod
VERSION:  v1

FIELD: securityContext <PodSecurityContext>

DESCRIPTION:
...output omitted...
```

3. From the terminal, log in to the OpenShift cluster as the `admin` user with the `redhatocp` password. Then, use the command line to identify the following cluster resources:

- List the cluster operators.
- Identify the available namespaced resources.
- Identify the resources that belong to the core API group.
- List the resource types that the `oauth.openshift.io` API group provides.
- List the events in the `openshift-kube-controller-manager` namespace.

## 3.1. Log in to the OpenShift cluster.

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

## 3.2. List the cluster operators.

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
authentication	4.14.0	True	False	False	12h
baremetal	4.14.0	True	False	False	31d
cloud-controller-manager	4.14.0	True	False	False	31d
cloud-credential	4.14.0	True	False	False	31d
cluster-autoscaler	4.14.0	True	False	False	31d
config-operator	4.14.0	True	False	False	31d
console	4.14.0	True	False	False	31d
...output omitted...					

## 3.3. List the available namespaced resources.

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
bindings		v1	true	Binding
configmaps	cm	v1	true	ConfigMap
endpoints	ep	v1	true	Endpoints
events	ev	v1	true	Event
limitranges	limits	v1	true	LimitRange
persistentvolumeclaims	pvc	v1	true	PersistentVolumeClaim
pods	po	v1	true	Pod
...output omitted...				

## 3.4. Identify the resources that belong to the core API group.

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
bindings		v1	true	Binding
componentstatuses	cs	v1	false	ComponentStatus
configmaps	cm	v1	true	ConfigMap
endpoints	ep	v1	true	Endpoints
events	ev	v1	true	Event
limitranges	limits	v1	true	LimitRange
namespaces	ns	v1	false	Namespace
nodes	no	v1	false	Node
...output omitted...				

## 3.5. List the resource types that the oauth.openshift.io API group provides.

```
[student@workstation ~]$ oc api-resources --api-group oauth.openshift.io
NAME           SHORTNAMES   APIVERSION      NAMESPACED KIND
oauthaccesstokens   oauth.openshift.io/v1  false    OAuthAccessToken
oauthauthorizationtokens   oauth.openshift.io/v1  false    OAuthAuthorizationToken
OAuthAuthorizationToken
...output omitted...
```

3.6. Retrieve the events for the `openshift-kube-controller-manager` namespace.

```
[student@workstation ~]$ oc get events -n openshift-kube-controller-manager
LAST SEEN TYPE      REASON          OBJECT            ...
48m       Normal   CreatedSCCRanges pod/kube-controller-manager-master ...
21m       Normal   CreatedSCCRanges pod/kube-controller-manager-master ...
14m       Normal   CreatedSCCRanges pod/kube-controller-manager-master ...
```

4. Identify the following information about the cluster services and its nodes:

- Retrieve the `conditions` status of the `etcd-master01` pod in the `openshift-etcd` namespace by using `jq` filters to limit the output.
- List the compute resource usage of the containers in the `etcd-master01` pod in the `openshift-etcd` namespace.
- Get the number of allocatable pods for the `master01` node by using a JSONPath filter.
- List the memory and CPU usage of all pods in the cluster.
- Retrieve the compute resource consumption of the `master01` node.
- Retrieve the capacity and allocatable CPU for the `master01` node by using a JSONPath filter.

- 4.1. Retrieve the `conditions` status of the `etcd-master01` pod in the `openshift-etcd` namespace. Use `jq` filters to limit the output to the `.status.conditions` attribute of the pod.

```
[student@workstation ~]$ oc get pods etcd-master01 -n openshift-etcd \
-o json | jq .status.conditions
[
  {
    "lastProbeTime": null,
    "lastTransitionTime": "2023-03-12T16:40:35Z",
    "status": "True",
    "type": "Initialized"
  },
  {
    "lastProbeTime": null,
    "lastTransitionTime": "2023-03-12T16:40:47Z",
    "status": "True",
    "type": "Ready"
  },
  {
    "lastProbeTime": null,
    "lastTransitionTime": "2023-03-12T16:40:47Z",
    "status": "True",
    "type": "PodScheduled"
  }
]
```

```

        "status": "True",
        "type": "ContainersReady"
    },
    {
        "lastProbeTime": null,
        "lastTransitionTime": "2023-03-12T16:40:23Z",
        "status": "True",
        "type": "PodScheduled"
    }
]

```

- 4.2. List the resource usage of the containers in the `etcd-master01` pod in the `openshift-etcd` namespace.

```
[student@workstation ~]$ oc adm top pods etcd-master01 \
-n openshift-etcd --containers
POD          NAME      CPU(cores)   MEMORY(bytes)
etcd-master01 POD        0m          0Mi
etcd-master01 etcd       54m         1513Mi
etcd-master01 etcd-metrics 5m          24Mi
etcd-master01 etcd-readyz 4m          39Mi
etcd-master01 etcdctl    0m          0Mi
```

- 4.3. Use a JSONPath filter to determine the number of allocatable pods for the `master01` node.

```
[student@workstation ~]$ oc get node master01 \
-o jsonpath='{.status.allocatable.pods}{"\n"}'
250
```

- 4.4. List the memory and CPU usage of all pods in the cluster. Use the `--sum` option to print the sum of the resource usage. The resource usage on your system probably differs.

```
[student@workstation ~]$ oc adm top pods -A --sum
NAMESPACE     NAME      CPU(cores)   MEMORY(bytes)
metallb-system controller-5f6dfd8c4f-ddr8v  0m          56Mi
metallb-system metallb-operator-controller-manager-... 0m          50Mi
metallb-system metallb-operator-webhook-server-...  0m          26Mi
metallb-system speaker-2dds4        9m          210Mi
...output omitted...
----- -----
505m          8982Mi
```

- 4.5. Retrieve the resource consumption of the `master01` node.

```
[student@workstation ~]$ oc adm top node
NAME      CPU(cores)   CPU%    MEMORY(bytes)   MEMORY%
master01  1199m       15%    12555Mi        66%
```

- 4.6. Use a JSONPath filter to determine the capacity and allocatable CPU for the `master01` node.

```
[student@workstation ~]$ oc get node master01 -o jsonpath=\
'Allocatable: {.status.allocatable.cpu}{"\n"}'\
'Capacity: {.status.capacity.cpu}{"\n"}'
Allocatable: 7500m
Capacity: 8
```

5. Retrieve debugging information for the cluster. Specify the /home/student/D0180/labs/cli-review/debugging directory as the destination directory.

Then, generate debugging information for the kube-apiserver cluster operator. Specify the /home/student/D0180/labs/cli-review/inspect directory as the destination directory. Limit the debugging information to the last five minutes.

- 5.1. Retrieve debugging information for the cluster. Save the output to the /home/student/D0180/labs/cli-review/debugging directory.

```
[student@workstation ~]$ oc adm must-gather \
--dest-dir /home/student/D0180/labs/cli-review/debugging
[must-gather      ] OUT Using must-gather plug-in image: quay.io/openshift-
release-dev/ocp-v4.0-art-dev@sha256:07d3...e94c
...output omitted...
Reprinting Cluster State:
When opening a support case, bugzilla, or issue please include the following
summary data along with any other requested information:
ClusterID: 94ff22c1-88a0-44cf-90f6-0b7b8b545434
ClusterVersion: Stable at "4.14.0"
ClusterOperators:
    All healthy and stable
```

- 5.2. Generate debugging information for the kube-apiserver cluster operator. Save the output to the /home/student/D0180/labs/cli-review/inspect directory, and limit the debugging information to the last five minutes.

```
[student@workstation ~]$ oc adm inspect clusteroperator kube-apiserver \
--dest-dir /home/student/D0180/labs/cli-review/inspect --since 5m
Gathering data for ns/metallb-system...
...output omitted...
Wrote inspect data to /home/student/D0180/labs/cli-review/inspect.
```

## 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 cli-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 cli-review
```

## ► Quiz

# Kubernetes and OpenShift Command-line Interfaces and APIs

Choose the correct answers to the following questions:

As the student user on the workstation machine, 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
```

Then, use OC commands to answer the following questions:

- ▶ **1. Which supported API resource is a member of the oauth.openshift.io api-group?**
  - a. groupoauthaccesstokens
  - b. tokenreviews
  - c. cloudcredentials
  - d. networkpolicies
  
- ▶ **2. Which two fields are members of the pod.spec.securityContext object? (Choose two.)**
  - a. readinessGates
  - b. runAsUser
  - c. sysctls
  - d. priorityClassName
  
- ▶ **3. Which three commands display the conditions of the master01 node? (Choose three.)**
  - a. oc get node/master01 -o json | jq '.status.conditions'
  - b. oc get node/master01 -o wide
  - c. oc get node/master01 -o yaml
  - d. oc get node/master01 -o json
  
- ▶ **4. Select the two valid condition types for a control plane node. (Choose two)**
  - a. PIDPressure
  - b. DiskIOPressure
  - c. OutOfMemory
  - d. Ready

► **5. Select three valid options for the `oc adm top pods` command. (Choose three.)**

- a. -A
- b. --sum
- c. --pod-selector
- d. --containers

► **6. Which command determines the assigned internal IP address on the master01 node?**

- a. `oc get nodes master01 -o=jsonpath='{{.status.addresses}}{"\n"}}`
- b. `oc get nodes master01 -o json | jq .status.nodeInfo`
- c. `oc get nodes master01 -o=jsonpath='{{.status.conditions}}{"\n"}}`

► **7. Which command displays only the conditions for the authentication cluster operator?**

- a. `oc get clusteroperators.config.openshift.io authentication -o json | jq .status.versions`
- b. `oc get clusteroperators.config.openshift.io authentication -o json | jq .status`
- c. `oc get clusteroperators.config.openshift.io authentication -o json | jq .status.conditions`

## ► Solution

# Kubernetes and OpenShift Command-line Interfaces and APIs

Choose the correct answers to the following questions:

As the student user on the workstation machine, 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
```

Then, use OC commands to answer the following questions:

- ▶ **1. Which supported API resource is a member of the oauth.openshift.io api-group?**
  - a. groupoauthaccesstokens
  - b. tokenreviews
  - c. cloudcredentials
  - d. networkpolicies
  
- ▶ **2. Which two fields are members of the pod.spec.securityContext object? (Choose two.)**
  - a. readinessGates
  - b. runAsUser
  - c. sysctls
  - d. priorityClassName
  
- ▶ **3. Which three commands display the conditions of the master01 node? (Choose three.)**
  - a. oc get node/master01 -o json | jq '.status.conditions'
  - b. oc get node/master01 -o wide
  - c. oc get node/master01 -o yaml
  - d. oc get node/master01 -o json
  
- ▶ **4. Select the two valid condition types for a control plane node. (Choose two)**
  - a. PIDPressure
  - b. DiskIOPressure
  - c. OutOfMemory
  - d. Ready

► **5. Select three valid options for the `oc adm top pods` command. (Choose three.)**

- a. -A
- b. --sum
- c. --pod-selector
- d. --containers

► **6. Which command determines the assigned internal IP address on the master01 node?**

- a. `oc get nodes master01 -o=jsonpath='{{.status.addresses}}{"\n"}}`
- b. `oc get nodes master01 -o json | jq .status.nodeInfo`
- c. `oc get nodes master01 -o=jsonpath='{{.status.conditions}}{"\n"}}`

► **7. Which command displays only the conditions for the authentication cluster operator?**

- a. `oc get clusteroperators.config.openshift.io authentication -o json | jq .status.versions`
- b. `oc get clusteroperators.config.openshift.io authentication -o json | jq .status`
- c. `oc get clusteroperators.config.openshift.io authentication -o json | jq .status.conditions`

# Summary

---

- An RHOCP cluster can be managed from the web console or by using the `kubectl` or `oc` command-line interfaces (CLI).
- Use the `--help` option on any command to view detailed information about the command.
- Projects provide isolation between your application resources.
- Token authentication is the only guaranteed method to work with any RHOCP cluster, because enterprise SSO might replace the login form of the web console.
- All administrative tasks require creating, viewing, and changing the API resources.
- Kubernetes provides YAML- and JSON-formatted output options, which are ideal for parsing or scripting.
- Operators provide the means of monitoring applications, performing health checks, managing over-the-air (OTA) updates, and ensuring that applications remain in your specified state.
- The RHOCP web console incorporates useful graphs to visualize cluster and resource analytics.
- The RHOCP web console provides an interface for executing Prometheus queries, visualizing metrics, and configuring alerts.
- The monitoring stack is based on the Prometheus project, and it is configured to monitor the core RHOCP cluster components, by default.
- RHOCP provides the ability to view logs in running containers and pods to ease troubleshooting.
- You can collect resource definitions and service logs from your cluster by using the `oc adm must-gather` command.

## Chapter 3

# Run Applications as Containers and Pods

### Goal

Run and troubleshoot containerized applications as unmanaged Kubernetes pods.

### Objectives

- Run containers inside pods and identify the host OS processes and namespaces that the containers use.
- Find containerized applications in container registries and get information about the runtime parameters of supported and community container images.
- Troubleshoot a pod by starting additional processes on its containers, changing their ephemeral file systems, and opening short-lived network tunnels.

### Sections

- Create Linux Containers and Kubernetes Pods (and Guided Exercise)
- Find and Inspect Container Images (and Guided Exercise)
- Troubleshoot Containers and Pods (and Guided Exercise)

### Lab

- Run Applications as Containers and Pods

# Create Linux Containers and Kubernetes Pods

---

## Objectives

- Run containers inside pods and identify the host OS processes and namespaces that the containers use.

## Creating Containers and Pods

Kubernetes and OpenShift offer many ways to create containers in pods. You can use one such way, the `run` command, with the `kubectl` or `oc` CLI to create and deploy an application in a pod from a container image. A *container image* contains immutable data that defines an application and its libraries.



### Note

Container images are discussed in more detail elsewhere in the course.

The `run` command uses the following syntax:

```
oc run RESOURCE/NAME --image IMAGE [options]
```

For example, the following command deploys an Apache HTTPD application in a pod named `web-server` that uses the `registry.access.redhat.com/ubi8/httpd-24` container image.

```
[user@host ~]$ kubectl run web-server --image
registry.access.redhat.com/ubi8/httpd-24
```

You can use several options and flags with the `run` command. The `--command` option executes a custom command and its arguments in a container, rather than the default command that is defined in the container image. You must follow the `--command` option with a double dash (`--`) to separate the custom command and its arguments from the `run` command options. The following syntax is used with the `--command` option:

```
oc run RESOURCE/NAME --image IMAGE --command -- cmd arg1 ... argN
```

You can also use the double dash option to provide custom arguments to a default command in the container image.

```
kubectl run RESOURCE/NAME --image IMAGE -- arg1 arg2 ... argN
```

To start an interactive session with a container in a pod, include the `-it` options before the pod name. The `-i` option tells Kubernetes to keep open the standard input (`stdin`) on the container in the pod. The `-t` option tells Kubernetes to open a TTY session for the container in the pod. You can use the `-it` options to start an interactive, remote shell in a container. From the remote shell, you can then execute additional commands in the container.

The following example starts an interactive remote shell, /bin/bash, in the default container in the `my-app` pod.

```
[user@host ~]$ oc run -it my-app --image registry.access.redhat.com/ubi9/ubi \
--command -- /bin/bash
If you don't see a command prompt, try pressing enter.
bash-5.1$
```



### Note

Unless you include the `--namespace` or `-n` options, the `run` command creates containers in pods in the current selected project.

You can also define a restart policy for containers in a pod by including the `--restart` option. A pod restart policy determines how the cluster should respond when containers in that pod exit. The `--restart` option has the following accepted values: `Always`, `OnFailure`, and `Never`.

#### Always

If the restart policy is set to `Always`, then the cluster continuously tries to restart a successfully exited container, for up to five minutes. The default pod restart policy is `Always`. If the `--restart` option is omitted, then the pod is configured with the `Always` policy.

#### OnFailure

Setting the pod restart policy to `OnFailure` tells the cluster to restart only failed containers in the pod, for up to five minutes.

#### Never

If the restart policy is set to `Never`, then the cluster does not try to restart exited or failed containers in a pod. Instead, the pods immediately fail and exit.

The following example command executes the `date` command in the container of the pod named `my-app`, redirects the `date` command output to the terminal, and defines `Never` as the pod restart policy.

```
[user@host ~]$ oc run -it my-app \
--image registry.access.redhat.com/ubi9/ubi \
--restart Never --command -- date
Mon Feb 20 22:36:55 UTC 2023
```

To automatically delete a pod after it exits, include the `--rm` option with the `run` command.

```
[user@host ~]$ kubectl run -it my-app --rm \
--image registry.access.redhat.com/ubi9/ubi \
--restart Never --command -- date
Mon Feb 20 22:38:50 UTC 2023
pod "date" deleted
```

For some containerized applications, you might need to specify environment variables for the application to work. To specify an environment variable and its value, include the `--env=` option with the `run` command.

```
[user@host ~]$ oc run mysql \
--image registry.redhat.io/rhel9/mysql-80 \
--env MYSQL_ROOT_PASSWORD=myP@$$123
pod/mysql created
```

## User and Group IDs Assignment

When a project is created, OpenShift adds annotations to the project that determine the user ID (UID) range and supplemental group ID (GID) for pods and their containers in the project. You can retrieve the annotations with the `oc describe project project-name` command.

```
[user@host ~]$ oc describe project my-app
Name:    my-app
...output omitted...
Annotations:  openshift.io/description=
              openshift.io/display-name=
              openshift.io/requester=developer
              openshift.io/sa.scc.mcs=s0:c27,c4
openshift.io/sa.scc.supplemental-groups=1000710000/10000
openshift.io/sa.scc.uid-range=1000710000/10000
...output omitted...
```

With OpenShift default security policies, regular cluster users cannot choose the USER or UIDs for their containers. When a regular cluster user creates a pod, OpenShift ignores the USER instruction in the container image. Instead, OpenShift assigns to the user in the container a UID and a supplemental GID from the identified range in the project annotations. The GID of the user is always 0, which means that the user belongs to the `root` group. Any files and directories that the container processes might write to must have read and write permissions by `GID=0` and have the `root` group as the owner. Although the user in the container belongs to the `root` group, the user is an unprivileged account.

In contrast, when a cluster administrator creates a pod, the USER instruction in the container image is processed. For example, if the USER instruction for the container image is set to 0, then the user in the container is the `root` privileged account, with a 0 value for the UID. Executing a container as a privileged account is a security risk. A privileged account in a container has unrestricted access to the container's host system. Unrestricted access means that the container could modify or delete system files, install software, or otherwise compromise its host. Red Hat therefore recommends that you run containers as `root less`, or as an unprivileged user with only the necessary privileges for the container to run.

Red Hat also recommends that you run containers from different applications with unique user IDs. Running containers from different applications with the same UID, even an unprivileged one, is a security risk. If the UID for two containers is the same, then the processes in one container could access the resources and files of the other container. By assigning a distinct range of UIDs and GIDs for each project, OpenShift ensures that applications in different projects do not run as the same UID or GID.

## Pod Security

The Kubernetes Pod Security Admission controller issues a warning when a pod is created without a defined security context. Security contexts grant or deny OS-level privileges to pods. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security. You can ignore pod security warnings in these course exercises. Security Context Constraints (SCC)

are discussed in more detail in course DO280: *Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster*.

## Execute Commands in Running Containers

To execute a command in a running container in a pod, you can use the `exec` command with the `kubectl` or `oc` CLI. The `exec` command uses the following syntax:

```
oc exec RESOURCE/NAME -- COMMAND [args...] [options]
```

The output of the executed command is sent to your terminal. In the following example, the `exec` command executes the `date` command in the `my-app` pod.

```
[user@host ~]$ oc exec my-app -- date  
Tue Feb 21 20:43:53 UTC 2023
```

The specified command is executed in the first container of a pod. For multicontainer pods, include the `-c` or `--container=` options to specify which container is used to execute the command. The following example executes the `date` command in a container named `ruby-container` in the `my-app` pod.

```
[user@host ~]$ kubectl exec my-app -c ruby-container -- date  
Tue Feb 21 20:46:50 UTC 2023
```

The `exec` command also accepts the `-i` and `-t` options to create an interactive session with a container in a pod. In the following example, Kubernetes sends `stdin` to the `bash` shell in the `ruby-container` container from the `my-app` pod, and sends `stdout` and `stderr` from the `bash` shell back to the terminal.

```
[user@host ~]$ oc exec my-app -c ruby-container -it -- bash -il  
[1000780000@ruby-container /]$
```

In the previous example, a raw terminal is opened in the `ruby-container` container. From this interactive session, you can execute additional commands in the container. To terminate the interactive session, you must execute the `exit` command in the raw terminal.

```
[user@host ~]$ kubectl exec my-app -c ruby-container -it -- bash -il  
[1000780000@ruby-container /]$ date  
Tue Feb 21 21:16:00 UTC 2023  
[1000780000@ruby-container] exit
```

## Container Logs

Container logs are the standard output (`stdout`) and standard error (`stderr`) output of a container. You can retrieve logs with the `logs pod pod-name` command that the `kubectl` and `oc` CLIs provide. The command includes the following options:

**`-l` or `--selector=''`**

Filter objects based on the specified key:`:value` label constraint.

**--tail=**

Specify the number of lines of recent log files to display; the default value is -1 with no selectors, which displays all log lines.

**-c or --container=**

Print the logs of a particular container in a multicontainer pod.

**-f or --follow**

Follow, or stream, logs for a container.

**-p or --previous=true**

Print the logs of a previous container instance in the pod, if it exists. This option is helpful for troubleshooting a pod that failed to start, because it prints the logs of the last attempt.

The following example restricts `oc logs` command output to the 10 most recent log files:

```
[user@host ~]$ oc logs postgresql-1-jw89j --tail=10
done
server stopped
Starting server...
2023-01-04 22:00:16.945 UTC [1] LOG:  starting PostgreSQL 12.11 on x86_64-redhat-
linux-gnu, compiled by gcc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-10), 64-bit
2023-01-04 22:00:16.946 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port
      5432
2023-01-04 22:00:16.946 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
2023-01-04 22:00:16.953 UTC [1] LOG:  listening on Unix socket "/var/run/
postgresql/.s.PGSQL.5432"
2023-01-04 22:00:16.960 UTC [1] LOG:  listening on Unix socket "/"
      tmp/.s.PGSQL.5432"
2023-01-04 22:00:16.968 UTC [1] LOG:  redirecting log output to logging collector
      process
2023-01-04 22:00:16.968 UTC [1] HINT:  Future log output will appear in directory
      "log".
```

You can also use the `attach pod-name -c container-name -it` command to connect to and start an interactive session on a running container in a pod. The `-c container-name` option is required for multicontainer pods. If the container name is omitted, then Kubernetes uses the `kubectl.kubernetes.io/default-container` annotation on the pod to select the container. Otherwise, the first container in the pod is chosen. You can use the interactive session to retrieve application log files and to troubleshoot application issues.

```
[user@host ~]$ oc attach my-app -it
If you don't see a command prompt, try pressing enter.

bash-4.4$
```

You can also retrieve logs from the web console by clicking the **Logs** tab of any pod.

Project: metallb-system

Pods > Pod details

P controller-58c464cdb6-g7bx7 Running

Actions ▾

Details Metrics YAML Environment Logs Events Terminal

**Pod details**

Name	controller-58c464cdb6-g7bx7	Status	Running
Namespace	metallb-system	Restart policy	Always restart
Labels	app=metallb, component=controller, pod-template-hash=58c464cdb6	Edit	Active deadline seconds Not configured
Pod IP			

If you have more than one container, then you can change between them to list the logs of each one.

Project: metallb-system

Pods > Pod details

P controller-58c464cdb6-g7bx7 Running

Actions ▾

Details Metrics YAML Environment Logs Events Terminal

Log streaming... controller Current log Search

Wrap lines | Raw

208 lines

```

202 |     controller : ServiceReconciler , end reconcile : openshift-storage/lvms-
203 |     {"caller": "service_controller_reload.go:61", "controller": "ServiceReconciler - reprocessAll", "level": "info", "stt
204 |     {"caller": "service_controller_reload.go:61", "controller": "ServiceReconciler - reprocessAll", "end reconcile": "r
205 |     {"caller": "service_controller.go:60", "controller": "ServiceReconciler", "level": "info", "start reconcile": "opensh
206 |     {"caller": "service_controller.go:103", "controller": "ServiceReconciler", "end reconcile": "openshift-storage/lvms-
207 |     {"caller": "service_controller.go:60", "controller": "ServiceReconciler", "level": "info", "start reconcile": "opensh
208 |     {"caller": "service_controller.go:103", "controller": "ServiceReconciler", "end reconcile": "openshift-storage/lvms-

```

## Deleting Resources

You can delete Kubernetes resources, such as pod resources, with the `delete` command. The `delete` command can delete resources by resource type and name, resource type and label, standard input (`stdin`), and with JSON- or YAML-formatted files. The command accepts only one argument type at a time.

For example, you can supply the resource type and name as a command argument.

```
[user@host ~]$ oc delete pod php-app
```

You can also delete pods from the web console by clicking **Actions** and then **Delete Pod** in the pod's principal menu.

The screenshot shows the OpenShift web console interface. In the top left, it says 'metallb-system'. Below that, 'Pod details' and 'controller-7ff8d94567-dhs2s' are shown as 'Running'. A navigation bar includes 'Labels', 'Metrics', 'YAML', 'Environment', 'Logs', 'Events', and 'Terminal'. On the right, there's an 'Actions' dropdown with options: 'Edit labels', 'Edit annotations', 'Edit Pod', and 'Delete Pod' (which is highlighted with a red box). The main content area shows 'Labels' for the pod, including 'app=controller' and 'pod-template-hash=7ff8d94567'. It also shows 'Status' (Running), 'Restart policy' (Always restart), and 'Active deadline seconds' (Not configured).

To select resources based on labels, you can include the `-l` option and the `key:value` label as a command argument.

```
[user@host ~]$ kubectl delete pod -l app=my-app
pod "php-app" deleted
pod "mysql-db" deleted
```

You can also provide the resource type and a JSON- or YAML-formatted file that specifies the name of the resource. To use a file, you must include the `-f` option and provide the full path to the JSON- or YAML-formatted file.

```
[user@host ~]$ oc delete pod -f ~/php-app.json
pod "php-app" deleted
```

You can also use `stdin` and a JSON- or YAML-formatted file that includes the resource type and resource name with the `delete` command.

```
[user@host ~]$ cat ~/php-app.json | kubectl delete -f -
pod "php-app" deleted
```

Pods support graceful termination, which means that pods try to terminate their processes first before Kubernetes forcibly terminates the pods. To change the time period before a pod is forcibly terminated, you can include the `--grace-period` flag and a time period in seconds in your `delete` command. For example, to change the grace period to 10 seconds, use the following command:

```
[user@host ~]$ oc delete pod php-app --grace-period=10
```

To shut down the pod immediately, set the grace period to 1 second. You can also use the `--now` flag to set the grace period to 1 second.

```
[user@host ~]$ oc delete pod php-app --now
```

You can also forcibly delete a pod with the `--force` option. If you forcibly delete a pod, Kubernetes does not wait for a confirmation that the pod's processes ended, which can leave the pod's processes running until its node detects the deletion. Therefore, forcibly deleting a pod could result in inconsistency or data loss. Forcibly delete pods only if you are sure that the pod's processes are terminated.

```
[user@host ~]$ kubectl delete pod php-app --force
```

To delete all pods in a project, you can include the `--all` option.

```
[user@host ~]$ kubectl delete pods --all
pod "php-app" deleted
pod "mysql-db" deleted
```

Likewise, you can delete a project and its resources with the `oc delete project project-name` command.

```
[user@host ~]$ oc delete project my-app
project.project.openshift.io "my-app" deleted
```

## The CRI-O Container Engine

A container engine is required to run containers. Worker and control plane nodes in an OpenShift Container Platform cluster use the CRI-O container engine to run containers. Unlike tools such as Podman or Docker, the CRI-O container engine is a runtime that is designed and optimized specifically for running containers in a Kubernetes cluster. Because CRI-O meets the Kubernetes Container Runtime Interface (CRI) standards, the container engine can integrate with other Kubernetes and OpenShift tools, such as networking and storage plug-ins.



### Note

For more information about the Kubernetes Container Runtime Interface (CRI) standards, refer to the *CRI-API* repository at <https://github.com/kubernetes/cri-api>.

CRI-O provides a command-line interface to manage containers with the `crictl` command. The `crictl` command includes several subcommands to help you to manage containers. The following subcommands are commonly used with the `crictl` command:

#### **crictl pods**

Lists all pods on a node.

#### **crictl image**

Lists all images on a node.

#### **crictl inspect**

Retrieve the status of one or more containers.

#### **crictl exec**

Run a command in a running container.

#### **crictl logs**

Retrieve the logs of a container.

### **cricctl ps**

List running containers on a node.

To manage containers with the `cricctl` command, you must first identify the node that is hosting your containers.

```
[user@host ~]$ kubectl get pods -o wide
NAME          READY STATUS    RESTARTS AGE IP           NODE
postgresql-1-8lzf2 1/1   Running   0        20m 10.8.0.64 master01
postgresql-1-deploy 0/1  Completed 0        21m 10.8.0.63 master01
```

```
[user@host ~]$ oc get pod postgresql-1-8lzf2 -o jsonpath='{.spec.nodeName}{ "\n"}'
master01
```

Next, you must connect to the identified node as a cluster administrator. Cluster administrators can use SSH to connect to a node or create a debug pod for the node. Regular users cannot connect to or create debug pods for cluster nodes.

As a cluster administrator, you can create a debug pod for a node with the `oc debug node/node-name` command. OpenShift creates the `pod/node-name-debug` pod in your currently selected project and automatically connects you to the pod. You must then enable access host binaries, such as the `cricctl` command, with the `chroot /host` command. This command mounts the host's root file system in the `/host` directory within the debug pod shell. By changing the root directory to the `/host` directory, you can run binaries contained in the host's executable path.

```
[user@host ~]$ oc debug node/master01
Starting pod/master01-debug ...
To use host binaries, run chroot /host
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
```

After enabling host binaries, you can use the `cricctl` command to manage the containers on the node. For example, you can use the `cricctl ps` and `cricctl inspect` commands to retrieve the process ID (PID) of a running container. You can then use the PID to retrieve or enter the namespaces within a container, which is useful for troubleshooting application issues.

To find the PID of a running container, you must first determine the container's ID. You can use the `cricctl ps` command with the `--name` option to filter the command output to a specific container.

```
sh-5.1# crictl ps --name postgresql
CONTAINER      IMAGE      CREATED STATE      NAME      ATTEMPT POD ID      POD
27943ae4f3024 image...7104 5...ago Running postgresql 0      5768...f015
postgresql-1...
```

The default output of the `cricctl ps` command is a table. You can find the short container ID under the `CONTAINER` column. You can also use the `-o` or `--output` options to specify the format of the `cricctl ps` command as JSON or YAML and then parse the output. The parsed output displays the full container ID.

```
sh-5.1# crictl ps --name postgresql -o json | jq .containers[0].id
"2794...29a4"
```

After identifying the container ID, you can use the `crictl inspect` command and the container ID to retrieve the PID of the running container. By default, the `crictl inspect` command displays verbose output. You can use the `-o` or `--output` options to format the command output as JSON, YAML, a table, or as a Go template. If you specify the JSON format, you can then parse the output with the `jq` command. Likewise, you can use the `grep` command to limit the command output.

```
sh-5.1# crictl inspect -o json 27943ae4f3024 | jq .info.pid
43453
sh-5.1# crictl inspect 27943ae4f3024 | grep pid
  "pid": 43453,
...output omitted...
```

After determining the PID of a running container, you can use the `lsns -p PID` command to list the system namespaces of a container.

```
sh-5.1# lsns -p 43453
      NS TYPE    NPROCS   PID USER        COMMAND
4026531835 cgroup     530     1 root      /usr/lib/systemd/systemd --switched-root
--system --deserialize 17
4026531837 user       530     1 root      /usr/lib/systemd/systemd --switched-root
--system --deserialize 17
4026537853 uts        8 43453 1000690000 postgres
4026537854 ipc        8 43453 1000690000 postgres
4026537856 net        8 43453 1000690000 postgres
4026538013 mnt        8 43453 1000690000 postgres
4026538014 pid        8 43453 1000690000 postgres
```

You can also use the PID of a running container with the `nsenter` command to enter a specific namespace of a running container. For example, you can use the `nsenter` command to execute a command within a specified namespace on a running container. The following example executes the `ps -ef` command within the process namespace of a running container.

```
sh-5.1# nsenter -t 43453 -p -r ps -ef
UID      PID  PPID C STIME TTY          TIME CMD
1000690+    1      0 0 18:49 ?          00:00:00 postgres
1000690+    58     1 0 18:49 ?          00:00:00 postgres: logger
1000690+    60     1 0 18:49 ?          00:00:00 postgres: checkpointer
1000690+    61     1 0 18:49 ?          00:00:00 postgres: background writer
1000690+    62     1 0 18:49 ?          00:00:00 postgres: walwriter
1000690+    63     1 0 18:49 ?          00:00:00 postgres: autovacuum launcher
1000690+    64     1 0 18:49 ?          00:00:00 postgres: stats collector
1000690+    65     1 0 18:49 ?          00:00:00 postgres: logical replication
launcher
root      7414     0 0 20:14 ?          00:00:00 ps -ef
```

The `-t` option specifies the PID of the running container as the target PID for the `nsenter` command. The `-p` option directs the `nsenter` command to enter the process or pid namespace.

The `-r` option sets the top-level directory of the process namespace as the root directory, thus enabling commands to execute in the context of the namespace.

You can also use the `-a` option to execute a command in all of the container's namespaces.

```
sh-5.1# nsenter -t 43453 -a ps -ef
UID      PID  PPID  C STIME TTY      TIME CMD
1000690+    1      0  0 18:49 ?        00:00:00 postgres
1000690+    58     1  0 18:49 ?        00:00:00 postgres: logger
1000690+    60     1  0 18:49 ?        00:00:00 postgres: checkpointer
1000690+    61     1  0 18:49 ?        00:00:00 postgres: background writer
1000690+    62     1  0 18:49 ?        00:00:00 postgres: walwriter
1000690+    63     1  0 18:49 ?        00:00:00 postgres: autovacuum launcher
1000690+    64     1  0 18:49 ?        00:00:00 postgres: stats collector
1000690+    65     1  0 18:49 ?        00:00:00 postgres: logical replication
      launcher
root      10058      0  0 20:45 ?        00:00:00 ps -ef
```



## References

### Container Runtime Interface (CRI) CLI

<https://github.com/kubernetes-sigs/cri-tools/blob/master/docs/crictl.md>

### A Guide to OpenShift and UIDs

<https://cloud.redhat.com/blog/a-guide-to-openshift-and-uids>

For more information about resource log files, refer to the *Viewing Logs for a Resource* chapter in the Red Hat OpenShift Container Platform 4.14 *Logging* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/logging/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/logging/index)

### Manage Containers in Namespaces by Using nsenter

<https://www.redhat.com/sysadmin/container-namespaces-nsenter>

## ► Guided Exercise

# Create Linux Containers and Kubernetes Pods

Run a base OS container in a pod and compare the environment inside the container with its host node.

### Outcomes

- Create a pod with a single container, and identify the pod and its container within the container engine of an OpenShift node.
- View the logs of a running container.
- Retrieve information inside a container, such as the operating system (OS) release and running processes.
- Identify the process ID (PID) and namespaces for a container.
- Identify the User ID (UID) and supplemental group ID (GID) ranges of a project.
- Compare the namespaces of containers in one pod versus in another pod.
- Inspect a pod with multiple containers, and identify the purpose of each container.

### 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 pods-containers
```

### Instructions

- 1. Log in to the OpenShift cluster and create the `pods-containers` project. Determine the UID and GID ranges for pods in the `pods-containers` project.
- 1.1. Log in to the OpenShift cluster as the `developer` user with the `oc` command.

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

- 1.2. Create the `pods-containers` project.

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

- 1.3. Identify the UID and GID ranges for pods in the pods-containers project.

```
[student@workstation ~]$ oc describe project pods-containers
Name:      pods-containers
Created:   28 seconds ago
Labels:    kubernetes.io/metadata.name=pods-containers
           pod-security.kubernetes.io/audit=restricted
           pod-security.kubernetes.io/audit-version=v1.24
           pod-security.kubernetes.io/warn=restricted
           pod-security.kubernetes.io/warn-version=v1.24
Annotations: openshift.io/description=
             openshift.io/display-name=
             openshift.io/requester=developer
             openshift.io/sa.scc.mcs=s0:c28,c22
             openshift.io/sa.scc.supplemental-groups=1000800000/10000
             openshift.io/sa.scc.uid-range=1000800000/10000
Display Name: <none>
Description: <none>
Status:     Active
Node Selector: <none>
Quota:      <none>
Resource limits: <none>
```

Your UID and GID range values might differ from the previous output.

- ▶ 2. As the developer user, create a pod called ubi9-user from a UBI9 base container image. The image is available in the registry.ocp4.example.com:8443/ubi9/ubi container registry. Set the restart policy to Never and start an interactive session. Configure the pod to execute the whoami and id commands to determine the UIDs, supplemental groups, and GIDs of the container user in the pod. Delete the pod afterward.
- After the ubi-user pod is deleted, log in as the admin user and then re-create the ubi9-user pod. Retrieve the UIDs and GIDs of the container user. Compare the values to the values of the ubi9-user pod that the developer user created.
- Afterward, delete the ubi9-user pod.

- 2.1. Use the oc run command to create the ubi9-user pod. Configure the pod to execute the whoami and id commands through an interactive bash shell session.

```
[student@workstation ~]$ oc run -it ubi9-user --restart 'Never' \
--image registry.ocp4.example.com:8443/ubi9/ubi \
-- /bin/bash -c "whoami && id"
1000800000
uid=1000800000(1000800000) gid=0(root) groups=0(root),1000800000
```

Your values might differ from the previous output.

Notice that the user in the container has the same UID that is identified in the pods-containers project. However, the GID of the user in the container is 0, which means that the user belongs to the root group. Any files and directories that the container

processes might write to must have read and write permissions by GID=0 and have the root group as the owner.

Although the user in the container belongs to the root group, a UID value over 1000 means that the user is an unprivileged account. When a regular OpenShift user, such as the developer user, creates a pod, the containers within the pod run as unprivileged accounts.

2.2. Delete the pod.

```
[student@workstation ~]$ oc delete pod ubi9-user
pod "ubi9-user" deleted
```

2.3. Log in as the admin user with the redhatocp password.

```
[student@workstation ~]$ oc login -u admin -p redhatocp
Login successful.

You have access to 71 projects, the list has been suppressed. You can list all
projects with 'oc projects'

Using project "pods-containers".
```

2.4. Re-create the ubi9-user pod as the admin user. Configure the pod to execute the whoami and id commands through an interactive bash shell session. Compare the values of the UID and GID for the container user to the values of the ubi9-user pod that the developer user created.



**Note**

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

```
[student@workstation ~]$ oc run -it ubi9-user --restart 'Never' \
--image registry.ocp4.example.com:8443/ubi9/ubi \
-- /bin/bash -c "whoami && id"
Warning: would violate PodSecurity "restricted:v1.24":
allowPrivilegeEscalation != false (container "ubi9-user" must set
securityContext.allowPrivilegeEscalation=false), unrestricted capabilities
(container "ubi9-user" must set securityContext.capabilities.drop=["ALL"]),
runAsNonRoot != true (pod or container "ubi9-user" must set
securityContext.runAsNonRoot=true), seccompProfile (pod or container "ubi9-user"
must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
root
uid=0(root) gid=0(root) groups=0(root)
```

Notice that the value of the UID is 0, which differs from the UID range value of the pods-containers project. The user in the container is the privileged account root user and belongs to the root group. When a cluster administrator creates a pod, the containers within the pod run as a privileged account by default.

2.5. Delete the ubi9-user pod.

```
[student@workstation ~]$ oc delete pod ubi9-user
pod "ubi9-user" deleted
```

- 3. As the `developer` user, use the `oc run` command to create a `ubi9-date` pod from a UBI9 base container image. The image is available in the `registry.ocp4.example.com:8443/ubi9/ubi` container registry. Set the restart policy to Never, and configure the pod to execute the `date` command. Retrieve the logs of the `ubi9-date` pod to confirm that the `date` command executed. Delete the pod afterward.

3.1. Log in as the `developer` user with the `developer` password.

```
[student@workstation ~]$ oc login -u developer -p developer
Login successful.

You have one project on this server: "pods-containers"

Using project "pods-containers".
```

3.2. Create a pod called `ubi9-date` that executes the `date` command.

```
[student@workstation ~]$ oc run ubi9-date --restart 'Never' \
--image registry.ocp4.example.com:8443/ubi9/ubi -- date
pod/ubi9-date created
```

3.3. Wait a few moments for the creation of the pod. Then, retrieve the logs of the `ubi9-date` pod.

```
[student@workstation ~]$ oc logs ubi9-date
Mon Nov 28 15:02:55 UTC 2022
```

3.4. Delete the `ubi9-date` pod.

```
[student@workstation ~]$ oc delete pod ubi9-date
pod "ubi9-date" deleted
```

- 4. Use the `oc run ubi9-command -it` command to create a `ubi9-command` pod with the `registry.ocp4.example.com:8443/ubi9/ubi` container image. Add the `/bin/bash` in the `oc run` command to start an interactive shell. Exit the pods and view the logs for the `ubi9-command` pod with the `oc logs` command. Then, connect to the `ubi9-command` pod with the `oc attach` command, and issue the following command:

```
while true; do echo $(date); sleep 2; done
```

This command executes the `date` and `sleep` commands to generate output to the console every two seconds. Use the `oc logs` command to retrieve the logs of the `ubi9` pod, and confirm that the logs display the executed `date` and `sleep` commands.

4.1. Create a pod called `ubi9-command` and start an interactive shell.

```
[student@workstation ~]$ oc run ubi9-command -it \
--image registry.ocp4.example.com:8443/ubi9/ubi -- /bin/bash
If you don't see a command prompt, try pressing enter.
bash-5.1$
```

- 4.2. Exit the shell session.

```
bash-5.1$ exit
exit
Session ended, resume using 'oc attach ubi9-command -c ubi9-command -i -t' command
when the pod is running
```

- 4.3. Use the `oc logs` command to view the logs of the `ubi9-command` pod.

```
[student@workstation ~]$ oc logs ubi9-command
bash-5.1$ [student@workstation ~]$
```

The pod's command prompt is returned. The `oc logs` command displays the pod's current `stdout` and `stderr` output in the console. Because you disconnected from the interactive session, the pod's current `stdout` is the command prompt, and not the commands that you executed previously.

- 4.4. Use the `oc attach` command to connect to the `ubi9-command` pod again. In the shell, execute the `while true; do echo $(date); sleep 2; done` command to continuously generate `stdout` output.

```
[student@workstation ~]$ oc attach ubi9-command -it
If you don't see a command prompt, try pressing enter.
```

```
bash-5.1$ while true; do echo $(date); sleep 2; done
Mon Nov 28 15:15:16 UTC 2022
Mon Nov 28 15:15:18 UTC 2022
Mon Nov 28 15:15:20 UTC 2022
Mon Nov 28 15:15:22 UTC 2022
...output omitted...
```

- 4.5. Open another terminal window and view the logs for the `ubi9-command` pod with the `oc logs` command. Limit the log output to the last 10 entries with the `--tail` option. Confirm that the logs display the results of the command that you executed in the container.

```
[student@workstation ~]$ oc logs ubi9-command --tail=10
Mon Nov 28 15:15:16 UTC 2022
Mon Nov 28 15:15:18 UTC 2022
Mon Nov 28 15:15:20 UTC 2022
Mon Nov 28 15:15:22 UTC 2022
Mon Nov 28 15:15:24 UTC 2022
Mon Nov 28 15:15:26 UTC 2022
Mon Nov 28 15:15:28 UTC 2022
```

```
Mon Nov 28 15:15:30 UTC 2022
Mon Nov 28 15:15:32 UTC 2022
Mon Nov 28 15:15:34 UTC 2022
```

- 5. Identify the name for the container in the `ubi9-command` pod. Identify the process ID (PID) for the container in the `ubi9-command` pod by using a debug pod for the pod's host node. Use the `crlctl` command to identify the PID of the container in the `ubi9-command` pod. Then, retrieve the PID of the container in the debug pod.
- 5.1. Identify the container name in the `ubi9-command` pod with the `oc get` command. Specify the JSON format for the command output. Parse the JSON output with the `jq` command to retrieve the value of the `.status.containerStatuses[].name` object.

```
[student@workstation ~]$ oc get pod ubi9-command -o json | \
  jq .status.containerStatuses[].name
"ubi9-command"
```

The `ubi9-command` pod has a single container of the same name.

- 5.2. Find the host node for the `ubi9-command` pod. Start a debug pod for the host with the `oc debug` command.

```
[student@workstation ~]$ oc get pods ubi9-command -o wide
NAME           READY STATUS   RESTARTS   AGE   IP          NODE      NOMINATED NODE
ubi9-command   1/1   Running  2 (16m ago) 27m  10.8.0.26  master01 <none>
<none>
```

```
[student@workstation ~]$ oc debug node/master01
Error from server (Forbidden): nodes "master01" is forbidden: User "developer"
cannot get resource "nodes" in API group "" at the cluster scope
```

The debug pod fails because the `developer` user does not have the required permission to debug a host node.

- 5.3. Log in as the `admin` user with the `redhatocp` password. Start a debug pod for the host with the `oc debug` command. After connecting to the debug pod, run the `chroot /host` command to use host binaries, such as the `crlctl` command-line tool.

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

```
[student@workstation ~]$ oc debug node/master01
Starting pod/master01-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter
```

```
sh-4.4# chroot /host
```

- 5.4. Use the `cricctl ps` command to retrieve the `ubi9-command` container ID. Specify the `ubi9-command` container with the `--name` option and use the JSON output format. Parse the JSON output with the `jq -r` command to get the RAW JSON output. Export the container ID as the `$CID` environment variable.

**Note**

When using `jq` without the `-r` flag, the container ID is wrapped in double quotes, which does not work with `cricctl` commands. If the `-r` flag is not used, then you can add `| tr -d '"'` to the end of the command to trim the double quotes.

```
sh-5.1# crictl ps --name ubi9-command -o json | jq -r .containers[0].id
81adbc6222d79ed9ba195af4e9d36309c18bb71bc04b2e8b5612be632220e0d6
```

```
sh-5.1# CID=$(cricctl ps --name ubi9-command -o json | jq -r .containers[0].id)
```

```
sh-5.1# echo $CID
```

```
81adbc6222d79ed9ba195af4e9d36309c18bb71bc04b2e8b5612be632220e0d6
```

Your container ID value might differ from the previous output.

- 5.5. Use the `cricctl inspect` command to find the PID of the `ubi9-command` container. The PID value is in the `.info.pid` object in the `cricctl inspect` output. Export the `ubi9-command` container PID as the `$PID` environment variable.

```
sh-5.1# crictl inspect $CID | grep pid
"pid": 365297,
"pids": {
    "type": "pid"
...output omitted...
}
...output omitted...
```

```
sh-5.1# PID=365297
```

Your PID values might differ from the previous output.

- 6. Use the `lsns` command to list the system namespaces of the `ubi9-command` container. Confirm that the running processes in the container are isolated to different system namespaces.
- 6.1. View the system namespaces of the `ubi9-command` container with the `lsns` command. Specify the PID with the `-p` option and use the `$PID` environment variable. In the resulting table, the `NS` column contains the namespace values for the container.

```
sh-5.1# lsns -p $PID
      NS TYPE    NPROCS   PID USER        COMMAND
4026531835 cgroup     540      1 root      /usr/lib/systemd/systemd --switched-
root --system --deserialize 16
4026531837 user      540      1 root      /usr/lib/systemd/systemd --switched-
root --system --deserialize 16
4026536117 uts       1 153168 1000800000 /bin/bash
4026536118 ipc       1 153168 1000800000 /bin/bash
4026536120 net       1 153168 1000800000 /bin/bash
4026537680 mnt      1 153168 1000800000 /bin/bash
4026537823 pid      1 153168 1000800000 /bin/bash
```

Your namespace values might differ from the previous output.

- 7. Use the host debug pod to retrieve and compare the operating system (OS) and the GNU C Library (`glibc`) package version of the `ubi9-command` container and the host node.

- 7.1. Retrieve the OS for the host node with the `cat /etc/redhat-release` command.

```
sh-5.1# cat /etc/redhat-release
Red Hat Enterprise Linux CoreOS release 4.14
```

- 7.2. Use the `cricctl exec` command and the `$CID` container ID variable to retrieve the OS of the `ubi9-command` container. Use the `-it` options to create an interactive terminal to execute the `cat /etc/redhat-release` command.

```
sh-5.1# crictl exec -it $CID cat /etc/redhat-release
Red Hat Enterprise Linux release 9.1 (Plow)
```

The `ubi9-command` container has a different OS from the host node.

- 7.3. Use the `ldd --version` command to retrieve the `glibc` package version of the host node.

```
sh-5.1$ ldd --version
ldd (GNU libc) 2.28
Copyright (C) 2018 Free Software Foundation, Inc.
...output omitted...
```

- 7.4. Use the `cricctl exec` command and the `$CID` container ID variable to retrieve the `glibc` package version of the `ubi9-command` container. Use the `-it` options to create an interactive terminal to execute the `ldd --version` command.

```
sh-5.1# crictl exec -it $CID ldd --version
ldd (GNU libc) 2.34
Copyright (C) 2021 Free Software Foundation, Inc.
...output omitted...
```

The `ubi9-command` container has a different version of the `glibc` package from its host.

- 8. Exit the `master01-debug` pod and the `ubi9-command` pod.

- 8.1. Exit the `master01-debug` pod. You must issue the `exit` command to end the host binary access. Execute the `exit` command again to exit and remove the `master01-debug` pod.

```
sh-5.1# exit  
exit
```

```
sh-4.4# exit  
exit  
  
Removing debug pod ...  
Temporary namespace openshift-debug-bg7kn was removed.
```

- 8.2. Return to the terminal window that is connected to the `ubi9-command` pod. Press `Ctrl+C` and then execute the `exit` command. Confirm that the pod is still running.

```
...output omitted...  
^C  
bash-5.1$ exit  
exit  
Session ended, resume using 'oc attach ubi9-command -c ubi9-command -i -t' command  
when the pod is running
```

```
[student@workstation ~]$ oc get pods  
NAME          READY   STATUS    RESTARTS   AGE  
ubi9-command   1/1     Running   2 (6s ago)  35m
```

## 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 pods-containers
```

# Find and Inspect Container Images

---

## Objectives

- Find containerized applications in container registries and get information about the runtime parameters of supported and community container images.

## Container Image Overview

A container is an isolated runtime environment where applications are executed as isolated processes. The isolation of the runtime environment ensures that applications do not interfere with other containers or system processes.

A container image contains a packaged version of your application, with all the necessary dependencies for the application to run. Images can exist without containers. However, containers depend on images, because containers use container images to build a runtime environment to execute applications.

Containers can be split into two similar but distinct concepts: *container images* and *container instances*. A *container image* contains immutable data that defines an application and its libraries. You can use container images to create *container instances*, which are running processes that are isolated by a set of kernel namespaces.

You can use each container image many times to create many distinct container instances. These replicas can be split across multiple hosts. The application within a container is independent of the host environment.

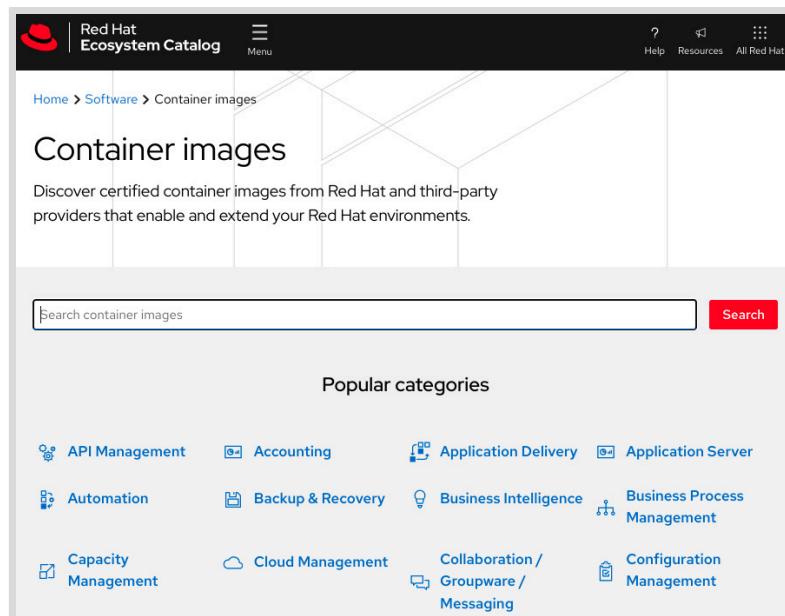
## Container Image Registries

Image registries are services that offer container images to download. Image creators and maintainers can store and distribute container images in a controlled manner to public or private audiences. Some examples of image registries include Quay.io, Red Hat Registry, Docker Hub, and Amazon ECR.

## Red Hat Registry

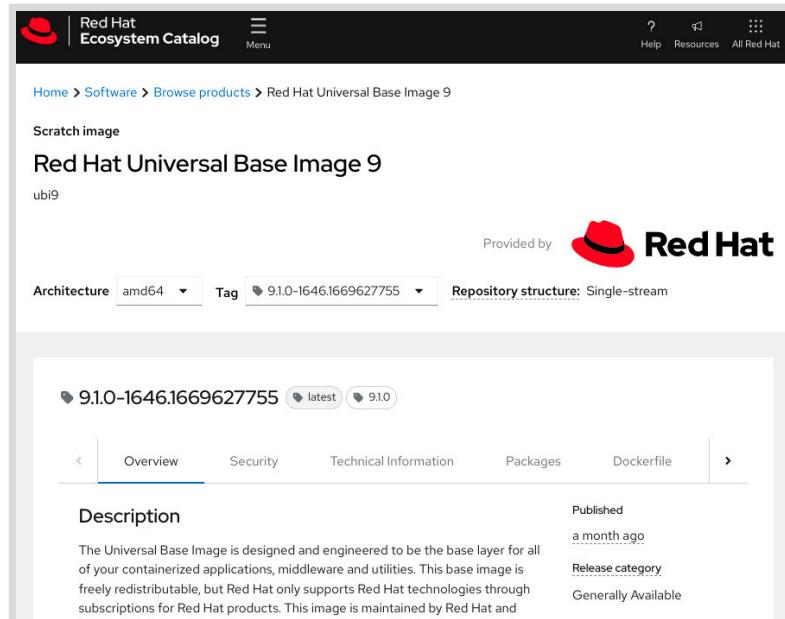
Red Hat distributes container images by using two registries: `registry.access.redhat.com` (where no authentication is required), and `registry.redhat.io` (where authentication is required). The Red Hat Ecosystem Catalog, <https://catalog.redhat.com/>, provides centralized searching utility for both registries. You can search the Red Hat Ecosystem Catalog for technical details about container images. The catalog hosts a large set of container images, including from major open source projects, such as Apache, MySQL, and Jenkins.

Because the Red Hat Ecosystem Catalog is also searched for software products other than container images, you must navigate to <https://catalog.redhat.com/software/containers/explore> to specifically search for container images.



**Figure 3.4: Red Hat Ecosystem Catalog**

The details page of a container image gives relevant information, such as technical data, the installed packages within the image, or a security scan. You can navigate through these options by using the tabs on the website. You can also change the image version by selecting a specific tag.



**Figure 3.5: The Red Hat Universal Base Image 9**

The Red Hat internal security team vets all images in the container catalog. Red Hat rebuilds all components to avoid known security vulnerabilities.

Red Hat container images provide the following benefits:

- Trusted source: All container images use sources that Red Hat knows and trusts.
- Original dependencies: None of the container packages are tampered with, and include only known libraries.

- Vulnerability-free: Container images are free of known critical vulnerabilities in the platform components or layers.
- Runtime protection: All applications in container images run as non-root users, to minimize the exposure surface to malicious or faulty applications.
- Red Hat Enterprise Linux (RHEL) compatible: Container images are compatible with all RHEL platforms, from bare metal to cloud.
- Red Hat support: Red Hat commercially supports the complete stack.



**Note**

You must log in to the `registry.redhat.io` registry with a customer portal account or a Red Hat Developer account to use the stored container images in the registry.

## Quay.io

Although the Red Hat Registry stores only images from Red Hat and certified providers, you can store your own images with Quay.io, another public image registry that Red Hat sponsors. Although storing public images in Quay is free of charge, some options are available only for paying customers. Quay also offers an on-premise version of the product, which you can use to set up an image registry in your own servers.

Quay.io introduces features such as server-side image building, fine-grained access controls, and automatic scanning of images for known vulnerabilities.

Quay.io offers live images that creators regularly update. Quay.io users can create their namespaces, with fine-grained access control, and publish their created images to that namespace. Container Catalog users rarely or never push new images, but consume trusted images from the Red Hat team.



**Figure 3.6: The Quay.io welcome page**

## Private Registries

Image creators or maintainers might want to make their images publicly available. However, other image creators might prefer to keep their images private, for the following reasons:

- Company privacy and secret protection
- Legal restrictions and laws

- Avoidance of publishing images in development

In some cases, private images are preferred. Private registries give image creators control over image placement, distribution, and usage. Private images are more secure than images in public registries.

## Public Registries

Other public registries, such as Docker Hub and Amazon ECR, are also available for storing, sharing, and consuming container images. These registries can include official images that the registry owners or the registry community users create and maintain. For example, Docker Hub hosts a Docker Official Image of a WordPress container image. Although the `docker.io/library/wordpress` container image is a Docker Official Image, the container image is not supported by WordPress, Docker, or Red Hat. Instead, the Docker Community, a global group of Docker Hub users, supports and maintains the container image. Support for this container image depends on the availability and skills of the Docker Community users.

Consuming container images from public registries brings risks. For example, a container image might include malicious code or vulnerabilities, which can compromise the host system that executes the container image. A host system can also be compromised by public container images, because the images are often configured with the privileged `root` user. Additionally, the software in a container image might not be correctly licensed, or might violate licensing terms.

Before you use a container image from a public registry, review and verify the container image. Also ensure that you have the correct permissions to use the software in the container image.

## Container Image Identifiers

Several objects provide identifying information about a container image.

### Registry

It is a content server, such as `registry.access.redhat.com`, that is used to store and share container images. A registry consists of one or more repositories that contain tagged container images.

### Name

It identifies the container image repository; it is a string that is composed of letters, numbers, and some special characters. This component refers to the name of the directory, or the container repository, within the container registry where the container image is.

For example, consider the fully qualified domain name (FQDN) of the `registry.access.redhat.com/ubi9/httpd-24:1-233` container image. The container image is in the `ubi9/httpd-24` repository in the `registry.access.redhat.com` container registry.

### ID/Hash

It is the SHA (Secure Hash Algorithm) code to pull or verify an image. The SHA image ID cannot change, and always references the same container image content. The ID/hash is the true, unique identifier of an image. For example, the `sha256:4186a1ead13fc30796f951694c494e7630b82c320b81e20c020b3b07c888985b` image ID always refers to the `registry.access.redhat.com/ubi9/httpd-24:1-233` container image.

### Tag

It is a label for a container image in a repository, to distinguish from other images, for version control. The tag comes after the image repository name and is delimited by a colon (:).

When an image tag is omitted, the floating tag, `latest`, is used as the default tag. A floating tag is an alias to another tag. In contrast, a fixed tag points to a specific container build. For the `registry.access.redhat.com/ubi9/httpd-24:1-233.1669634588` container image, `1-233.1669634588` is the fixed tag for the image, and at the time of writing, corresponds to the floating `latest` flag.

## Container Image Components

A container image is composed of multiple components.

### Layers

Container images are created from instructions. Each instruction adds a layer to the container image. Each layer consists of the differences between it and the following layer. The layers are then stacked to create a read-only container image.

### Metadata

Metadata includes the instructions and documentation for a container image.

## Container Image Instructions and Metadata

Container image layers consist of instructions, or steps, and metadata for building the image. You can override instructions during container creation to adjust the container image according to your needs. Some instructions can affect the running container, and other instructions are for informational purposes only.

The following instructions affect the state of a running container:

### ENV

Defines the available environment variables in the container. A container image might include multiple ENV instructions. Any container can recognize additional environment variables that are not listed in its metadata.

### ARG

It defines build-time variables, typically to make a customizable container build. Developers commonly configure the ENV instructions by using the ARG instruction. It is useful for preserving the build-time variables for run time.

### USER

Defines the active user in the container. Later instructions run as this user. It is a good practice to define a user other than `root` for security purposes. OpenShift does not honor the user in a container image, for regular cluster users. Only cluster administrators can run containers (pods) with their chosen `user ID (UIDs)` and `group IDs (GIDs)`.

### ENTRYPOINT

It defines the executable to run when the container is started.

### CMD

It defines the command to execute when the container is started. This command is passed to the executable that the ENTRYPOINT instruction defines. Base images define a default ENTRYPOINT executable, which is usually a shell executable, such as Bash.

### WORKDIR

It sets the current working directory within the container. Later instructions execute within this directory.

Metadata is used for documentation purposes, and does not affect the state of a running container. You can also override the metadata values during container creation.

The following metadata is for information only, and does not affect the state of the running container:

#### **EXPOSE**

It indicates the network port that the application binds to within the container. This metadata does not automatically bind the port on the host, and is used only for documentation purposes.

#### **VOLUME**

It defines where to store data outside the container. The value shows the path where your container runtime mounts the directory inside the container. More than one path can be defined to create multiple volumes.

#### **LABEL**

Adds a key-value pair to the metadata of the image for organization and image selection.

Container engines are not required to honor metadata in a container image, such as USER or EXPOSE. A container engine can also recognize additional environment variables that are not listed in the container image metadata.

## **Base Images**

A *base image* is the image that your resulting container image is built on. Your chosen base image determines the Linux distribution, and any of the following components:

- Package manager
- Init system
- File system layout
- Preinstalled dependencies and runtimes

The base image can also influence factors such as image size, vendor support, and processor compatibility.

Red Hat provides enterprise-grade container images that are engineered to be the base operating system layer for your containerized applications. These container images are intended as a common starting point for containers, and are known as *universal base images* (UBI). Red Hat UBI container images are *Open Container Initiative (OCI)* compliant images that contain portions of Red Hat Enterprise Linux (RHEL). UBI container images include a subset of RHEL content. They provide a set of prebuilt runtime languages, such as Python and Node.js, and associated DNF repositories that you can use to add application dependencies. UBI-based images can be distributed without cost or restriction. They can be deployed to both Red Hat and non-Red Hat platforms, and be pushed to your chosen container registry.

A Red Hat subscription is not required to use or distribute UBI-based images. However, Red Hat provides full support only for containers that are built on UBI if the containers are deployed to a Red Hat platform, such as a Red Hat OpenShift Container Platform (RHOC) cluster or RHEL.

Red Hat provides four UBI variants: `standard`, `init`, `minimal`, and `micro`. All UBI variants and UBI-based images use Red Hat Enterprise Linux (RHEL) at their core and are available from the Red Hat Container Catalog. The main differences are as follows:

#### **Standard**

This image is the primary UBI, which includes DNF, systemd, and utilities such as `gzip` and `tar`.

#### **Init**

This image simplifies running multiple applications within a single container by managing them with systemd.

### Minimal

This image is smaller than the `init` image and provides nice-to-have features. This image uses the `microdnf` minimal package manager instead of the full-sized version of DNF.

### Micro

This image is the smallest available UBI, and includes only the minimum packages. For example, this image does not include a package manager.

## Inspecting and Managing Container Images

Various tools can inspect and manage container images, including the `oc image` command and Skopeo.

### Skopeo

Skopeo is another tool to inspect and manage remote container images. With Skopeo, you can copy and sync container images from different container registries and repositories. You can also copy an image from a remote repository and save it to a local disk. If you have the appropriate repository permissions, then you can also delete an image from container registry. You also can use Skopeo to inspect the configuration and contents of a container image, and to list the available tags for a container image. Unlike other container image tools, Skopeo can execute without a privileged account, such as `root`. Skopeo does not require a running daemon to execute various operations.

Skopeo is executed with the `skopeo` command-line utility, which you can install with various package managers, such as DNF, Brew, and APT. The `skopeo` utility might already be installed on some Linux-based distributions. You can install the `skopeo` utility on Fedora, CentOS Stream 8 and later, and Red Hat Enterprise Linux 8 and later systems by using the DNF package manager.

```
[user@host ~]$ sudo dnf -y install skopeo
```

The `skopeo` utility is currently not available as a packaged binary for Windows-based systems. However, the `skopeo` utility is available as a container image from the `quay.io/skopeo/stable` container repository. For more information about the Skopeo container image, refer to the `skopeoimage` overview guide in the Skopeo repository (<https://github.com/containers/skopeo/blob/main/contrib/skopeoimage/README.md>).

You can also build `skopeo` from source code in a container, or build it locally without using a container. Refer to the installation guide in the Skopeo repository (<https://github.com/containers/skopeo/blob/main/install.md#container-images>) for more information about installing or building Skopeo from source code.

The `skopeo` utility provides commands to help you to manage and inspect container images and container image registries. For container registries that require authentication, you must first log in to the registry before you can execute additional `skopeo` commands.

```
[user@host ~]$ skopeo login quay.io
```

**Note**

OpenShift clusters are typically configured with registry credentials. When a pod is created from a container image in a remote repository, OpenShift authenticates to the container registry with the configured registry credentials, and then pulls, or copies, the image. Because OpenShift automatically uses the registry credentials, you typically do not need to manually authenticate to a container registry when you create a pod. By contrast, the `oc image` command and the `skopeo` utility require you first to log in to a container registry.

After you log in to a container registry (if required), you can execute additional `skopeo` commands against container images in a repository. When you execute a `skopeo` command, you must specify the transport and the repository name. A *transport* is the mechanism to transfer or move container images between locations. Two common transports are `docker` and `dir`. The `docker` transport is used for container registries, and the `dir` transport is used for local directories.

The `oc image` command and other tools default to the `docker` transport, and so you do not need to specify the transport when executing commands. However, the `skopeo` utility does not define a default transport; you must specify the transport with the container image name. Most `skopeo` commands use the `skopeo command [command options] transport://IMAGE-NAME` format. For example, the following `skopeo list-tags` command lists all available tags in a `registry.access.redhat.com/ubi9/httpd-24` container repository by using the `docker` transport:

```
[user@host ~]$ skopeo list-tags docker://registry.access.redhat.com/ubi9/httpd-24
{
    "Repository": "registry.access.redhat.com/ubi9/httpd-24",
    "Tags": [
        "1-229",
        "1-217.1666632462",
        "1-201",
        "1-194.1655192191-source",
        "1-229-source",
        "1-194-source",
        "1-201-source",
        "1-217.1664813224",
        "1-217.1664813224-source",
        "1-210-source",
        "1-233.1669634588",
        "1",
        "1-217.1666632462-source",
        "1-233",
        "1-194.1655192191",
        "1-217.1665076049-source",
        "1-217",
        "1-233.1669634588-source",
        "1-210",
        "1-217.1665076049",
        "1-217-source",
        "1-233-source",
        "1-194",
```

```

        "latest"
    ]
}
```

The `skopeo` utility includes other useful commands for container image management.

### `skopeo inspect`

View low-level information for an image name, such as environment variables and available tags. Use the `skopeo inspect [command options] transport://IMAGE-NAME` command format. You can include the `--config` flag to view the configuration, metadata, and history of a container repository. The following example retrieves the configuration information for the `registry.access.redhat.com/ubi9/httpd-24` container repository:

```
[user@host ~]$ skopeo inspect --config
docker://registry.access.redhat.com/ubi9/httpd-24
...output omitted...
{
  "config": {
    "User": "1001",
    "ExposedPorts": {
      "8080/tcp": {},
      "8443/tcp": {}
    },
    "Env": [
      ...output omitted...
      "HTTPD_MAIN_CONF_PATH=/etc/httpd/conf",
      "HTTPD_MAIN_CONF_MODULES_D_PATH=/etc/httpd/conf.modules.d",
      "HTTPD_MAIN_CONF_D_PATH=/etc/httpd/conf.d",
      "HTTPD_TLS_CERT_PATH=/etc/httpd/tls",
      "HTTPD_VAR_RUN=/var/run/httpd",
      "HTTPD_DATA_PATH=/var/www",
      "HTTPD_DATA_ORIG_PATH=/var/www",
      "HTTPD_LOG_PATH=/var/log/httpd"
    ],
    "Entrypoint": [
      "container-entrypoint"
    ],
    "Cmd": [
      "/usr/bin/run-httpd"
    ],
    "WorkingDir": "/opt/app-root/src",
    ...output omitted...
  }
}
...output omitted...
```

### `skopeo copy`

Copy an image from one location or repository to another. Use the `skopeo copy transport://SOURCE-IMAGE transport://DESTINATION-IMAGE` format. For example, the following command copies the `quay.io/skopeo/stable:latest` container image to the `skopeo` repository in the `registry.example.com` container registry:

```
[user@host ~]$ skopeo copy docker://quay.io/skopeo/stable:latest \
docker://registry.example.com/skopeo:latest
```

**skopeo delete**

Delete a container image from a repository. You must use the `skopeo delete [command options] transport://IMAGE-NAME` format. The following command deletes the `skopeo:latest` image from the `registry.example.com` container registry:

```
[user@host ~]$ skopeo delete docker://registry.example.com/skopeo:latest
```

**skopeo sync**

Synchronize one or more images from one location to another. Use this command to copy all container images from a source to a destination. The command uses the `skopeo sync [command options] --src transport --dest transport SOURCE DESTINATION` format. The following command synchronizes the `registry.access.redhat.com/ubi8/httpd-24` container repository to the `registry.example.com/httpd-24` container repository:

```
[user@host ~]$ skopeo sync --src docker --dest docker \
registry.access.redhat.com/ubi8/httpd-24 registry.example.com/httpd-24
```

## Registry Credentials

Some registries require users to authenticate. For example, Red Hat containers that are based on RHEL typically require authenticated access:

```
[user@host ~]$ skopeo inspect docker://registry.redhat.io/rhel8/httpd-24
FATA[0000] Error parsing image name "docker://registry.redhat.io/rhel8/
httpd-24": unable to retrieve auth token: invalid username/password:
unauthorized: Please login to the Red Hat Registry using your Customer Portal
credentials. Further instructions can be found here: https://access.redhat.com/
RegistryAuthentication
```

You might choose a different image that does not require authentication, such as the UBI 8 image:

```
[user@host ~]$ skopeo inspect docker://registry.access.redhat.com/ubi8:latest
{
  "Name": "registry.access.redhat.com/ubi8",
  "Digest": "sha256:70fc...1173",
  "RepoTags": [
    "8.7-1054-source",
    "8.6-990-source",
    "8.6-754",
    "8.4-203.1622660121-source",
    ...output omitted....
```

Alternatively, you must execute the `skopeo login` command for the registry before you can access the RHEL 8 image.

```
[user@host ~]$ skopeo login registry.redhat.io
Username: YOUR_USER
Password: YOUR_PASSWORD
Login Succeeded!
[user@host ~]$ skopeo list-tags docker://registry.redhat.io/rhel8/httpd-24
{
```

```
"Repository": "registry.redhat.io/rhel8/httpd-24",
"Tags": [
    "1-166.1645816922",
    "1-209",
    "1-160-source",
    "1-112",
    ...output omitted...
```

Skopeo stores the credentials in the  `${XDG_RUNTIME_DIR}/containers/auth.json` file, where the  `${XDG_RUNTIME_DIR}` refers to a directory that is specific to the current user. The credentials are encoded in the base64 format:

```
[user@host ~]$ cat ${XDG_RUNTIME_DIR}/containers/auth.json
{
  "auths": {
    "registry.redhat.io": {
      "auth": "dXNlcjpodW50ZXIy"
    }
  }
}
[user@host ~]$ echo -n dXNlcjpodW50ZXIy | base64 -d
user:hunter2
```



### Note

For security reasons, the `skopeo login` command does not show your password in the interactive session. Although you do not see what you are typing, Skopeo registers every key stroke. After typing your full password in the interactive session, press `Enter` to start the login.

## The `oc image` Command

The OpenShift command-line interface provides the `oc image` command. You can use this command to inspect, configure, and retrieve information about container images.

The `oc image info` command inspects and retrieves information about a container image. You can use the `oc image info` command to identify the ID/hash SHA and to list the image layers of a container image. You can also review container image metadata, such as environment variables, network ports, and commands. If a container image repository provides a container image in multiple architectures, such as `amd64` or `arm64`, then you must include the `--filter-by-os` tag. For example, you can execute the following command to retrieve information about the `registry.access.redhat.com/ubi9/httpd-24:1-233` container image that is based on the `amd64` architecture:

```
[user@host ~]$ oc image info registry.access.redhat.com/ubi9/httpd-24:1-233 \
--filter-by-os amd64
Name:      registry.access.redhat.com/ubi9/httpd-24:1-233
Digest:    sha256:4186...985b
...output omitted...
Image Size: 130.8MB in 3 layers
Layers:    79.12MB sha256:d74e...1cad
           17.32MB sha256:dac0...a283
           34.39MB sha256:47d8...5550
```

```

OS:          linux
Arch:        amd64
Entrypoint:  container-entrypoint
Command:     /usr/bin/run-httpd
Working Dir: /opt/app-root/src
User:         1001
Expose Ports: 8080/tcp, 8443/tcp
Environment: container=oci
...output omitted...
HTTPD_CONTAINER_SCRIPTS_PATH=/usr/share/container-scripts/httpd/
HTTPD_APP_ROOT=/opt/app-root
HTTPD_CONFIGURATION_PATH=/opt/app-root/etc/httpd.d
HTTPD_MAIN_CONF_PATH=/etc/httpd/conf
HTTPD_MAIN_CONF_MODULES_D_PATH=/etc/httpd/conf.modules.d
HTTPD_MAIN_CONF_D_PATH=/etc/httpd/conf.d
HTTPD_TLS_CERT_PATH=/etc/httpd/tls
HTTPD_VAR_RUN=/var/run/httpd
HTTPD_DATA_PATH=/var/www
HTTPD_DATA_ORIG_PATH=/var/www
HTTPD_LOG_PATH=/var/log/httpd
...output omitted...

```

The `oc image` command provides more options to manage container images.

#### **oc image append**

Use this command to add layers to container images, and then push the container image to a registry.

#### **oc image extract**

You can use this command to extract or copy files from a container image to a local disk. Use this command to access the contents of a container image without first running the image as a container. A running container engine is not required.

#### **oc image mirror**

Copy or mirror container images from one container registry or repository to another. For example, you can use this command to mirror container images between public and private registries. You can also use this command to copy a container image from a registry to a disk. The command mirrors the HTTP structure of a container registry to a directory on a disk. The directory on the disk can then be served as a container registry.

## Running Containers as Root

Running containers as the root user is a security risk, because an attacker could exploit the application, access the container, and exploit further vulnerabilities to escape from the containerized environment into the host system. Attackers might escape the containerized environment by exploiting bugs and vulnerabilities that are typically in the kernel or container runtime.

Traditionally, when an attacker gains access to the container file system by using an exploit, the root user inside the container corresponds to the root user outside the container. If an attacker escapes the container isolation, then they have elevated privileges on the host system, which potentially causes more damage.

Containers that do not run as the root user have limitations that might prove unsuitable for use in your application, such as the following limitations:

## Non-trivial Containerization

Some applications might require the root user. Depending on the application architecture, some applications might not be suitable for non-root containers, or might require a deeper understanding to containerize.

For example, applications such as HTTPd and Nginx start a bootstrap process and then create a process with a non-privileged user, which interacts with external users. Such applications are non-trivial to containerize for rootless use.

Red Hat provides containerized versions of HTTPd and Nginx that do not require root privileges for production usage. You can find the containers in the Red Hat container registry (<https://catalog.redhat.com/software/containers/explore>).

## Required Use of Privileged Utilities

Non-root containers cannot bind to privileged ports, such as the 80 or 443 ports. Red Hat advises against using privileged ports, but to use port forwarding instead.

Similarly, non-root containers cannot use the ping utility by default, because it requires elevated privileges to establish raw sockets.



### References

#### Skopeo GitHub Repository

<https://github.com/containers/skopeo>  
skopeo(1) man page

## ► Guided Exercise

# Find and Inspect Container Images

Use a supported MySQL container image to run server and client pods in Kubernetes; also test two community images and compare their runtime requirements.

### Outcomes

- Locate and run container images from a container registry.
- Inspect remote container images and container logs.
- Set environment variables and override entry points for a container.
- Access files and directories within a container.

### 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 pods-images
```

### Instructions

- 1. Log in to the OpenShift cluster and create the `pods-images` project.
- 1.1. Log in to the OpenShift cluster as the `developer` user with the `oc` command.

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

- 1.2. Create the `pods-images` project.
- ```
[student@workstation ~]$ oc new-project pods-images
...output omitted...
```
- 2. Authenticate to `registry.ocp4.example.com:8443`, which is the classroom container registry. This private registry hosts certain copies and tags of community images from Docker and Bitnami, as well as some supported images from Red Hat. Use `skopeo` to log in as the `developer` user, and then retrieve a list of available tags for the `registry.ocp4.example.com:8443/redhattraining/docker-nginx` container repository.
- 2.1. Use the `skopeo login` command to log in as the `developer` user with the `developer` password.

```
[student@workstation ~]$ skopeo login registry.ocp4.example.com:8443
Username: developer
Password: developer
Login Succeeded!
```

- 2.2. The classroom registry contains a copy and specific tags of the docker.io/library/nginx container repository. Use the `skopeo list-tags` command to retrieve a list of available tags for the `registry.ocp4.example.com:8443/redhattraining/docker-nginx` container repository.

```
[student@workstation ~]$ skopeo list-tags \
  docker://registry.ocp4.example.com:8443/redhattraining/docker-nginx
{
  "Repository": "registry.ocp4.example.com:8443/redhattraining/docker-nginx",
  "Tags": [
    "1.23",
    "1.23-alpine",
    "1.23-perl",
    "1.23-alpine-perl",
    "latest"
  ]
}
```

- 3. Create a `docker-nginx` pod from the `registry.ocp4.example.com:8443/redhattraining/docker-nginx:1.23` container image. Investigate any pod failures.

- 3.1. Use the `oc run` command to create the `docker-nginx` pod.

```
[student@workstation ~]$ oc run docker-nginx \
  --image registry.ocp4.example.com:8443/redhattraining/docker-nginx:1.23
pod/docker-nginx created
```

- 3.2. After a few moments, verify the status of the `docker-nginx` pod.

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
docker-nginx  0/1     Error      0          4s
```

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS             RESTARTS   AGE
docker-nginx  0/1     CrashLoopBackOff  2 (17s ago)  38s
```

The `docker-nginx` pod failed to start.

- 3.3. Investigate the pod failure. Retrieve the logs of the `docker-nginx` pod to identify a possible cause of the pod failure.

```
[student@workstation ~]$ oc logs docker-nginx
...output omitted...
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/12/02 18:51:45 [warn] 1#1: the "user" directive makes sense only if
the master process runs with super-user privileges, ignored in /etc/nginx/
nginx.conf:2
nginx: [warn] the "user" directive makes sense only if the master process runs
with super-user privileges, ignored in /etc/nginx/nginx.conf:2
2022/12/02 18:51:45 [emerg] 1#1: mkdir() "/var/cache/nginx/client_temp" failed
(13: Permission denied)
nginx: [emerg] mkdir() "/var/cache/nginx/client_temp" failed (13: Permission
denied)
```

The pod failed to start because of permission issues for the nginx directories.

3.4. Create a debug pod for the docker-nginx pod.

```
[student@workstation ~]$ oc debug pod/docker-nginx
Starting pod/docker-nginx-debug ...
Pod IP: 10.8.0.72
If you don't see a command prompt, try pressing enter.

$
```

3.5. From the debug pod, verify the permissions of the /etc/nginx and /var/cache/nginx directories.

```
$ ls -la /etc/ | grep nginx
drwxr-xr-x. 3 root root    132 Nov 15 13:14 nginx
```

```
$ ls -la /var/cache | grep nginx
drwxr-xr-x. 2 root root    6 Oct 19 09:32 nginx
```

Only the `root` user has permission to the nginx directories. The pod must therefore run as the privileged `root` user to work.

3.6. Retrieve the user ID (UID) of the docker-nginx user to determine whether the user is a privileged or unprivileged account. Then, exit the debug pod.

```
$ whoami
1000820000
```

```
$ exit
```

```
Removing debug pod ...
```

Your UID value might differ from the previous output.

A UID over 0 means that the container's user is a `non-root` account. Recall that OpenShift default security policies prevent regular user accounts, such as the `developer` user, from running pods and their containers as privileged accounts.

- 3.7. Confirm that the docker-nginx:1.23 image requires the `root` privileged account. Use the `skopeo inspect --config` command to view the configuration for the image.

```
[student@workstation ~]$ skopeo inspect --config \
  docker://registry.ocp4.example.com:8443/redhattraining/docker-nginx:1.23
...output omitted...
{
  "config": {
    "ExposedPorts": {
      "80/tcp": {}
    },
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "NGINX_VERSION=1.23.3",
      "NJS_VERSION=0.7.9",
      "PKG_RELEASE=1~bullseye"
    ],
    "Entrypoint": [
      "/docker-entrypoint.sh"
    ],
    "Cmd": [
      "nginx",
      "-g",
      "daemon off;"
    ],
    "Labels": {
      "maintainer": "NGINX Docker Maintainers \u003cdocker-maint@nginx.com\u003e"
    },
    "StopSignal": "SIGQUIT"
  },
  ...output omitted...
}
```

The image configuration does not define `USER` metadata, which confirms that the image must run as the `root` privileged user.

- 3.8. The `docker-nginx:1-23` container image must run as the `root` privileged user. OpenShift security policies prevent regular cluster users, such as the `developer` user, from running containers as the `root` user. Delete the `docker-nginx` pod.

```
[student@workstation ~]$ oc delete pod docker-nginx
pod "docker-nginx" deleted
```

- 4. Create a `bitnami-mysql` pod, which uses a copy of the Bitnami community MySQL image. The image is available in the `registry.ocp4.example.com:8443/redhattraining/bitnami-mysql` container repository.
- 4.1. A copy and specific tags of the `docker.io/bitnami/mysql` container repository are hosted in the classroom registry. Use the `skopeo list-tags` command to identify available tags for the Bitnami MySQL community image in the `registry.ocp4.example.com:8443/redhattraining/bitnami-mysql` container repository.

```
[student@workstation ~]$ skopeo list-tags \
  docker://registry.ocp4.example.com:8443/redhattraining/bitnami-mysql
{
  "Repository": "registry.ocp4.example.com:8443/redhattraining/bitnami-mysql",
  "Tags": [
    "8.0.31",
    "8.0.30",
    "8.0.29",
    "8.0.28",
    "latest"
  ]
}
```

- 4.2. Retrieve the configuration of the `bitnami-mysql:8.0.31` container image. Determine whether the image requires a privileged account by inspecting image configuration for USER metadata.

```
[student@workstation ~]$ skopeo inspect --config \
  docker://registry.ocp4.example.com:8443/redhattraining/bitnami-mysql:8.0.31
...output omitted...
{
  "config": {
    "User": "1001",
    "ExposedPorts": {
      "3306/tcp": {}
    },
    ...
  },
  ...output omitted...
```

The image defines the `1001` UID, which means that the image does not require a privileged account.

- 4.3. Create the `bitnami-mysql` pod with the `oc run` command. Use the `registry.ocp4.example.com:8443/redhattraining/bitnami-mysql:8.0.31` container image. Then, wait a few moments and then retrieve the pod's status with the `oc get` command.

```
[student@workstation ~]$ oc run bitnami-mysql \
  --image registry.ocp4.example.com:8443/redhattraining/bitnami-mysql:8.0.31
pod/bitnami-mysql created
```

| NAME          | READY | STATUS           | RESTARTS    | AGE |
|---------------|-------|------------------|-------------|-----|
| bitnami-mysql | 0/1   | CrashLoopBackoff | 2 (19s ago) | 23s |

The pod failed to start.

- 4.4. Examine the logs of the `bitnami-mysql` pod to determine the cause of the failure.

```
[student@workstation ~]$ oc logs bitnami-mysql
mysql 16:18:00.40
mysql 16:18:00.40 Welcome to the Bitnami mysql container
mysql 16:18:00.40 Subscribe to project updates by watching https://github.com/
bitnami/containers
```

```
mysql 16:18:00.40 Submit issues and feature requests at https://github.com/
bitnami/containers/issues
mysql 16:18:00.40
mysql 16:18:00.41 INFO  ==> ** Starting MySQL setup **
mysql 16:18:00.42 INFO  ==> Validating settings in MYSQL_*_MARIADB_* env vars
mysql 16:18:00.42 ERROR ==> The MYSQL_ROOT_PASSWORD environment variable is empty
or not set. Set the environment variable ALLOW_EMPTY_PASSWORD=yes to allow
the container to be started with blank passwords. This is recommended only for
development.
```

The `MYSQL_ROOT_PASSWORD` environment variable must be set for the pod to start.

- 4.5. Delete and then re-create the `bitnami-mysql` pod. Specify `redhat123` as the value for the `MYSQL_ROOT_PASSWORD` environment variable. After a few moments, verify the status of the pod.

```
[student@workstation ~]$ oc delete pod bitnami-mysql
pod "bitnami-mysql" deleted
```

```
[student@workstation ~]$ oc run bitnami-mysql \
--image registry.ocp4.example.com:8443/redhattraining/bitnami-mysql:8.0.31 \
--env MYSQL_ROOT_PASSWORD=redhat123
pod/bitnami-mysql created
```

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
bitnami-mysql  1/1     Running   0          20s
```

The `bitnami-mysql` pod successfully started.

- 4.6. Determine the UID of the container user in the `bitnami-mysql` pod. Compare this value to the UID in the container image and to the UID range of the `pods-images` project.

```
[student@workstation ~]$ oc exec -it bitnami-mysql -- /bin/bash -c "whoami && id"
1000820000
uid=1000820000(1000820000) gid=0(root) groups=0(root),1000820000
```

```
[student@workstation ~]$ oc describe project pods-images
Name:      pods-images
...output omitted...
Annotations: openshift.io/description=
...output omitted...
          openshift.io/sa.scc.supplemental-groups=1000820000/10000
          openshift.io/sa.scc.uid-range=1000820000/10000
...output omitted...
```

Your values for the UID of the container and the UID range of the project might differ from the previous output.

The container user UID is the same as the specified UID range in the namespace. Notice that the container user UID does not match the 1001 UID of the container

image. For a container to use the specified UID of a container image, the pod must be created with a privileged OpenShift user account, such as the `admin` user.

- ▶ 5. The private classroom registry hosts a copy of a supported MySQL image from Red Hat. Retrieve the list of available tags for the `registry.ocp4.example.com:8443/rhel9/mysql-80` container repository. Compare the `rhel9/mysql-80` container image release version that is associated with each tag.
- 5.1. Use the `skopeo list-tags` command to list the available tags for the `rhel9/mysql-80` container image.

```
[student@workstation ~]$ skopeo list-tags \
  docker://registry.ocp4.example.com:8443/rhel9/mysql-80
{
  "Repository": "registry.ocp4.example.com:8443/rhel9/mysql-80",
  "Tags": [
    "1-237",
    "1-228",
    "1-228-source",
    "1-224",
    "1-224-source",
    "latest",
    "1"
  ]
}
```

Several tags are available:

- The `latest` and `1` tags are floating tags, which are aliases to other tags, such as the `1-237` tag.
- The `1-228` and `1-224` tags are fixed tags, which point to a build of a container.
- The `1-228-source` and `1-224-source` tags are source containers, which provide the necessary sources and license terms to rebuild and distribute the images.

- 5.2. Use the `skopeo inspect` command to compare the `rhel9/mysql-80` container image release version and SHA IDs that are associated with the identified tags.



### Note

To improve readability, the instructions truncate the SHA-256 strings.

On your system, the commands return the full SHA-256 strings.

```
[student@workstation ~]$ skopeo inspect \
  docker://registry.ocp4.example.com:8443/rhel9/mysql-80:latest
...output omitted...
  "Name": "registry.ocp4.example.com:8443/rhel9/mysql-80",
  "Digest": "sha256:d282...f38f",
...output omitted...
  "Labels":
...output omitted...
```

```
"name": "rhel9/mysql-80",
"release": "237",
...output omitted...
```

You can also format the output of the `skopeo inspect` command with a Go template. Append the template objects with \n to add new lines between the results.

```
[student@workstation ~]$ skopeo inspect --format \
"Name: {{.Name}}\n Digest: {{.Digest}}\n Release: {{.Labels.release}}" \
docker://registry.ocp4.example.com:8443/rhel9/mysql-80:latest
Name: registry.ocp4.example.com:8443/rhel9/mysql-80
Digest: sha256:d282...f38f
Release: 237
```

```
[student@workstation ~]$ skopeo inspect --format \
"Name: {{.Name}}\n Digest: {{.Digest}}\n Release: {{.Labels.release}}" \
docker://registry.ocp4.example.com:8443/rhel9/mysql-80:1
Name: registry.ocp4.example.com:8443/rhel9/mysql-80
Digest: sha256:d282...f38f
Release: 237
```

```
[student@workstation ~]$ skopeo inspect --format \
"Name: {{.Name}}\n Digest: {{.Digest}}\n Release: {{.Labels.release}}" \
docker://registry.ocp4.example.com:8443/rhel9/mysql-80:1-237
Name: registry.ocp4.example.com:8443/rhel9/mysql-80
Digest: sha256:d282...f38f
Release: 237
```

The `latest`, `1`, and `1-237` tags resolve to the same release versions and SHA IDs.  
The `latest` and `1` tags are floating tags for the `1-237` fixed tag.

- ▶ 6. The classroom registry hosts a copy and certain tags of the `registry.redhat.io/rhel9/mysql-80` container repository. Use the `oc run` command to create a `rhel9-mysql` pod from the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-228` container image. Verify the status of the pod and then inspect the container logs for any errors.
  - 6.1. Create a `rhel9-mysql` pod with the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237` container image.

```
[student@workstation ~]$ oc run rhel9-mysql \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1-237
pod/rhel9-mysql created
```

- 6.2. After a few moments, retrieve the pod's status with the `oc get` command.

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS            RESTARTS   AGE
bitnami-mysql 1/1    Running          0          5m16s
rhel9-mysql   0/1    CrashLoopBackoff  2 (29s ago) 49s
```

The pod failed to start.

- 6.3. Retrieve the logs for the `rhel9-mysql` pod to determine why the pod failed.

```
[student@workstation ~]$ oc logs rhel9-mysql
=> sourcing 20-validate-variables.sh ...
You must either specify the following environment variables:
  MYSQL_USER (regex: '^[_a-zA-Z0-9]+$')
  MYSQL_PASSWORD (regex: '^[_a-zA-Z0-9~!@#$%^&*()-=;<,>,.?;:|]+$')
  MYSQL_DATABASE (regex: '^[_a-zA-Z0-9]+$')
Or the following environment variable:
  MYSQL_ROOT_PASSWORD (regex: '^[_a-zA-Z0-9~!@#$%^&*()-=;<,>,.?;:|]+$')
Or both.
Optional Settings:
  MYSQL_LOWER_CASE_TABLE_NAMES (default: 0)
...output omitted...
```

The pod failed because the required environment variables were not set for the container.

- ▶ 7. Delete the `rhel9-mysql` pod. Create another `rhel9-mysql` pod and specify the necessary environment variables. Retrieve the status of the pod and inspect the container logs to confirm that the new pod is working.
- 7.1. Delete the `rhel9-mysql` pod with the `oc delete` command. Wait for the pod to delete before continuing to the next step.

```
[student@workstation ~]$ oc delete pod rhel9-mysql
pod "rhel9-mysql" deleted
```

- 7.2. Create another `rhel9-mysql` pod from the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237` container image. Use the `oc run` command with the `--env` option to specify the following environment variables and their values:

| Variable       | Value     |
|----------------|-----------|
| MYSQL_USER     | redhat    |
| MYSQL_PASSWORD | redhat123 |
| MYSQL_DATABASE | worlidx   |

```
[student@workstation ~]$ oc run rhel9-mysql \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1-237 \
--env MYSQL_USER=redhat \
--env MYSQL_PASSWORD=redhat123 \
--env MYSQL_DATABASE=worlidx
pod/rhel9-mysql created
```

- 7.3. After a few moments, retrieve the status of the `rhel9-mysql` pod with the `oc get` command. View the container logs to confirm that the database on the `rhel9-mysql` pod is ready to accept connections.

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
bitnami-mysql 1/1     Running   0          10m
rhel9-mysql   1/1     Running   0          20s
```

```
[student@workstation ~]$ oc logs rhel9-mysql
...output omitted...
2022-11-02T20:14:14.333599Z 0 [System] [MY-011323] [Server] X Plugin ready for
connections. Bind-address: '::' port: 33060, socket: /var/lib/mysql/mysqlx.sock
2022-11-02T20:14:14.333641Z 0 [System] [MY-010931] [Server] /usr/libexec/
mysqld: ready for connections. Version: '8.0.30' socket: '/var/lib/mysql/
mysql.sock' port: 3306 Source distribution.
```

The `rhel9-mysql` pod is ready to accept connections.

- ▶ 8. Determine the location of the MySQL database files for the `rhel9-mysql` pod. Confirm that the directory contains the `worldx` database.
  - 8.1. Use the `oc image` command to inspect the `rhel9/mysql-80:1-228` image in the `registry.ocp4.example.com:8443` classroom registry.

```
[student@workstation ~]$ oc image info \
  registry.ocp4.example.com:8443/rhel9/mysql-80:1-237
Name:           registry.ocp4.example.com:8443/rhel9/mysql-80:1-237
...output omitted...
Command:        run-mysqld
Working Dir:    /opt/app-root/src
User:           27
Exposes Ports: 3306/tcp
Environment:    container=oci
                 STI_SCRIPTS_URL=image:///usr/libexec/s2i
                 STI_SCRIPTS_PATH=/usr/libexec/s2i
                 APP_ROOT=/opt/app-root
                 PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/
local/bin:/usr/sbin:/usr/bin:/sbin:/bin
                 PLATFORM=e19
                 MYSQL_VERSION=8.0
                 APP_DATA=/opt/app-root/src
                 HOME=/var/lib/mysql
```

The container manifest sets the `HOME` environment variable for the container user to the `/var/lib/mysql` directory.

- 8.2. Use the `oc exec` command to list the contents of the `/var/lib/mysql` directory.

```
[student@workstation ~]$ oc exec -it rhel9-mysql -- ls -la /var/lib/mysql
total 12
drwxrwxr-x. 1 mysql root          102 Nov  2 20:41 .
drwxr-xr-x. 1 root  root         19 Oct 24 18:47 ..
drwxrwxr-x. 1 mysql root        4096 Nov  2 20:54 data
srwxrwxrwx. 1 mysql 1000820000  0 Nov  2 20:41 mysql.sock
```

```
-rw----- 1 mysql 1000820000 2 Nov 2 20:41 mysql.sock.lock
srwxrwxrwx 1 mysql 1000820000 0 Nov 2 20:41 mysqlx.sock
-rw----- 1 mysql 1000820000 2 Nov 2 20:41 mysqlx.sock.lock
```

A data directory exists in the /var/lib/mysql directory.

- 8.3. Use the oc exec command again to list the contents of the /var/lib/mysql/data directory.

```
[student@workstation ~]$ oc exec -it rhel9-mysql \
-- ls -la /var/lib/mysql/data | grep worldx
drwxr-x--- 2 1000820000 root 6 Nov 2 20:41 worldx
```

The /var/lib/mysql/data directory contains the worldx database with the worldx directory.

- 9. Determine the IP address of the rhel9-mysql pod. Next, create another MySQL pod, named mysqlclient, to access the rhel9-mysql pod. Confirm that the mysqlclient pod can view the available databases on the rhel9-mysql pod with the mysqlshow command.

- 9.1. Identify the IP address of the rhel9-mysql pod.

```
[student@workstation ~]$ oc get pods rhel9-mysql -o json | jq .status.podIP
"10.8.0.109"
```

Note the IP address. Your IP address might differ from the previous output.

- 9.2. Use the oc run command to create a pod named mysqlclient that uses the registry.ocp4.example.com:8443/rhel9/mysql-80:1-237 container image. Set the value of the MYSQL\_ROOT\_PASSWORD environment variable to redhat123, and then confirm that the pod is running.

```
[student@workstation ~]$ oc run mysqlclient \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1-237 \
--env MYSQL_ROOT_PASSWORD=redhat123
pod/mysqlclient created
```

```
[student@workstation ~]$ oc get pods
NAME      READY   STATUS    RESTARTS   AGE
bitnami-mysql 1/1   Running   0          15m
mysqlclient   1/1   Running   0          19s
rhel9-mysql   1/1   Running   0          5m
```

- 9.3. Use the oc exec command with the -it options to execute the mysqlshow command on the mysqlclient pod. Connect as the redhat user and specify the host as the IP address of the rhel9-mysql pod. When prompted, enter redhat123 for the password.

```
[student@workstation ~]$ oc exec -it mysqlclient \
-- mysqlshow -u redhat -p -h 10.8.0.109
Enter password: redhat123
+-----+
| Databases |
```

```
+-----+
| information_schema |
| performance_schema |
| worldx           |
+-----+
```

The `worldx` database on the `rhel9-mysql` pod is accessible to the `mysql-client` pod.

## 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 pods-images
```

# Troubleshoot Containers and Pods

## Objectives

- Troubleshoot a pod by starting additional processes on its containers, changing their ephemeral file systems, and opening short-lived network tunnels.

## Container Troubleshooting Overview

Containers are designed to be immutable and ephemeral. A running container must be redeployed when changes are needed or when a new container image is available. However, you can change a running container without redeployment.

Updating a running container is best reserved for troubleshooting problematic containers. Red Hat does not generally recommend editing a running container to fix errors in a deployment. Changes to a running container are not captured in source control, but help to identify the needed corrections to the source code for the container functions. Capture these container updates in version control after you identify the necessary changes. Then, build a new container image and redeploy the application.

Custom alterations to a running container are incompatible with elegant architecture, reliability, and resilience for the environment.

## CLI Troubleshooting Tools

Administrators use various tools to interact with, inspect, and alter running containers.

Administrators can use commands such as `oc get` to gather initial details for a specified resource type. Other commands are available for detailed inspection of a resource, or to update a resource in real time.



### Note

When interacting with the cluster containers, take suitable precautions with actively running components, services, and applications.

Use these tools to validate the functions and environment for a running container:

- The `kubectl` CLI provides the following commands:
  - `kubectl describe`: Display the details of a resource.
  - `kubectl edit`: Edit a resource configuration by using the system editor.
  - `kubectl patch`: Update a specific attribute or field for a resource.
  - `kubectl replace`: Deploy a new instance of the resource.
  - `kubectl cp`: Copy files and directories to and from containers.
  - `kubectl exec`: Execute a command within a specified container.
  - `kubectl explain`: Display documentation for a specified resource.

- `kubectl port-forward`: Configure a port forwarder for a specified container.
- `kubectl logs`: Retrieve the logs for a specified container.

Besides supporting the previous `kubectl` commands, the `oc` CLI adds the following commands for inspecting and troubleshooting running containers:

- The `oc` CLI provides the following commands:
  - `oc status`: Display the status of the containers in the selected namespace.
  - `oc rsync`: Synchronize files and directories to and from containers.
  - `oc rsh`: Start a remote shell within a specified container.

## Editing Resources

Troubleshooting and remediation often begin with a phase of inspection and data gathering. When solving issues, the `describe` command can provide helpful details about the running resource, such as the definition of a container and its purpose.

The following example demonstrates use of the `oc describe RESOURCE NAME` command to retrieve information about a pod in the `openshift-dns` namespace:

```
[user@host ~]$ oc describe pod dns-default-lt13h
Name:           dns-default-lt13h
Namespace:      openshift-dns
Priority:       2000001000
Priority Class: system-node-critical
...output omitted...
```

Various CLI tools can apply a change that you determine is needed to a running container. The `edit` command opens the specified resource in the default editor for your environment. This editor is specified by setting either the `KUBE_EDITOR` or the `EDITOR` environment variable, or otherwise with the `vi` editor in Linux or the Notepad application in Windows.

The following example demonstrates use of the `oc edit RESOURCE NAME` command to edit a running container:

```
[user@host ~]$ oc edit pod mongo-app-sw88b

# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
# will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Pod
metadata:
  annotations:
...output omitted...
```

You can also use the `patch` command to update fields of a resource.

The following example uses the `patch` command to update the container image that a pod uses:

```
[user@host ~]$ oc patch pod valid-pod --type='json' \
-p='[{"op": "replace", "path": "/spec/containers/0/image", \
"value":"http://registry.access.redhat.com/ubi8/httpd-24"}]'
```

**Note**

For more information about patching resources and the different merge methods, refer to Update API Objects in Place Using `kubectl patch` [<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/update-api-object-kubectl-patch/>].

## Copy Files to and from Containers

Administrators can copy files and directories to or from a container to inspect, update, or correct functionality. Adding a configuration file or retrieving an application log are common use cases.

**Note**

To use the `cp` command with the `kubectl` CLI or the `oc` CLI, the `tar` binary must be present in the container. If the binary is absent, then an error message appears and the operation fails.

The following example demonstrates copying a file from a running container to a local directory by using the `oc cp SOURCE DEST` command:

```
[user@host ~]$ oc cp apache-app-kc82c:/var/www/html/index.html /tmp/index.bak

[user@host ~]$ ls /tmp
index.bak
```

The following example demonstrates use of the `oc cp SOURCE DEST` command to copy a file from a local directory to a directory in a running container:

```
[user@host ~]$ oc cp /tmp/index.html apache-app-kc82c:/var/www/html/

[user@host ~]$ oc exec -it apache-app-kc82c -- ls /var/www/html
index.bak
```

**Note**

Targeting a file path within a pod for either the `SOURCE` or `DEST` argument uses the `pod_name:path` format, and can include the `-c container_name` option to specify a container within the pod. If you omit the `-c container_name` option, then the command targets the first container in the pod.

Additionally, when using the `oc` CLI, file and directory synchronization is available by using the `oc rsync` command.

The following example demonstrates use of the `oc rsync SOURCE_NAME DEST` command to synchronize files from a running container to a local directory.

```
[user@host ~]$ oc rsync apache-app-kc82c:/var/www/ /tmp/web_files  
  
[user@host ~]$ ls /tmp/web_files  
cgi-bin  
html
```

The `oc rsync` command uses the `rsync` client on your local system to copy changed files to and from a pod container. The `rsync` binary must be available locally and within the container for this approach. If the `rsync` binary is not found, then a `tar` archive is created on the local system and is sent to the container. The container then uses the `tar` utility to extract files from the archive. Without the `rsync` and `tar` binaries, an error message occurs and the `oc rsync` command fails.



#### Note

For Linux-based systems, you can install the `rsync` client and the `tar` utility on a local system by using a package manager, such as DNF. For Windows-based systems, you can install the `cwRsync` client. For more information about the `cwRsync` client, refer to <https://www.itefix.net/cwrsync>.

## Remote Container Access

Exposing a network port for a container is routine, especially for containers that provide a service. In a cluster, port forwarding connections are made through the kubelet, which maps a local port on your system to a port on a pod. Configuring port forwarding creates a request through the Kubernetes API, and creates a multiplexed stream, such as HTTP/2, with a `port` header that specifies the target port in the pod. The kubelet delivers the stream data to the target pod and port, and vice versa for egress data from the pod.

When troubleshooting an application that typically runs without a need to connect locally, you can use the port-forwarding function to expose connectivity to the pod for investigation. With this function, an administrator can connect on the new port and inspect the problematic application. After you remediate the issue, the application can be redeployed without the port-forward connection.

The following example demonstrates use of the `oc port-forward RESOURCE EXTERNAL_PORT:CONTAINER_PORT` command to listen locally on port 8080 and to forward connections to port 80 on the pod:

```
[user@host ~]$ oc port-forward nginx-app-cc78k 8080:80  
Forwarding from 127.0.0.1:8080 -> 80  
Forwarding from [::1]:8080 -> 80
```

## Connect to Running Containers

Administrators use CLI tools to connect to a container via a shell for forensic inspections. With this approach, you can connect to, inspect, and run any available commands within the specified container.

The following example demonstrates use of the `oc rsh POD_NAME` command to connect to a container via a shell:

```
[user@host ~]$ oc rsh tomcat-app-jw53r
sh-4.4#
```

The `oc rsh` command does not accept the `-n namespace` option. Therefore, you must change to the namespace of the pod before you execute the `oc rsh` command. If you need to connect to a specific container in a pod, then use the `-c container_name` option to specify the container name. If you omit this option, then the command connects to the first container in the pod.

You can also connect to running containers from the web console by clicking the **Terminal** tab in the pod's principal menu.

Project: metallb-system

Pods > Pod details

**P controller-58c464cdb6-g7bx7** Running

Actions ▾

Details Metrics YAML Environment Logs Events **Terminal**

**Pod details**

|           |                                                                 |                         |                |
|-----------|-----------------------------------------------------------------|-------------------------|----------------|
| Name      | controller-58c464cdb6-g7bx7                                     | Status                  | Running        |
| Namespace | metallb-system                                                  | Restart policy          | Always restart |
| Labels    | app=metallb, component=controller, pod-template-hash=58c464cdb6 | Active deadline seconds | Not configured |
| Pod IP    |                                                                 |                         |                |

If you have more than one container, then you can change between them to connect to the CLI.

Project: metallb-system

Pods > Pod details

**P controller-58c464cdb6-g7bx7** Running

Actions ▾

Details Metrics YAML Environment Logs Events Terminal

Connecting to **controller** ▾

sh-4.4\$ [ ]

Expand

- controller
- kube-rbac-proxy

## Execute Commands in a Container

Passing commands to execute within a container from the CLI is another method for troubleshooting a running container. Use this method to send a command to run within the container, or to connect to the container, when further investigation is necessary.

Use the following command to pass and execute commands in a container:

```
oc exec POD | TYPE/NAME [-c container_name] -- COMMAND [arg1 ... argN]
```

If you omit the `-c container_name` option, then the command targets the first container in the pod.

The following examples demonstrate the use of the `oc exec` command to execute the `ls` command in a container to list the contents of the container's root directory:

```
[user@host ~]$ oc exec -it mariadb-lc78h -- ls /
bin  boot  dev  etc  help.1  home  lib  lib64 ...
...output omitted...
```

```
[user@host ~]$ oc exec mariadb-lc78h -- ls /
bin
boot
dev
etc
...output omitted...
```



#### Note

It is common to add the `-it` flags to the `kubectl exec` or `oc exec` commands. These flags instruct the command to send STDIN to the container and STDOUT/STDERR back to the terminal. The format of the command output is impacted by the inclusion of the `-it` flags.

## Container Events and Logs

Reviewing the historical actions for a container can offer insights into both the lifecycle and health of the deployment. Retrieving the cluster logs provides the chronological details of the container actions. Administrators inspect this log output for information and issues that occur in the running container.

For the following commands, use the `-c container_name` to specify a container in the pod. If you omit this option, then the command targets the first container in the pod.

The following examples demonstrate use of the `oc logs POD_NAME` command to retrieve the logs for a pod:

```
[user@host ~]$ oc logs BIND9-app-rw43j
Defaulted container "dns" out of: dns
.:5353
[INFO] plugin/reload: Running configuration SHA512 = 7c3d...3587
CoreDNS-1.9.2
...output omitted...
```

In Kubernetes, an event resource is a report of an event somewhere in the cluster. You can use the `kubectl get events` and `oc get events` commands to view pod events in a namespace:

```
[user@host ~]$ oc get events
LAST SEEN    TYPE      REASON          OBJECT                MESSAGE
...output omitted...
21m         Normal    AddedInterface   pod/php-app-5d9b84b588-kzfdx   Add eth0
[10.8.0.93/23] from ovn-kubernetes
21m         Normal    Pulled          pod/php-app-5d9b84b588-kzfdx   Container
image "registry.ocp4.example.com:8443/redhattraining/php-webapp:v4" already
present on machine
21m         Normal    Created         pod/php-app-5d9b84b588-kzfdx   Created
container php-webapp
21m         Normal    Started        pod/php-app-5d9b84b588-kzfdx   Started
container php-webapp
```

## Available Linux Commands in Containers

The use of Linux commands for troubleshooting applications can also help with troubleshooting containers. However, when connecting to a container, only the defined tools and applications within the container are available. You can augment the environment inside the container by adding the tools from this section or any other remedial tools to the container image.

Before you add tools to a container image, consider how the tools affect your container image.

- Additional tools increase the size of the image, which might impact container performance.
- Tools might require additional update packages and licensing terms, which can impact the ease of updating and distributing the container image.
- Hackers might exploit tools in the image.

## Troubleshooting from Inside the Cluster

It is routine to troubleshoot a cluster, its components, or the running applications by connecting remotely. This approach assumes that the administrator's computer contains the necessary tools for the work. When unanticipated issues arise, the necessary tools might not be available from an administrator's computer or an alternative machine.

Administrators can alternatively author and deploy a container within the cluster for investigation and remediation. By creating a container image that includes the cluster troubleshooting tools, you have a reliable environment to perform these tasks from any computer with access to the cluster. This approach ensures that an administrator always has access to the tools for reliable troubleshooting and remediation of issues.

Additionally, administrators should plan to author a container image that provides the most valuable troubleshooting tools for containerized applications. In this way, you deploy this "toolbox" container to supplement the forensic process and to provide an environment with the required commands and tools for troubleshooting problematic containers. For example, the "toolbox" container can test how resources operate inside a cluster, such as to confirm whether a pod can connect to resources outside the cluster. Regular cluster users can also create a "toolbox" container to help with application troubleshooting. For example, a regular user could run a pod with a MySQL client to connect to another pod that runs a MySQL server.

Although this approach falls outside the focus of this course, because it is more application-level remediation than container-level troubleshooting, it is important to realize that containers have such capacity.



## References

### Kubernetes Documentation - kubectl edit

<https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/#kubectl-edit>

### Update API Objects in Place Using kubectl patch

<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/update-api-object-kubectl-patch/>

For more information about troubleshooting pod issues, refer to the *Investigating Pod Issues* section in the *Troubleshooting* chapter in the Red Hat OpenShift Container Platform 4.14 Support documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/support/index#investigating-pod-issues](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/support/index#investigating-pod-issues)

For more information about how to copy files to and from pods, refer to the *Copying Files to or from an OpenShift Container Platform Container* section in the *Working with Containers* chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-containers-copying-files](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-containers-copying-files)

For more information about port forwarding, refer to the *Using Port Forwarding to Access Applications in a Container* section in the *Working with Containers* chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-containers-port-forwarding](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-containers-port-forwarding)

### Kubernetes Documentation - Use Port Forwarding to Access Applications in a Cluster

<https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/>

For more information about executing remote commands in containers, refer to the *Executing Remote Commands in an OpenShift Container Platform Container* section in the *Working with Containers* chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-containers-remote-commands](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-containers-remote-commands)

## ► Guided Exercise

# Troubleshoot Containers and Pods

Troubleshoot and fix a failed MySQL pod and manually initialize a database with test data.

### Outcomes

- Investigate errors with creating a pod.
- View the status, logs, and events for a pod.
- Copy files into a running pod.
- Connect to a running pod by using port forwarding.

### 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 and all exercise resources are available.

```
[student@workstation ~]$ lab start pods-troubleshooting
```

### Instructions

- 1. Log in to the OpenShift cluster and create the `pods-troubleshooting` project.

1.1. Log in to the OpenShift cluster as the `developer` user with the `oc` command.

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

1.2. Create the `pods-troubleshooting` project.

```
[student@workstation ~]$ oc new-project pods-troubleshooting
...output omitted...
```

- 2. Create a MySQL pod called `mysql-server` with the `oc run` command. Use the `registry.ocp4.example.com:8443/rhel9/mysql-80:1228` container image for the pod. Specify the environment variables with the following values:

| Variable       | Value     |
|----------------|-----------|
| MYSQL_USER     | redhat    |
| MYSQL_PASSWORD | redhat123 |
| MYSQL_DATABASE | world     |

Then, view the status of the pod with the `oc get` command.

- 2.1. Create the `mysql-server` pod with the `oc run` command. Specify the environment values with the `--env` option.

```
[student@workstation ~]$ oc run mysql-server \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1228 \
--env MYSQL_USER=redhat \
--env MYSQL_PASSWORD=redhat123 \
--env MYSQL_DATABASE=world
pod/mysql created
```

- 2.2. After a few moments, retrieve the status of the pod. Execute the `oc get pods` command a few times to view the status updates for the pod.

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS      RESTARTS   AGE
mysql-server 0/1     ErrImagePull    0          30s
```

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS      RESTARTS   AGE
mysql-server 0/1     ImagePullBackoff 0          45s
```

The pod failed to start.

► 3. Identify the root cause of the pod's failure.

- 3.1. Retrieve the pod's logs with the `oc logs` command.

```
[student@workstation ~]$ oc logs mysql-server
Error from server (BadRequest): container "mysql-server" in pod "mysql-server" is
waiting to start: trying and failing to pull image
```

The logs state that the pod cannot pull the container image.

- 3.2. Retrieve the events log with the `oc get events` command.

```
student@workstation ~]$ oc get events
LAST SEEN   TYPE      REASON           OBJECT      MESSAGE
33s         Normal    Scheduled        pod/mysql-server  Successfully assigned
            to pods-troubleshooting/mysql-server by master01
            master01 by master01
31s         Normal    AddedInterface   pod/mysql-server  Add eth0 [10.8.0.68/23]
            from ovn-kubernetes
16s         Normal    Pulling          pod/mysql-server  Pulling image
            "registry.ocp4.example.com:8443/rhel9/mysql-80:1228"
16s         Warning   Failed           pod/mysql-server  Failed to pull image
            "registry.ocp4.example.com:8443/rhel9/mysql-80:1228": rpc error: code = Unknown
            desc = reading manifest 1228 in registry.ocp4.example.com:8443/rhel9/mysql-80:
            manifest unknown: manifest unknown
```

```

16s      Warning  Failed        pod/mysql-server  Error: ErrImagePull
4s       Normal   BackOff     pod/mysql-server  Back-off pulling image
"registry.ocp4.example.com:8443/rhel9/mysql-80:1228"
4s      Warning  Failed        pod/mysql-server  Error: ImagePullBackOff

```

The output states that the image pull failed because the 1228 manifest is unknown. This failure could mean that the manifest, or image tag, does not exist in the repository.

- 3.3. Inspect the available manifest in the `registry.ocp4.example.com:8443/rhel9/mysql-80` container repository. Authenticate to the container repository with the `skopeo login` command as the `developer` user with the `developer` password. Then, use the `skopeo inspect` command to retrieve the available manifests in the `registry.ocp4.example.com:8443/rhel9/mysql-80` repository.

```
[student@workstation ~]$ skopeo login registry.ocp4.example.com:8443
Username: developer
Password: developer
Login Succeeded!
```

```
[student@workstation ~]$ skopeo inspect \
docker://registry.ocp4.example.com:8443/rhel9/mysql-80
...output omitted...
{
  "Name": "registry.ocp4.example.com:8443/rhel9/mysql-80",
  "Digest": "sha256:d282...f38f",
  "RepoTags": [
    "1-237",
    "1-228",
    "1-228-source",
    "1-224",
    "1-224-source",
    "latest",
    "1"
  ],
  ...output omitted...
}
```

The 1228 manifest, or tag, is not available in the repository, which means that the `registry.ocp4.example.com:8443/rhel9/mysql-80:1228` image does not exist. However, the 1-228 tag does exist.

- ▶ 4. The pod failed to start because of a typing error in the image tag. Update the pod's configuration to use the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-228` container image. Confirm that the pod is re-created after editing the resource.
- 4.1. Edit the pod's configuration with the `oc edit` command. Locate the `.spec.containers.image` object. Update the value to the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-228` container image and save the change.

```
[student@workstation ~]$ oc edit pod/mysql-server
```

```
...output omitted...
apiVersion: v1
kind: Pod
metadata:
...output omitted...
spec:
  containers:
    - image: registry.ocp4.example.com:8443/rhel9/mysql-80:1-228
...output omitted...
```

- 4.2. Verify the status of the mysql-server pod with the `oc get` command. The pod's status might take a few moments to update after the resource edit. Repeat the `oc get` command until the pod's status changes.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql-server   1/1     Running   0          10m
```

The mysql-server pod successfully pulled the image and created the container. The pod now shows a `Running` status.

- 5. Prepare the database on the mysql-server pod. Copy the `~/D0180/labs/pods-troubleshooting/world_x.sql` file to the mysql-server pod. Then, connect to the pod and execute the `world_x.sql` file to populate the world database.
- 5.1. Use the `oc cp` command to copy the `world_x.sql` file in the `~/D0180/labs/pods-troubleshooting` directory to the `/tmp/` directory on the mysql-server pod.

```
[student@workstation ~]$ oc cp ~/D0180/labs/pods-troubleshooting/world_x.sql \
mysql-server:/tmp/
```

- 5.2. Confirm that the `world_x.sql` file is accessible within the mysql-server pod with the `oc exec` command.

```
[student@workstation ~]$ oc exec -it pod/mysql-server -- ls -la /tmp/
total 0
drwxrwxrwx. 1 root      root  22 Nov  4 14:45 .
dr-xr-xr-x. 1 root      root  50 Nov  4 14:34 ..
-rw-rw-r--. 1 1000680000 root 558791 Nov  4 14:45 world_x.sql
```

- 5.3. Connect to the mysql-server pod with the `oc rsh` command. Then, log in to MySQL as the redhat user with the `redhat123` password.

```
[student@workstation ~]$ oc rsh mysql-server
```

```
sh-5.1$ mysql -u redhat -p
Enter password: redhat123
Welcome to the MySQL monitor. Commands end with ; or \g.
...output omitted...
mysql>
```

- 5.4. From the MySQL prompt, select the `world` database. Source the `world_x.sql` script inside the pod to initialize and populate the `world` database.

```
mysql> USE world;
Database changed
```

```
mysql> SOURCE /tmp/world_x.sql;
...output omitted...
```

- 5.5. Execute the `SHOW TABLES;` command to confirm that the database now contains tables. Then, exit the database and the pod.

```
mysql> SHOW TABLES;
-----
Tables_in_test
city
country
countryinfo
countrylanguage
-----
4 rows in set (0.00 sec)
```

```
mysql> exit;
Bye
sh-5.1$ exit
exit
[student@workstation ~]$
```

- 6. Configure port forwarding and then use the MySQL client on the `workstation` machine to connect to the `world` database on the `mysql-server` pod. Confirm that you can access data within the `world` database from the `workstation` machine.

- 6.1. From the `workstation` machine, use the `oc port-forward` command to forward the 3306 local port to the 3306 port on the `mysql-server` pod.

```
[student@workstation ~]$ oc port-forward mysql-server 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
```

- 6.2. Open another terminal window on the `workstation` machine. Connect to the `world` database with the local MySQL client on the `workstation` machine. Log in as the `redhat` user with the `redhat123` password. Specify the host as the localhost `127.0.0.1` IP address and use `3306` as the port.

```
[student@workstation ~]$ mysql -u redhat -p -h 127.0.0.1 -P 3306
Enter password: redhat123
Welcome to the MySQL monitor. Commands end with ; or \g.
...output omitted...
mysql>
```

- 6.3. Select the `world` database and execute the `SHOW TABLES;` command.

```
mysql> USE world;
Database changed

mysql> SHOW TABLES;
-----
Tables_in_test
city
country
countryinfo
countrylanguage
-----
4 rows in set (0.01 sec)
```

- 6.4. Confirm that you can retrieve data from the `country` table. Execute the `SELECT COUNT(*) FROM country` command to retrieve the number of countries within the `country` table.

```
mysql> SELECT COUNT(*) FROM country;
-----
COUNT(*)
239
-----
1 row in set (0.01 sec)
```

- 6.5. Exit the database.

```
mysql> exit;
Bye
[student@workstation ~]$
```

- 6.6. Return to the terminal that is executing the `oc port-forward` command. Press `Ctrl+C` to end the connection.

```
[student@workstation ~]$ oc port-forward mysql-server 3306:3306
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306
Handling connection for 3306
Handling connection for 3306
^C[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 pods-troubleshooting
```

## ► Lab

# Run Applications as Containers and Pods

Run a web server as a pod and insert a debug page that displays diagnostic information.

### Outcomes

- Deploy a pod from a container image.
- Retrieve the status and events of a pod.
- Troubleshoot a failed pod.
- Edit pod resources.
- Copy files to a running pod for diagnostic purposes.
- Use port forwarding to connect to a running 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 exercise resources are available.

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

### Instructions

The API URL of your OpenShift cluster is <https://api.ocp4.example.com:6443>, and the `oc` command is already installed on your `workstation` machine.

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

Use the `pods-review` project for your work.

1. Log in to the OpenShift cluster and change to the `pods-review` project.
2. Deploy a pod named `webphp` that uses the `registry.ocp4.example.com:8443/redhat-training/webphp:v1` container image. Determine why the pod fails to start.
3. Troubleshoot the failed `webphp` pod by creating a debug pod.
4. The application developer resolved the identified issue in the `registry.ocp4.example.com:8443/redhat-training/webphp:v2` container image. In a terminal window, edit the `webphp` pod resource to use the `v2` image tag. Retrieve the status of the `webphp` pod. Then, confirm that the user in the container is an unprivileged user and belongs to the `root` group. Confirm that the `root` group permissions are correct for the `/run/httpd` directory.
5. Connect port `8080` on the `Workstation` machine to port `8080` on the `webphp` pod. In a new terminal window, retrieve the content of the pod's `127.0.0.1:8080/index.php` web page to confirm that the pod is operational.



### Note

The terminal window that you connect to the `webphp` pod must remain open for the remainder of the lab. This connection is necessary for the final lab step and for the `lab grade` command.

6. An issue occurs with the PHP application that is running on the `webphp` pod. To debug the issue, the application developer requires diagnostic and configuration information for the PHP instance that is running on the `webphp` pod.

The `~/D0180/labs/pods-review` directory contains a `phpinfo.php` file to generate debugging information for a PHP instance. Copy the `phpinfo.php` file to the `/var/www/html/` directory on the `webphp` pod.

Then, confirm that the PHP debugging information is displayed when accessing the `127.0.0.1:8080/phpinfo.php` from a web browser.

## 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 pods-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 pods-review
```

## ► Solution

# Run Applications as Containers and Pods

Run a web server as a pod and insert a debug page that displays diagnostic information.

### Outcomes

- Deploy a pod from a container image.
- Retrieve the status and events of a pod.
- Troubleshoot a failed pod.
- Edit pod resources.
- Copy files to a running pod for diagnostic purposes.
- Use port forwarding to connect to a running 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 exercise resources are available.

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

### Instructions

The API URL of your OpenShift cluster is <https://api.ocp4.example.com:6443>, and the **oc** command is already installed on your **workstation** machine.

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

Use the **pods-review** project for your work.

1. Log in to the OpenShift cluster and change to the **pods-review** project.

1.1. Log in to the OpenShift cluster.

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

1.2. Select the **pods-review** project.

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

2. Deploy a pod named webphp that uses the `registry.ocp4.example.com:8443/redhattraining/webphp:v1` container image. Determine why the pod fails to start.
  - 2.1. Deploy a pod named webphp that uses the `registry.ocp4.example.com:8443/redhattraining/webphp:v1` container image.

```
[student@workstation ~]$ oc run webphp \
--image=registry.ocp4.example.com:8443/redhattraining/webphp:v1
pod/webphp created
```

- 2.2. After a few moments, observe the status of the webphp pod.

```
[student@workstation ~]$ oc get pods
NAME      READY     STATUS            RESTARTS   AGE
webphp   0/1      CrashLoopBackOff   1 (4s ago)  7s
[student@workstation ~]$ oc get pods
NAME      READY     STATUS            RESTARTS   AGE
webphp   0/1      Error            2 (24s ago)  7s
```

The pod failed to start.

- 2.3. Retrieve the cluster events.

```
[student@workstation ~]$ oc get events
LAST SEEN    TYPE      REASON          OBJECT      MESSAGE
3m25s       Normal    Scheduled        pod/webphp  Successfully assigned pods-
review/webphp to master01 by master01
3m23s       Normal    AddedInterface   pod/webphp  Add eth0 [10.8.0.73/23] from
ovn-kubernetes
3m23s       Normal    Pulling         pod/webphp  Pulling image
"registry.ocp4.example.com:8443/redhattraining/webphp:v1"
3m15s       Normal    Pulled          pod/webphp  Successfully pulled image
"registry.ocp4.example.com:8443/redhattraining/webphp:v1" in 7.894992669s
104s        Normal    Created         pod/webphp  Created container webphp
104s        Normal    Started         pod/webphp  Started container webphp
104s        Normal    Pulled          pod/webphp  Container image
"registry.ocp4.example.com:8443/redhattraining/webphp:v1" already present on
machine
103s        Warning   BackOff        pod/webphp  Back-off restarting failed
container
```

- 2.4. Retrieve the logs for the webphp pod.

```
[student@workstation ~]$ oc logs webphp
[20-Dec-2022 18:46:56] NOTICE: [pool www] 'user' directive is ignored when FPM is
not running as root
[20-Dec-2022 18:46:56] NOTICE: [pool www] 'group' directive is ignored when FPM is
not running as root
[20-Dec-2022 18:46:56] ERROR: unable to bind listening socket for address '/run/
php-fpm/www.sock': Permission denied (13)
[20-Dec-2022 18:46:56] ERROR: FPM initialization failed
AH00558: httpd: Could not reliably determine the server's fully qualified domain
name, using 10.8.0.62. Set the 'ServerName' directive globally to suppress this
message
(13)Permission denied: AH00058: Error retrieving pid file run/httpd.pid
AH00059: Remove it before continuing if it is corrupted.
```

The logs indicate permission issues with the /run directory within the pod.

### 3. Troubleshoot the failed webphp pod by creating a debug pod.

#### 3.1. Create a debug pod to troubleshoot the failed webphp pod.

```
[student@workstation ~]$ oc debug pod/webphp
Starting pod/webphp-debug ...
Pod IP: 10.8.0.63
If you don't see a command prompt, try pressing enter.
sh-4.4$
```

#### 3.2. List the contents of the /run directory to retrieve the permissions, owners, and groups.

```
sh-4.4$ ls -la /run
total 0
drwxr-xr-x. 1 root root 42 Dec 20 18:47 .
dr-xr-xr-x. 1 root root 17 Dec 20 18:47 ..
-rw-r--r--. 1 root root 0 Dec 20 18:47 .containerenv
drwx--x---. 3 root apache 26 Dec 20 18:42 httpd
drwxr-xr-x. 2 root root 6 Oct 26 11:10 lock
drwxr-xr-x. 2 root root 6 Dec 20 18:42 php-fpm
drwxr-xr-x. 4 root root 80 Dec 20 18:47 secrets
```

The /run/httpd directory grants read, write, and execute permissions to the `root` user, but does not provide permissions for the `root` group.

#### 3.3. Retrieve the UID and GID of the user in the container. Determine whether the user is a privileged user and belongs to the `root` group.

```
sh-4.4$ id
uid=1000680000(1000680000) gid=0(root) groups=0(root),1000680000
```

Your UID and GID values might differ from the previous output.

The user is an unprivileged, non-`root` user and belongs to the `root` group, which does not have access to the /run directory. Therefore, the user in the container cannot access the files and directories that the container processes use, which is required for arbitrarily assigned UIDs.

#### 3.4. Exit the debug pod.

```
sh-4.4$ exit  
exit  
  
Removing debug pod ...
```

4. The application developer resolved the identified issue in the `registry.ocp4.example.com:8443/redhattraining/webphp:v2` container image. In a terminal window, edit the `webphp` pod resource to use the `v2` image tag. Retrieve the status of the `webphp` pod. Then, confirm that the user in the container is an unprivileged user and belongs to the `root` group. Confirm that the `root` group permissions are correct for the `/run/httpd` directory.

- 4.1. Use the terminal to edit the `webphp` pod resource.

```
[student@workstation ~]$ oc edit pod/webphp
```

- 4.2. Update the `.spec.containers.image` object value to use the `:v2` image tag.

```
...output omitted...  
spec:  
  containers:  
    - image: registry.ocp4.example.com:8443/redhattraining/webphp:v2  
      imagePullPolicy: IfNotPresent  
  ...output omitted...
```

- 4.3. Verify the status of the `webphp` pod.

```
[student@workstation ~]$ oc get pods  
NAME      READY   STATUS    RESTARTS   AGE  
webphp   1/1     Running   9 (2m9s ago)  18m
```

- 4.4. Retrieve the UID and GID of the user in the container to confirm that the user is an unprivileged user.

```
[student@workstation ~]$ oc exec -it webphp -- id  
uid=1000680000(1000680000) gid=0(root) groups=0(root),1000680000
```

Your UID and GID values might differ from the previous output.

- 4.5. Confirm that the permissions for the `/run/httpd` directory are correct.

```
[student@workstation ~]$ oc exec -it webphp -- ls -la /run/  
total 0  
drwxr-xr-x. 1 root root 70 Dec 20 19:01 .  
dr-xr-xr-x. 1 root root 39 Dec 20 19:01 ..  
-rw-r--r--. 1 root root 0 Dec 20 18:45 .containerenv  
drwxrwx---. 1 root root 41 Dec 20 19:01 httpd  
drwxr-xr-x. 2 root root 6 Oct 26 11:10 lock  
drwxrwxr-x. 1 root root 41 Dec 20 19:01 php-fpm  
drwxr-xr-x. 4 root root 80 Dec 20 19:01 secrets
```

5. Connect port 8080 on the **Workstation** machine to port 8080 on the **webphp** pod. In a new terminal window, retrieve the content of the pod's `127.0.0.1:8080/index.php` web page to confirm that the pod is operational.



**Note**

The terminal window that you connect to the **webphp** pod must remain open for the remainder of the lab. This connection is necessary for the final lab step and for the `lab grade` command.

- 5.1. Connect to port 8080 on the **webphp** pod.

```
[student@workstation ~]$ oc port-forward pod/webphp 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

- 5.2. Open a second terminal window and then retrieve the `127.0.0.1:8080/index.php` web page on the **webphp** pod.

```
[student@workstation ~]$ curl 127.0.0.1:8080/index.php
<html>
<body>
Hello, World!
</body>
</html>
```

6. An issue occurs with the PHP application that is running on the **webphp** pod. To debug the issue, the application developer requires diagnostic and configuration information for the PHP instance that is running on the **webphp** pod.

The `~/D0180/labs/pods-review` directory contains a `phpinfo.php` file to generate debugging information for a PHP instance. Copy the `phpinfo.php` file to the `/var/www/html/` directory on the **webphp** pod.

Then, confirm that the PHP debugging information is displayed when accessing the `127.0.0.1:8080/phpinfo.php` from a web browser.

- 6.1. In the second terminal, copy the `~/D0180/labs/pods-review/phpinfo.php` file to the **webphp** pod as the `/var/www/html/phpinfo.php` file.

```
[student@workstation ~]$ oc cp ~/D0180/labs/pods-review/phpinfo.php \
webphp:/var/www/html/phpinfo.php
```

- 6.2. Open a web browser and access the `127.0.0.1:8080/phpinfo.php` web page. Confirm that PHP debugging information is displayed.

PHP Version 7.4.30	
<b>System</b>	Linux webphp 4.18.0-372.19.1.el8_6.x86_64 #1 SMP Mon Jul 18 11:14:02 EDT 2022 x86_64
<b>Build Date</b>	Jun 7 2022 08:38:19
<b>Server API</b>	FPM/FastCGI
<b>Virtual Directory Support</b>	disabled
<b>Configuration File (php.ini) Path</b>	/etc
<b>Loaded Configuration File</b>	/etc/php.ini
<b>Scan this dir for additional .ini files</b>	/etc/php.d
<b>Additional .ini files parsed</b>	/etc/php.d/10-opcache.ini, /etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-dom.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-json.ini, /etc/php.d/20-mbstring.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-simplexml.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/20-xml.ini, /etc/php.d/20-xmlwriter.ini, /etc/php.d/20-xsl.ini, /etc/php.d/30-pdo_sqlite.ini, /etc/php.d/30-xmlreader.ini
<b>PHP API</b>	20190902
<b>PHP Extension</b>	20190902

**Figure 3.9: PHP debugging information**

- 6.3. Return to the terminal that is executing the `oc port-forward` command. Press `Ctrl+C` to end the connection.

```
[student@workstation ~]$ oc port-forward pod/webphp 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
^C[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 pods-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 pods-review
```

## ► Quiz

# Run Applications as Containers and Pods

Choose the correct answers to the following questions:

As the student user on the workstation machine, use Skopeo to log in to the `registry.ocp4.example.com:8443` classroom container registry as the developer user with the developer password. Then, use the `skopeo` and `oc image` commands to answer the following questions.

► 1. Which command is executed in the `registry.ocp4.example.com:8443/ubi8/php-74:latest` container image?

- a. `httpd -D FOREGROUND`
- b. `run-mysqld`
- c. `php-fpm -D, httpd -D FOREGROUND`
- d. `/bin/sh -c $STI_SCRIPTS_PATH/usage`
- e. `/bin/sh -c php-fpm -D`

► 2. Using the UID value for the `registry.ocp4.example.com:8443/ubi8/php-74:latest` container image, identify the two correct deployment scenarios. (Choose two.)

- a. The container executes as the root privileged user when a regular OpenShift user account deploys it in a pod.
- b. The container user UID value is 1001 when a regular OpenShift user account deploys it in a pod.
- c. The container executes as a non-privileged user when it is deployed in a pod.
- d. The container executes as the root privileged user when an OpenShift cluster administrator deploys it in a pod.
- e. The container user UID value is 1001 when an OpenShift cluster administrator deploys it in a pod.

► 3. What is the ID/hash for the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237` container image?

- a. sha256:90b4...a3a4
- b. sha256:4243...71a6
- c. sha256:3843...7caf
- d. sha256:d282...f38f
- e. sha256:0027...8c72

- 4. Which two environment variables and their values are specified in the `registry.ocp4.example.com:8443/rhel8/postgresql-13:latest` container image? (Choose two.)
- a. NAME=PHP\_7.4
  - b. APP\_DATA=/opt/app-root/src/bin
  - c. HOME=/var/lib/pgsql
  - d. PHP\_SYSCONF\_FILE=/etc/
  - e. PGUSER=postgres
- 5. Which three tags are available for the `registry.ocp4.example.com:8443/ubi8/ubi` container image? (Choose three.)
- a. v1
  - b. 1
  - c. latest
  - d. 8.1
  - e. 8.4
  - f. 1.8
  - g. 8.0
- 6. Which two container images run as a privileged user when an OpenShift cluster administrator deploys a pod from the image? (Choose two.)
- a. `registry.ocp4.example.com:8443/rhel8/postgresql-13:latest`
  - b. `registry.ocp4.example.com:8443/ubi8/php-74:latest`
  - c. `registry.ocp4.example.com:8443/ubi8/nodejs-16:latest`
  - d. `registry.ocp4.example.com:8443/rhel8/mysql-80:latest`
  - e. `registry.ocp4.example.com:8443/ubi8/python-39`
- 7. Which ports are exposed in the `registry.ocp4.example.com:8443/rhel8/mysql-80` container image?
- a. 8080/tcp
  - b. 3363/tcp
  - c. 8443/tcp
  - d. 3306/tcp
  - e. 3360/tcp

## ► Solution

# Run Applications as Containers and Pods

Choose the correct answers to the following questions:

As the student user on the workstation machine, use Skopeo to log in to the `registry.ocp4.example.com:8443` classroom container registry as the developer user with the developer password. Then, use the `skopeo` and `oc image` commands to answer the following questions.

► 1. Which command is executed in the `registry.ocp4.example.com:8443/ubi8/php-74:latest` container image?

- a. `httpd -D FOREGROUND`
- b. `run-mysqld`
- c. `php-fpm -D, httpd -D FOREGROUND`
- d. `/bin/sh -c $STI_SCRIPTS_PATH/usage`
- e. `/bin/sh -c php-fpm -D`

► 2. Using the UID value for the `registry.ocp4.example.com:8443/ubi8/php-74:latest` container image, identify the two correct deployment scenarios. (Choose two.)

- a. The container executes as the root privileged user when a regular OpenShift user account deploys it in a pod.
- b. The container user UID value is 1001 when a regular OpenShift user account deploys it in a pod.
- c. The container executes as a non-privileged user when it is deployed in a pod.
- d. The container executes as the root privileged user when an OpenShift cluster administrator deploys it in a pod.
- e. The container user UID value is 1001 when an OpenShift cluster administrator deploys it in a pod.

► 3. What is the ID/hash for the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237` container image?

- a. sha256:90b4...a3a4
- b. sha256:4243...71a6
- c. sha256:3843...7caf
- d. sha256:d282...f38f
- e. sha256:0027...8c72

- 4. Which two environment variables and their values are specified in the `registry.ocp4.example.com:8443/rhel8/postgresql-13:latest` container image? (Choose two.)
- a. NAME=PHP\_7\_4
  - b. APP\_DATA=/opt/app-root/src/bin
  - c. HOME=/var/lib/pgsql
  - d. PHP\_SYSCONF\_FILE=/etc/
  - e. PGUSER=postgres
- 5. Which three tags are available for the `registry.ocp4.example.com:8443/ubi8/ubi` container image? (Choose three.)
- a. v1
  - b. 1
  - c. latest
  - d. 8.1
  - e. 8.4
  - f. 1.8
  - g. 8.0
- 6. Which two container images run as a privileged user when an OpenShift cluster administrator deploys a pod from the image? (Choose two.)
- a. `registry.ocp4.example.com:8443/rhel8/postgresql-13:latest`
  - b. `registry.ocp4.example.com:8443/ubi8/php-74:latest`
  - c. `registry.ocp4.example.com:8443/ubi8/nodejs-16:latest`
  - d. `registry.ocp4.example.com:8443/rhel8/mysql-80:latest`
  - e. `registry.ocp4.example.com:8443/ubi8/python-39`
- 7. Which ports are exposed in the `registry.ocp4.example.com:8443/rhel8/mysql-80` container image?
- a. 8080/tcp
  - b. 3363/tcp
  - c. 8443/tcp
  - d. 3306/tcp
  - e. 3360/tcp

# Summary

---

- A container is an encapsulated process that includes the required runtime dependencies for an application to run.
- OpenShift uses Kubernetes to manage pods. Pods consist of one or more containers that share resources, such as selected namespaces and networking, and represent a single application.
- Container images can create container instances, which are executable versions of the image, and include references to networking, disks, and other runtime necessities.
- Container image registries, such as Quay.io and the Red Hat Container Catalog, are the preferred way to distribute container images to many users and hosts.
- The `oc image` command and Skopeo, among other tools, can inspect and manage container images.
- Containers are immutable and ephemeral. Thus, updating a running container is best reserved for troubleshooting problematic containers.



## Chapter 4

# Deploy Managed and Networked Applications on Kubernetes

### Goal

Deploy applications and expose them to network access from inside and outside a Kubernetes cluster.

### Objectives

- Identify the main resources and settings that Kubernetes uses to manage long-lived applications and demonstrate how OpenShift simplifies common application deployment workflows.
- Deploy containerized applications as pods that Kubernetes workload resources manage.
- Interconnect applications pods inside the same cluster by using Kubernetes services.
- Expose applications to clients outside the cluster by using Kubernetes ingress and OpenShift routes.

### Sections

- Deploy Applications from Images and Templates (and Guided Exercise)
- Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API (and Guided Exercise)
- Kubernetes Pod and Service Networks (and Guided Exercise)
- Scale and Expose Applications to External Access (and Guided Exercise)

### Lab

- Deploy Managed and Networked Applications on Kubernetes

# Deploy Applications from Images and Templates

---

## Objectives

- Identify the main resources and settings that Kubernetes uses to manage long-lived applications and demonstrate how OpenShift simplifies common application deployment workflows.

## Deploying Applications

Microservices and DevOps are growing trends in enterprise software. Containers and Kubernetes gained popularity alongside those trends, but have become categories of their own. Container-based infrastructures support most types of traditional and modern applications.

The term *application* can refer to your software system or to a service within it. Given this ambiguity, it is clearer to refer directly to resources, services, and other components.

## Resources and Resource Definitions

Kubernetes manages applications, or services, as a loose collection of resources. Resources are configuration pieces for components in the cluster. When you create a resource, the corresponding component is not created immediately. Instead, the cluster is responsible for eventually creating the component.

A resource type represents a specific component type, such as a pod. Kubernetes ships with many default resource types, some of which overlap in function. Red Hat OpenShift Container Platform (RHOC) includes the default Kubernetes resource types, and provides other resource types of its own. To add resource types, you can create or import custom resource definitions (CRDs).

## Managing Resources

You can add, view, and edit resources in various formats, including YAML and JSON. Traditionally, YAML is the most common format.

You can delete resources in batch by using label selectors or by deleting the entire project or namespace. For example, the following command deletes only deployments with the `app=my-app` label.

```
[user@host ~]$ oc delete deployment -l app=my-app  
...output omitted...
```

Similar to creation, deleting a resource is not immediate, but is instead a request for eventual deletion.



### Note

Commands that are executed without specifying a namespace are executed in the user's current namespace.

## Common Resource Types and Their Uses

The following resources are standard Kubernetes resources unless otherwise noted.

### Templates

Similar to projects, templates are an RHOCP addition to Kubernetes. A *template* is a YAML manifest that contains parameterized definitions of one or more resources. RHOCP provides predefined templates in the `openshift` namespace.

Process a template into a list of resources by using the `oc process` command, which replaces values and generates resource definitions. The resulting resource definitions create or update resources in the cluster by supplying them to the `oc apply` command.

For example, the following command processes a `mysql-template.yaml` template file and generates four resource definitions.

```
[user@host ~]$ oc process -f mysql-template.yaml -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Secret
  ...output omitted...
- apiVersion: v1
  kind: Service
  ...output omitted...
- apiVersion: v1
  kind: PersistentVolumeClaim
  ...output omitted...
- apiVersion: apps.openshift.io/v1
  kind: DeploymentConfig
  ...output omitted...
```

The `--parameters` option instead displays the parameters of a template. For example, the following command lists the parameters of the `mysql-template.yaml` file.

```
[user@host ~]$ oc process -f mysql-template.yaml --parameters
NAME                  DESCRIPTION ...output omitted...
...output omitted...
MYSQL_USER            Username for MySQL user    ...output omitted...
MYSQL_PASSWORD        Password for the MySQL connection    ...output omitted...
MYSQL_ROOT_PASSWORD  Password for the MySQL root user.    ...output omitted...
MYSQL_DATABASE        Name of the MySQL database accessed. ...output omitted...
VOLUME_CAPACITY       Volume space available for data,    ...output omitted...
```

You can use templates with the `new-app` command from RHOCP. In the following example, the `new-app` command uses the `mysql-persistent` template to create a MySQL application and its supporting resources.

```
[user@host ~]$ oc new-app --template mysql-persistent
--> Deploying template "db-app/mysql-persistent" to project db-app
...output omitted...
The following service(s) have been created in your project: mysql.
```

```
Username: userQSL
Password: pyf0yElPvFWYQQou
Database Name: sampledb
Connection URL: mysql://mysql:3306/
...output omitted...
* With parameters:
* Memory Limit=512Mi
* Namespace=openshift
* Database Service Name=mysql
* MySQL Connection Username=userQSL # generated
* MySQL Connection Password=pyf0yElPvFWYQQou # generated
* MySQL root user Password=HHbdurqW05gAog2m # generated
* MySQL Database Name=sampledb
* Volume Capacity=1Gi
* Version of MySQL Image=8.0-e18

--> Creating resources ... ①
secret "mysql" created
service "mysql" created
persistentvolumeclaim "mysql" created
deploymentconfig.apps.openshift.io "mysql" created
--> Success
...output omitted...
```

- ① Notice that several resources are created to meet the requirements of the deployment, including a secret, a service, and a persistent volume claim.



**Note**

You can specify environment variables to be configured in creating your application.

You can also use templates when creating applications from the web console by using the Developer Catalog. From the **Developer** perspective, navigate to the **+Add** menu and click **All Services** in the **Developer Catalog** section to open the catalog. Then, enter the application name in the filter to search for a template for your application. You can instantiate the template and change the default values, such as the Git repository, the memory limits, the application version, and much more.

The screenshot shows the 'Instantiate Template' dialog for the 'Apache HTTP Server' template. It includes fields for Namespace (PR testing), Name (httpd-example), Namespace (openshift), HTTPD Version (2.4-el8), and Memory Limit (512Mi). To the right, there's a summary of resources to be created: BuildConfig, DeploymentConfig, ImageStream, Route, and Service.

You can add an application based on a template by changing to the **Developer** perspective, and then moving to the **Topology** menu and clicking **Start building application** or clicking the book icon.

The screenshot shows the OpenShift developer interface. The left sidebar has 'Topology' selected. The main area displays a network icon and the message 'No resources found'. A link at the bottom says 'Start building your application or visit the Add page for more details.'

## Pod

From the RHOCP documentation, a *pod* is defined as "the smallest compute unit that can be defined, deployed, and managed". A pod runs one or more containers that represent a single application. Containers in the pod share resources, such as networking and storage.

The following example shows a definition of a pod:

```
apiVersion: v1
kind: Pod ①
metadata: ②
  annotations: { ... }
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
  name: registry
```

```

namespace: pod-registries
spec: ③
  containers:
    - env:
        - name: OPENSHIFT_CA_DATA
          value: ...
      image: openshift/origin-docker-registry:v0.6.2
      imagePullPolicy: IfNotPresent
      name: registry
      ports:
        - containerPort: 5000
          protocol: TCP
      resources: {}
      securityContext: { ... }
      volumeMounts:
        - mountPath: /registry
          name: registry-storage
    dnsPolicy: ClusterFirst
    imagePullSecrets:
      - name: default-dockercfg-at06w
    restartPolicy: Always
    serviceAccount: default
    volumes:
      - emptyDir: {}
        name: registry-storage
  status: ④
    conditions: { ... }

```

- ① Resource kind set to Pod.
- ② Information that describes your application, such as the name, project, attached labels, and annotations.
- ③ Section where the application requirements are specified, such as the container name, the container image, environment variables, volume mounts, network configuration, and volumes.
- ④ Indicates the last condition of the pod, such as the last probe time, the last transition time, the status setting as `true` or `false`, and more.

## Deployment

Similar to deployment configurations, *deployments* define the intended state of a replica set. *Replica sets* maintain a configurable number of pods that match a specification.

Replica sets are generally similar to and a successor to replication controllers. This difference in intermediate resources is the primary difference between deployments and deployment configurations.

Deployments and replica sets are a Kubernetes-standard replacement for deployment configurations and replication controllers, respectively. Use a deployment unless you need a feature that is specific to deployment configurations.

The following example shows the definition of a deployment:

```

apiVersion: apps/v1
kind: Deployment 1
metadata:
  name: hello-openshift 2
spec:
  replicas: 1 3
  selector:
    matchLabels:
      app: hello-openshift
  template: 4
    metadata:
      labels:
        app: hello-openshift
    spec:
      containers:
        - name: hello-openshift 5
          image: openshift/hello-openshift:latest 6
          ports: 7
            - containerPort: 80

```

- 1** Resource kind set to `Deployment`.
- 2** Name of the deployment resource.
- 3** Number of running instances.
- 4** Section to define the metadata, labels, and the container information of the deployment resource.
- 5** Name of the container.
- 6** Resource of the image to use for creating the deployment resource.
- 7** Port configuration, such as the port number, name of the port, and the protocol.

## Deployment Configurations

*Deployment configurations* define the specification of a pod. They manage pods by creating *replication controllers*, which manage the number of replicas of a pod. Deployment configurations and replication controllers are an RHOCP addition to Kubernetes.



### Note

As of OpenShift Container Platform 4.14, deployment configuration objects are deprecated. Deployment configurations objects are still supported, but are not recommended for new installations.

Instead, use Deployment objects or another alternative to provide declarative updates for pods.

## Projects

RHOCP adds projects to enhance the function of Kubernetes namespaces. A *project* is a Kubernetes namespace with additional annotations, and is the primary method for managing access to resources for regular users. Projects can be created from templates and must use Role

Based Access Control (RBAC) for organization and permission management. Administrators must grant cluster users access to a project. If a cluster user is allowed to create projects, then the user automatically has access to their created projects.

Projects provide logical and organizational isolation to separate your application component resources. Resources in one project can access resources in other projects, but not by default.

The following example shows the definition of a project:

```
apiVersion: project.openshift.io/v1
kind: Project ①
metadata:
  name: test ②
spec:
  finalizers: ③
    - kubernetes
```

- ① Resource kind set to **Project**.
- ② Name of the project.
- ③ A finalizer is a special metadata key that tells Kubernetes to wait until a specific condition is met before it fully deletes a resource.

## Services

You can configure internal pod-to-pod network communication in RHOCP by using the **Service** object. Applications send requests to the service name and port. RHOCP provides a virtual network, which reroutes such requests to the pods that the service targets by using labels.

The following example shows the definition of a service:

```
apiVersion: v1
kind: Service ①
metadata:
  name: docker-registry ②
  namespace: test ③
spec:
  selector:
    app: MyApp ④
  ports:
    - protocol: TCP ⑤
      port: 80 ⑥
      targetPort: 9376 ⑦
```

- ① Resource kind set to **Service**.
- ② Name of the service.
- ③ Project name where the service resource exists.
- ④ The label selector identifies all pods with the attached `app=MyApp` label and adds the pods to the service endpoints.
- ⑤ Internet protocol set to TCP.

- ⑥ Port that the service listens on.
- ⑦ Port on the backing pods, which the service forwards connections to.

## Persistent Volume Claims

RHOCP uses the Kubernetes persistent volume (PV) framework to enable cluster administrators to provision persistent storage for a cluster. Developers can use persistent volume claims (PVCs) to request PV resources without having specific knowledge of the underlying storage infrastructure. After a PV is bound to a PVC, that PV cannot then be bound to additional PVCs. Because PVCs are namespaced objects and PVs are globally scoped objects, this binding effectively scopes a bound PV to a single namespace until the binding PVC is deleted.

The following example shows the definition of a persistent volume claim:

```
apiVersion: v1
kind: PersistentVolumeClaim 1
metadata:
  name: mysql-pvc 2
spec:
  accessModes:
    - ReadWriteOnce 3
  resources:
    requests:
      storage: 1Gi 4
  storageClassName: nfs-storage 5
status:
  ...
  
```

- ① Resource kind set to `PersistentVolumeClaim`.
- ② Name of the service.
- ③ The access mode, to define the read/write and mount permissions.
- ④ The amount of storage that is available to the PVC.
- ⑤ Name of the StorageClass that the claim requires.

## Secrets

Secrets provide a mechanism to hold sensitive information, such as passwords, private source repository credentials, sensitive configuration files, and credentials to an external resource, such as an SSH key or OAuth token. You can mount secrets into containers by using a volume plug-in. Kubernetes can also use secrets to perform actions, such as declaring environment variables, on a pod. Secrets can store any type of data. Kubernetes and OpenShift support different types of secrets, such as service account tokens, SSH keys, and TLS certificates.

The following example shows the definition of a secret:

```
apiVersion: v1
kind: Secret 1
metadata:
  name: example-secret 2
  namespace: my-app 3
  
```

```

type: Opaque ④
data: ⑤
  username: bXl1c2VyCg==
  password: bXlQQDU1Cg==
stringData: ⑥
  hostname: myapp.mydomain.com
  secret.properties: |
    property1=valueA
    property2=valueB

```

- ①** Resource kind set to `Secret`.
- ②** Name of the service.
- ③** Project name where the service resource exists.
- ④** Specifies the type of secret.
- ⑤** Specifies the encoded string and data.
- ⑥** Specifies the decoded string and data.

## Managing Resources from the Command Line

Kubernetes and RHOC have many commands to create and modify cluster resources. Some commands are part of core Kubernetes, whereas others are exclusive additions to RHOC.

The resource management commands generally fall into one of two categories: declarative or imperative. An *imperative* command instructs what the cluster does. A *declarative* command defines the state that the cluster attempts to match.

### Imperative Resource Management

The `create` command is an imperative way to create resources, and is included in both of the `oc` and `kubectl` commands. For example, the following command creates a deployment called `my-app` that creates pods that are based on the specified image.

```
[user@host ~]$ oc create deployment my-app --image example.com/my-image:dev
deployment.apps/my-app created
```

Use the `set` command to define attributes on a resource, such as environment variables. For example, the following command adds the `TEAM=red` environment variable to the preceding deployment.

```
[user@host ~]$ oc set env deployment/my-app TEAM=red
deployment.apps/my-app updated
```

Another imperative approach to creating a resource is the `run` command. In the following example, the `run` command creates the `example-pod` pod.

```
[user@host ~]$ oc run example-pod \
--image=registry.access.redhat.com/ubi8/httpd-24 \
--env GREETING='Hello from the awesome container' \
--port 8080
pod/example-pod created
```

- ❶ The pod `.metadata.name` definition.
- ❷ The image for the single container in this pod.
- ❸ The environment variable for the single container in this pod.
- ❹ The port metadata definition.

The imperative commands are a faster way of creating pods, because such commands do not require a pod object definition. However, developers cannot handle versioning or incrementally change the pod definition.

Generally, developers test a deployment by using imperative commands, and then use the imperative commands to generate the pod object definition. Use the `--dry-run=client` option to avoid creating the object in RHOC. Additionally, use the `-o yaml` or `-o json` option to configure the definition format.

The following command is an example of generating the YAML definition for the `example-pod` pod:

```
[user@host ~]$ oc run example-pod \
--image=registry.access.redhat.com/ubi8/httpd-24 \
--env GREETING='Hello from the awesome container' \
--port 8080 \
--dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: example-pod
    name: example-pod
spec:
  containers:
    ...output omitted...
```

Managing resources in this way is imperative, because you are instructing the cluster what to do rather than declaring the intended outcomes.

## Declarative Resource Management

Using the `create` command does not guarantee that you are creating resources imperatively. For example, providing a manifest declaratively creates the resources that are defined in the YAML file, such as in the following command.

```
[user@host ~]$ oc create -f my-app-deployment.yaml
deployment.apps/my-app created
```

RHOCP adds the `new-app` command, which provides another declarative way to create resources. This command uses heuristics to automatically determine which types of resources to create based on the specified parameters. For example, the following command deploys the `my-app` application by creating several resources, including a deployment resource, from a YAML manifest file.

```
[user@host ~]$ oc new-app --file=./example/my-app.yaml
...output omitted...
--> Creating resources ...
  imagestream.image.openshift.io "my-app" created
  deployment.apps "my-app" created
  services "my-app" created
...output omitted...
```

In both of the preceding `create` and `new-app` examples, you are *declaring* the intended state of the resources, and so they are declarative.

You can also use the `new-app` command with templates for resource management. A template describes the intended state of resources that must be created for an application to run, such as deployment configurations and services. Supplying a template to the `new-app` command is a form of declarative resource management.

The `new-app` command also includes options, such as the `--param` option, that customize an application deployment declaratively. For example, when the `new-app` is used with a template, you can include the `--param` option to override a parameter value in the template.

```
[user@host ~]$ oc new-app --template mysql-persistent \
--param MYSQL_USER=operator --param MYSQL_PASSWORD=myP@55 \
--param MYSQL_DATABASE=mydata \
--param DATABASE_SERVICE_NAME=db
--> Deploying template "db-app/mysql-persistent" to project db-app
...output omitted...
The following service(s) have been created in your project: db.

  Username: operator
  Password: myP@55
  Database Name: mydata
  Connection URL: mysql://db:3306/
...output omitted...
  * With parameters:
    * Memory Limit=512Mi
    * Namespace=openshift
    * Database Service Name=db
    * MySQL Connection Username=operator
    * MySQL Connection Password=myP@55
    * MySQL root user Password=tLH8BThuVgnIrCon # generated
    * MySQL Database Name=mydata
    * Volume Capacity=1Gi
    * Version of MySQL Image=8.0-e18

--> Creating resources ...
  secret "db" created
  service "db" created
  persistentvolumeclaim "db" created
```

```
deploymentconfig.apps.openshift.io "db" created
--> Success
...output omitted...
```

Similar to the `create` command, you can use the `new-app` command imperatively. When using a container image with the `new-app`, you are instructing the cluster what to do, rather than declaring the intended outcomes. For example, the following command deploys the `example.com/my-app:dev` image by creating several resources, including a deployment resource.

```
[user@host ~]$ oc new-app --image example.com/my-app:dev
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "my-app" created
deployment.apps "my-app" created
...output omitted...
```

You can also supply a Git repository to the `new-app` command. The following command creates an application named `httpd24` by using a Git repository.

```
[user@host ~]$ oc new-app https://github.com/apache/httpd.git#2.4.56
...output omitted...
--> Creating resources ...
imagestream.image.openshift.io "httpd24" created
deployment.apps "httpd24" created
...output omitted...
```

## Retrieving Resource Information

You can supply an `all` argument to commands, to specify a list of common resource types. However, the `all` option does not include every available resource type. Instead, `all` is a shorthand form for a predefined subset of types. When you use this command argument, ensure that `all` includes any intended types to address.

You can view detailed information about a resource, such as the defined parameters, by using the `describe` command. For example, RHOC provides templates in the `openshift` project to use with the `oc new-app` command. The following example command displays detailed information about the `mysql-ephemeral` template:

```
[user@host ~]$ oc describe template mysql-ephemeral -n openshift
Name: mysql-ephemeral
Namespace: openshift
...output omitted...
Parameters:
  Name: MEMORY_LIMIT
  Display Name: Memory Limit
  Description: Maximum amount of memory the container can use.
  Required: true
  Value: 512Mi

  Name: NAMESPACE
  Display Name: Namespace
  Description: The OpenShift Namespace where the ImageStream resides.
```

```
Required: false
Value: openshift
...output omitted...
Objects:
Secret ${DATABASE_SERVICE_NAME}
Service ${DATABASE_SERVICE_NAME}
DeploymentConfig.apps.openshift.io ${DATABASE_SERVICE_NAME}
```

The `describe` command cannot generate structured output, such as the YAML or JSON formats. Without a structured format, the `describe` command cannot parse or filter the output with tools such as JSONPath or Go templates. Instead, use the `get` command to generate and then to parse the structured output of a resource.



## References

### **OpenShift Container Platform Documentation - Understanding Deployment and DeploymentConfig Objects**

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#what-deployments-are](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#what-deployments-are)

### **OpenShift Container Platform Documentation - Working with Projects**

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#working-with-projects](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#working-with-projects)

### **OpenShift Container Platform Documentation - Creating Applications Using the CLI**

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#creating-applications-using-cli](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#creating-applications-using-cli)

### **OpenShift Container Platform Documentation - Using Pods**

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-pods-using-pp](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-pods-using-pp)

## ► Guided Exercise

# Deploy Applications from Images and Templates

Deploy a database from a container image and from a template by using the OpenShift command-line interface and compare the resources and attributes generated by each method.

### Outcomes

In this exercise, you deploy two MySQL database server pods to compare deployment methods and the resources that each creates.

- Deploy a database from a template.
- Deploy a database from a container 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 resources are available for the exercise.

```
[student@workstation ~]$ lab start deploy-newapp
```

## Instructions

- 1. As the developer user, create a project and verify that it is not empty after creation.

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

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

- 1.2. Create a project named `deploy-newapp`.

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

The new project is automatically selected.

- 1.3. Observe that resources for the new project are not returned with the `oc get all` command.

```
[student@workstation ~]$ oc get all
No resources found in deploy-newapp namespace.
```

**Important**

Commands that use `all` for the resource type do not include every available resource type. Instead, `all` is a shorthand form for a predefined subset of types. When you use this command argument, ensure that `all` includes any types that you intend to address.

- 1.4. Observe that the new project contains other types of resources.

```
[student@workstation ~]$ oc get serviceaccounts
NAME      SECRETS   AGE
builder   1          20s
default   1          20s
deployer  1          20s
```

```
[student@workstation ~]$ oc get secrets
NAME           TYPE          DATA   AGE
builder-dockercfg-sczxg  kubernetes.io/dockercfg    1      36m
builder-token-gsnqj     kubernetes.io/service-account-token 4      36m
default-dockercfg-6f8nm kubernetes.io/dockercfg    1      36m
...output omitted...
```

- ▶ 2. Create two MySQL instances by using the `oc new-app` command with different options.

- 2.1. View the `mysql-persistent` template definition to see the resources that it creates. Specify the project that houses the template by using the `-n openshift` option.

```
[student@workstation ~]$ oc describe template mysql-persistent -n openshift
Name:  mysql-persistent
Namespace:  openshift
...output omitted...
Objects:
  Secret          ${DATABASE_SERVICE_NAME}
  Service         ${DATABASE_SERVICE_NAME}
  PersistentVolumeClaim  ${DATABASE_SERVICE_NAME}
  DeploymentConfig.apps.openshift.io  ${DATABASE_SERVICE_NAME}
```

The `objects` attribute specifies several resource definitions that are applied on using the template. These resources include one or each of the following types: secret, service (svc), persistent volume claim (pvc), and deployment configuration (dc).

- 2.2. Create an instance by using the `mysql-persistent` template. Specify a `name` option and attach a custom `team=red` label to the created resources.

```
[student@workstation ~]$ oc new-app -l team=red --template mysql-persistent \
-p MYSQL_USER=developer \
-p MYSQL_PASSWORD=developer
...output omitted...
--> Creating resources with label team=red ...
secret "mysql" created
```

```
service "mysql" created
persistentvolumeclaim "mysql" created
deploymentconfig.apps.openshift.io "mysql" created
--> Success
...output omitted...
```

The template creates resources of the types from the preceding step.

- View and wait for the pod to start, which takes a few minutes to complete. You might need to run the command several times before the status changes to **Running**.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
mysql-1-deploy 0/1     Completed  0          93s
mysql-1-qmxbf  1/1     Running   1          60s
```

Your pod names might differ from the previous output.

- Create an instance by using a container image. Specify a name option and attach a custom `team=blue` label to the created resources.

```
[student@workstation ~]$ oc new-app --name db-image -l team=blue \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1 \
-e MYSQL_USER=developer \
-e MYSQL_PASSWORD=developer \
-e MYSQL_ROOT_PASSWORD=redhat
...output omitted...
--> Creating resources with label team=blue ...
imagestream.image.openshift.io "db-image" created
deployment.apps "db-image" created
service "db-image" created
--> Success
...output omitted...
```

The command creates predefined resources that are needed to deploy an image. These resource types are image stream (`is`), deployment, and service (`svc`). Image streams and services are discussed in more detail elsewhere in the course.



### Note

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

- Wait for the pod to start. After a few moments, list all pods that contain `team` as a label.

```
[student@workstation ~]$ oc get pods -L team
NAME           READY   STATUS    RESTARTS   AGE   TEAM
db-image-8d4b97594-6jb85  1/1     Running   0      55s   blue
mysql-1-deploy      0/1     Completed  0      2m
mysql-1-hn64v        1/1     Running   0      1m30s
```

Your pod name might differ from the previous output. Without a `readinessProbe`, this pod shows as ready before the MySQL service is ready for requests. Readiness probes are discussed in more detail elsewhere in the course.

Notice that only the db-image pod has a label that contains the word team. Pods that the mysql-persistent template creates do not have the team=red label, because the template does not define this label in its pod specification template.

► 3. Compare the resources that each image and template method creates.

3.1. View the template-created pod and observe that it contains a readiness probe.

```
[student@workstation ~]$ oc get pods -l deploymentconfig=mysql \
-o jsonpath='{.items[0].spec.containers[0].readinessProbe}' | jq
{
  "exec": {
    "command": [
      "/bin/sh",
      "-i",
      "-c",
      "MYSQL_PWD=\"$MYSQL_PASSWORD\" mysqladmin -u $MYSQL_USER ping"
    ]
  },
  "failureThreshold": 3,
  "initialDelaySeconds": 5,
  "periodSeconds": 10,
  "successThreshold": 1,
  "timeoutSeconds": 1
}
```



**Note**

The results of the preceding oc command are passed to the jq command, which formats the JSON output.

3.2. Observe that the image-based pod does not contain a readiness probe.

```
[student@workstation ~]$ oc get pods -l deployment=db-image \
-o jsonpath='{.items[0].spec.containers[0].readinessProbe}' | jq
no output expected
```

3.3. Observe that the template-based pod has a memory resource limit, which restricts allocated memory to the resulting pods. Resource limits are discussed in more detail elsewhere in the course.

```
[student@workstation ~]$ oc get pods -l deploymentconfig=mysql \
-o jsonpath='{.items[0].spec.containers[0].resources.limits}' | jq
{
  "memory": "512Mi"
}
```

3.4. Observe that the image-based pod has no resource limits.

```
[student@workstation ~]$ oc get pods -l deployment=db-image \
-o jsonpath='{.items[0].spec.containers[0].resources}' | jq
{}
```

- 3.5. Retrieve secrets in the project. Notice that the template produced a secret, whereas the pod that was created with only an image did not.

```
[student@workstation ~]$ oc get secrets
NAME          TYPE
...output omitted...
mysql         Opaque
DATA          AGE
4             3m
```

► 4. Explore filtering resources via labels.

- 4.1. Observe that not supplying a label shows all services.

```
[student@workstation ~]$ oc get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
db-image  ClusterIP  172.30.38.113  <none>        3306/TCP    1m30s
mysql     ClusterIP  172.30.95.52   <none>        3306/TCP    2m30s
```

- 4.2. Observe that supplying a label shows only the services with the label.

```
[student@workstation ~]$ oc get services -l team=red
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
mysql     ClusterIP  172.30.95.52   <none>        3306/TCP    2m43s
```

- 4.3. Observe that not all resources include the label, such as the pods that are created with the template.

```
[student@workstation ~]$ oc get pods -l team=red
No resources found in deploy-newapp namespace.
```

► 5. Use labels to delete only the resources that are associated with the image-based deployment.

- 5.1. Delete only the resources that use the `team=red` label by using it with the `oc delete` command. List the resource types from the template to ensure that all relevant resources are deleted.

```
[student@workstation ~]$ oc delete all -l team=red
replicationcontroller "mysql-1" deleted
service "mysql" deleted
...output omitted...
deploymentconfig.apps.openshift.io "mysql" deleted
```

```
[student@workstation ~]$ oc delete secret,pvc -l team=red
secret "mysql" deleted
persistentvolumeclaim "mysql" deleted
```



**Note**

By using the `oc delete all -l team=red` command, some resources are deleted, but the persistent volume claim and secret remain.

5.2. Observe that the resources that the template created are deleted.

```
[student@workstation ~]$ oc get secret,svc,pvc,dc -l team=red  
...output omitted...  
No resources found in deploy-newapp namespace.
```

5.3. Observe that the image-based resources remain unchanged.

```
[student@workstation ~]$ oc get is,deployment,svc  
NAME                                     IMAGE REPOSITORY ...output omitted...  
imagestream.image.openshift.io/db-image   image-registry.openshift...output  
omitted...  
NAME          READY  UP-TO-DATE  AVAILABLE  AGE  
deployment.apps/db-image     1/1      1           1        46m  
  
NAME          TYPE    CLUSTER-IP      EXTERNAL-IP    PORT(S)    AGE  
service/db-image  ClusterIP  172.30.71.0  <none>       3306/TCP  46m
```

## 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 deploy-newapp
```

# Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API

## Objectives

- Deploy containerized applications as pods that Kubernetes workload resources manage.

## Kubernetes Workload Resources

The Kubernetes Workloads API consists of resources that represent various types of cluster tasks, jobs, and workloads. This API is composed of various Kubernetes APIs and extensions that simplify deploying and managing applications. These resource types of the Workloads API are most often used:

- Jobs
- Deployments
- Stateful sets

## Jobs

A *job* resource represents a one-time task to perform on the cluster. As with most cluster tasks, jobs are executed via pods. If a job's pod fails, then the cluster retries a number of times that the job specifies. The job does not run again after a specified number of successful completions.

Jobs differ from using the `kubectl run` and `oc run` commands; both of the latter create only a pod.

Common uses for jobs include the following tasks:

- Initializing or migrating a database
- Calculating one-off metrics from information within the cluster
- Creating or restoring from a data backup

The following example command creates a job that logs the date and time:

```
[user@host ~]$ oc create job \ ①  
date-job \ ②  
--image registry.access.redhat.com/ubi8/ubi \ ③  
-- /bin/bash -c "date" ④
```

- ① Creates a job resource.
- ② Specifies a job name of `date-job`.
- ③ Sets `registry.access.redhat.com/ubi8/ubi` as the container image for the job pods.
- ④ Specifies the command to run within the pods.

You can also create a job from the web console by clicking the **Workloads > Jobs** menu. Click **Create Job** and customize the YAML manifest.

## Cron Jobs

A *cron job* resource builds on a regular job resource by enabling you to specify how often the job should run. Cron jobs are useful for creating periodic and recurring tasks, such as backups or report generation. Cron jobs can also schedule individual tasks for a specific time, such as to schedule a job for a low activity period. Similar to the `crontab` (cron table) file on a Linux system, the `CronJob` resource uses the `Cron` format for scheduling. A `CronJob` resource creates a job resource based on the configured time zone on the control plane node that runs the cron job controller.

The following example command creates a cron job named `date` that prints the system date and time every minute:

```
[user@host ~]$ oc create cronjob date-cronjob \ ①
--image registry.access.redhat.com/ubi8/ubi \ ②
--schedule */1 * * * * \ ③
--date ④
```

- ① Creates a cron job resource with a name of `date-cronjob`.
- ② Sets the `registry.access.redhat.com/ubi8/ubi` as the container image for the job pods.
- ③ Specifies the schedule for the job in `Cron` format.
- ④ The command to execute within the pods.

You can also create a `cronjob` from the web console by clicking the `Workloads > CronJobs` menu. Click `Create CronJob` and customize the YAML manifest.

## Deployments

A *deployment* creates a replica set to maintain pods. A *replica set* maintains a specified number of replicas of a pod. Replica sets use selectors, such as a label, to identify pods that are part of the set. Pods are created or removed until the replicas reach the number that the deployment specifies. Replica sets are not managed directly. Instead, deployments indirectly manage replica sets.

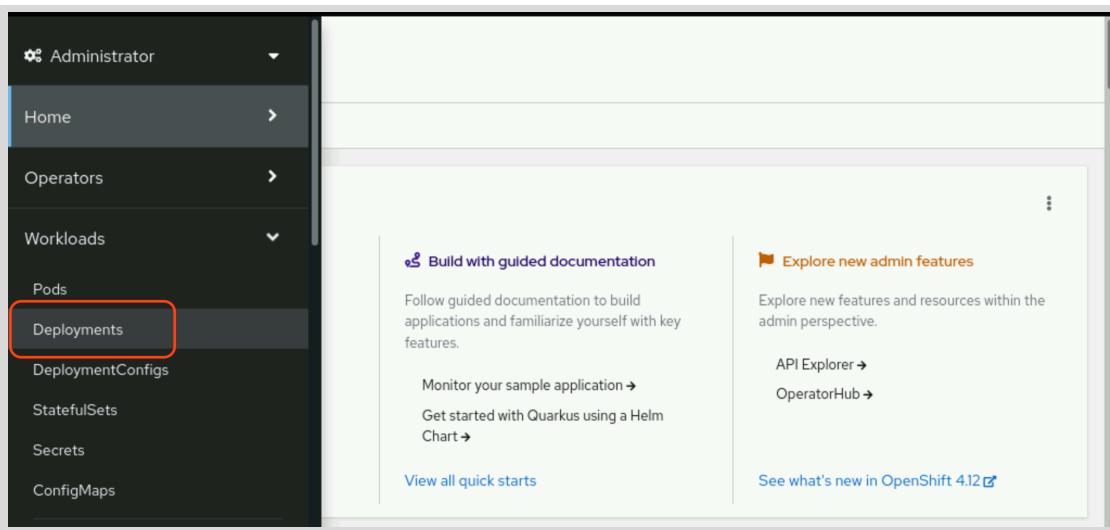
Unlike a job, a deployment's pods are re-created after crashing or deletion. The reason is that deployments use replica sets.

The following example command creates a deployment:

```
[user@host ~]$ oc create deployment \ ①
my-deployment \ ②
--image registry.access.redhat.com/ubi8/ubi \ ③
--replicas 3 ④
```

- ① Creates a deployment resource.
- ② Specifies `my-deployment` as the deployment name.
- ③ Sets `registry.access.redhat.com/ubi8/ubi` as the container image for the pods.
- ④ Sets the deployment to maintain three instances of the pod.

You can also create a deployment from the web console by clicking the **Deployments** tab in the **Workloads** menu.



Click **Create Deployment** and customize the form or the YAML manifest.

A screenshot of the 'Create Deployment' form. At the top, it says 'Create Deployment' and has a 'Configure via:' section with 'Form view' (selected) and 'YAML view' radio buttons. Below this is a blue header bar with a warning message: 'DeploymentConfig is being deprecated with OpenShift 4.14'. The message states that feature development of DeploymentConfigs will be deprecated in OpenShift Container Platform 4.14, and DeploymentConfigs will continue to be supported for security and critical fixes, but users should migrate to Deployments wherever it is possible. A 'Learn more about Deployments' link is provided. Another blue header bar below says 'Note: Some fields may not be represented in this form view. Please select "YAML view" for full control.' At the bottom, there is a 'Name' field with a red asterisk indicating it is required, followed by a text input field.

Pods in a replica set are identical and match the pod template in the replica set definition. If the number of replicas is not met, then a new pod is created by using the template. For example, if a pod crashes or is otherwise deleted, then a new one is created to replace it.

*Labels* are a type of resource metadata that are represented as string key-value pairs. A label indicates a common trait for resources with that label. For example, you might attach a `layer=frontend` label to resources that relate to an application's UI components.

Many `oc` and `kubectl` commands accept a label to filter affected resources. For example, the following command deletes all pods with the `environment=testing` label:

```
[user@host ~]$ oc delete pod -l environment=testing
```

By liberally applying labels to resources, you can cross-reference resources and craft precise selectors. A *selector* is a query object that describes the attributes of matching resources.

Certain resources use selectors to find other resources. In the following YAML excerpt, an example replica set uses a selector to match its pods.

```
apiVersion: apps/v1
kind: ReplicaSet
...output omitted...
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpd
      pod-template-hash: 7c84fbdb57
...output omitted...
```

## Stateful Sets

Like deployments, *stateful sets* manage a set of pods based on a container specification. However, each pod that a stateful set creates is unique. Pod uniqueness is useful when, for example, a pod needs a unique network identifier or persistent storage.

As their name implies, stateful sets are for pods that require state within the cluster. Deployments are used for stateless pods.

Stateful sets are covered further in a later chapter.



### References

#### Kubernetes Workloads

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

#### Kubernetes Labels and Selectors

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

## ► Guided Exercise

# Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API

Deploy a batch application managed by a job resource and a database server that a deployment resource manages by using the Kubernetes command-line interface.

## Outcomes

In this exercise, you deploy a database server and a batch application that are both managed by workload resources.

- Create deployments.
- Update environment variables on a pod template.
- Create and run job resources.
- Retrieve the logs and termination status of a job.
- View the pod template of a job resource.

## 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 resources are available for the exercise.

```
[student@workstation ~]$ lab start deploy-workloads
```

## Instructions

- 1. As the developer user, create a MySQL deployment in a new project.

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

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

- 1.2. Create a project named `deploy-workloads`.

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

13. Create a deployment that runs an ephemeral MySQL server.

```
[student@workstation ~]$ oc create deployment my-db \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1
Warning: would violate PodSecurity "restricted:v1.24"
...output omitted...
deployment.apps/my-db created
```

**Note**

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

14. Retrieve the status of the deployment.

```
[student@workstation ~]$ oc get deployments
NAME      READY     UP-TO-DATE   AVAILABLE   AGE
my-db    0/1       1           0           67s
```

The deployment never has a ready instance.

15. Retrieve the status of the created pod. Your pod name might differ from the output.

```
[student@workstation ~]$ oc get pods
NAME                  READY     STATUS            RESTARTS   AGE
my-db-8567b478dd-d28f7  0/1     CrashLoopBackOff   4 (60s ago)  2m35s
```

The pod fails to start and repeatedly crashes.

16. Review the logs for the pod to determine why it fails to start.

```
[student@workstation ~]$ oc logs deploy/my-db
...output omitted...
You must either specify the following environment variables:
  MYSQL_USER (regex: '^$')
  MYSQL_PASSWORD (regex: '^[_a-zA-Z0-9~!@#$%^&*()-=<>,.?;:]$')
  MYSQL_DATABASE (regex: '^$')
Or the following environment variable:
  MYSQL_ROOT_PASSWORD (regex: '^[_a-zA-Z0-9~!@#$%^&*()-=<>,.?;:]$')
...output omitted...
```

Note that the container fails to start due to missing environment variables.

- 2. Fix the database deployment and verify that the server is running.

- 2.1. Set the `MYSQL_USER`, `MYSQL_PASSWORD`, and `MYSQL_DATABASE` environment variables.

```
[student@workstation ~]$ oc set env deployment/my-db \
  MYSQL_USER=developer \
  MYSQL_PASSWORD=developer \
  MYSQL_DATABASE=sampled
deployment.apps/my-db updated
```

- 2.2. Retrieve the list of deployments and observe that the `my-db` deployment has a running pod.

```
[student@workstation ~]$ oc get deployments
NAME      READY     UP-TO-DATE   AVAILABLE   AGE
my-db    1/1       1           1           4m50s
```

- 2.3. Retrieve the internal IP address of the MySQL pod within the list of all pods.

```
[student@workstation ~]$ oc get pods -o wide
NAME                  READY   STATUS    RESTARTS   AGE   IP          ...
my-db-748c97d478-g8xc9  1/1    Running   0          64s   10.8.0.91 ...
```

The `-o wide` option enables additional output, such as IP addresses. Your IP address value might differ from the previous output.

- 2.4. Verify that the database server is running, by running a query. Replace the IP address with the one that you retrieved in the preceding step.

```
[student@workstation ~]$ oc run -it db-test --restart=Never \
--image registry.ocp4.example.com:8443/rhel9/mysql-80:1 \
-- mysql sampledb -h 10.8.0.91 -u developer --password=developer \
-e "select 1;" \
...output omitted...
...
| 1 |
...
| 1 |
...
```

- 3. Delete the database server pod and observe that the deployment causes the pod to be re-created.

- 3.1. Delete the existing MySQL pod by using the label that is associated with the deployment.

```
[student@workstation ~]$ oc delete pod -l app=my-db
pod "my-db-84c8995d5-2sss1" deleted
```

- 3.2. Retrieve the information for the MySQL pod and observe that it is newly created. Your pod name might differ in your output.

```
[student@workstation ~]$ oc get pod -l app=my-db
NAME      READY   STATUS    RESTARTS   AGE
my-db-fbccb9447-p99jd  1/1    Running   0          6s
```

- 4. Create and apply a job resource that prints the time and date repeatedly.

- 4.1. Create a job resource called `date-loop` that runs a script. Ignore the warning.

```
[student@workstation ~]$ oc create job date-loop \
--image registry.ocp4.example.com:8443/ubi9/ubi \
-- /bin/bash -c "for i in {1..30}; do date; done"
job.batch/date-loop created
```

4.2. Retrieve the job resource to review the pod specification.

```
[student@workstation ~]$ oc get job date-loop -o yaml
...output omitted...
spec:
  containers:
    - command: ❶
      - /bin/bash
      - -c
      - for i in {1..30}; do date; done
    image: registry.ocp4.example.com:8443/ubi9/ubi ❷
    imagePullPolicy: Always
    name: date-loop
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    restartPolicy: Never ❸
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
...output omitted...
```

- ❶ The command object, which specifies the defined script to execute within the pod.
- ❷ Sets the container image for the pod.
- ❸ Defines the restart policy for the pod. Kubernetes does not restart the job pod after the pod exits.

4.3. List the jobs to see that the date-loop job completed successfully.

```
[student@workstation ~]$ oc get jobs
NAME      COMPLETIONS   DURATION   AGE
date-loop  1/1          7s          8s
```

You might need to wait for the script to finish and run the command again.

4.4. Retrieve the logs for the associated pod. The log values might differ in your output.

```
[student@workstation ~]$ oc logs job/date-loop
Fri Nov 18 14:50:56 UTC 2022
Fri Nov 18 14:50:59 UTC 2022
...output omitted...
```

- 5. Delete the pod for the date-loop job and observe that the pod is not created again.

- 5.1. Delete the associated pod.

```
[student@workstation ~]$ oc delete pod -l job-name=date-loop  
pod "date-loop-wvn2q" deleted
```

- 5.2. View the list of pods and observe that the pod is not re-created for the job.

```
[student@workstation ~]$ oc get pod -l job-name=date-loop  
No resources found in deploy-workloads namespace.
```

- 5.3. Verify that the job status is still listed as successfully completed.

```
[student@workstation ~]$ oc get job -l job-name=date-loop  
NAME      COMPLETIONS      DURATION      AGE  
date-loop  1/1            7s            7m36s
```

## 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 deploy-workloads
```

# Kubernetes Pod and Service Networks

---

## Objectives

- Interconnect applications pods inside the same cluster by using Kubernetes services.

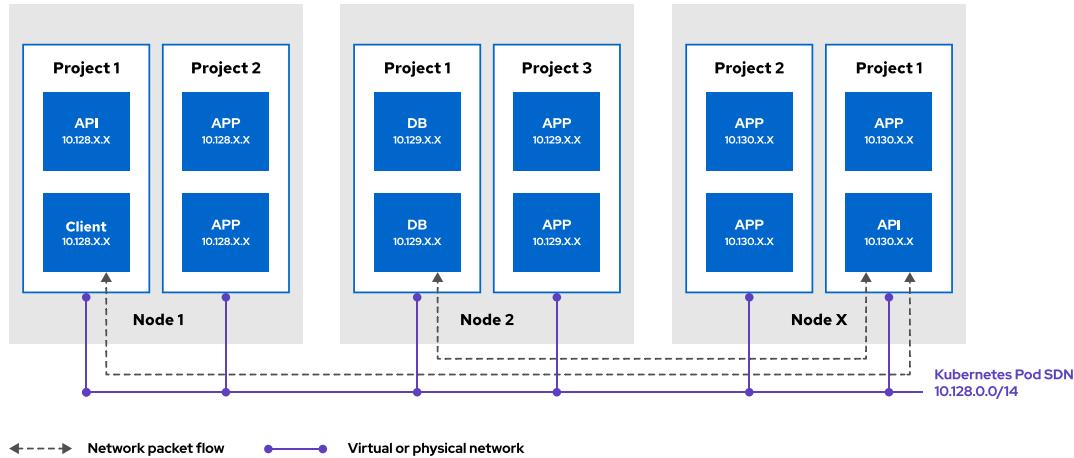
## The Software-defined Network

Kubernetes implements software-defined networking (SDN) to manage the network infrastructure of the cluster and user applications. The SDN is a virtual network that encompasses all cluster nodes. The virtual network enables communication between any container or pod inside the cluster. Cluster node processes that Kubernetes pods manage can access the SDN. However, the SDN is not accessible from outside the cluster, nor to regular processes on cluster nodes. With the software-defined networking model, you can manage network services through the abstraction of several networking layers.

With the SDN, you can manage the network traffic and network resources programmatically, so that the organization teams can decide how to expose their applications. The SDN implementation creates a model that is compatible with traditional networking practices. It makes pods akin to virtual machines in terms of port allocation, IP address leasing, and reservation.

With the SDN design, you do not need to change how application components communicate with each other, which helps to containerize legacy applications. If your application is composed of many services that communicate over the TCP/UDP stack, then this approach still works, because containers in a pod use the same network stack.

The following diagram shows how all pods are connected to a shared network:



**Figure 4.5: How the Kubernetes SDN manages the network**

Among the many features of SDN, with open standards, vendors can propose their solutions for centralized management, dynamic routing, and tenant isolation.

## Kubernetes Networking

Networking in Kubernetes provides a scalable means of communication between containers.

Kubernetes networking provides the following capabilities:

- Highly coupled container-to-container communications
- Pod-to-pod communications
- Pod-to-service communications
- External-to-service communication: covered in *Scale and Expose Applications to External Access*

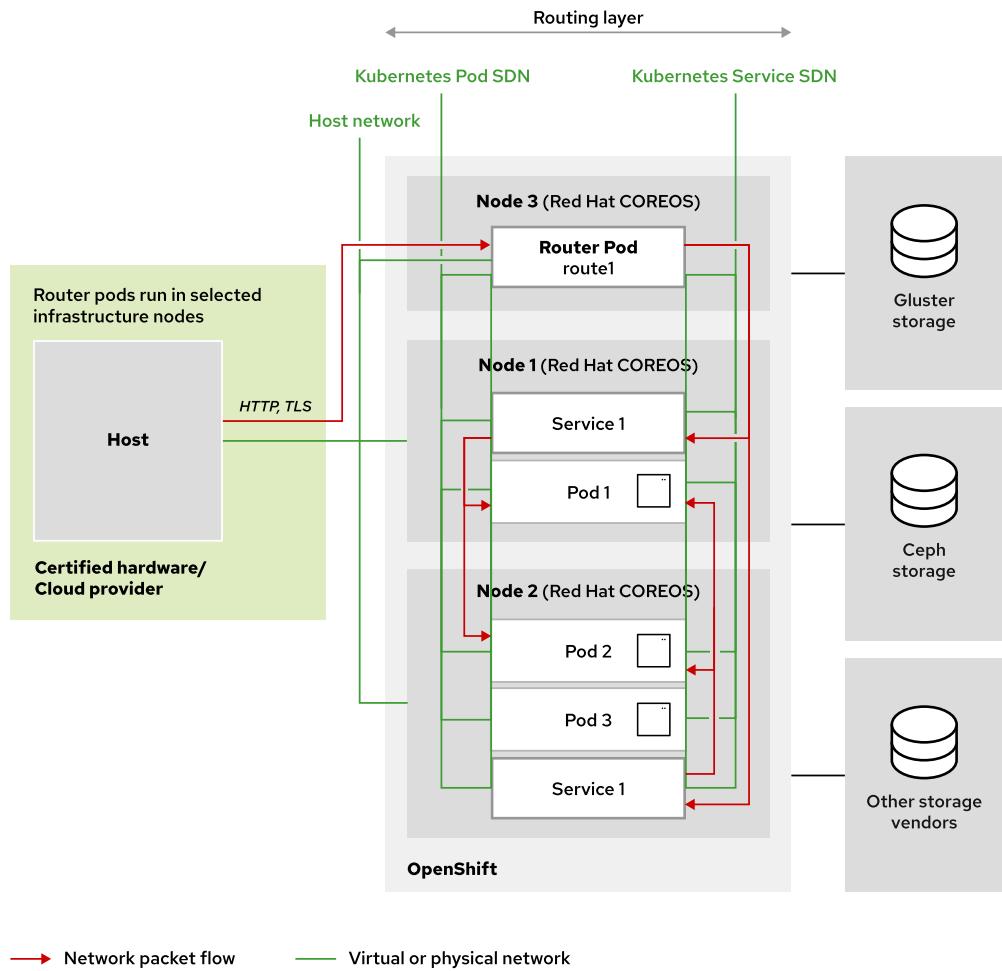
Kubernetes automatically assigns an IP address to every pod. However, pod IP addresses are unstable, because pods are ephemeral. Pods are constantly created and destroyed across the nodes in the cluster. For example, when you deploy a new version of your application, Kubernetes destroys the existing pods and then deploys new ones.

All containers within a pod share networking resources. The IP address and MAC address that are assigned to the pod are shared among all containers in the pod. Thus, all containers within a pod can reach each other's ports through the loopback address, `localhost`. Ports that are bound to `localhost` are available to all containers that run within the pod, but never to containers outside it.

By default, the pods can communicate with each other even if they run on different cluster nodes or belong to different Kubernetes namespaces. Every pod is assigned an IP address in a flat shared networking namespace that has full communication with other physical computers and containers across the network. All pods are assigned a unique IP address from a Classless Inter-Domain Routing (CIDR) range of host addresses. The shared address range places all pods in the same subnet.

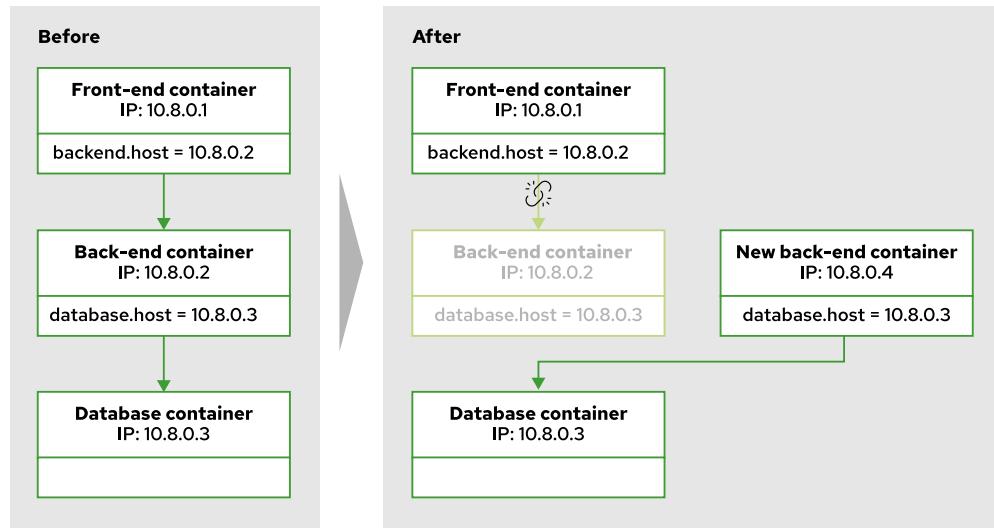
Because all the pods are on the same subnet, pods on all nodes can communicate with pods on any other node without the aid of Network Address Translation (NAT). Kubernetes also provides a service subnet, which links the stable IP address of a service resource to a set of specified pods. The traffic is forwarded in a transparent way to the pods; an agent (depending on the network mode that you use) manages routing rules to route traffic to the pods that match the service resource selectors. Thus, pods can be treated much like Virtual Machines (VMs) or physical hosts from the perspective of port allocation, networking, naming, service discovery, load balancing, application configuration, and migration. Kubernetes implements this infrastructure by managing the SDN.

The following illustration gives further insight into how the infrastructure components work along with the pod and service subnets to enable network access between pods inside an OpenShift instance.



**Figure 4.6: Network access between pods in a cluster**

The shared networking namespace of pods enables a straightforward communication model. However, the dynamic nature of pods presents a problem. Pods can be added on the fly to handle increased traffic. Likewise, pods can be dynamically scaled down. If a pod fails, then Kubernetes automatically replaces the pod with a new one. These events change pod IP addresses.

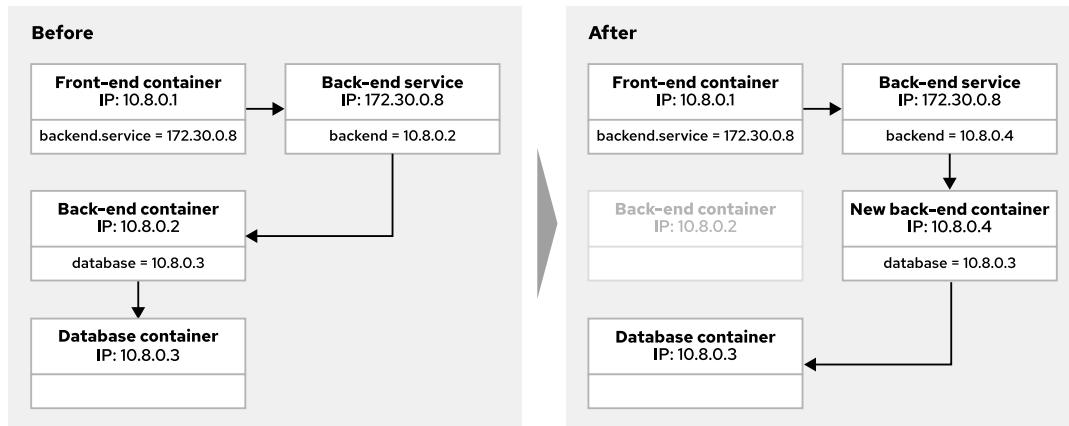
**Figure 4.7: Problem with direct access to pods**

In the diagram, the **Before** side shows the Front-end container that is running in a pod with a 10.8.0.1 IP address. The container also refers to a Back-end container that is running in a pod with a 10.8.0.2 IP address. In this example, an event occurs that causes the Back-end container to fail. A pod can fail for many reasons. In response to the failure, Kubernetes creates a pod for the Back-end container that uses a new IP address of 10.8.0.4. From the **After** side of the diagram, the Front-end container now has an invalid reference to the Back-end container because of the IP address change. Kubernetes resolves this problem with service resources.

## Using Services

Containers inside Kubernetes pods must not connect directly to each other's dynamic IP address. Instead, Kubernetes assigns a stable IP address to a service resource that is linked to a set of specified pods. The service then acts as a virtual network load balancer for the pods that are linked to the service.

If the pods are restarted, replicated, or rescheduled to different nodes, then the service endpoints are updated, thus providing scalability and fault tolerance for your applications. Unlike the IP addresses of pods, the IP addresses of services do not change.

**Figure 4.8: Services resolve pod failure issues**

In the diagram, the **Before** side shows that the **Front-end container** now holds a reference to the stable IP address of the **Back-end service**, instead of to the IP address of the pod that is running the **Back-end container**. When the **Back-end container** fails, Kubernetes creates a pod with the **New back-end container** to replace the failed pod. In response to the change, Kubernetes removes the failed pod from the service's host list, or service endpoints, and then adds the IP address of the **New back-end container** pod to the service endpoints. With the addition of the service, requests from the **Front-end container** to the **Back-end container** continue to work, because the service is dynamically updated with the IP address change. A service provides a permanent, static IP address for a group of pods that belong to the same deployment or replica set for an application. Until you delete the service, the assigned IP address does not change, and the cluster does not reuse it.

Most real-world applications do not run as a single pod. Applications need to scale horizontally. Multiple pods run the same containers to meet a growing user demand. A *Deployment* resource manages multiple pods that execute the same container. A service provides a single IP address for the whole set, and provides load-balancing for client requests among the member pods.

With services, containers in one pod can open network connections to containers in another pod. The pods, which the service tracks, are not required to exist on the same compute node or in the same namespace or project. Because a service provides a stable IP address for other pods to use, a pod also does not need to discover the new IP address of another pod after a restart. The service provides a stable IP address to use, no matter which compute node runs the pod after each restart.

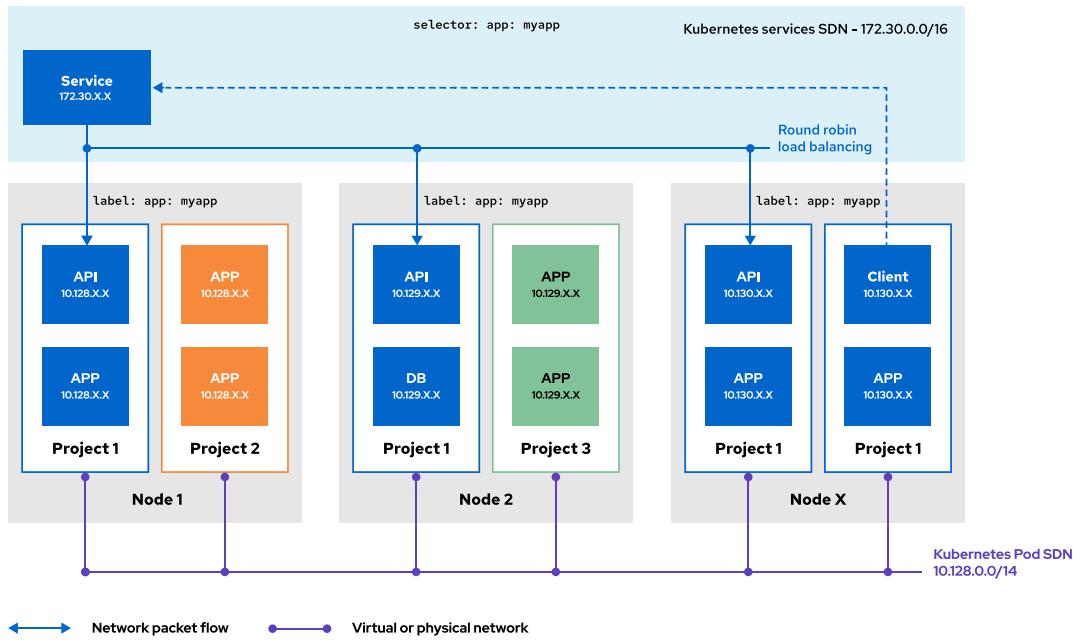


Figure 4.9: Service with pods on many nodes

The **SERVICE** object provides a stable IP address for the **CLIENT** container on **NODE X** to send a request to any one of the **API** containers.

Kubernetes uses labels on the pods to select the pods that are associated with a service. To include a pod in a service, the pod labels must include each of the **selector** fields of the service.

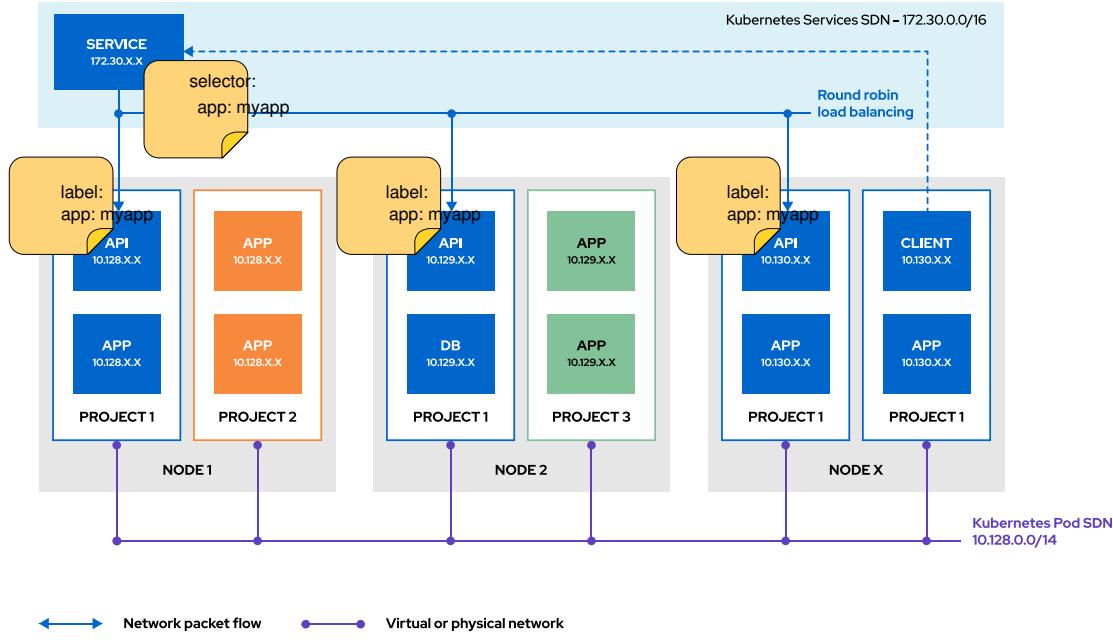


Figure 4.10: Service selector match to pod labels

In this example, the selector has a key-value pair of **app: myapp**. Thus, pods with a matching label of **app: myapp** are included in the set that is associated with the service. The **selector** attribute of a service is used to identify the set of pods that form the endpoints for the service. Each pod in the set is an endpoint for the service.

To create a service for a deployment, use the `oc expose` command:

```
[user@host ~]$ oc expose deployment/<deployment-name> [--selector <selector>]
[--port <port>][--target-port <target port>][--protocol <protocol>][--name <name>]
```

The `oc expose` command can use the `--selector` option to specify the label selectors to use. When the command is used without the `--selector` option, the command applies a selector to match the replication controller or replica set.

The `--port` option of the `oc expose` command specifies the port that the service listens on. This port is available only to pods within the cluster. If a port value is not provided, then the port is copied from the deployment configuration.

The `--target-port` option of the `oc expose` command specifies the name or number of the container port that the service uses to communicate with the pods.

The `--protocol` option determines the network protocol for the service. TCP is used by default.

The `--name` option of the `oc expose` command can explicitly name the service. If not specified, the service uses the same name that is provided for the deployment.

To view the selector that a service uses, use the `-o wide` option with the `oc get` command.

```
[user@host ~]$ oc get service db-pod -o wide
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP PORT(S)      AGE      SELECTOR
db-pod   ClusterIP  172.30.108.92  <none>        3306/TCP   108s    app=db-pod
```

In this example, `db-pod` is the name of the service. Pods must use the `app=db-pod` label to be included in the host list for the `db-pod` service. To see the endpoints that a service uses, use the `oc get endpoints` command.

```
[user@host ~]$ oc get endpoints
NAME      ENDPOINTS      AGE
db-pod   10.8.0.86:3306,10.8.0.88:3306  27s
```

This example illustrates a service with two pods in the host list. The `oc get endpoints` command returns the service endpoints in the current selected project. Add the name of the service to the command to show only the endpoints of a single service. Use the `--namespace` option to view the endpoints in a different namespace.

Use the `oc describe deployment <deployment name>` command to view the deployment selector.

```
[user@host ~]$ oc describe deployment db-pod
Name:           db-pod
Namespace:      deploy-services
CreationTimestamp:  Wed, 18 Jan 2023 17:46:03 -0500
Labels:          app=db-pod
Annotations:    deployment.kubernetes.io/revision: 2
Selector:        app=db-pod
...output omitted...
```

You can view or parse the selector from the YAML or JSON output for the deployment resource from the `spec.selector.matchLabels` object. In this example, the `-o yaml` option of the `oc get` command returns the selector label that the deployment uses.

```
[user@host ~]$ oc get deployment/<deployment_name> -o yaml
...output omitted...
selector:
  matchLabels:
    app: db-pod
...output omitted...
```

## Kubernetes DNS for Service Discovery

Kubernetes uses an internal Domain Name System (DNS) server that the DNS operator deploys. The DNS operator creates a default cluster DNS name, and assigns DNS names to services that you define. The DNS operator implements the DNS API from the `operator.openshift.io` API group. The operator deploys CoreDNS; creates a service resource for the CoreDNS; and then configures the `kubelet` to instruct pods to use the CoreDNS service IP for name resolution. When a service does not have a cluster IP address, the DNS operator assigns to the service a DNS record that resolves to the set of IP addresses of the pods behind the service.

The DNS server discovers a service from a pod by using the internal DNS server, which is visible only to pods. Each service is dynamically assigned a *Fully Qualified Domain Name* (FQDN) that uses the following format:

```
SVC-NAME.PROJECT-NAME.svc.CLUSTER-DOMAIN
```

When a pod is created, Kubernetes provides the container with a `/etc/resolv.conf` file with similar contents to the following items:

```
[user@host ~]$ cat /etc/resolv.conf
search deploy-services.svc.cluster.local svc.cluster.local ...
nameserver 172.30.0.10
options ndots:5
```

In this example, `deploy-services` is the project name for the pod, and `cluster.local` is the cluster domain.

The `nameserver` directive provides the IP address of the Kubernetes internal DNS server. The `options ndots` directive specifies the number of dots that must appear in a name to qualify for an initial absolute query. Alternative hostname values are derived by appending values from the `Search` directive to the name that is sent to the DNS server.

In the `search` directive in this example, the `svc.cluster.local` entry enables any pod to communicate with another pod in the same cluster by using the service name and project name:

```
SVC-NAME.PROJECT-NAME
```

The first entry in the `search` directive enables a pod to use the service name to specify another pod in the same project. In RHOCP, a project is also the namespace for the pod. The service name alone is sufficient for pods in the same RHOCP project:

SVC-NAME

## Kubernetes Networking Drivers

Container Network Interface (CNI) plug-ins provide a common interface between the network provider and the container runtime. CNI defines the specifications for plug-ins that configure network interfaces inside containers. Plug-ins that are written to the specification enable different network providers to control the RHOCP cluster network.

Red Hat provides the following CNI plug-ins for a RHOCP cluster:

- OVN-Kubernetes: The default plug-in for first-time installations of RHOCP, starting with RHOCP 4.10.
- OpenShift SDN: An earlier plug-in from RHOCP 3.x; it is incompatible with some later features of RHOCP 4.x.
- Kuryr: A plug-in for integration and performance on OpenStack deployments.

Certified CNI-plugins from other vendors are also compatible with an RHOCP cluster.

The SDN uses CNI plug-ins to create Linux namespaces to partition the usage of resources and processes on physical and virtual hosts. With this implementation, containers inside pods can share network resources, such as devices, IP stacks, firewall rules, and routing tables. The SDN allocates a unique routable IP to each pod, so that you can access the pod from any other service in the same network.

In OpenShift 4.14, OVN-Kubernetes is the default network provider.

OVN-Kubernetes uses Open Virtual Network (OVN) to manage the cluster network. A cluster that uses the OVN-Kubernetes plug-in also runs Open vSwitch (OVS) on each node. OVN configures OVS on each node to implement the declared network configuration.

## The OpenShift Cluster Network Operator

RHOCP provides a *Cluster Network Operator* (CNO) that configures OpenShift cluster networking. The CNO is a OpenShift cluster operator that loads and configures Container Network Interface (CNI) plug-ins. As a cluster administrator, execute the following command to observe the status of the CNO:

```
[user@host ~]$ oc get -n openshift-network-operator deployment/network-operator
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
network-operator   1/1      1           1        41d
```

An administrator configures the cluster network operator at installation time. To see the configuration, use the following command:

```
[user@host ~]$ oc describe network.config/cluster
Name: cluster
...output omitted...
Spec:
Cluster Network:
Cidr: 10.8.0.0/14 ①
Host Prefix: 23
External IP:
```

```
Policy:  
Network Type: OVNKubernetes  
Service Network:  
172.30.0.0/16 ②  
...output omitted...
```

- ① The Cluster Network CIDR defines the range of IPs for all pods in the cluster.
- ② The Service Network CIDR defines the range of IPs for all services in the cluster.



## References

For more information, refer to the *About Kubernetes Pods and Services* chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/architecture/index#building-simple-container](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/architecture/index#building-simple-container)

For more information, refer to the *Cluster Network Operator in OpenShift Container Platform* chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/networking/index#cluster-network-operator](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index#cluster-network-operator)

For more information, refer to the *About the OVN-Kubernetes Network Plug-in* chapter in the Red Hat OpenShift Container Platform 4.14 Networking documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/networking/index#about-ovn-kubernetes](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index#about-ovn-kubernetes)

## Cluster Networking

<https://kubernetes.io/docs/concepts/cluster-administration/networking/>

## ► Guided Exercise

# Kubernetes Pod and Service Networks

Deploy a database server and access it through a Kubernetes service.

### Outcomes

You should be able to deploy a database server, and access it indirectly through a Kubernetes service, and also directly pod-to-pod for troubleshooting.

### 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. It also creates the `deploy-services` project and the `/home/student/D0180/labs/deploy-services/resources.txt` file. The `resources.txt` file contains some commands that you use during the exercise. You can use the file to copy and paste these commands.

```
[student@workstation ~]$ lab start deploy-services
```



#### Note

It is safe to ignore pod security warnings for exercises in this course. OpenShift uses the Security Context Constraints controller to provide safe defaults for pod security.

## Instructions

- 1. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Use the `deploy-services` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `deploy-services` project as the active project.

```
[student@workstation ~]$ oc project deploy-services
...output omitted...
```

- 2. Use the `registry.ocp4.example.com:8443/rhel8/mysql-80` container image to create a MySQL deployment named db-pod. Add the missing environment variables for the pod to run.

- 2.1. Create the db-pod deployment.

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

- 2.2. Add the environment variables.

```
[student@workstation ~]$ oc set env deployment/db-pod \
MYSQL_USER=user1 \
MYSQL_PASSWORD=mypa55w0rd \
MYSQL_DATABASE=items
deployment.apps/db-pod updated
```

- 2.3. Confirm that the pod is running.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
db-pod-6ccc485cfcc-vrc4r   1/1     Running   0          2m30s
```

Your pod name might differ from the previous output.

- 3. Expose the db-pod deployment to create a ClusterIP service.

- 3.1. View the deployment for the pod.

```
[student@workstation ~]$ oc get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
db-pod    1/1     1           1           3m36s
```

- 3.2. Expose the db-pod deployment to create a service.

```
[student@workstation ~]$ oc expose deployment/db-pod
service/db-pod exposed
```

- 4. Validate the service. Confirm that the service selector matches the label on the pod. Then, confirm that the db-pod service endpoint matches the IP of the pod.

- 4.1. Identify the selector for the db-pod service. Use the `oc get service` command with the `-o wide` option to retrieve the selector that the service uses.

```
[student@workstation ~]$ oc get service db-pod -o wide
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE      SELECTOR
db-pod    ClusterIP  172.30.108.92  <none>        3306/TCP    108s    app=db-pod
```

The selector shows an `app=db-pod` key:value pair.

- 4.2. Capture the name of the pod in a variable.

```
[student@workstation ~]$ PODNAME=$(oc get pods \
-o jsonpath='{.items[0].metadata.name}' )
```

4.3. Query the label on the pod.

```
[student@workstation ~]$ oc get pod $PODNAME --show-labels
NAME           READY   STATUS    RESTARTS   AGE     LABELS
db-pod-6ccc485cf-vrc4r  1/1     Running   0          6m50s  app=db-pod ...
```

Notice that the label list includes the `app=db-pod` key-value pair, which is the selector for the db-pod service.

4.4. Retrieve the endpoints for the db-pod service.

```
[student@workstation ~]$ oc get endpoints
NAME      ENDPOINTS      AGE
db-pod   10.8.0.85:3306  4m38s
```

Your endpoints values might differ from the previous output.

4.5. Verify that the service endpoint matches the db-pod IP address. Use the `oc get pods` command with the `-o wide` option to view the pod IP address.

```
[student@workstation ~]$ oc get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP          ...
db-pod-6ccc485cf-vrc4r  1/1     Running   0          54m   10.8.0.85 ...
```

The service endpoint resolves to the IP address that is assigned to the pod.

- 5. Delete and then re-create the db-pod deployment. Confirm that the db-pod service endpoint automatically resolves to the IP address of the new pod.

5.1. Delete the db-pod deployment.

```
[student@workstation ~]$ oc delete deployment.apps/db-pod
deployment.apps "db-pod" deleted
```

5.2. Verify that the service still exists without the deployment.

```
[student@workstation ~]$ oc get service
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
db-pod   ClusterIP  172.30.108.92  <none>        3306/TCP    9m53s
```

5.3. The list of endpoints for the service is now empty.

```
[student@workstation ~]$ oc get endpoints
NAME      ENDPOINTS      AGE
db-pod   <none>        12m
```

5.4. Re-create the db-pod deployment.

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

- 5.5. Add the environment variables.

```
[student@workstation ~]$ oc set env deployment/db-pod \
MYSQL_USER=user1 \
MYSQL_PASSWORD=mypa55w0rd \
MYSQL_DATABASE=items
deployment.apps/db-pod updated
```

- 5.6. Confirm that the newly created pod has the app=db-pod selector.

```
[student@workstation ~]$ oc get pods --selector app=db-pod -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP          ...
db-pod-6ccc485cfcc-12x   1/1     Running   0          32s   10.8.0.85 ...
```

Notice the change in the pod name. The pod IP address might also change. Your pod name and IP address might differ from the previous output.

- 5.7. Confirm that the endpoints for the db-pod service include the newly created pod.

```
[student@workstation ~]$ oc get endpoints
NAME      ENDPOINTS      AGE
db-pod    10.8.0.85:3306   16m
```

## ▶ 6. Create a pod to identify the available DNS name assignments for the service.

- 6.1. Create a pod named shell to use for troubleshooting. Use the oc run command and the registry.ocp4.example.com:8443/openshift4/network-tools-rhel8 container image.

```
[student@workstation ~]$ oc run shell -it \
--image registry.ocp4.example.com:8443/openshift4/network-tools-rhel8
If you don't see a command prompt, try pressing enter.
bash-4.4$
```

- 6.2. From the prompt inside the shell pod, view the /etc/resolv.conf file to identify the cluster-domain name.

```
bash-4.4$ cat /etc/resolv.conf
search deploy-services.svc.cluster.local svc.cluster.local ...
nameserver 172.30.0.10
options ndots:5
```

The container uses the values from the search directive as suffix values on DNS searches. The container appends these values to a DNS query, in the written order, to resolve the search. The cluster-domain name is the last few components of these values that start after SVC.

- 6.3. Use the nc and echo commands to test the available DNS names for the service.

```
nc -z <service>_<server>_ <port>
```

The long version of the DNS name is required when accessing the service from a different project. When the pod is in the same project, you can use a shorter version of the DNS name.

```
bash-4.4$ nc -z db-pod.deploy-services 3306 && \
echo "Connection success to db-pod.deploy-services:3306" || \
echo "Connection failed"
Connection success to db-pod.deploy-services:3306
```

6.4. Exit the interactive session.

```
bash-4.4$ exit
Session ended, resume using 'oc attach shell -c shell -i -t' command when the pod
is running
```

6.5. Delete the pod for the shell.

```
[student@workstation ~]$ oc delete pod shell
pod "shell" deleted
```

► 7. Use a new project to test pod communications across namespaces.

7.1. Create a second namespace with the `oc new-project` command.

```
[student@workstation ~]$ oc new-project deploy-services-2
Now using project "deploy-services-2" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

7.2. Execute the `nc` and `echo` commands from a pod to test the DNS name access to another namespace.

```
[student@workstation ~]$ oc run shell -it --rm \
--image registry.ocp4.example.com:8443/openshift4/network-tools-rhel8 \
--restart Never -- nc -z db-pod.deploy-services.svc.cluster.local 3306 && \
echo "Connection success to db-pod.deploy-services.svc.cluster.local:3306" \
|| echo "Connection failed"
pod "shell" deleted
Connection success to db-pod.deploy-services.svc.cluster.local:3306
```

7.3. Return to the `deploy-services` project.

```
[student@workstation ~]$ oc project deploy-services
Now using project "deploy-services" on server "https://api.ocp4.example.com:6443".
```

► 8. Use a Kubernetes job to add initialization data to the database.

- 8.1. Create a job named `mysql-init` that uses the `registry.ocp4.example.com:8443/redhattraining/do180-dbinit:v1` container image. This image uses the `mysql-80` container image as a base image, and it includes a script that adds a few initial records to the database.

```
[student@workstation ~]$ oc create job mysql-init \
--image registry.ocp4.example.com:8443/redhattraining/do180-dbinit:v1 \
-- /bin/bash -c "mysql -uuser1 -pmypa55w0rd --protocol tcp \
-h db-pod -P3306 items </tmp/db-init.sql"
job.batch/mysql-init created
```

The `-h` option of the `mysql` command directs the command to communicate with the DNS short name of the `db-pod` service. The `db-pod` short name can be used here, because the pod for the job is created in the same namespace as the service.

The double dash `--` before `/bin/bash` separates the `oc` command arguments from the command in the pod. The `-c` option of `/bin/bash` directs the command interpreter in the container to execute the command string. The `/tmp/db-init.sql` file is redirected as input for the command. The `db-init.sql` file is included in the image, and contains the following script.

```
DROP TABLE IF EXISTS `Item`;
CREATE TABLE `Item` (`id` BIGINT not null auto_increment primary key,
`description` VARCHAR(100), `done` BIT);
INSERT INTO `Item` (`id`, `description`, `done`) VALUES (1,'Pick up newspaper', 0);
INSERT INTO `Item` (`id`, `description`, `done`) VALUES (2,'Buy groceries', 1);
```

- 8.2. Confirm the status of the `mysql-init` job. Wait for the job to complete.

```
[student@workstation ~]$ oc get job
NAME      COMPLETIONS  DURATION   AGE
mysql-init  1/1          22m
```

- 8.3. Retrieve the status of the `mysql-init` job pod, to confirm that the pod has a `Completed` status.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
db-pod-6ccc485cf8-2lklx  1/1     Running   0          4h24m
mysql-init-1n9cg   0/1     Completed  0          23m
```

- 8.4. Delete the `mysql-init` job, because it is no longer needed.

```
[student@workstation ~]$ oc delete job mysql-init
job.batch "mysql-init" deleted
```

- 8.5. Verify that the corresponding `mysql-init` pod is also deleted.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
db-pod-6ccc485cf8-2lklx  1/1     Running   0          4h2
```

- 9. Create a query-db pod by using the `oc run` command and the `registry.ocp4.example.com:8443/redhattraining/do180-dbinit:v1` container image. Use the pod to execute a query against the database service.
- 9.1. Create the query-db pod. Configure the pod to use the MySQL client to execute a query against the db-pod service. You can use the db-pod service short name, which provides a stable reference.

```
[student@workstation ~]$ oc run query-db -it --rm \
--image registry.ocp4.example.com:8443/redhattraining/do180-dbinit:v1 \
--restart Never \
-- mysql -uuser1 -pmypa55w0rd --protocol tcp \
-h db-pod -P3306 items -e 'select * from Item;' \
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+-----+
| id | description      | done   |
+-----+-----+
| 1  | Pick up newspaper | 0x00   |
| 2  | Buy groceries     | 0x01   |
+-----+-----+
pod "query-db" deleted
```

- 10. It might be necessary to use pod-to-pod communications for troubleshooting. Use the `oc run` command to create a pod that executes a network test against the IP address of the database pod.
- 10.1. Confirm the IP address of the MySQL database pod. Your pod IP address might differ from the output.

```
[student@workstation ~]$ oc get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP          ...
db-pod-6ccc485cf-2lklx  1/1     Running   0          4h5  10.8.0.69 ...
```

- 10.2. Capture the IP address in an environment variable.

```
[student@workstation ~]$ POD_IP=$(oc get pod -l app=db-pod \
-o jsonpath='{.items[0].status.podIP}')
```

- 10.3. Create a test pod named shell with the `oc run` command. Execute the `nc` command to test against the `$POD_IP` environment variable and the 3306 port for the database.

```
[student@workstation ~]$ oc run shell --env POD_IP=$POD_IP -it --rm \
--image registry.ocp4.example.com:8443/openshift4/network-tools-rhel8 \
--restart Never \
-- nc -z $POD_IP 3306 && echo "Connection success to $POD_IP:3306" \
|| echo "Connection failed"
pod "shell" deleted
Connection success to 10.8.0.69:3306
```

## 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 deploy-services
```

# Scale and Expose Applications to External Access

---

## Objectives

- Expose applications to clients outside the cluster by using Kubernetes ingress and OpenShift routes.

## IP Addresses for Pods and Services

Most real-world applications do not run as a single pod. Because applications need to scale horizontally, many pods run the same containers from the same pod resource definition, to meet growing user demand. A service defines a single IP/port combination, and provides a single IP address to a pool of pods, and a load-balancing client request among member pods.

By default, services connect clients to pods in a round-robin fashion, and each service is assigned a unique IP address for clients to connect to. This IP address comes from an internal OpenShift virtual network, which although distinct from the pods' internal network, is visible only to pods. Each pod that matches the selector is added to the service resource as an endpoint.

Containers inside Kubernetes pods must not connect to each other's dynamic IP address directly. Services resolve this problem by linking more stable IP addresses from the SDN to the pods. If pods are restarted, replicated, or rescheduled to different nodes, then services are updated, to provide scalability and fault tolerance.

## Service Types

You can choose between several service types depending on your application needs, cluster infrastructure, and security requirements.

### ClusterIP

This type is the default, unless you explicitly specify a type for a service. The ClusterIP type exposes the service on a cluster-internal IP address. If you choose this value, then the service is reachable only from within the cluster.

The ClusterIP service type is used for pod-to-pod routing within the RHOCP cluster, and enables pods to communicate with and to access each other. IP addresses for the ClusterIP services are assigned from a dedicated service network that is accessible only from inside the cluster. Most applications should use this service type, for which Kubernetes automates the management.

### Load balancer

This resource instructs RHOCP to activate a load balancer in a cloud environment. A load balancer instructs Kubernetes to interact with the cloud provider that the cluster is running in, to provision a load balancer. The load balancer then provides an externally accessible IP address to the application.

Take all necessary precautions before deploying this service type. Load balancers are typically too expensive to assign one for each application in a cluster. Furthermore, applications that use this service type become accessible from networks outside the cluster. Additional security configuration is required to prevent unintended access.

**NodePort**

With this method, Kubernetes exposes a service on a port on the node IP address. The port is exposed on all cluster nodes, and each node redirects traffic to the endpoints (pods) of the service.

A NodePort service requires allowing direct network connections to a cluster node, which is a security risk.

**ExternalName**

This service tells Kubernetes that the DNS name in the `externalName` field is the location of the resource that backs the service. When a DNS request is made against the Kubernetes DNS server, it returns the `externalName` in a *Canonical Name (CNAME)* record, and directs the client to look up the returned name to get the IP address.

## Using Routes for External Connectivity

RHOCP provides resources to expose your applications to external networks outside the cluster. You can expose HTTP and HTTPS traffic, TCP applications, and also non-TCP traffic. However, you should expose only HTTP and TLS-based applications to external access. Applications that use other protocols, such as databases, are usually not exposed to external access (from outside a cluster). Routes and ingress are the main resources for handling ingress traffic.

RHOCP provides the `route` resource to expose your applications to external networks. With routes, you can access your application with a unique hostname that is publicly accessible. Routes rely on a Kubernetes ingress controller to redirect the traffic from the public IP address to pods. By default, Kubernetes provides an ingress controller, starting from the 1.24 release. For RHOCP clusters, the OpenShift ingress operator provides the ingress controller. RHOCP clusters can also use various third-party ingress controllers that can be deployed in parallel with the OpenShift ingress controller.

Routes provide ingress traffic to services in the cluster. Routes were created before Kubernetes ingress objects, and provide more features. Routes provide advanced features that Kubernetes ingress controllers might not support through a standard interface, such as TLS re-encryption, TLS passthrough, and split traffic for blue-green deployments.

To create a route (secure or insecure) with the `oc` CLI, use the `oc expose service service-name` command. Include the `--hostname` option to provide a custom hostname for the route.

```
[user@host ~]$ oc expose service api-frontend \
--hostname api.apps.acme.com
```

If you omit the hostname, then RHOCP automatically generates a hostname with the following structure: <route-name>-<project-name>. <default-domain>. For example, if you create a `frontend` route in an `api` project, in a cluster that uses `apps.example.com` as the wildcard domain, then the route hostname is as follows:

```
frontend-api.apps.example.com
```

 **Important**

The DNS server that hosts the wildcard domain is unaware of any route hostnames; it resolves any name only to the configured IPs. Only the RHOCP router knows about route hostnames, and treats each one as an HTTP virtual host.

Invalid wildcard domain hostnames, or hostnames that do not correspond to any route, are blocked by the RHOCP router and result in an HTTP 503 error.

Consider the following settings when creating a route:

- The name of a service. The route uses the service to determine the pods to direct the traffic to.
- A hostname for the route. A route is always a subdomain of your cluster wildcard domain. For example, if you are using a wildcard domain of `apps.dev-cluster.acme.com`, and need to expose a `frontend` service through a route, then the route name is as follows:

```
frontend.apps.dev-cluster.acme.com.
```

RHOCP can also automatically generate a hostname for the route.

- An optional path, for path-based routes.
- A target port that the application listens to. The target port corresponds to the port that you define in the `targetPort` key of the service.
- An encryption strategy, depending on whether you need a secure or an insecure route.

The following listing shows a minimal definition for a route:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: a-simple-route ①
  labels: ②
    app: API
    name: api-frontend
spec:
  host: api.apps.acme.com ③
  to:
    kind: Service
    name: api-frontend ④
    port: 8080 ⑤
    targetPort: 8443
```

- ➊ The name of the route. This name must be unique.
- ➋ A set of labels that you can use as selectors.
- ➌ The hostname of the route. This hostname must be a subdomain of your wildcard domain, because RHOCP routes the wildcard domain to the routers.
- ➍ The service to redirect the traffic to. Although you use a service name, the route uses this information only to determine the list of pods that receive the traffic.

- ⑤ Port mapping from a router to an endpoint in the service endpoints. The target port on pods that are selected by the service that this route points to.

**Note**

Some ecosystem components have an integration with ingress resources, but not with route resources. For this case, RHOCP automatically creates managed route objects when an ingress object is created. These route objects are deleted when the corresponding ingress objects are deleted.

You can delete a route by using the `oc delete route route-name` command.

```
[user@host ~]$ oc delete route myapp-route
```

You can also expose a service from the web console by clicking the **Networking > Routes** menu. Click **Create Route** and customize the name, the hostname, the path, and the service to route to by using the form view or the YAML manifest.

The screenshot shows the 'Create Route' interface. At the top, it says 'Project: metallb-system'. Below that is the title 'Create Route' with a sub-instruction: 'Routing is a way to make your application publicly visible.' Underneath, there's a 'Configure via:' dropdown with 'Form view' selected. The main form area has three fields: 'Name \*' containing 'my-route', 'Hostname' containing 'www.example.com', and 'Path' which is empty. At the bottom are 'Create' and 'Cancel' buttons.

## Using Ingress Objects for External Connectivity

An ingress is a Kubernetes resource that provides some of the same features as routes (which are an RHOCP resource). Ingress objects accept external requests and transfer the requests based on the route. You can enable only certain types of traffic: HTTP, HTTPS and server name identification (SNI), and TLS with SNI. Standard Kubernetes ingress resources are typically minimal. Many common features that applications rely on, such as TLS termination, path redirecting, and sticky sessions, depend on the ingress controller. Kubernetes does not define the configuration syntax. In RHOCP, routes are generated to meet the conditions that the ingress object specifies.

**Note**

The ingress resource is commonly used for Kubernetes. However, the route resource is the preferred method for external connectivity in RHOCP.

To create an ingress object, use the `oc create ingress ingress-name --rule=URL_route=service-name:port-number` command. Use the `--rule` option to provide a custom rule in the `host/path=service:port[,tls=secretname]` format. If the TLS option is omitted, then an insecure route is created.

```
[user@host ~]$ oc create ingress ingr-sakila \
--rule="ingr-sakila.apps.ocp4.example.com/*=sakila-service:8080"
```

The following listing shows a minimal definition for an ingress object:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend 1
spec:
  rules: 2
  - host: "www.example.com" 3
    http:
      paths:
        - backend: 4
          service:
            name: frontend
            port:
              number: 80
            pathType: Prefix
            path: /
  tls: 5
  - hosts:
    - www.example.com
    secretName: example-com-tls-certificate
```

- 1** The name of the ingress object. This name must be unique.
- 2** The HTTP or HTTPS rule for the ingress object.
- 3** The host for the ingress object. Applies the HTTP rule to the inbound HTTP traffic of the specified host.
- 4** The backend to redirect traffic to. Defines the service name, port number, and port names for the ingress object. To connect to the back end, incoming requests must match the host and path of the rule.
- 5** The configuration of TLS for the ingress object; it is required for secured paths. The host in the TLS object must match the host in the rules object.

You can delete an ingress object by using the `oc delete ingress ingress-name` command.

```
[user@host ~]$ oc delete ingress example-ingress
```

## Sticky Sessions

Sticky sessions enable stateful application traffic by ensuring that all requests reach the same endpoint. RHOCP uses cookies to configure session persistence for ingress and route resources. The ingress controller selects an endpoint to handle any user requests, and creates a cookie for the session. The cookie is passed back in response to the request, and the user sends back the cookie with the next request in the session. The cookie tells the ingress controller which endpoint is handling the session, to ensure that client requests use the cookie so that they are routed to the same pod.

RHOCP auto-generates the cookie name for ingress and route resources. You can overwrite the default cookie name by using the `annotate` command with either the `kubectl` or the `oc` commands. With this annotation, the application that receives route traffic knows the cookie name.

The following example configures a cookie for an ingress object:

```
[user@host ~]$ oc annotate ingress ingr-example \
ingress.kubernetes.io/affinity=cookie
```

The following example configures a cookie named `myapp` for a route object:

```
[user@host ~]$ oc annotate route route-example \
router.openshift.io/cookie_name=myapp
```

After you annotate the route, capture the route hostname in a variable:

```
[user@host ~]$ ROUTE_NAME=$(oc get route <route_name> \
-o jsonpath='{.spec.host}' )
```

Then, use the `curl` command to save the cookie and access the route:

```
[user@host ~]$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

The cookie is passed back in response to the request, and is saved to the `/tmp/cookie_jar` directory. Use the `curl` command and the cookie that was saved from the previous command to connect to the route:

```
[user@host ~]$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

By using the saved cookie, the request is sent to the same pod as the previous request.

## Load Balance and Scale Applications

Developers and administrators can choose to manually scale the number of replica pods in a deployment. More pods might be needed for an anticipated surge in traffic, or the pod count might be reduced to reclaim resources that the cluster can use elsewhere.

You can change the number of replicas in a deployment resource manually by using the `oc scale` command.

```
[user@host ~]$ oc scale --replicas 5 deployment/scale
```

The deployment resource propagates the change to the replica set. The replica set reacts to the change by creating pods (replicas) or by deleting existing ones, depending on whether the new intended replica count is less than or greater than the existing count.

Although you can manipulate a replica set resource directly, the recommended practice is to manipulate the deployment resource instead. A new deployment creates either a replica set or a replication controller, and direct changes to a previous replica set or replication controller are ignored.

## Load Balance Pods

A Kubernetes service serves as an internal load balancer. Standard services act as a load balancer or a proxy, and give access to the workload object by using the service name. A service identifies a set of replicated pods to transfer the connections that it receives.

A router uses the service selector to find the service and the endpoints, or pods, that back the service. When both a router and a service provide load balancing, RHOCP uses the router to load-balance traffic to pods. A router detects relevant changes in the IP addresses of its services, and adapts its configuration accordingly. Custom routers can thereby communicate modifications of API objects to an external routing solution.

RHOCP routers map external hostnames, and load-balance service endpoints over protocols that pass distinguishing information directly to the router. The hostname must exist in the protocol for the router to determine where to send it.



### References

For more information, refer to the *About Networking* section in the Red Hat OpenShift Container Platform 4.14 *Networking* documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/networking/index#about-networking](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/networking/index#about-networking)

## ► Guided Exercise

# Scale and Expose Applications to External Access

Deploy one web server and access it through a Kubernetes ingress; and deploy another web server and access it through an OpenShift route.

### Outcomes

In this exercise, you deploy two web applications to access them through an ingress object and a route, and scale them to verify the load-balance between the pods.

- Deploy two web applications.
- Create a route and an ingress object to access the web applications.
- Enable the sticky sessions for the web applications.
- Scale the web applications to load-balance the service.

### 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 accessible.

```
[student@workstation ~]$ lab start deploy-routes
```

## Instructions

- 1. Create two web application deployments, named `satir-app` and `sakila-app`. Use the `registry.ocp4.example.com:8443/httpd-app:v1` container image for both deployments.
- 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. Change to the `web-applications` project.

```
[student@workstation ~]$ oc project web-applications
Now using project "web-applications" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

13. Create the `satir-app` web application deployment by using the `registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1` container image. Ignore the warning message.

```
[student@workstation ~]$ oc create deployment satir-app \
--image registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1
deployment.apps/satir-app created
```

14. After a few moments, verify that the deployment is successful.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   ...
satir-app-787b7d7858-5dfsh   1/1     Running   0          ...
```

```
[student@workstation ~]$ oc status
...output omitted...
deployment/satir-app deploys registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1
  deployment #1 running for 20 seconds - 1 pod
...output omitted...
```

15. Create the `sakila-app` web application deployment by using the `registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1` image. Ignore the warning message.

```
[student@workstation ~]$ oc create deployment sakila-app \
--image registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1
deployment.apps/sakila-app created
```

16. Wait a few moments and then verify that the deployment is successful.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   ...
sakila-app-6694...5kpd   1/1     Running   0          ...
satir-app-787b7...dfsh   1/1     Running   0          ...
```

```
[student@workstation ~]$ oc status
...output omitted...
deployment/satir-app deploys registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1
  deployment #1 running for 5 minutes - 1 pod

deployment/sakila-app deploys registry.ocp4.example.com:8443/redhattraining/do180-httppd-app:v1
  deployment #1 running for 2 minutes - 1 pod
...output omitted...
```

- ▶ 2. Create services for the web application deployments. Then, use the services to create a route for the `satir-app` application and an ingress object for the `sakila-app` application.

- 2.1. Expose the `satir-app` deployment. Name the service `satir-svc`, and specify port `8080` as the port and target port.

```
[student@workstation ~]$ oc expose deployment satir-app --name satir-svc \
--port 8080 --target-port 8080
service/satir-svc exposed
```

- 2.2. Expose the `sakila-app` deployment to create the `sakila-svc` service.

```
[student@workstation ~]$ oc expose deployment sakila-app --name sakila-svc \
--port 8080 --target-port 8080
service/sakila-svc exposed
```

- 2.3. Verify the status of the services.

```
[student@workstation ~]$ oc get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      ...
sakila-svc   ClusterIP   172.30.230.41   <none>        8080/TCP      ...
satir-svc    ClusterIP   172.30.143.15   <none>        8080/TCP      ...
```

```
[student@workstation ~]$ oc get endpoints
NAME      ENDPOINTS      ...
sakila-svc  10.8.0.66:8080  ...
satir-svc   10.8.0.65:8080  ...
```

```
[student@workstation ~]$ oc get pods -o wide
NAME          READY  STATUS      RESTARTS  AGE      IP           ...
sakila-app-6694...5kpd  1/1    Running    0          92s    10.8.0.66 ...
satir-app-787b7...dfsh  1/1    Running    0          2m49s  10.8.0.65 ...
```

- 2.4. Create a route named `satir` for the `satir-app` web application by exposing the `satir-svc` service.

```
[student@workstation ~]$ oc expose service satir-svc --name satir
route.route.openshift.io/satir exposed
```

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT      ... SERVICES      PORT      ...
satir     satir-web-applications.apps.ocp4.example.com ... satir-svc  8080      ...
```

- 2.5. Create an ingress object named `ingr-sakila` for the `sakila-svc` service. Configure the `--rule` option with the following values:

Field	Value
Host	<code>ingr-sakila.apps.ocp4.example.com</code>
Service name	<code>sakila-svc</code>
Port number	<code>8080</code>

```
[student@workstation ~]$ oc create ingress ingr-sakila \
--rule "ingr-sakila.apps.ocp4.example.com/*=sakila-svc:8080"
ingress.networking.k8s.io/ingr-sakila created
```

```
[student@workstation ~]$ oc get ingress
NAME      ... HOSTS          ADDRESS        PORTS ...
ingr-sakila ... ingr-sakila.apps.ocp4.example.com  router...com  80    ...
```

2.6. Confirm that a route exists for the `ingr-sakila` ingress object.

```
[student@workstation ~]$ oc get routes
NAME           HOST/PORT          ... SERVICES   PORT
ingr-sakila... ingr-sakila.apps.ocp4.example.com ... sakila-svc <all>
satir         satir-web-applications.apps.ocp4.example.com ... satir-svc  8080
```

A specific port is not assigned to routes that ingress objects created. By contrast, a route that an exposed service created is assigned the same ports as the service.

2.7. Use the `curl` command to access the `ingr-sakila` ingress object and the `satir` route. The output states the name of the pod that is servicing the request.

```
[student@workstation ~]$ curl ingr-sakila.apps.ocp4.example.com
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
```

```
[student@workstation ~]$ curl satir-web-applications.apps.ocp4.example.com
Welcome to Red Hat Training, from satir-app-787b7d7858-bdfsh
```

- 3. Scale the web application deployments to load-balance their services. Scale the `sakila-app` to two replicas, and the `satir-app` to three replicas.

3.1. Scale the `sakila-app` deployment with two replicas.

```
[student@workstation ~]$ oc scale deployment sakila-app --replicas 2
deployment.apps/sakila-app scaled
```

3.2. Wait a few moments and then verify the status of the replica pods.

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    RESTARTS   ...
sakila-app-6694...5kpd  1/1     Running   0          ...
sakila-app-6694...rfzg  1/1     Running   0          ...
satir-app-787b...dfsh  1/1     Running   0          ...
```

3.3. Scale the `satir-app` deployment with three replicas.

```
[student@workstation ~]$ oc scale deployment satir-app --replicas 3
deployment.apps/satir-app scaled
```

3.4. Wait a few moments and then verify the status of the replica pods.

```
[student@workstation ~]$ oc get pods -o wide
NAME          READY   STATUS    RESTARTS   ... IP      ...
sakila-app-6694...5kpd  1/1     Running   0          ... 10.8.0.66 ...
sakila-app-6694...rfgzg  1/1     Running   0          ... 10.8.0.67 ...
satir-app-787b...dfsh   1/1     Running   0          ... 10.8.0.65 ...
satir-app-787b...z8xm   1/1     Running   0          ... 10.8.0.69 ...
satir-app-787b...7bhj   1/1     Running   0          ... 10.8.0.70 ...
```

- 3.5. Retrieve the service endpoints to confirm that the services are load-balanced between the additional replica pods.

```
[student@workstation ~]$ oc get endpoints
NAME      ENDPOINTS      ...
sakila-svc  10.8.0.66:8080,10.8.0.67:8080  ...
satir-svc   10.8.0.65:8080,10.8.0.69:8080,10.8.0.70:8080  ...
```

- ▶ 4. Enable the sticky sessions for the `sakila-app` web application. Then, use the `curl` command to confirm that the sticky sessions are working for the `ingr-sakila` object.

- 4.1. Configure a cookie for the `ingr-sakila` ingress object.

```
[student@workstation ~]$ oc annotate ingress ingr-sakila \
  ingress.kubernetes.io/affinity=cookie
ingress.networking.k8s.io/ingr-sakila annotated
```

- 4.2. Use the `curl` command to access the `ingr-sakila` ingress object. The output states the name of the pod that is servicing the request. Notice that the connection is load-balanced between the replicas.

```
[student@workstation ~]$ for i in {1..3}; do \
  curl ingr-sakila.apps.ocp4.example.com ; done
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
```

- 4.3. Use the `curl` command to save the `ingr-sakila` ingress object cookie to the `/tmp/cookie_jar` file. Confirm that the cookie exists in the `/tmp/cookie_jar` file.

```
[student@workstation ~]$ curl ingr-sakila.apps.ocp4.example.com \
  -c /tmp/cookie_jar
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
```

```
[student@workstation ~]$ cat /tmp/cookie_jar
...output omitted...
#HttpOnly_ingr-sakila.apps.ocp4.example.com FALSE / FALSE 0
b9b484110526b4b1b3159860d3aebe04 921e139c5145950d00424bf3b0a46d22
```

- 4.4. The cookie provides session stickiness for connections to the `ingr-sakila` route. Use the `curl` command and the cookie in the `/tmp/cookie_jar` file to connect to

the `ingr-sakila` route again. Confirm that you are connected to the same pod that handled the request in the previous step.

```
[student@workstation ~]$ for i in {1..3}; do \
  curl ingr-sakila.apps.ocp4.example.com -b /tmp/cookie_jar; done
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
```

- 4.5. Use the `curl` command to connect to the `ingr-sakila` route without the cookie. Observe that session stickiness occurs only with the cookie.

```
[student@workstation ~]$ for i in {1..3}; do \
  curl ingr-sakila.apps.ocp4.example.com ; done
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
```

- ▶ 5. Enable the sticky sessions for the `satir-app` web application. Then, use the `curl` command to confirm that sticky sessions are active for the `satir` route.

- 5.1. Configure a cookie with a `hello` value for the `satir` route.

```
[student@workstation ~]$ oc annotate route satir \
  router.openshift.io/cookie_name="hello"
route.route.openshift.io/satir annotated
```

- 5.2. Use the `curl` command to access the `satir` route. The output states the name of the pod that is servicing the request. Notice that the connection is load-balanced between the three replica pods.

```
[student@workstation ~]$ for i in {1..3}; do \
  curl satir-web-applications.apps.ocp4.example.com; done
Welcome to Red Hat Training, from satir-app-787b7d7858-bdfsh
Welcome to Red Hat Training, from satir-app-787b7d7858-gz8xm
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
```

- 5.3. Use the `curl` command to save the `hello` cookie to the `/tmp/cookie_jar` file. Afterward, confirm that the `hello` cookie exists in the `/tmp/cookie_jar` file.

```
[student@workstation ~]$ curl satir-web-applications.apps.ocp4.example.com \
  -c /tmp/cookie_jar
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
```

```
[student@workstation ~]$ cat /tmp/cookie_jar
...output omitted...
#HttpOnly_satir-web-applications.apps.ocp4.example.com FALSE / FALSE 0 hello
b7dd73d32003e513a072e25a32b6c881
```

- 5.4. The `hello` cookie provides session stickiness for connections to the `satir` route. Use the `curl` command and the `hello` cookie in the `/tmp/cookie_jar` file to

connect to the `satir` route again. Confirm that you are connected to the same pod that handled the request in the previous step.

```
[student@workstation ~]$ for i in {1..3}; do \
    curl satir-web-applications.apps.ocp4.example.com -b /tmp/cookie_jar; done
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
```

- 5.5. Use the `curl` command to connect to the `satir` route without the `hello` cookie. Observe that session stickiness occurs only with the cookie.

```
[student@workstation ~]$ for i in {1..3}; do \
    curl satir-web-applications.apps.ocp4.example.com; done
Welcome to Red Hat Training, from satir-app-787b7d7858-gz8xm
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
Welcome to Red Hat Training, from satir-app-787b7d7858-bdfsh
```

## 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 deploy-routes
```

## ► Lab

# Deploy Managed and Networked Applications on Kubernetes

Deploy a database server and a web application that connects to that database and expose the web application to external access.

### Outcomes

- Deploy a MySQL database from a container image.
- Deploy a web application from a container image.
- Configure environment variables for a deployment.
- Expose the web application for external access.

### 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 accessible and that all exercise resources are available. It also creates the `database-applications` project.

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

## Instructions

The API URL of your OpenShift cluster is <https://api.ocp4.example.com:6443>, and the `oc` command is already installed on your `workstation` machine.

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

Use the `database-applications` project for your work.

1. Log in to the OpenShift cluster and change to the `database-applications` project.
2. Create a MySQL database deployment named `mysql-app` by using the `registry.ocp4.example.com:8443/redhattraining/mysql-app:v1` image, and identify the root cause of the failure.
3. Configure the environment variables for the `mysql-app` deployment by using the following information:

Field	Value
MYSQL_USER	redhat
MYSQL_PASSWORD	redhat123
MYSQL_DATABASE	world_x

Then, execute the following command in the `mysql-app` deployment pod to load the `world_x` database:

```
/bin/bash -c "mysql -uroot -p12345 </tmp/world_x.sql"
```

4. Create a service for the `mysql-app` deployment by using the following information:

Field	Value
Name	<code>mysql-service</code>
Port	3306
Target port	3306

5. Create a web application deployment named `php-app` by using the `registry.ocp4.example.com:8443/redhat-training/php-webapp:v1` image.
6. Create a service for the `php-app` deployment by using the following information:

Field	Value
Name	<code>php-svc</code>
Port	8080
Target port	8080

Then, create a route named `phpapp` to expose the web application to external access.

7. Test the connectivity between the web application and the MySQL database. In a web browser, navigate to the `phpapp-database-applications.apps.ocp4.example.com` route, and verify that the application retrieves data from the MySQL database.

## 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 deploy-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 deploy-review
```

## ► Solution

# Deploy Managed and Networked Applications on Kubernetes

Deploy a database server and a web application that connects to that database and expose the web application to external access.

### Outcomes

- Deploy a MySQL database from a container image.
- Deploy a web application from a container image.
- Configure environment variables for a deployment.
- Expose the web application for external access.

### 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 accessible and that all exercise resources are available. It also creates the `database-applications` project.

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

### Instructions

The API URL of your OpenShift cluster is <https://api.ocp4.example.com:6443>, and the `oc` command is already installed on your `workstation` machine.

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

Use the `database-applications` project for your work.

1. Log in to the OpenShift cluster and change to the `database-applications` project.

1.1. Log in to the OpenShift cluster.

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

1.2. Change to the `database-applications` project.

```
[student@workstation ~]$ oc project database-applications
Now using project "database-applications" on server "https://
api.ocp4.example.com:6443".
...output omitted...
```

2. Create a MySQL database deployment named mysql-app by using the `registry.ocp4.example.com:8443/redhattraining/mysql-app:v1` image, and identify the root cause of the failure.

- 2.1. Create the MySQL database deployment. Ignore the warning message.

```
[student@workstation ~]$ oc create deployment mysql-app \
--image registry.ocp4.example.com:8443/redhattraining/mysql-app:v1
deployment.apps/mysql-app created
```

- 2.2. Verify the deployment status. The pod name might differ in your output.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    ...
mysql-app-75dfd58f99-5xfqc   0/1     Error   ...
[student@workstation ~]$ oc status
...output omitted...
Errors:
pod/mysql-app-75dfd58f99-5xfqc is crash-looping

1 error, 1 info identified, use 'oc status --suggest' to see details.
```

- 2.3. Identify the root cause of the deployment failure.

```
[student@workstation ~]$ oc logs mysql-app-75dfd58f99-5xfqc
...output omitted...
You must either specify the following environment variables:
MYSQL_USER
MYSQL_PASSWORD
MYSQL_DATABASE
Or the following environment variable:
MYSQL_ROOT_PASSWORD (regex: '^[_a-zA-Z0-9~!@#$%^&*()-=<>, .?;:|]+$')
...output omitted...
```

3. Configure the environment variables for the mysql-app deployment by using the following information:

Field	Value
MYSQL_USER	redhat
MYSQL_PASSWORD	redhat123
MYSQL_DATABASE	world_x

Then, execute the following command in the mysql-app deployment pod to load the world\_x database:

```
/bin/bash -c "mysql -uredhat -predhat123 </tmp/world_x.sql"
```

- 3.1. Update the environment variables for the mysql-app deployment.

```
[student@workstation ~]$ oc set env deployment/mysql-app \
  MYSQL_USER=redhat MYSQL_PASSWORD=redhat123 MYSQL_DATABASE=world_x
deployment.apps/mysql-app updated
```

- 3.2. Verify that the mysql-app application pod is in the RUNNING state. The pod name might differ in your output.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    ...
mysql-app-57c44f646-5qt2k  1/1     Running  ...
```

- 3.3. Load the world\_x database.

```
[student@workstation ~]$ oc exec -it mysql-app-57c44f646-5qt2k \
  -- /bin/bash -c "mysql -uredhat -predhat123 </tmp/world_x.sql"
...output omitted..
[student@workstation ~]$
```

- 3.4. Confirm that you can access the MySQL database.

```
[student@workstation ~]$ oc rsh mysql-app-57c44f646-5qt2k
sh-4.4$ mysql -uredhat -predhat123 world_x
...output omitted...
mysql>
```

- 3.5. Exit the MySQL database, and then exit the container.

```
mysql> exit
Bye
sh-4.4$ exit
```

- 4.** Create a service for the mysql-app deployment by using the following information:

Field	Value
Name	mysql-service
Port	3306
Target port	3306

- 4.1. Expose the mysql-app deployment.

```
[student@workstation ~]$ oc expose deployment mysql-app --name mysql-service \
--port 3306 --target-port 3306
service/mysql-service created
```

4.2. Verify the service configuration. The endpoint IP address might differ in your output.

```
[student@workstation ~]$ oc get services
NAME          TYPE      CLUSTER-IP      EXTERNAL-IP     PORT(S)      AGE
mysql-service ClusterIP  172.30.146.213  <none>        3306/TCP    10s
[student@workstation ~]$ oc get endpoints
NAME          ENDPOINTS      AGE
mysql-service  10.8.0.102:3306  19s
```

5. Create a web application deployment named php-app by using the `registry.ocp4.example.com:8443/redhattraining/php-webapp:v1` image.

5.1. Create the web application deployment. Ignore the warning message.

```
[student@workstation ~]$ oc create deployment php-app \
--image registry.ocp4.example.com:8443/redhattraining/php-webapp:v1
deployment.apps/php-app created
```

5.2. Verify the deployment status. Verify that the `php-app` application pod is in the `RUNNING` state.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    ...
php-app-725... 1/1     Running   ...
mysql-app-57c... 1/1     Running   ...
```

```
[student@workstation ~]$ oc status
...output omitted...
deployment/php-app deploys registry.ocp4.example.com:8443/redhattraining/php-
webapp:v1
  deployment #1 running for about a minute - 1 pod
...output omitted...
```

6. Create a service for the `php-app` deployment by using the following information:

Field	Value
Name	php-svc
Port	8080
Target port	8080

Then, create a route named `phpapp` to expose the web application to external access.

6.1. Expose the `php-app` deployment.

```
[student@workstation ~]$ oc expose deployment php-app --name php-svc \
--port 8080 --target-port 8080
service/php-svc exposed
```

6.2. Verify the service configuration. The endpoint IP address might differ in your output.

```
[student@workstation ~]$ oc get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)      AGE
mysql-service ClusterIP  172.30.146.213  <none>        3306/TCP    7m47s
php-svc       ClusterIP  172.30.228.80   <none>        8080/TCP    4m34s
[student@workstation ~]$ oc get endpoints
NAME          ENDPOINTS      AGE
mysql-service 10.8.0.102:3306  7m50s
php-svc       10.8.0.107:8080  4m37s
```

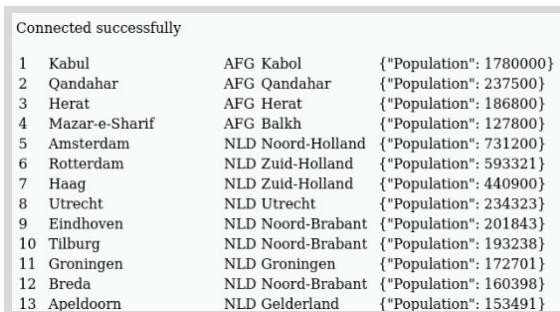
6.3. Expose the php-svc service.

```
[student@workstation ~]$ oc expose service/php-svc --name phpapp
route.route.openshift.io/phpapp exposed
```

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT      ...
phpapp   phpapp-database-applications.apps.ocp4.example.com ...
```

7. Test the connectivity between the web application and the MySQL database. In a web browser, navigate to the `phpapp-database-applications.apps.ocp4.example.com` route, and verify that the application retrieves data from the MySQL database.

7.1. Navigate to the `phpapp-database-applications.apps.ocp4.example.com` route in the web browser.



## 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 deploy-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 deploy-review
```

# Summary

---

- Many resources in Kubernetes and RHOCUP create or affect pods.
- Resources are created imperatively or declaratively. The imperative strategy instructs the cluster what to do. The declarative strategy defines the state that the cluster matches.
- The `oc new-app` command creates resources that are determined via heuristics.
- The main way to deploy an application is by creating a deployment.
- The workload API includes several resources to create pods. The choice between resources depends on for how long and how often the pod needs to run.
- A *job* resource executes a one-time task on the cluster via a pod. The cluster retries the job until it succeeds, or it retries a specified number of attempts.
- Resources are organized into projects and are selected via labels.
- A *route* connects a public-facing IP address and a DNS hostname to an internal-facing service IP address. Services provide network access between pods, whereas routes provide network access to pods from users and applications outside the RHOCUP cluster.

## Chapter 5

# Manage Storage for Application Configuration and Data

### Goal

Externalize application configurations in Kubernetes resources and provision storage volumes for persistent data files.

### Objectives

- Configure applications by using Kubernetes secrets and configuration maps to initialize environment variables and to provide text and binary configuration files.
- Provide applications with persistent storage volumes for block and file-based data.
- Match applications with storage classes that provide storage services to satisfy application requirements.
- Deploy applications that scale without sharing storage.

### Sections

- Externalize the Configuration of Applications (and Guided Exercise)
- Provision Persistent Data Volumes (and Guided Exercise)
- Select a Storage Class for an Application (and Guided Exercise)
- Manage Non-shared Storage with Stateful Sets (and Guided Exercise)

### Lab

- Manage Storage for Application Configuration and Data

# Externalize the Configuration of Applications

---

## Objectives

- Configure applications by using Kubernetes secrets and configuration maps to initialize environment variables and to provide text and binary configuration files.

## Configuring Kubernetes Applications

When an application is run in Kubernetes with a pre-existing image, the application uses the default configuration. This action is valid for testing purposes. However, for production environments, you might need to customize your applications before deploying them.

With Kubernetes, you can use manifests in JSON and YAML formats to specify the intended configuration for each application. You can define the name of the application, labels, the image source, storage, environment variables, and more.

The following snippet shows an example of a YAML manifest file of a deployment:

```
apiVersion: apps/v1 ①
kind: Deployment ②
metadata: ③
  name: hello-deployment
spec: ④
  replicas: 1
  selector:
    matchLabels:
      app: hello-deployment
  template:
    metadata:
      labels:
        app: hello-deployment
    spec: ⑤
      containers:
        - env: ⑥
          - name: ENV_VARIABLE_1
            valueFrom:
              secretKeyRef:
                key: hello
                name: world
      image: quay.io/hello-image:latest
```

- API version of the resource.
- Deployment resource type.
- In this section, you specify the metadata of your application, such as the name.
- You can define the general configuration of the resource that is applied to the deployment, such as the number of replicas (pods), the selector label, and the template data.

- ⑤ In this section, you specify the configuration for your application, such as the image name, the container name, ports, environment variables, and more.
- ⑥ You can define the environment variables to configure your application needs.

Sometimes your application requires configuring a combination of files. For example, at the time of creation, a database deployment must have preloaded databases and data. You most commonly configure applications by using environment variables, external files, or command-line arguments. This process of configuration externalization ensures that the application is portable across environments when the container image, external files, and environment variables are available in the environment where the application runs.

Kubernetes provides a mechanism to externalize the configuration of your applications by using configuration maps and secrets.

You can use configuration maps to inject containers with configuration data. The *ConfigMap* (configuration map) namespaced objects provide ways to inject configuration data into containers, which helps to maintain platform independence of the containers. These objects can store fine-grained information, such as individual properties, or coarse-grained information, such as entire configuration files or JSON blobs (JSON sections). The information in configuration maps does not require protection.

The following listing shows an example of a configuration map:

```
apiVersion: v1
kind: ConfigMap ①
metadata:
  name: example-configmap
  namespace: my-app
data: ②
  example.property.1: hello
  example.property.2: world
  example.property.file: |-  
    property.1=value-1  
    property.2=value-2  
    property.3=value-3
binaryData: ③
bar: L3Jvb3QvMTAw
```

- ① ConfigMap resource type.
- ② Contains the configuration data.
- ③ Points to an encoded file in base64 that contains non-UTF-8 data, for example, a binary Java keystore file. Place a key followed by the encoded file.

Applications often require access to sensitive information. For example, a back-end web application requires access to database credentials to query a database. Kubernetes and OpenShift use secrets to hold sensitive information. For example, you can use secrets to store the following types of sensitive information:

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

The following listing shows an example of a secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: example-secret
  namespace: my-app
type: Opaque ①
data: ②
  username: bXl1c2VyCg==
  password: bXlQQDU1Cg==
stringData: ③
  hostname: myapp.mydomain.com
  secret.properties: |
    property1=valueA
    property2=valueB
```

- ①** Specifies the type of secret.
- ②** Specifies the encoded string and data.
- ③** Specifies the decoded string and data.

A secret is a namespaced object and it can store any type of data. Data in a secret is Base64-encoded, and is not stored in plain text. Secret data is not encrypted; you can decode the secret from Base64 format to access the original data. The following example shows the decoded values for the `username` and `password` objects from the `example-secret` secret:

```
[user@host] echo bXl1c2VyCg== | base64 --decode
myuser
[user@host] echo bXlQQDU1Cg== | base64 --decode
myP@55
```

Kubernetes and OpenShift support the following types of secrets:

- Opaque secrets: An opaque secret store key and value pairs that contain arbitrary values, and are not validated to conform to any convention for key names or values.
- Service account tokens: Store a token credential for applications that authenticate to the Kubernetes API.
- Basic authentication secrets: Store the needed credentials for basic authentication. The `data` parameter of the secret object must contain the `user` and the `password` keys that are encoded in the Base64 format.
- SSH keys: Store data that is used for SSH authentication.
- TLS certificates: Store a certificate and a key that are used for TLS.
- Docker configuration secrets: Store the credentials for accessing a container image registry.

When you store information in a specific secret resource type, Kubernetes validates that the data conforms to the type of secret.

**Note**

By default, configuration maps and secrets are not encrypted. To encrypt your secret data at rest, you must encrypt the Etcd database. When enabled, Etcd encrypts the following resources: secrets, configuration maps, routes, OAuth access tokens, and OAuth authorization tokens. Encrypting the Etcd database is outside the scope of the course.

For more information, refer to the *Encrypting Etcd Data* chapter in the Red Hat OpenShift Container Platform 4.14 Security and Compliance documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/security\\_and\\_compliance/index#encrypting-etcd](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/security_and_compliance/index#encrypting-etcd)

## Creating Secrets

If a pod requires access to sensitive information, then create a secret for the information before you deploy the pod. Both the `oc` and `kubectl` command-line tools provide the `create secret` command. Use one of the following commands to create a secret:

- Create a generic secret that contains key-value pairs from literal values that are typed on the command line:

```
[user@host ~]$ oc create secret generic secret_name \
--from-literal key1=secret1 \
--from-literal key2=secret2
```

- Create a generic secret by using key names that are specified on the command line and values from files:

```
[user@host ~]$ kubectl create secret generic ssh-keys \
--from-file id_rsa=/path-to/id_rsa \
--from-file id_rsa.pub=/path-to/id_rsa.pub
```

- Create a TLS secret that specifies a certificate and the associated key:

```
[user@host ~]$ oc create secret tls secret-tls \
--cert /path-to-certificate --key /path-to-key
```

To create an opaque secret from the web console, click the **Workloads > Secrets** menu. Click **Create** and select **Key/value secret**. Complete the form with the key name, and specify the value by writing it in the following section, or by extracting it from a file.

**Create key/value secret**

Key/value secrets let you inject sensitive data into your application as files or environment variables.

**Secret name \***  
database-scret

Unique name of the new secret.

**Key \***  
user

**Value**

Drag and drop file with your value here or browse to upload it.  
developer

**Browse...**

To create a secret from the web console that stores the credentials for accessing a container image registry, click the **Workloads > Secrets** menu. Click **Create** and select **Image pull secret**. Complete the form or upload a configuration file with the secret name, select the authentication type, and add the registry server address, the username, password, and email credentials.

**Create image pull secret**

Image pull secrets let you authenticate against a private image registry.

**Secret name \***  
database-scret

Unique name of the new secret.

**Authentication type**  
Image registry credentials

**Registry server address \***  
quay.io

For example quay.io or docker.io

**Username \***  
developer

**Password \***  
\*\*\*\*\*

**Email**

## Creating Configuration Maps

The syntax for creating a configuration map and for creating a secret closely match. You can enter key-value pairs on the command line, or use the content of a file as the value of a specified key. You can use either the `oc` or `kubectl` command-line tools to create a configuration map. The following command shows how to create a configuration map:

```
[user@host ~]$ kubectl create configmap my-config \
--from-literal key1=config1 --from-literal key2=config2
```

You can also use the `cm` shortname to create a configuration map.

```
[user@host ~]$ oc create cm my-config \
--from-literal key1=config1 --from-literal key2=config2
```

To create a configuration map from the web console, click the **Workloads > ConfigMaps** menu. Click **Create ConfigMap** and complete the configuration map by using the form view or the YAML view.

The screenshot shows the configuration for a ConfigMap named 'database'. It includes a description of what a ConfigMap is, an 'Immutable' checkbox, and a 'Data' section. In the 'Data' section, there is a key 'database' with a value containing the file 'countries'. A note at the bottom says 'You can use files on each key that you add by clicking Browse beside the Value field. The Key field must be the name of the added file in the Value field.'

You can use files on each key that you add by clicking **Browse** beside the **Value** field. The **Key** field must be the name of the added file in the **Value** field.

The screenshot shows the configuration for a ConfigMap named 'index.html'. It includes a description of what a ConfigMap is, an 'Immutable' checkbox, and a 'Data' section. In the 'Data' section, there is a key 'index.html' with a value containing the file content '<a href="redhatlogo.png">click me!</a>'. A note at the bottom says 'You can use files on each key that you add by clicking Browse beside the Value field. The Key field must be the name of the added file in the Value field.'



### Note

Use a binary data key instead of a data key if the file uses the binary format, such as a PNG file.

## Using Configuration Maps and Secrets to Initialize Environment Variables

You can use configuration maps to populate individual environment variables that configure your application. Unlike secrets, the information in configuration maps does not require protection. The following listing shows an initialization example of environment variables:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-map-example
  namespace: example-app ①
data:
  database.name: sakila ②
  database.user: redhat ③
```

- ➊ The project where the configuration map resides. ConfigMap objects can be referenced only by pods in the same project.
- ➋ Initializes the database.name variable to the sakila value.
- ➌ Initializes the database.user variable to the redhat value.

You can then use the configuration map to populate environment variables for your application. The following example shows a pod resource that populates specific environment variables by using a configuration map.

```
apiVersion: v1
kind: Pod
metadata:
  name: config-map-example-pod
  namespace: example-app
spec:
  containers:
    - name: example-container
      image: registry.example.com/mysql-80:1-237
      command: [ "/bin/sh", "-c", "env" ]
      env: ➊
        - name: MYSQL_DATABASE ➋
          valueFrom:
            configMapKeyRef:
              name: config-map-example ➌
              key: database.name ➍
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: config-map-example ➎
              key: database.user ➏
              optional: true ➐
```

- ➊ The attribute to specify environment variables for the pod.
- ➋ The name of a pod environment variable where you are populating a key's value.
- ➌➎ Name of the ConfigMap object to pull the environment variables from.
- ➍➏ The environment variable to pull from the ConfigMap object.
- ➐ Sets the environment variable as optional. The pod is started even if the specified ConfigMap object and keys do not exist.

The following example shows a pod resource that injects all environment variables from a configuration map:

```
apiVersion: v1
kind: Pod
metadata:
  name: config-map-example-pod2
  namespace: example-app
spec:
  containers:
```

```

- name: example-container
  image: registry.example.com/mysql-80:1-237
  command: [ "/bin/sh", "-c", "env" ]
  envFrom: ①
    - configMapRef:
        name: config-map-example ②
  restartPolicy: Never

```

- ① The attribute to pull all environment variables from a **ConfigMap** object.
- ② The name of the **ConfigMap** object to pull environment variables from.

You can use secrets with other Kubernetes resources such as pods, deployments, builds, and more. You can specify secret keys or volumes with a mount path to store your secrets. The following snippet shows an example of a pod that populates environment variables with data from the `test-secret` Kubernetes secret:

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env: ①
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom: ②
            secretKeyRef: ③
              name: test-secret ④
              key: username ⑤

```

- ① Specifies the environment variables for the pod.
- ② Indicates the source of the environment variables.
- ③ The `secretKeyRef` source object of the environment variables.
- ④ Name of the secret, which must exist.
- ⑤ The key that is extracted from the secret is the username for authentication.

In contrast with configuration maps, the values in secrets are always encoded (not encrypted), and their access is restricted to fewer authorized users.

## Using Secrets and Configuration Maps as Volumes

To expose a secret to a pod, you must first create the secret in the same namespace, or project, as the pod. In the secret, assign each piece of sensitive data to a key. After creation, the secret contains key-value pairs.

The following command creates a generic secret that contains key-value pairs from literal values that are typed on the command line: `user` with the `demo-user` value, and `root_password` with the `zT1kTgk` value.

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

You can also create a generic secret by specifying key names on the command line and values from files:

```
[user@host ~]$ oc create secret generic demo-secret \
--from-file user=/tmp/demo/user \
--from-file root_password=/tmp/demo/root_password
```

You can mount a secret to a directory within a pod. Kubernetes creates a file for each key in the secret that uses the name of the key. The content of each file is the decoded value of the secret. The following command shows how to mount secrets in a pod:

```
[user@host ~]$ oc set volume deployment/demo \ ①
--add --type secret \ ②
--secret-name demo-secret \ ③
--mount-path /app-secrets ④
```

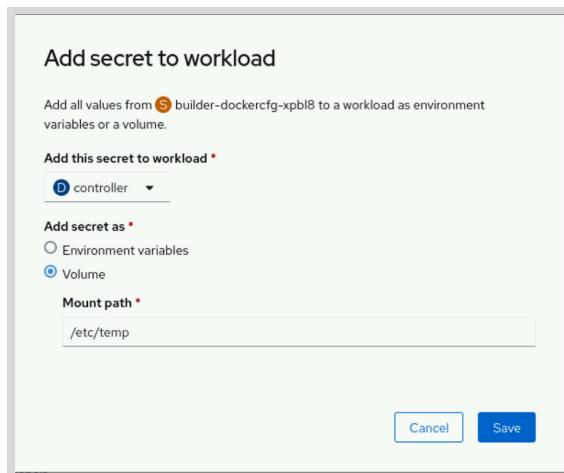
- ① Modify the volume configuration in the demo deployment.
- ② Add a new volume from a secret.
- ③ Use the demo-secret secret.
- ④ Make the secret data available in the /app-secrets directory in the pod. The content of the /app-secrets/user file is demo-user. The content of the /app/secrets/root\_password file is zT1KTgk.

To assign a secret as a volume to a deployment from the web console, list the available secrets from the **Workloads > Secrets** menu.

Name	Type	Size	Created
builder-dockercfg-xpbI8	kubernetes.io/dockercfg	1	Oct 16, 2023, 4:27 AM
builder-token-cwqj4	kubernetes.io/service-account-token	4	Oct 16, 2023, 4:27 AM
controller-certs-secret	kubernetes.io/tls	2	Oct 16, 2023, 4:28 AM
controller-dockercfg-zn946	kubernetes.io/dockercfg	1	Oct 16, 2023, 4:27 AM
controller-token-729in	kubernetes.io/service-account-token	4	Oct 16, 2023, 4:27 AM
default-dockercfg-fzq8c	kubernetes.io/dockercfg	1	Oct 16, 2023, 4:27 AM
default-token-w2684	kubernetes.io/service-account-token	4	Oct 16, 2023, 4:27 AM
deployer-dockercfg-4bq4z	kubernetes.io/dockercfg	1	Oct 16, 2023, 4:27 AM
deployer-token-ppbsh	kubernetes.io/service-account-token	4	Oct 16, 2023, 4:27 AM

Select a secret and click **Add Secret to workload**.

Select the workload, choose the **Volume** option, and define the mount path for the secret.



Similar to secrets, you must first create a configuration map before a pod can consume it. The configuration map must exist in the same namespace, or project, as the pod. The following command shows how to create a configuration map from an external configuration file:

```
[user@host ~]$ oc create configmap demo-map \
--from-file=config-files/httpd.conf
```

You can similarly add a configuration map as a volume by using the following command:

```
[user@host ~]$ oc set volume deployment/demo \
--add --type configmap \
--configmap-name demo-map \
--mount-path /app-secrets
```

To confirm that the volume is attached to the deployment, use the following command:

```
[user@host ~]$ oc get volume deployment/demo
demo
configMap/demo-map as volume-du9in
mounted at /app-secrets
```

You can also use the `oc set env` command to set application environment variables from either secrets or configuration maps. In some cases, you can modify the names of the keys to match the names of environment variables by using the `--prefix` option. In the following example, the `user` key from the `demo-secret` secret sets the `MYSQL_USER` environment variable, and the `root_password` key from the `demo-secret` secret sets the `MYSQL_ROOT_PASSWORD` environment variable. If the key name from the secret is lowercase, then the corresponding environment variable is converted to uppercase to match the pattern that the `--prefix` option defines.

```
[user@host ~]$ oc set env deployment/demo \
--from secret/demo-secret --prefix MYSQL_
```

**Note**

You cannot assign configuration maps by using the web console.

## Updating Secrets and Configuration Maps

Secrets and configuration maps occasionally require updates. OpenShift provides the `oc extract` command to ensure that you have the latest data. You can save the data to a specific directory by using the `--to` option. Each key in the secret or configuration map creates a file with the same name as the key. The content of each file is the value of the associated key. If you run the `oc extract` command more than one time, then you must use the `--confirm` option to overwrite the existing files. You can also use the `--confirm` option to create the target directory for the extracted content.

```
[user@host ~]$ oc extract secret/demo-secrets -n demo \
--to /tmp/demo --confirm
[user@host ~]$ ls /tmp/demo/
user root_password
[user@host ~]$ cat /tmp/demo/root_password
zT1KTgk
[user@host ~]$ echo k8qhcw3m0 > /tmp/demo/root_password
```

After updating the locally saved files, use the `oc set data` command to update the secret or configuration map. For each key that requires an update, specify the name of a key and the associated value. If a file contains the value, then use the `--from-file` option.

```
[user@host ~]$ oc set data secret/demo-secrets -n demo \
--from-file /tmp/demo/root_password
```

You must restart pods that use environment variables for the pods to read the updated secret or configuration map. Pods that use a volume mount to reference secrets or configuration maps receive the updates without a restart by using an eventually consistent approach. By default, the `kubelet` agent watches for changes to the keys and values that are used in volumes for pods on the node. The `kubelet` agent detects changes and propagates the changes to the pods to keep volume data consistent. Despite the automatic updates that Kubernetes provides, a restart of the pod is still required if the software reads configuration data only at startup time.

## Deleting Secrets and Configuration Maps

Similar to other Kubernetes resources, you can use the `delete` command to delete secrets and configuration maps that are no longer needed or in use.

```
[user@host ~]$ kubectl delete secret/demo-secrets -n demo
```

```
[user@host ~]$ oc delete configmap/demo-map -n demo
```



## References

For more information, refer to the *Using Config Maps with Applications* chapter in the Red Hat OpenShift Container Platform 4.14 *Building Applications* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#config-maps](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#config-maps)

For more information, refer to *Providing Sensitive Data to Pods* in the Red Hat OpenShift Container Platform 4.14 *Working with Pods* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-pods-secrets](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-pods-secrets)

For more information, refer to the *Encrypting Etcd Data* chapter in the Red Hat OpenShift Container Platform 4.14 *Security and Compliance* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/security\\_and\\_compliance/index#encrypting-etcd](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/security_and_compliance/index#encrypting-etcd)

## ► Guided Exercise

# Externalize the Configuration of Applications

Deploy a web server taking configuration files from a configuration map.

### Outcomes

In this exercise, you deploy a web application to mount the missing files from a configuration map.

- Create a web application deployment.
- Expose the web application deployment to external access.
- Create a configuration map from two files.
- Mount the configuration map in the web application deployment.

### 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 accessible.

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

### Instructions

- 1. Create a web application deployment named `webconfig` from the web console. Use the `registry.ocp4.example.com:8443/rhscl/httpd-24-rhel7:latest` container image.
- 1.1. Log in to the OpenShift cluster as the `developer` user with the `developer` password by using the OpenShift web console `https://console-openshift-console.apps.ocp4.example.com` URL.
  - 1.2. Change to the `Administrator` perspective, and change to the `storage-configs` project. Select the `Workloads > Deployments` option and click the `Create Deployments` button. Add the `webconfig` deployment name and the image name, and leave the default for the other values.

**Images**

Container: **C container**

Deploy image from an image stream tag

**Image Name \***

registry.ocp4.example.com:8443/rhscl/httpd-24-rhel7.latest

Container image name

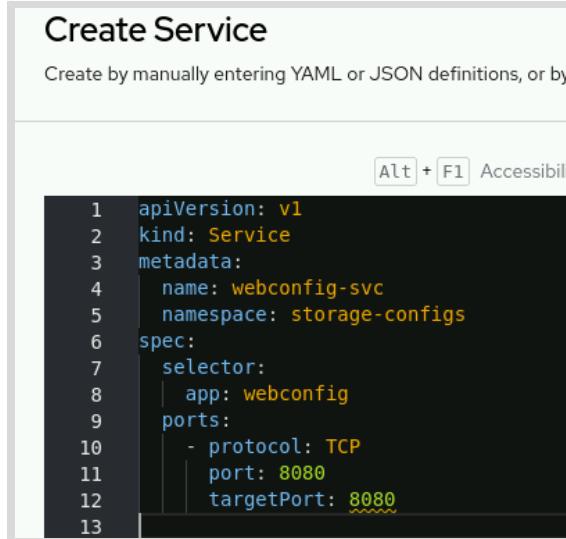
› Show advanced image options

- 1.3. Select the **Create** button. Verify the successful deployment of three pods.
- ▶ 2. Expose the web application to external access from the web console. Use the following information to create a service and a route for the web application, and leave the hostname and path fields blank.

Service field	Service value
Service name	webconfig-svc
App selector	webconfig
Port number	8080
Target port	8080

Route field	Route value
Route name	webconfig-rt
Service name	webconfig-svc
Target port	8080

- 2.1. Select the **Networking > Services** option and click the **Create Service** button to create the **webconfig-svc** service that exposes the **webconfig** deployment. Verify the status of the service.



```

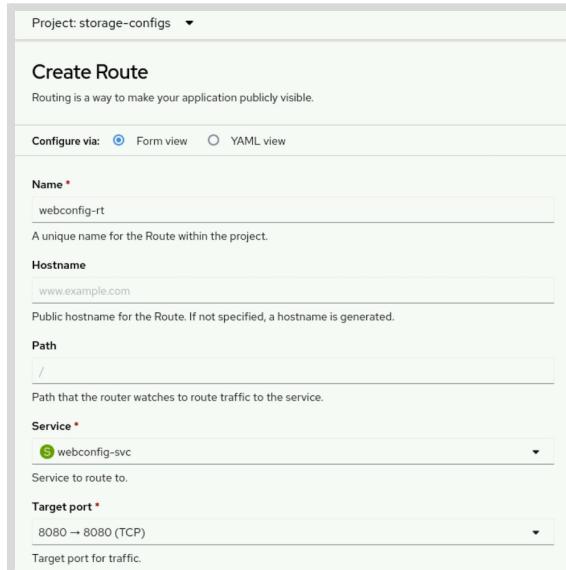
Create Service
Create by manually entering YAML or JSON definitions, or by

Alt + F1 Accessibility

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: webconfig-svc
5    namespace: storage-configs
6  spec:
7    selector:
8      app: webconfig
9    ports:
10   - protocol: TCP
11     port: 8080
12     targetPort: 8080
13

```

- 2.2. Select the **Networking > Routes** option and click the **Create Route** button to create the **webconfig-rt** service that exposes the **webconfig-svc** service.



The screenshot shows the 'Create Route' configuration form. The project is set to 'storage-configs'. The 'Name' field is filled with 'webconfig-rt'. The 'Hostname' field contains 'www.example.com'. The 'Path' field has a single slash ('/'). The 'Service' dropdown is set to 'webconfig-svc'. The 'Target port' dropdown shows '8080 → 8080 (TCP)'. The 'Configure via' section offers 'Form view' (selected) and 'YAML view' options.

- 2.3. Use a web browser to navigate to the `http://webconfig-rt-storage-configs.apps.ocp4.example.com` route. A testing page is displayed by default because of the missing files.



- 3. Use the missing files to create a configuration map from the web console.

- 3.1. Select the Workloads > ConfigMaps option and click the Create ConfigMap button to create the webfiles configuration map by using the redhatlogo.png file and the index.html file in the /home/D0180/labs/storage-configs directory, and verify the creation of the configuration map. Add a Data key, define the index.html name as the Key value, and open the /home/D0180/labs/storage-configs/index.html file.

The screenshot shows a 'Create ConfigMap' dialog. At the top, there's a 'Name' field with 'webfiles' entered. Below it is a description: 'A unique name for the ConfigMap within the project'. There's an 'Immutable' checkbox with a note: 'Immutable, if set to true, ensures that data stored in the ConfigMap cannot be updated'. The 'Data' section contains a 'Key' field with 'index.html' and a 'Value' field containing the content of 'index.html': '<a href="redhatlogo.png">click me!</a>'. At the bottom are 'Create' and 'Cancel' buttons.

Add a Binary Data key, define the redhatlogo.png name as the Key value, and open the /home/D0180/labs/storage-configs/redhatlogo.png file.

The screenshot shows a 'Binary Data' dialog. It has a 'Key' field with 'redhatlogo.png' and a 'Value' field with a 'Browse...' button. A note below says: 'Drag and drop file with your value here or browse to upload it.' A message box indicates: 'Non-printable file detected. File contains non-printable characters. Preview is not available.' At the bottom are 'Add key/value', 'Create', and 'Cancel' buttons.

- 4. Log in to the OpenShift cluster from the command line, and mount the webfiles configuration map as a volume in the webconfig deployment from the command line.

- 4.1. 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
..output omitted...
```

- 4.2. Mount the webfiles configuration map as a volume.

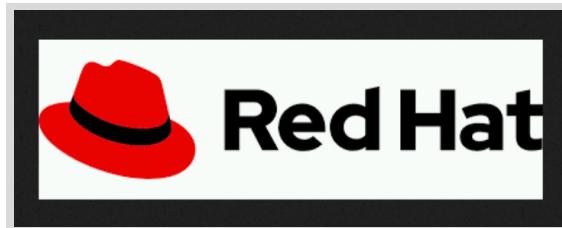
```
[student@workstation ~]$ oc set volume deployment/webconfig \
--add --type configmap --configmap-name webfiles \
--name webfiles-vol --mount-path /var/www/html/
deployment.apps/webconfig volume updated
```

- 4.3. Verify the deployment status. Verify that a new pod was created.

```
[student@workstation ~]$ oc status
...output omitted...
http://webconfig-rt-storage-configs.apps.ocp4.example.com to pod port 8080 (svc/
webconfig-svc)
deployment/webconfig deploys registry.ocp4.example.com:8443/redhattraining/
httpd-noimage:v2
    deployment #2 running for 2 minutes - 1 pod
    deployment #1 deployed 17 minutes ago
...output omitted...
```

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    ...
webconfig-654bcf6cf-wcnnk  1/1     Running   ...
...output omitted...
```

- 4.4. Return to the web browser, and navigate to the `webconfig-rt-storage-configs.apps.ocp4.example.com` route. Click the `Click me!` link to open the file.



The configuration map successfully added the missing files to the web application.

## 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 storage-configs
```

# Provision Persistent Data Volumes

## Objectives

- Provide applications with persistent storage volumes for block and file-based data.

## Kubernetes Persistent Storage

Containers have ephemeral storage by default. The lifetime of this ephemeral storage does not extend beyond the life of the individual pod, and this ephemeral storage cannot be shared across pods. When a container is deleted, all the files and data inside it are also deleted. To preserve the files, containers use persistent storage volumes.

Because OpenShift Container Platform uses the Kubernetes persistent volume (PV) framework, cluster administrators can provision persistent storage for a cluster. Developers can use persistent volume claims (PVCs) to request PV resources without specific knowledge of the underlying storage infrastructure.

Two ways exist to provision storage for the cluster: static and dynamic. Static provisioning requires the cluster administrator to create persistent volumes manually. Dynamic provisioning uses storage classes to create the persistent volumes on demand.

Administrators can use storage classes to provide persistent storage. Storage classes describe types of storage for the cluster. Cluster administrators create storage classes to manage storage services or storage tiers of a service. Rather than specifying provisioned storage, PVCs instead refer to a storage class.

Developers use PVCs to add persistent volumes to their applications. Developers need not know details of the storage infrastructure. With static provisioning, developers use previously created PVs, or ask a cluster administrator to manually create persistent volumes for their applications. With dynamic provisioning, developers declare the storage requirements of the application, and the cluster creates a PV to fill the request.

## Persistent Volumes

Not all storage is equal. Storage types vary in cost, performance, and reliability. Multiple storage types are usually available for each Kubernetes cluster.

The following list of commonly used storage volume types and their use cases is not exhaustive.

### configMap

The configMap volume externalizes the application configuration data. This use of the configMap resource ensures that the application configuration is portable across environments and can be version-controlled.

### emptyDir

An emptyDir volume provides a per-pod directory for scratch data. The directory is usually empty after provisioning. emptyDir volumes are often required for ephemeral storage.

**hostPath**

A hostPath volume mounts a file or directory from the host node into your pod. To use a hostPath volume, the cluster administrator must configure pods to run as privileged. This configuration grants access to other pods in the same node.

Red Hat does not recommend the use of hostPath volumes in production. Instead, Red Hat supports hostPath mounting for development and testing on a single-node cluster. Although most pods do not need a hostPath volume, it does offer a quick option for testing if an application requires it.

**iSCSI**

Internet Small Computer System Interface (iSCSI) is an IP-based standard that provides block-level access to storage devices. With iSCSI volumes, Kubernetes workloads can consume persistent storage from iSCSI targets.

**local**

You can use Local persistent volumes to access local storage devices, such as a disk or partition, by using the standard PVC interface. Local volumes are subject to the availability of the underlying node, and are not suitable for all applications.

**NFS**

An NFS (Network File System) volume can be accessed from multiple pods at the same time, and thus provides shared data between pods. The NFS volume type is commonly used because of its ability to share data safely. Red Hat recommends to use NFS only for non-production systems.

## Volume Access Mode

Persistent volume providers vary in capabilities. A volume uses access modes to specify the modes that it supports. For example, NFS can support multiple read/write clients, but a specific NFS PV might be exported on the server as read-only. OpenShift defines the following access modes, as summarized in the following table:

### Volume Access Modes

Access mode	Abbreviation	Description
ReadWriteOnce	RWO	A single node mounts the volume as read/write.
ReadOnlyMany	ROX	Many nodes mount the volume as read-only.
ReadWriteMany	RWX	Many nodes mount the volume as read/write.

Developers must select a volume type that supports the required access level by the application. The following table shows some example supported access modes:

### Access Mode Support

Volume type	RWO	ROX	RWX
configMap	Yes	No	No
emptyDir	Yes	No	No
hostPath	Yes	No	No

Volume type	RWO	ROX	RWX
iSCSI	Yes	Yes	No
local	Yes	No	No
NFS	Yes	Yes	Yes

## Volume Modes

Kubernetes supports two volume modes for persistent volumes: **Filesystem** and **Block**. If the volume mode is not defined for a volume, then Kubernetes assigns the default volume mode, **Filesystem**, to the volume.

OpenShift Container Platform can provision raw block volumes. These volumes do not have a file system, and can provide performance benefits for applications that either write to the disk directly or that implement their own storage service. Raw block volumes are provisioned by specifying **volumeMode: Block** in the PV and PVC specification.

The following table provides examples of storage options with block volume support:

### Block Volume Support

Volume plug-in	Manually provisioned	Dynamically provisioned
AWS EBS	Yes	Yes
Azure disk	Yes	Yes
Cinder	Yes	Yes
Fibre channel	Yes	No
GCP	Yes	Yes
iSCSI	Yes	No
local	Yes	No
Red Hat OpenShift Data Foundation	Yes	Yes
VMware vSphere	Yes	Yes

## Manually Creating a PV

Use a **PersistentVolume** manifest file to manually create a persistent volume. The following example creates a persistent volume from a fiber channel storage device that uses block mode.

```
apiVersion: v1
kind: PersistentVolume ①
metadata:
  name: block-pv ②
spec:
  capacity:
```

```

storage: 10Gi ③
accessModes:
  - ReadWriteOnce ④
volumeMode: Block ⑤
persistentVolumeReclaimPolicy: Retain ⑥
fc: ⑦
  targetWWNs: ["50060e801049cf1"]
  lun: 0
  readOnly: false

```

- ①** PersistentVolume is the resource type for PVs.
- ②** Provide a name for the PV, which subsequent claims use to access the PV.
- ③** Specify the amount of storage that is allocated to this volume.
- ④** The storage device must support the access mode that the PV specifies.
- ⑤** The volumeMode attribute is optional for Filesystem volumes, but is required for Block volumes.
- ⑥** The persistentVolumeReclaimPolicy determines how the cluster handles the PV when the PVC is deleted. Valid options are Retain or Delete.
- ⑦** The remaining attributes are specific to the storage type. In this example, the fc object specifies the Fiber Channel storage type attributes.

If the previous manifest is in a file named `my-fc-volume.yaml`, then the following command can create the PV resource on RHOCP:

```
[user@host]$ oc create -f my-fc-volume.yaml
```

To create a persistent volume from the web console, click the **Storage > PersistentVolumes** menu.

## Persistent Volume Claims

A persistent volume claim (PVC) resource represents a request from an application for storage. A PVC specifies the minimal storage characteristics, such as capacity and access mode. A PVC does not specify a storage technology, such as iSCSI or NFS.

The lifecycle of a PVC is not tied to a pod, but to a namespace. Multiple pods from the same namespace but with potentially different workload controllers can connect to the same PVC. You can also sequentially connect storage to and detach storage from different application pods, to initialize, convert, migrate, or back up data.

Kubernetes matches each PVC to a persistent volume (PV) resource that can satisfy the requirements of the claim. It is not an exact match. A PVC might be bound to a PV with a larger disk size than is requested. A PVC that specifies single access might be bound to a PV that is shareable for multiple concurrent accesses. Rather than enforcing policy, PVCs declare what an application needs, which Kubernetes provides on a best-effort basis.

## Creating a PVC

A PVC belongs to a specific project. To create a PVC, you must specify the access mode and size, among other options. A PVC cannot be shared between projects. Developers use a PVC to access a persistent volume (PV). Persistent volumes are not exclusive to projects, and are

accessible across the entire OpenShift cluster. When a PV binds to a PVC, the PV cannot be bound to another PVC.

To add a volume to an application deployment, create a `PersistentVolumeClaim` resource, and add it to the application as a volume. Create the PVC by using either a Kubernetes manifest or the `oc set volumes` command. In addition to either creating a PVC or using an existing PVC, the `oc set volumes` command can modify a deployment to mount the PVC as a volume within the pod.

To add a volume to an application deployment, use the `oc set volumes` command:

```
[user@host ~]$ oc set volumes deployment/example-application \ ①
--add \ ②
--name example-pv-storage \ ③
--type persistentVolumeClaim \ ④
--claim-mode rwo \ ⑤
--claim-size 15Gi \ ⑥
--mount-path /var/lib/example-app \ ⑦
--claim-name example-pv-claim ⑧
```

- ① Specify the name of the deployment that requires the PVC resource.
- ② Setting the `add` option to `true` adds volumes and volume mounts for containers.
- ③ The `name` option specifies a volume name. If not specified, a name is autogenerated.
- ④ The supported types, for the `add` operation, include `emptyDir`, `hostPath`, `secret`, `configMap`, and `persistentVolumeClaim`.
- ⑤ The `claim-mode` option defaults to `ReadWriteOnce`. The valid values are `ReadWriteOnce` (`RWO`), `ReadWriteMany` (`RWX`), and `ReadOnlyMany` (`ROX`).
- ⑥ Create a claim with the given size in bytes, if specified along with the persistent volume type. The size must use SI notation, for example, 15, 15 G, or 15 Gi.
- ⑦ The `mount-path` option specifies the mount path inside the container.
- ⑧ The `claim-name` option provides the name for the PVC, and is required for the `persistentVolumeClaim` type.

The command creates a PVC resource and adds it to the application as a volume within the pod.

The command updates the deployment for the application with `volumeMounts` and `volumes` specifications.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...output omitted...
  namespace: storage-volumes ①
  ...output omitted...
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      ...output omitted...
```

```

volumeMounts:
  - mountPath: /var/lib/example-app ②
    name: example-pv-storage ③
...output omitted...
volumes:
  - name: example-pv-storage ④
persistentVolumeClaim:
  claimName: example-pv-claim ⑤
...output omitted...

```

- ① The deployment, which must be in the same namespace as the PVC.
- ② The mount path in the container.
- ③ ④ The volume name, which is used to specify the volume that is associated with the mount.
- ⑤ The claim name that is bound to the volume.

The following example specifies a PVC by using a YAML manifest to create a `PersistentVolumeClaim` API object:

```

---
apiVersion: v1
kind: PersistentVolumeClaim ①
metadata:
  name: example-pv-claim ②
  labels:
    app: example-application
spec:
  accessModes:
    - ReadWriteOnce ③
  resources:
    requests:
      storage: 15Gi ④

```

- ① `PersistentVolumeClaim` is the resource type for a PVC.
- ② Use this name in the `claimName` field of the `persistentVolumeClaim` element in the `volumes` section of a deployment manifest.
- ③ Specify the access mode that this PVC requests. The storage class provisioner must provide this access mode. If persistent volumes are created statically, then an eligible persistent volume must provide this access mode.
- ④ The storage class creates a persistent volume that matches this size request. If persistent volumes are created statically, then an eligible persistent volume must be at least the requested size.

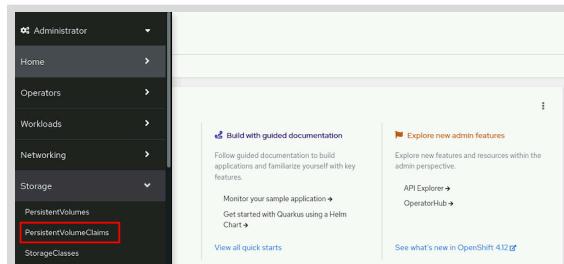
Use the `oc create` command to create the PVC from the manifest file.

```
[user@host ~]$ oc create -f pvc_file_name.yaml
```

Use `oc get pvc` to view the available PVCs in the current namespace.

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	...
db-pod-pvc	Bound	pvc-13...ca45	1Gi	RWO	nfs-storage	...

To create a persistent volume claim from the web console, click the **Storage > PersistentVolumeClaims** menu.



Click **Create PersistentVolumeClaim** and complete the form by adding the name, the storage class, the size, the access mode, and the volume mode.

Create PersistentVolumeClaim		Edit YAML
StorageClass	SC nfs-storage	
StorageClass for the new claim		
PersistentVolumeClaim name *	bd-maria-pvc	
A unique name for the storage claim within the project		
Access mode *	<input checked="" type="radio"/> Single user (RWO) <input type="radio"/> Shared access (RWX) <input type="radio"/> Read only (RO)	
Permissions to the mounted drive		
Size *	3	GiB
Desired storage capacity		

## Kubernetes Dynamic Provisioning

PVs are defined by a **PersistentVolume** API object, which is from existing storage in the cluster. The cluster administrator must statically provision some storage types. Alternatively, the Kubernetes persistent volume framework can use a **StorageClass** object to dynamically provision PVs.

When you create a PVC, you specify a storage amount, the required access mode, and a storage class to describe and classify the storage. The control loop in the RHOCP control node watches for new PVCs, and binds the new PVC to an appropriate PV. If an appropriate PV does not exist, then a provisioner for the storage class creates one.

Claims remain unbound indefinitely if a matching volume does not exist or if a volume cannot be created with any available provisioner that services a storage class. Claims are bound when matching volumes become available. For example, a cluster with many manually provisioned 50 Gi volumes would not match a PVC that requests 100 Gi. The PVC can be bound when a 100 Gi PV is added to the cluster.

Use `oc get storageclass` to view the storage classes that the cluster provides.

```
[user@host ~]$ oc get storageclass
NAME           PROVISIONER
nfs-storage   (default) k8s-sigs.io/nfs-subdir-external-provisioner ...
lvms-vg1       topolvm.io ...
```

In the example, the `nfs-storage` storage class is marked as the default storage class. When a default storage class is configured, the PVC must explicitly name any other storage class to use, or can set the `storageClassName` annotation to "", to be bound to a PV without a storage class.

The following `oc set volume` command example uses the `claim-class` option to specify a dynamically provisioned PV.

```
[user@host ~]$ oc set volumes deployment/example-application \
--add --name example-pv-storage --type pvc --claim-class nfs-storage \
--claim-mode rwo --claim-size 15Gi --mount-path /var/lib/example-app \
--claim-name example-pv-claim
```

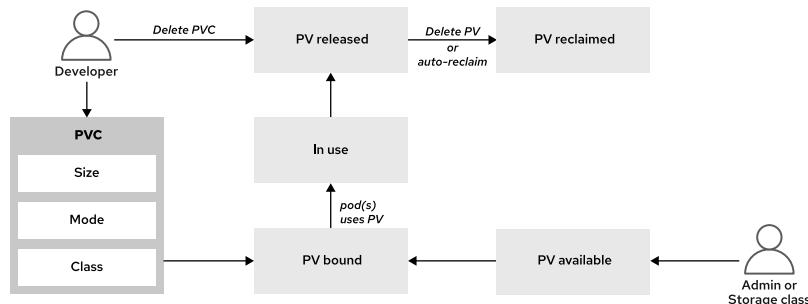


### Note

Because a cluster administrator can change the default storage class, Red Hat recommends that you always specify the storage class when you create a PVC.

## PV and PVC Lifecycles

When you create a PVC, you request a specific amount of storage, access mode, and storage class. Kubernetes binds the PVC to an appropriate PV. If an appropriate PV does not exist, then a provisioner for the storage class creates one.



**Figure 5.17: PV lifecycle**

PVs follow a lifecycle based on their relationship to the PVC.

### Available

After a PV is created, it becomes *available* for any PVC to use in the cluster in any namespace.

### Bound

A PV that is *bound* to a PVC is also bound to the same namespace as the PVC, and no other PVC can use it.

### In Use

You can delete a PVC if no pods actively use it. The *Storage Object in Use Protection* feature prevents the removal of bound PVs and PVCs that pods are actively using. Such removal can result in data loss. Storage Object in Use Protection is enabled by default.

If a user deletes a PVC that a pod actively uses, then the PVC is not removed immediately. PVC removal is postponed until no pods actively use the PVC. Also, if a cluster administrator deletes a PV that is bound to a PVC, then the PV is not removed immediately. PV removal is postponed until the PV is no longer bound to a PVC.

### **Released**

After the developer deletes the PVC that is bound to a PV, the PV is *released*, and the storage that the PV used can be reclaimed.

### **Reclaimed**

The reclaim policy of a persistent volume tells the cluster what to do with the volume after it is released. A volume's reclaim policy can be Retain or Delete.

#### **Volume Reclaim Policy**

Policy	Description
Retain	Enables manual reclamation of the resource for those volume plug-ins that support it.
Delete	Deletes both the PersistentVolume object from OpenShift Container Platform and the associated storage asset in external infrastructure.

## **Deleting a Persistent Volume Claim**

To delete a volume, use the `oc delete` command to delete the PVC. The storage class reclaims the volume after removing the PVC.

```
[user@host ~]$ oc delete pvc/example-pvc-storage
```



### **References**

For more information, refer to the *Understanding Ephemeral Storage* chapter in the Red Hat OpenShift Container Platform 4.14 Storage documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/storage/index#understanding-ephemeral-storage](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/storage/index#understanding-ephemeral-storage)

For more information, refer to the *Understanding Persistent Storage* chapter in the Red Hat OpenShift Container Platform 4.14 Storage documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/storage/index#pvcprotectionUnderstandingPersistentStorage](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/storage/index#pvcprotectionUnderstandingPersistentStorage)

[https://developers.redhat.com/articles/2022/10/06/kubernetes-improves-developer-agility#few\\_cloud\\_native\\_applications\\_are\\_stateless](https://developers.redhat.com/articles/2022/10/06/kubernetes-improves-developer-agility#few_cloud_native_applications_are_stateless)

[https://developers.redhat.com/articles/2022/10/06/kubernetes-storage-concepts#3\\_concepts\\_of\\_kubernetes\\_storage\\_for\\_developers\\_](https://developers.redhat.com/articles/2022/10/06/kubernetes-storage-concepts#3_concepts_of_kubernetes_storage_for_developers_)

<https://loft.sh/blog/kubernetes-persistent-volumes-examples-and-best-practices/>

## ► Guided Exercise

# Provision Persistent Data Volumes

Deploy a MySQL database with persistent storage from a PVC and identify the PV and storage provisioner that backs the application.

### Outcomes

You should be able to do the following tasks:

- Deploy a MySQL database with persistent storage from a PVC.
- Identify the PV that backs the application.
- Identify the storage provisioner that created the PV.

### 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 storage-volumes
```

## Instructions

- ▶ 1. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Select the `storage-volumes` project.
  - 1.1. Use a web browser to navigate to the OpenShift web console at `https://console.openshift-console.apps.ocp4.example.com`.
  - 1.2. Log in by using Red Hat Identity Management with the `developer` username and the `developer` password.
  - 1.3. Select the `Administrator` view to access the administrative menu.
  - 1.4. On the `Home > Projects` page, select the `storage-volumes` project.
- ▶ 2. Identify the default storage class for the cluster.
  - 2.1. Select the `Storage > StorageClasses` menu option.

StorageClasses		
Name	Provisioner	Reclaim policy
SC lvms-vg1	topolvm.io	Delete
SC nfs-storage - Default	k8s-sigs.io/nfs-subdir-external-provisioner	Delete

**Figure 5.18: Storage classes**

The nfs-storage storage class has the **Default** label.

- ▶ 3. Use the `registry.ocp4.example.com:8443/rhel8/mysql-80` container image to create a MySQL deployment named `db-pod`. Use the `storage-volumes` project. Add a service for the database.
  - 3.1. Select the **Workloads > Deployments** menu option.
  - 3.2. Verify that the `storage-volumes` project is active and select the **Create Deployment** button.
  - 3.3. Use `db-pod` for the deployment name and change the image name to `registry.ocp4.example.com:8443/rhel8/mysql-80`.
  - 3.4. Add the environment variables.

#### MySQL Environment Variables

Name	Value
MYSQL_USER	user1
MYSQL_PASSWORD	mypa55w0rd
MYSQL_DATABASE	items

- 3.5. Select the **Advanced options > Scaling** link and set the replicas to one.
- 3.6. Click **Create**. Wait for the blue circle to indicate that a single pod is running.
- 3.7. Click **Networking > Services > Create Service**.
- 3.8. Add the service values.

## Service Fields

Field name	Value
Name	db-pod
selector	app=db-pod
Port	3306
Target port	3306

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: db-pod
5    namespace: storage-volumes
6  spec:
7    selector:
8      app: db-pod
9    ports:
10      - protocol: TCP
11        port: 3306
12        targetPort: 3306
13

```

Figure 5.19: Add Service

- 3.9. Click the **Create** button.
- 4. Add a 1 Gi, RWO PVC named db-pod-pvc to the deployment. Set the /var/lib/mysql directory as the mount path.
- 4.1. Select the **Workloads > Deployments** menu item.
  - 4.2. Click the three vertical dots in the row with the db-pod deployment, and select the **Add storage** menu option.
  - 4.3. In the **Add Storage** form, click **Create new claim**.
  - 4.4. Add the following field values to the form.

## Add PVC Storage Fields

Field name	Value
PersistentVolumeClaim name	db-pod-pvc
Access mode	RWO
Size	1 GiB
Volume mode	Filesystem
Mount path	/var/lib/mysql

- 4.5. Click **Save**.
- 4.6. Scroll down in the deployment details to the **Volumes** section.
- 4.7. Select the **db-pod-pvc** link to see the PVC details.

Name	Mount path	SubPath	Type	Permissions	Utilized by	
db-pod-pvc	/var/lib/mysql	No subpath	PVC db-pod-pvc	Read/Write	container	⋮

Figure 5.20: PVC Link

- ▶ 5. Observe how the volume mount changed the deployment.
- 5.1. Select the **Workloads > Deployments > dp-pod > YAML** tab.
  - 5.2. Observe the **volumes** and **volumeMounts** additions to the deployment.

```
apiVersion: apps/v1
kind: Deployment

...output omitted...

volumes:
- name: nfs-volume-storage
  persistentVolumeClaim:
    claimName: db-pod-pvc

...output omitted...

volumeMounts:
- mountPath: /var/lib/mysql
  name: nfs-volume-storage

...output omitted...
```

- ▶ 6. Use a configuration map resource to add initialization data to the database.

- 6.1. Observe the contents of the **init-db.sql** script that initializes the database.

```
[student@workstation ~]$ cat \
~/DO180/labs/storage-volumes/configmap/init-db.sql
```

```
DROP TABLE IF EXISTS `Item`;
CREATE TABLE `Item` (`id` BIGINT not null auto_increment primary key,
`description` VARCHAR(100), `done` BIT);
INSERT INTO `Item` (`id`, `description`, `done`) VALUES (1,'Pick up newspaper', 0);
INSERT INTO `Item` (`id`, `description`, `done`) VALUES (2, 'Buy groceries', 1);
```

- 6.2. Log in to the OpenShift cluster.

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

- 6.3. Set the `storage-volumes` project as the active project.

```
[student@workstation ~]$ oc project storage-volumes
...output omitted...
```

- 6.4. Use the contents of the `init-db.sql` file to create a configuration map named `init-db-cm`.

```
[student@workstation ~]$ oc create configmap init-db-cm \
--from-file=/home/student/D0180/labs/storage-volumes/configmap/init-db.sql
configmap/init-db-cm created
```

- 6.5. Add the `init-db-cm` configuration map resource as a volume named `init-db-volume` to the deployment. Specify the volume type as `configmap`, and set the `/var/db/config` directory as the mount path.

```
[student@workstation ~]$ oc set volumes deployment/db-pod \
--add --name init-db-volume --type configmap --configmap-name init-db-cm \
--mount-path /var/db/config
deployment.apps/db-pod volume updated
```

- 6.6. Start a remote shell session inside the container.

```
[student@workstation ~]$ oc rsh deployment/db-pod
sh-4.4$
```

- 6.7. Use the `mysql` client to execute the database script in the `/var/db/config/init-db` volume.

```
sh-4.4$ mysql -uuser1 -pmypa55w0rd items </var/db/config/init-db.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
sh-4.4$
```

- 6.8. Execute a query to verify the database contents.

```
sh-4.4$ mysql -uuser1 -pmypa55w0rd items -e 'select * from Item;'
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| id | description      | done   |
+-----+
| 1  | Pick up newspaper | 0x00   |
| 2  | Buy groceries     | 0x01   |
+-----+
sh-4.4$
```

- 6.9. Exit the shell session.

```
sh-4.4$ exit
```

- 7. Delete and then re-create the db-pod deployment.

- 7.1. Delete the db-pod deployment.

```
[student@workstation ~]$ oc delete deployment/db-pod
deployment.apps "db-pod" deleted
```

- 7.2. Verify that the PVC still exists without the deployment.

NAME	STATUS	VOLUME	CAPACITY	...
db-pod-pvc	Bound	pvc-ab5b38c7-359a-4e99-b81c-f7d11ef91cc9	1Gi	...

- 7.3. Re-create the db-pod deployment.

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

- 7.4. Add the environment variables.

```
[student@workstation ~]$ oc set env deployment/db-pod \
MYSQL_USER=user1 \
MYSQL_PASSWORD=mypa55w0rd \
MYSQL_DATABASE=items
deployment.apps/db-pod updated
```

- 8. Use the oc set volume command to attach the existing PVC to the deployment.

```
[student@workstation ~]$ oc set volumes deployment/db-pod \
--add --type pvc \
--mount-path /var/lib/mysql \
--name db-pod-vol \
--claim-name db-pod-pvc
deployment.apps/db-pod volume updated
```

- 9. Create a query-db pod by using the oc run command and the registry.ocp4.example.com:8443/redhattraining/do180-dbinit container image. Use the pod to execute a query against the database service.

- 9.1. Create the query-db pod. Configure the pod to use the MySQL client to execute a query against the db-pod service.

```
[student@workstation ~]$ oc run query-db -it --rm \
--image registry.ocp4.example.com:8443/redhattraining/do180-dbinit \
--restart Never \
-- /bin/bash -c "mysql -uuser1 -pmypa55w0rd --protocol tcp \
-h db-pod -P3306 items -e 'select * from Item;'" \
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+-----+
| id | description      | done      |
+-----+-----+
| 1  | Pick up newspaper | 0x00      |
| 2  | Buy groceries     | 0x01      |
+-----+-----+
pod "query-db" deleted
```

▶ **10.** Delete the db-pod deployment and the db-pod-pvc PVC.

10.1. Delete the db-pod deployment.

```
[student@workstation ~]$ oc delete deployment/db-pod
deployment.apps "db-pod" deleted
```

10.2. Delete the db-pod-pvc PVC.

```
[student@workstation ~]$ oc delete pvc/db-pod-pvc
persistentvolumeclaim "db-pod-pvc" 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 storage-volumes
```

# Select a Storage Class for an Application

## Objectives

- Match applications with storage classes that provide storage services to satisfy application requirements.

## Storage Class Selection

Storage classes are a way to describe types of storage for the cluster and to provision dynamic storage on demand. The cluster administrator determines the meaning of a storage class, which is also called a profile in other storage systems. For example, an administrator can create one storage class for development use, and another for production use.

Kubernetes supports multiple storage back ends. The storage options differ in cost, performance, reliability, and function. An administrator can create different storage classes for these options. As a result, developers can select the storage solution that fits the needs of the application. Developers do not need to know the storage infrastructure details.

Recall that an administrator selects the default storage class for dynamic provisioning. A default storage class enables Kubernetes to automatically provision a persistent volume claim (PVC) that does not specify a storage class. Because an administrator can change the default storage class, a developer should explicitly set the storage class for an application.

## Reclaim Policy

Outside the application function, the developer must also consider the impact of the reclaim policy on storage requirements. A reclaim policy determines what happens to the data on a PVC after the PVC is deleted. When you are finished with a volume, you can delete the PVC object from the API, which enables reclamation of the resource. Kubernetes releases the volume when the PVC is deleted, but the volume is not yet available for another claim. The previous claimant's data remains on the volume and must be handled according to the policy. To keep your data, choose a storage class with a `retain` reclaim policy.

By using the `retain` reclaim policy, when you delete a PVC, only the PVC object is deleted from the cluster. The Persistent Volume (PV) that backed the PVC, the physical storage device that the PV used, and your data still exist. To reclaim the storage and use it in your cluster again, the cluster administrator must take manual steps.

To manually reclaim a PV as a cluster administrator, follow these steps:

1. Delete the PV.

```
[user@host ~]$ oc delete pv <pv-name>
```

The associated asset in the external storage infrastructure, such as an AWS EBS, GCE PD, Azure Disk, or Cinder volume, still exists after the PV is deleted.

2. At this point, the cluster administrator can create another PV by using the same storage and data from the previous PV. A developer could then mount the new PV and access the data from the previous PV.

3. Alternatively, the cluster administrator can remove the data on the storage asset, and then delete the storage asset.

To automatically delete the PV, the data, and the physical storage for a deleted PVC, you must choose a storage class that uses the `delete` reclaim policy. This reclaim policy automatically reclaims your storage volume when the PVC is deleted. The `delete` reclaim policy is the default setting for all storage provisioners that adhere to the Kubernetes Container Storage Interface (CSI) standards. If you use a storage class that does not specify a reclaim policy, then the `delete` reclaim policy is used.

For more information about the Kubernetes Container Storage Interface standards, refer to the *Kubernetes CSI Developer Documentation* website at <https://kubernetes-csi.github.io/docs/>.

## Kubernetes and Application Responsibilities

Kubernetes does not change how an application relates to storage. The application is responsible for working with its storage devices and for ensuring data integrity and consistency. Kubernetes storage does not prevent an application from doing dangerous things, such as sharing a data volume between two databases that require exclusive access to data.

Because a PVC is a storage device that your Linux host mounts, an improperly configured application could behave unexpectedly. For example, you could have an iSCSI LUN, which is expressed as an RWO PVC that is not supposed to be shared, and then mount that same PVC on two pods of the same host. Whether this situation is problematic depends on the applications.

Usually, it is fine for two processes on the same host to share a disk. After all, many applications on your personal machine share a local disk. However, nothing prevents one text editor from overwriting and losing all edits from another text editor. The use of Kubernetes storage must come with the same caution.

Single-node access (RWO) and shared access (RWX) do not ensure that files can be shared safely and reliably. RWO means that only one cluster node can read and write to the PVC. Alternatively, with RWX, Kubernetes provides a storage volume that any pod can access for reading or writing.

## Use Cases for Storage Classes

The administrator creates storage classes that serve the needs of the developers. A `storageClass` object defines each storage class, and the object contains information about the storage provider and the capabilities of the storage medium. The provider creates PVs to match the specifications of the storage class. Administrators can create storage classes with various functional levels, based on many factors.

### Storage volume modes

A storage class with `block` volume mode support can increase performance for applications that can use raw block devices. Consider using a storage class with `Filesystem` volume mode support for applications that share files or that provide file access.

### Quality of Service (QoS) levels

A Solid State Drive (SSD) provides excellent speed and support for frequently accessed files. Use a lower cost and a slower hard drive (HDD) for files that are accessed less often.

### Administrative support tier

A production-tier storage class can include volumes that are backed up often. In contrast, a development-tier storage class might include volumes that are not configured with a backup schedule.

Storage classes can use a combination of these factors and others to best fit the needs of the developers.

Kubernetes matches PVCs with the best available PV that is not bound to another PVC. The PV must provide the access mode that is specified in the PVC, and the volume must be at least as large as the requested size in the PVC. The supported access modes depend on the capabilities of the storage provider. A PVC can specify additional criteria, such as the name of a storage class. If a PVC cannot find a PV that matches all criteria, then the PVC enters a pending state and waits until an appropriate PV becomes available.

PVCs can request a specific storage class by specifying the `storageClassName` attribute. This method of selecting a specific storage class ensures that the storage medium is a good fit for the application requirements. Only PVs of the requested storage class can be bound to the PVC. The cluster administrator can configure dynamic provisioners to service storage classes. The cluster administrator can also create a PV on demand that matches the specifications in the PVC.

## Create a Storage Class

The following YAML excerpt describes the basic definition for a `StorageClass` object. A cluster administrator or a `storage-admin` user creates globally scoped `StorageClass` objects. The following resource shows the parameters for configuring a storage class. This example uses the AWS ElasticBlockStore (EBS) object definition.

```
apiVersion: storage.k8s.io/v1 ①
kind: StorageClass ②
metadata:
  name: io1-gold-storage ③
  annotations: ④
    storageclass.kubernetes.io/is-default-class: 'false'
    description:'Provides RWO and RWOP Filesystem & Block volumes'
    ...
parameters: ⑤
  type: io1
  iopsPerGB: "10"
  ...
provisioner: kubernetes.io/aws-ebs ⑥
reclaimPolicy: Delete ⑦
volumeBindingMode: Immediate ⑧
allowVolumeExpansion: true ⑨
```

- ① A required item that specifies the current API version.
- ② A required item that specifies the API object type.
- ③ A required item that specifies the name of the storage class.
- ④ An optional item that specifies annotations for the storage class.
- ⑤ An optional item that specifies the required parameters for the specific provisioner; this object differs between plug-ins.
- ⑥ A required item that specifies the type of provisioner that is associated with this storage class.
- ⑦ An optional item that specifies the selected reclaim policy for the storage class.

- ❸ An optional item that specifies the selected volume binding mode for the storage class.
- ❹ An optional item that specifies the volume expansion setting.

Several attributes, such as the API version, API object type, and annotations, are common for Kubernetes objects, whereas other attributes are specific to storage class objects.

### Parameters

Parameters can configure file types, change storage types, enable encryption, enable replication, and so on. Each provisioner has different parameter options. Accepted parameters depend on the storage provisioner. For example, the `io1` value for the `type` parameter, and the `iopsPerGB` parameter, are specific to EBS. When a parameter is omitted, the storage provisioner uses the default value.

### Provisioners

The `provisioner` attribute identifies the source of the storage medium plug-in. Provisioners with names that begin with a `kubernetes.io` value are available by default in a Kubernetes cluster.

### ReclaimPolicy

The default reclaim policy, `Delete`, automatically reclaims the storage volume when the PVC is deleted. Reclaiming storage in this way can reduce the storage costs. The `Retain` reclaim policy does not delete the storage volume, so that data is not lost if the wrong PVC is deleted. This reclaim policy can result in higher storage costs if space is not manually reclaimed.

### VolumeBindingMode

The `volumeBindingMode` attribute determines how volume attachments are handled for a requesting PVC. Using the default `Immediate` volume binding mode creates a PV to match the PVC when the PVC is created. This setting does not wait for the pod to use the PVC, and thus can be inefficient. The `Immediate` binding mode can also cause problems for storage back ends that are topology-constrained or are not globally accessible from all nodes in the cluster. PVs are also bound without the knowledge of a pod's scheduling requirements, which might result in unschedulable pods.

By using the `WaitForFirstConsumer` mode, the volume is created after the pod that uses the PVC is in use. With this mode, Kubernetes creates PVs that conform to the pod's scheduling constraints, such as resource requirements and selectors.

### AllowVolumeExpansion

When set to a `true` value, the storage class specifies that the underlying storage volume can be expanded if more storage is required. Users can resize the volume by editing the corresponding PVC object. This feature can be used only to grow a volume, not to shrink it.

The cluster administrator can use the `create` command to create a storage class from a YAML manifest file. The resulting storage class is non-namespaced, and thus is available to all projects in the cluster.

```
[user@host ~]$ oc create -f <storage-class-filename.yaml>
```

To create a storage class from the web console, click the **Storage** > **StorageClasses** menu. Click **Create StorageClass** and complete the form or the YAML manifest.

## Cluster Storage Classes

Use the `oc get storageclass` command to view the storage class options that are available in a cluster.

```
[user@host ~]$ oc get storageclass
```

A regular cluster user can view the attributes of a storage class by using the `describe` command. The following example queries the attributes of the storage class with the name `lvms-vg1`.

```
[user@host ~]$ oc describe storageclass lvms-vg1
IsDefaultClass:          No
Annotations:             description=Provides RWO and RWOP Filesystem & Block
                         volumes
Provisioner:              topolvm.io
Parameters:               csi.storage.k8s.io/fstype=xfs,topolvm.io/device-class=vg1
AllowVolumeExpansion:    True
MountOptions:             <none>
ReclaimPolicy:            Delete
VolumeBindingMode:        WaitForFirstConsumer
Events:                  <none>
```

The `describe` command can help a developer to decide whether the storage class is a good fit for an application. If none of the storage classes in the cluster are appropriate for the application, then the developer can request the cluster administrator to create a PV with the required features.

## Storage Class Usage

Recall that the `oc set volume` command can add a PVC and an associated PV to a deployment. A YAML manifest file can declare the parameters of a PVC independently from the deployment. This method is the preferred option to support repeatability, configuration management, and version control. Use the `storageClassName` attribute to specify the storage class for the PVC.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-block-pvc
spec:
  accessModes:
    - RWO
  volumeMode: Block
  storageClassName: <storage-class-name>
  resources:
    requests:
      storage: 10Gi
```

Use the `create` command to create the resource from the YAML manifest file.

```
[user@host ~]$ oc create -f <my-pvc-filename.yaml>
```

Use the `--claim-name` option with the `set volume` command to add the pre-existing PVC to a deployment.

```
[user@host ~]$ oc set volume deployment/<deployment-name> \
--add --name <my-volume-name> \
--claim-name my-block-pvc \
--mount-path /var/tmp
```



## References

For more information, refer to the *Understanding Persistent Storage* section in the Red Hat OpenShift Container Platform 4.14 Storage documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/storage/index#understanding-persistent-storage](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/storage/index#understanding-persistent-storage)

### Kubernetes Storage

<https://kubernetes.io/docs/concepts/storage/storage-classes/>

For more information about the Kubernetes Container Storage Interface standards, refer to the *Kubernetes CSI Developer Documentation* website at <https://kubernetes-csi.github.io/docs/>

.

## ► Guided Exercise

# Select a Storage Class for an Application

Deploy a MySQL database with persistent storage based on block storage by selecting block storage instead of a default file storage.

### Outcomes

You should be able to deploy applications with persistent storage and create volumes from a storage class. The storage class must meet the application storage requirements.

### 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 storage-classes
```

### Instructions

- 1. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Use the `storage-classes` project.
- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `storage-classes` project as the active project.

```
[student@workstation ~]$ oc project storage-classes
...output omitted...
```

- 2. Examine the available storage classes on the cluster. Identify an appropriate storage class to use for a database application.
- 2.1. Use the `get` command to retrieve a list of storage classes in the cluster. You can use the `storageclass` short name, `sc`, in the command.

```
[student@workstation ~]$ oc get sc
NAME          PROVISIONER      ...
nfs-storage   (default)    k8s-sigs.io/nfs-subdir-external-provisioner ...
lvms-vg1       topolvm.io     ...
```

Because an administrator can change the default storage, applications must specify a storage class that meets the application requirements.

- 2.2. Use the `oc describe` command to view the details of the `lvms-vg1` storage class.

```
[student@workstation ~]$ oc describe sc lvms-vg1
Name:           lvms-vg1
IsDefaultClass: No
Annotations:    description=Provides RWO and RWOP Filesystem & Block
                 volumes
Provisioner:   topolvm.io
Parameters:    csi.storage.k8s.io/fstype=xfs,topolvm.io/device-class=vg1
AllowVolumeExpansion: True
MountOptions:  <none>
ReclaimPolicy: Delete
VolumeBindingMode: WaitForFirstConsumer
Events:        <none>
```

The description annotation states that the storage class provides support for block volumes. For some applications, such as databases, block volumes can provide a performance advantage over file system volumes. In the `lvms-vg1` storage class, the `AllowVolumeExpansion` field is set to `True`. With volume expansion, cluster users can edit their PVC objects and specify a new size for the PVC. Kubernetes then uses the storage back end to automatically expand the volume to the requested size. Kubernetes also expands the file system of pods that use the PVC. Enabling volume expansion can help to protect an application from failing due to the data growing too fast. With these features, the `lvms-vg1` storage class is a good choice for the database application.

- ▶ 3. Use the `registry.ocp4.example.com:8443/rhel8/mysql-80` container image to create a MySQL deployment named `db-pod`. Add the missing environment variables for the pod to run.

- 3.1. Create the `db-pod` deployment.

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

- 3.2. Add the environment variables.

```
[student@workstation ~]$ oc set env deployment/db-pod \
  MYSQL_USER=user1 \
  MYSQL_PASSWORD=mypa55w0rd \
  MYSQL_DATABASE=items
deployment.apps/db-pod updated
```

- 3.3. Verify that the pod is running.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
db-pod-b4ccfb74-nn4s5  1/1     Running   0          5s
```

## 3.4. Expose the db-pod deployment to create a service.

```
[student@workstation ~]$ oc expose deployment/db-pod
service/db-pod exposed
```

- ▶ 4. Add a 1 Gi, RWO PVC named db-pod-odf-pvc to the deployment. Specify the volume name as odf-lvm-storage, and set the /var/lib/mysql directory as the mount path. Use the lvms-vg1 storage class to create a block mode volume.

## 4.1. Use the oc set volume command to create a PVC for the deployment.

```
[student@workstation ~]$ oc set volumes deployment/db-pod \
--add --name odf-lvm-storage --type pvc \
--claim-mode rwo --claim-size 1Gi --mount-path /var/lib/mysql \
--claim-class lvms-vg1 \
--claim-name db-pod-odf-pvc
deployment.apps/db-pod volume updated
```

The claim-class option specifies a non-default storage class.

- 4.2. Use the oc get pvc command to view the status of the PVC. Identify the name of the PV, and confirm that the PVC uses the lvms-vg1 non-default storage class.

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS
...					
db-pod-odf-pvc	Bound	pvc-72...035a	1Gi	RWO	lvms-vg1 ...

- 4.3. Use the oc describe pvc command to inspect the details of the db-pod-odf-pvc PVC.

```
[student@workstation ~]$ oc describe pvc db-pod-odf-pvc
Name: db-pod-odf-pvc
Namespace: storage-classes
StorageClass: lvms-vg1
Status: Bound
Volume: pvc-b3084a46-3f32-435e-987b-4ad3b9026021
Labels: <none>
Annotations: pv.kubernetes.io/bind-completed: yes
pv.kubernetes.io/bound-by-controller: yes
volume.beta.kubernetes.io/storage-provisioner: topolvm.io
volume.kubernetes.io/selected-node: master01
volume.kubernetes.io/storage-provisioner: topolvm.io
Finalizers: [kubernetes.io/pvc-protection]
Capacity: 1Gi
Access Modes: RWO
VolumeMode: Filesystem
Used By: db-pod-56888457d-qmxn9
Events:
...output omitted...
```

The Used By attribute confirms that the PVC is bound to a pod.

- 5. 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 -pmypa55w0rd --protocol tcp \
-h db-pod -P3306 items -e 'show databases;'" \
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| performance_schema |
+-----+
pod "query-db" deleted
```

- 6. Delete the db-pod deployment and the db-pod-odf-pvc PVC.

#### 6.1. Delete the db-pod deployment.

```
[student@workstation ~]$ oc delete all -l app=db-pod
pod "db-pod-568888457d-fvklp" deleted
service "db-pod" deleted
deployment.apps "db-pod" deleted
replicaset.apps "db-pod-568888457d" deleted
...output omitted...
```

#### 6.2. Verify that the PVC still exists without the deployment.

```
[student@workstation ~]$ oc get pvc
NAME           STATUS  VOLUME        CAPACITY  ACCESS MODES  STORAGECLASS ...
db-pod-odf-pvc Bound   pvc-72...035a  1Gi       RWO          lvms-vg1    ...
```

#### 6.3. Delete the db-pod-odf-pvc PVC.

```
[student@workstation ~]$ oc delete pvc db-pod-odf-pvc
persistentvolumeclaim "db-pod-odf-pvc" deleted
```

- 7. Create a PVC for an application that requires a shared storage volume from the nfs-storage storage class.

#### 7.1. Create a PVC YAML manifest file named nfs-pvc.yaml with the following contents:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
```

```
requests:
  storage: 1Gi
storageClassName: nfs-storage
```

- 7.2. Create the PVC by using the `oc create -f` command and the YAML manifest file.

```
[student@workstation ~]$ oc create -f nfs-pvc.yaml
persistentvolumeclaim/nfs-pvc created
```

- 7.3. Use the `oc describe pvc` command to view the details of the PVC resource.

```
[student@workstation ~]$ oc describe pvc nfs-pvc
Name:           nfs-pvc
Namespace:      storage-classes
StorageClass:   nfs-storage
Status:         Bound
Volume:         pvc-9f462124-d96e-43e5-96a7-f96dd9375579
Labels:         <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
                pv.kubernetes.io/bound-by-controller: yes
                volume.beta.kubernetes.io/storage-provisioner: k8s-sigs.io/nfs-
                subdir-external-provisioner
                volume.kubernetes.io/storage-provisioner: k8s-sigs.io/nfs-subdir-
                external-provisioner
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Used By:       <none>
Events:
...output omitted...
```

The `Used By: <none>` attribute shows that no pod is using the PVC. The `Status: Bound` value and the `Volume` attribute assignment confirm that the storage class has its `VolumeBindingMode` set to `Immediate`.

- 8. Use the `registry.ocp4.example.com:8443/ubi9/httpd-24:1-215` container image to create a web application deployment named `web-pod`.

- 8.1. Create the `web-pod` deployment.

```
[student@workstation ~]$ oc create deployment web-pod --port 8080 \
--image registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
deployment.apps/web-pod created
```

- 8.2. Create a service for the `web-pod` application.

```
[student@workstation ~]$ oc expose deployment web-pod
service/web-pod exposed
```

- 8.3. Expose the service to create a route for the `web-pod` application. Specify `web-pod.apps.ocp4.example.com` as the hostname.

```
[student@workstation ~]$ oc expose svc web-pod \
--hostname web-pod.apps.ocp4.example.com
route.route.openshift.io/web-pod exposed
```

- 8.4. View the route that is assigned to the web-pod application.

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT          PATH  SERVICES  PORT  ...
web-pod   web-pod.apps.ocp4.example.com    web-pod  8080  ...
```

- 8.5. Use the curl command to view the index page of the web-pod application.

```
[student@workstation ~]$ curl http://web-pod.apps.ocp4.example.com/
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Test Page for the HTTP Server on Red Hat Enterprise Linux</title>
  ...output omitted...
```

► 9. Add the nfs-pvc PVC to the web-pod deployment.

- 9.1. Use the oc set volume command to add the PVC to the deployment. Specify the volume name as nfs-volume, and set the mount path to the /var/www/html directory.

```
[student@workstation ~]$ oc set volumes deployment/web-pod \
--add --name nfs-volume \
--claim-name nfs-pvc \
--mount-path /var/www/html
deployment.apps/web-pod volume updated
```

The volume mount-path is set to the /var/www/html directory. The server uses this path to serve HTML content.

► 10. Use the registry.ocp4.example.com:8443/redhattraining/do180-roster container image to create a custom web application deployment named app-pod.

- 10.1. Create the app-pod deployment and specify port 9090 as the target port.

```
[student@workstation ~]$ oc create deployment app-pod --port 9090 \
--image registry.ocp4.example.com:8443/redhattraining/do180-roster
deployment.apps/app-pod created
```

- 10.2. Create a service for the app-pod application.

```
[student@workstation ~]$ oc expose deployment app-pod
service/app-pod exposed
```

- 10.3. Expose the service to create a route for the app-pod application. Use app-pod.apps.ocp4.example.com for the hostname.

```
[student@workstation ~]$ oc expose svc app-pod \
--hostname app-pod.apps.ocp4.example.com
route.route.openshift.io/app-pod exposed
```

- 11. Add the nfs-pvc PVC to the app-pod application deployment.

- 11.1. Use the oc set volume command to add the PVC to the deployment. Set the volume name to nfs-volume and the mount path to the /var/tmp directory.

```
[student@workstation ~]$ oc set volumes deployment/app-pod \
--add --name nfs-volume \
--claim-name nfs-pvc \
--mount-path /var/tmp
deployment.apps/app-pod volume updated
```

At this point, the web-pod and the app-pod applications are sharing a PVC. Kubernetes does not have a mechanism to prevent data conflicts between the two applications. In this case, the app-pod application is a writer and the web-pod application is a reader, and thus they do not have a conflict. The application implementation, not Kubernetes, prevents data corruption from the two applications that use the same PVC. The RWO access mode does not protect data integrity. The RWO access mode means that a single node can mount the volume as read/write, and pods that share the volume must exist on the same node.

- 12. Use the app-pod application to add content to the shared volume.

- 12.1. Open a web browser to the <http://app-pod.apps.ocp4.example.com/> page

The screenshot shows a web form with five input fields: First Name, Last Name, State, Hobby, and Contact. Below the form is a summary line showing the entered data: First: Antone Last: Adamson State: Florinese Hobby: sword-fighting Contact: aadamson@example.com. A 'Delete' button is next to the contact email. At the bottom is a 'push' button.

First Name:	<input type="text"/>
Last Name:	<input type="text"/>
State:	<input type="text"/>
Hobby:	<input type="text"/>
Contact:	<input type="text"/>
<input type="button" value="save"/>	
First: Antone Last: Adamson State: Florinese Hobby: sword-fighting Contact: aadamson@example.com <input type="button" value="Delete"/>	
<input type="button" value="push"/>	

- 12.2. In the form, enter your information and then click **save**. The application adds your information to the list after the form.

- 12.3. Click **push** to create the /var/tmp/People.html file on the shared volume.

- 13. Open another tab on the browser and navigate to the `http://web-pod.apps.ocp4.example.com/People.html` page. The web-pod application displays the `People.html` file from the shared volume.

## 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 storage-classes
```

# Manage Non-shared Storage with Stateful Sets

---

## Objectives

- Deploy applications that scale without sharing storage.

## Application Clustering

Clustering applications, such as MySQL and Cassandra, typically require persistent storage to maintain the integrity of the data and files that the application uses. When many applications require persistent storage at the same time, multi-disk provisioning might not be possible due to the limited amount of available resources.

Shared storage solves this problem by allocating the same resources from a single device to multiple services.

## Storage Services

File storage solutions provide the directory structure that is found in many environments. Using file storage is ideal when applications generate or consume reasonable volumes of organized data. Applications that use file-based implementations are prevalent, easy to manage, and provide an affordable storage solution.

File-based solutions are a good fit for data backup and archiving, due to their reliability, as are also file sharing and collaboration services. Most data centers provide file storage solutions, such as a network-attached storage (NAS) cluster, for these scenarios.

*Network-attached storage (NAS)* is a file-based storage architecture that makes stored data accessible to networked devices. NAS gives networks a single access point for storage with built-in security, management, and fault-tolerant capabilities. Out of the multiple data transfer protocols that networks can run, two protocols are fundamental to most networks: *internet protocol (IP)* and *transmission control protocol (TCP)*.

The files that are transferred across these protocols can be formatted as one of the following protocols:

- *Network File Systems (NFS)*: This protocol enables remote hosts to mount file systems over a network and to interact with those file systems as though they are mounted locally.
- *Server Message Blocks (SMB)*: This protocol implements an application-layer network protocol that is used to access resources on a server, such as file shares and shared printers.

NAS solutions can provide file-based storage to applications within the same data center. This approach is common to many application architectures, including the following architectures:

- Web server content
- File share services
- FTP storage
- Backup archives

These applications take advantage of data reliability and the ease of file sharing that is available by using file storage. Additionally, for file storage data, the OS and file system handle the locking and caching of the files.

Although familiar and prevalent, file storage solutions are not ideal for all application scenarios. One particular pitfall of file storage is poor handling of large data sets or unstructured data.

Block storage solutions, such as Storage Area Network (SAN) and iSCSI technologies, provide access to raw block devices for application storage. These block devices function as independent storage volumes, such as the physical drives in servers, and typically require formatting and mounting for application access.

Using block storage is ideal when applications require faster access for optimizing computationally heavy data workloads. Applications that use block-level storage implementations gain efficiencies by communicating at the raw device level, instead of relying on operating system layer access.

Block-level approaches enable data distribution on blocks across the storage volume. Blocks also use basic metadata, including a unique identification number for each block of data, for quick retrieval and reassembly of blocks for reading.

SAN and iSCSI technologies provide applications with block-level volumes from network-based storage pools. Using block-level access to storage volumes is common for application architectures, including the following architectures:

- SQL Databases (single node access).
- Virtual Machines (multinode access).
- High-performance data access.
- Server-side processing applications.
- Multiple block device RAID configurations.

Application storage that uses several block devices in a RAID configuration benefits from the data integrity and performance that the various arrays provide.

With Red Hat OpenShift Container Platform (RHOCP), you can create customized storage classes for your applications. With the NAS and the SAN storage technologies, RHOCP applications can use either the NFS protocol for file-based storage, or the block-level protocol for block storage.

## Introduction to Stateful Sets

A stateful application is characterized by acting according to past states or transactions, which affect the current state and future ones of the application. Using a stateful application simplifies recovery from failures by starting from a certain point in time.

A stateful set is the representation of a set of pods with consistent identities. These identities are defined as a network with a single stable DNS, hostname, and storage from as many volume claims as the stateful set specifies. A stateful set guarantees that a given network identity maps to the same storage identity.

Deployments represent a set of containers within a pod. Each deployment can have many active replicas, depending on the user specification. These replicas can be scaled up or down, as needed. A replica set is a native Kubernetes API object that ensures that the specified number of pod replicas are running. Deployments are used for stateless applications by default, and they can be used for stateful application by attaching a persistent volume. All pods in a deployment share a volume and PVC.

In contrast with deployments, stateful set pods do not share a persistent volume. Instead, stateful set pods each have their own unique persistent volumes. Pods are created without a replica

set, and each replica records its own transactions. Each replica has its own identifier, which is maintained in any rescheduling. You must configure application-level clustering so that stateful set pods have the same data.

Stateful sets are the best option for applications, such as databases, that require consistent identities and non-shared persistent storage.

## Working with Stateful Sets

With Kubernetes, you can use manifest files to specify the intended configuration of a stateful set. You can define the name of the application, labels, the image source, storage, environment variables, and more.

The following snippet shows an example of a YAML manifest file for a stateful set:

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: dbserver ①
spec:
  selector:
    matchLabels:
      app: database ②
  replicas: 3 ③
  template:
    metadata:
      labels:
        app: database ④
    spec:
      containers:
        - env: ⑤
          - name: MYSQL_USER
            valueFrom:
              secretKeyRef:
                key: user
                name: sakila-cred
            image: registry.ocp4.example.com:8443/redhattraining/mysql-app:v1 ⑥
            name: database ⑦
            ports: ⑧
              - containerPort: 3306
                name: database
            volumeMounts: ⑨
              - mountPath: /var/lib/mysql
                name: data
            terminationGracePeriodSeconds: 10
      volumeClaimTemplates:
        - metadata:
            name: data
      spec:
        accessModes: [ "ReadWriteOnce" ] ⑩
        storageClassName: "lvms-vg1" ⑪
        resources:
          requests:
            storage: 1Gi ⑫

```

- ① Name of the stateful set.
- ② ④ Application labels.
- ③ Number of replicas.
- ⑤ Environment variables, which can be explicitly defined, or by using a secret object.
- ⑥ Image source.
- ⑦ Container name.
- ⑧ Container ports.
- ⑨ Mount path information for the persistent volumes for each replica. Each persistent volume has the same configuration.
- ⑩ The access mode of the persistent volume. You can choose between the `ReadWriteOnce`, `ReadWriteMany`, and `ReadOnlyMany` options.
- ⑪ The storage class that the persistent volume uses.
- ⑫ Size of the persistent volume.



**Note**

Stateful sets can be created only by using manifest files. The `oc` and `kubectl` CLI do not have commands to create stateful sets imperatively.

Create the stateful set by using the `create` command:

```
[user@host ~]$ oc create -f statefulset-dbserver.yml
```

Verify the creation of the stateful set named dbserver:

```
[user@host ~]$ kubectl get statefulset  
NAME      READY   AGE  
dbserver  3/3    6s
```

Verify the status of the instances:

```
[user@host ~]$ oc get pods  
NAME        READY   STATUS    RESTARTS   AGE  
dbserver-0  1/1     Running   0          85s  
dbserver-1  1/1     Running   0          82s  
dbserver-2  1/1     Running   0          79s
```

Verify the status of the persistent volumes:

```
[user@host ~]$ kubectl get pvc
NAME           STATUS  VOLUME          CAPACITY  ACCESS MODES
STORAGECLASS ...
data-dbserver-0  Bound   pvc-c28f61ee...  1Gi       RWO        nfs-
storage ...
data-dbserver-1  Bound   pvc-ddbe6af1...  1Gi       RWO        nfs-
storage ...
data-dbserver-2  Bound   pvc-8302924a...  1Gi       RWO        nfs-
storage ...
```

Notice that three PVCs were created. Confirm that persistent volumes are attached to each instance:

```
[user@host ~]$ oc describe pod dbserver-0
...output omitted...
Volumes:
  data:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
    the same namespace)
    ClaimName: data-dbserver-0
...output omitted...
```

```
[user@host ~]$ oc describe pod dbserver-1
...output omitted...
Volumes:
  data:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
    the same namespace)
    ClaimName: data-dbserver-1
...output omitted...
```

```
[user@host ~]$ oc describe pod dbserver-2
...output omitted...
Volumes:
  data:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
    the same namespace)
    ClaimName: data-dbserver-2
...output omitted...
```



### Note

You must configure application-level clustering for stateful set pods to have the same data.

You can update the number of replicas of the stateful set by using the `scale` command:

```
[user@host ~]$ oc scale statefulset/dbserver --replicas 1
NAME      READY  STATUS  RESTARTS ...
dbserver-0 1/1    Running  0        ...
```

To delete the stateful set, use the `delete statefulset` command:

```
[user@host ~]$ kubectl delete statefulset dbserver
statefulset.apps "dbserver" deleted
```

Notice that the PVCs are not deleted after the execution of the `oc delete statefulset` command:

```
[user@host ~]$ oc get pvc
NAME           STATUS    VOLUME          CAPACITY   ACCESS MODES
STORAGECLASS ...
data-dbserver-0 Bound    pvc-c28f61ee-...  1Gi        RWO        nfs-
storage ...
data-dbserver-1 Bound    pvc-ddbe6af1-...  1Gi        RWO        nfs-
storage ...
data-dbserver-2 Bound    pvc-8302924a-...  1Gi        RWO        nfs-
storage ...
```

You can create a stateful set from the web console by clicking the **Workloads > StatefulSets** menu. Click **Create StatefulSet** and customize the YAML manifest.



## References

### Kubernetes Documentation - StatefulSets

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

For more information, refer to the *What Is Network-Attached Storage?* section in the *Understanding Data Storage* chapter at

<https://www.redhat.com/en/topics/data-storage/network-attached-storage#how-does-it-work>

## ► Guided Exercise

# Manage Non-shared Storage with Stateful Sets

Deploy a replicated web server by using a deployment and verify that all web server pods share a PV; and deploy a replicated MySQL database by using a stateful set and verify that each database instance gets a dedicated PV.

## Outcomes

In this exercise, you deploy a web server with a shared persistent volume between the replicas, and a database server from a stateful set with dedicated persistent volumes for each instance.

- Deploy a web server with persistent storage.
- Add data to the persistent storage.
- Scale the web server deployment and observe the data that is shared with the replicas.
- Create a database server with a stateful set by using a YAML manifest file.
- Verify that each instance from the stateful set has a persistent volume claim.

## 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 storage-statefulsets
```

## Instructions

- 1. Create a web server deployment named `web-server`. Use the `registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:latest` container image.
- 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
...output omitted...
```

- 1.2. Change to the `storage-statefulsets` project.

```
[student@workstation]$ oc project storage-statefulsets
Now using project "storage-statefulsets" on server ...output omitted...
```

## 1.3. Create the web-server deployment.

```
[student@workstation ~]$ oc create deployment web-server \
--image registry.ocp4.example.com:8443/redhattraining/hello-world-nginx:latest
deployment.apps/web-server created
```

## 1.4. Verify the deployment status.

```
[student@workstation ~]$ oc get pods -l app=web-server
NAME           READY   STATUS    RESTARTS   AGE
web-server-7d7cb4cdc7-t7hx8   1/1     Running   0          4s
```

- 2. Add the web-pv persistent volume to the web-server deployment. Use the default storage class and the following information to create the persistent volume:

Field	Value
Name	web-pv
Type	persistentVolumeClaim
Claim mode	rwo
Claim size	5Gi
Mount path	/usr/share/nginx/html
Claim name	web-pv-claim

## 2.1. Add the web-pv persistent volume to the web-server deployment.

```
[student@workstation ~]$ oc set volumes deployment/web-server \
--add --name web-pv --type persistentVolumeClaim --claim-mode rwo \
--claim-size 5Gi --mount-path /usr/share/nginx/html --claim-name web-pv-claim
deployment.apps/web-server volume updated
```

Because a storage class was not specified with the --claim-class option, the command uses the default storage class to create the persistent volume.

## 2.2. Verify the deployment status. Notice that a new pod is created.

```
[student@workstation ~]$ oc get pods -l app=web-server
NAME           READY   STATUS    RESTARTS   AGE
web-server-64689877c6-mdr6f   1/1     Running   0          5s
```

## 2.3. Verify the persistent volume status.

```
[student@workstation ~]$ oc get pvc
NAME      STATUS    VOLUME      CAPACITY   ACCESS MODES  STORAGECLASS   AGE
web-pv-claim  Bound    pvc-42...63ab  5Gi        RWO          nfs-storage   29s
```

The default storage class, nfs-storage, provisioned the persistent volume.

- 3. Add data to the PV by using the exec command.

- 3.1. List pods to retrieve the web-server pod name.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
web-server-64689877c6-mdr6f   1/1     Running   0          17m
```

The pod name might differ in your output.

- 3.2. Use the exec command to add the pod name that you retrieved from the previous step to the /usr/share/nginx/html/index.html file on the pod. Then, retrieve the contents of the /var/www/html/index.html file to confirm that the pod name is in the file.

```
[student@workstation ~]$ oc exec -it pod/web-server-64689877c6-mdr6f \
-- /bin/bash -c \
'echo "Hello, World from ${HOSTNAME}" > /usr/share/nginx/html/index.html'
```

```
[student@workstation ~]$ oc exec -it pod/web-server-64689877c6-mdr6f \
-- cat /usr/share/nginx/html/index.html
Hello, World from web-server-64689877c6-mdr6f
```

- 4. Scale the web-server deployment to two replicas and confirm that an additional pod is created.

- 4.1. Scale the web-server deployment to two replicas.

```
[student@workstation ~]$ oc scale deployment web-server --replicas 2
deployment.apps/web-server scaled
```

- 4.2. Verify the replica status and retrieve the pod names.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
web-server-64689877c6-mbj6g   1/1     Running   0          2s
web-server-64689877c6-mdr6f   1/1     Running   0          17m
```

The pod names might differ from your output.

- 5. Retrieve the content of the /usr/share/nginx/html/index.html file on the web-server pods by using the oc exec command to verify that the file is the same in both pods.

- 5.1. Verify that the /usr/share/nginx/html/index.html file is the same in both pods.

```
[student@workstation ~]$ oc exec -it pod/web-server-64689877c6-mbj6g \
-- cat /usr/share/nginx/html/index.html
Hello, World from web-server-64689877c6-mdr6f
```

```
[student@workstation ~]$ oc exec -it pod/web-server-64689877c6-mdr6f \
-- cat /usr/share/nginx/html/index.html
Hello, World from web-server-64689877c6-mdr6f
```

Notice that both files show the name of the first instance, because they share the persistent volume.

- ▶ 6. Create a database server with a stateful set by using the `statefulset-db.yml` file in the `/home/student/D0180/labs/storage-statefulsets` directory. Update the file with the following information:

Field	Value
<code>metadata.name</code>	<code>dbserver</code>
<code>spec.selector.matchLabels.app</code>	<code>database</code>
<code>spec.template.metadata.labels.app</code>	<code>database</code>
<code>spec.template.spec.containers.name</code>	<code>dbserver</code>
<code>spec.template.spec.containers.volumeMounts.name</code>	<code>data</code>
<code>spec.template.spec.containers.volumeMounts.mountPath</code>	<code>/var/lib/mysql</code>
<code>spec.volumeClaimTemplates.metadata.name</code>	<code>data</code>
<code>spec.volumeClaimTemplates.spec.storageClassName</code>	<code>lvms-vg1</code>

- 6.1. Open the `/home/student/D0180/labs/storage-statefulsets/statefulset-db.yml` file in an editor. Replace the `<CHANGE_ME>` objects with values from the previous table:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: dbserver
spec:
  selector:
    matchLabels:
      app: database
  replicas: 2
  template:
    metadata:
      labels:
        app: database
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: dbserver
          image: registry.ocp4.example.com:8443/redhattraining/mysql-app:v1
          ports:
            - name: database
              containerPort: 3306
```

```

env:
- name: MYSQL_USER
  value: "redhat"
- name: MYSQL_PASSWORD
  value: "redhat123"
- name: MYSQL_DATABASE
  value: "sakila"
volumeMounts:
- name: data
  mountPath: /var/lib/mysql
volumeClaimTemplates:
- metadata:
    name: data
spec:
  accessModes: [ "ReadWriteOnce" ]
  storageClassName: "lvms-vg1"
  resources:
    requests:
      storage: 1Gi

```

- 6.2. Create the database server by using the `oc create -f /home/student/D0180/labs/storage-statefulsets/statefulset-db.yml` command.

```
[student@workstation ~]$ oc create -f \
/home/student/D0180/labs/storage-statefulsets/statefulset-db.yml
statefulset.apps/bdserver created
```

- 6.3. Wait a few moments and then verify the status of the stateful set and its instances.

```
[student@workstation ~]$ oc get statefulset
NAME      READY   AGE
dbserver  2/2    10s
```

```
[student@workstation ~]$ oc get pods -l app=database
NAME        READY   STATUS ...
dbserver-0  1/1    Running ...
dbserver-1  1/1    Running ...
```

- 6.4. Use the `exec` command to add data to each of the stateful set pods.

```
[student@workstation ~]$ oc exec -it pod/dbserver-0 -- /bin/bash -c \
"mysql -uredhat -predhat123 sakila -e 'create table items (count INT);'"
mysql: [Warning] Using a password on the command line interface can be insecure.
```

```
[student@workstation ~]$ oc exec -it pod/dbserver-1 -- /bin/bash -c \
"mysql -uredhat -predhat123 sakila -e 'create table inventory (count INT);'"
mysql: [Warning] Using a password on the command line interface can be insecure.
```

- 7. Confirm that each instance from the dbserver stateful set has a persistent volume claim. Then, verify that each persistent volume claim contains unique data.

- 7.1. Confirm that the persistent volume claims have a Bound status.

```
[student@workstation ~]$ oc get pvc -l app=database
NAME          STATUS  ...  CAPACITY ACCESS MODE ...
data-dbserver-0  Bound   ...  1Gi      RWO       ...
data-dbserver-1  Bound   ...  1Gi      RWO       ...
```

- 7.2. Verify that each instance from the dbserver stateful set has its own persistent volume claim by using the `oc get pod pod-name -o json | jq .spec.volumes[0].persistentVolumeClaim.claimName` command.

```
[student@workstation ~]$ oc get pod dbserver-0 -o json | \
jq .spec.volumes[0].persistentVolumeClaim.claimName
"data-dbserver-0"
```

```
[student@workstation ~]$ oc get pod dbserver-1 -o json | \
jq .spec.volumes[0].persistentVolumeClaim.claimName
"data-dbserver-1"
```

- 7.3. Application-level clustering is not enabled for the dbserver stateful set. Verify that each instance of the dbserver stateful set has unique data.

```
[student@workstation ~]$ oc exec -it pod/dbserver-0 -- /bin/bash -c \
"mysql -uredhat -predhat123 sakila -e 'show tables;'" 
mysql: [Warning] Using a password on the command line interface can be insecure.
-----
| Tables_in_sakila |
-----
| items            |
```

```
[student@workstation ~]$ oc exec -it pod/dbserver-1 -- /bin/bash -c \
"mysql -uredhat -predhat123 sakila -e 'show tables;'" 
mysql: [Warning] Using a password on the command line interface can be insecure.
-----
| Tables_in_sakila |
-----
| inventory        |
```

- 8. Delete a pod in the dbserver stateful set. Confirm that a new pod is created and that the pod uses the PVC from the previous pod. Verify that the previously added table exists in the sakila database.

- 8.1. Delete the dbserver-0 pod in the dbserver stateful set. Confirm that a new pod is generated for the stateful set. Then, confirm that the data-dbserver-0 PVC still exists.

```
[student@workstation ~]$ oc delete pod dbserver-0  
pod "dbserver-0" deleted
```

```
[student@workstation ~]$ oc get pods -l app=database
```

NAME	READY	STATUS	RESTARTS	AGE
dbserver-0	1/1	Running	0	4s
dbserver-1	1/1	Running	0	5m

```
[student@workstation ~]$ oc get pvc -l app=database
```

NAME	STATUS	CAPACITY	ACCESS MODE	...
data-dbserver-0	Bound	1Gi	RWO	...
data-dbserver-1	Bound	1Gi	RWO	...

- 8.2. Use the `exec` command to verify that the new `dbserver-0` pod has the `items` table in the `sakila` database.

```
[student@workstation ~]$ oc exec -it pod/dbserver-0 -- /bin/bash -c \  
"mysql -uredhat -predhat123 sakila -e 'show tables;'"  
mysql: [Warning] Using a password on the command line interface can be insecure.  
-----  
| Tables_in_sakila |  
-----  
| items |  
-----
```

## 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 storage-statefulsets
```

## ► Lab

# Manage Storage for Application Configuration and Data

Deploy a web application and its database that share database credentials from a secret. The database should use the default storage for the cluster. Also deploy a file-sharing application that runs with multiple replicas and shares its storage volume with a file uploader application. The file sharing and file uploader applications take configuration files from a config map and should use NFS file storage for shareability. The database should use local storage for increased and performance.

## Outcomes

- Deploy a database server.
- Deploy a web application.
- Create a secret that contains the database server credentials.
- Create a configuration map that contains an SQL file.
- Add and remove a volume on the database server and the web application.
- Expose the database server and the web application.
- Scale up the web application.
- Mount the configuration map as a volume.

## 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 accessible and that all exercise resources are available. It also creates the `storage-review` project, and it creates files that this lab uses, in the `/home/student/D0180/labs/storage-review` directory.

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

## Instructions

The API URL of your OpenShift cluster is <https://api.ocp4.example.com:6443>, and the `oc` command is already installed on your `workstation` machine.

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

Use the `storage-review` project for your work.

1. Log in to the OpenShift cluster and change to the `storage-review` project.
2. Create a secret named `world-cred` that contains the following data:

Field	Value
user	redhat
password	redhat123
database	world_x

3. Create a configuration map named dbfiles by using the ~/DO180/labs/storage-review/insertdata.sql file.
4. Create a database server deployment named dbserver by using the registry.ocp4.example.com:8443/redhattraining/mysql-app:v1 container image. Then, set the missing environment variables by using the world-cred secret.
5. Add a volume to the dbserver deployment by using the following information:

Field	Value
name	dbserver-lvm
type	persistentVolumeClaim
claim mode	rwo
claim size	1Gi
mount path	/var/lib/mysql
claim class	lvms-vg1
claim name	dbserver-lvm-pvc

6. Create a service for the dbserver deployment by using the following information:

Field	Value
Name	mysql-service
Port	3306
Target port	3306

7. Create a web application deployment named file-sharing by using the registry.ocp4.example.com:8443/redhattraining/php-webapp-mysql:v1

container image. Scale the deployment to two replicas. Then, expose the deployment by using the following information:

Field	Value
Name	file-sharing
Port	8080
Target port	8080

Create a route named **file-sharing** to expose the **file-sharing** web application to external access. Access the **file-sharing** route in a web browser to test the connection between the web application and the database server.

8. Mount the **dbfiles** configuration map to the **file-sharing** deployment as a volume named **config-map-pvc**. Set the mount path to the **/home/database-files** directory. Then, verify the content of the **insertdata.sql** file.
9. Add a shared volume to the **file-sharing** deployment. Use the following information to create the volume:

Field	Value
Name	shared-volume
Type	persistentVolumeClaim
Claim mode	rwo
Claim size	1Gi
Mount path	/home/sharedfiles
Claim class	nfs-storage
Claim name	shared-pvc

Next, connect to a **file-sharing** deployment pod and then use the **cp** command to copy the **/home/database-files/insertdata.sql** file to the **/home/sharedfiles** directory. Then, remove the **config-map-pvc** volume from the **file-sharing** deployment.

10. Add the **shared-volume** PVC to the **dbserver** deployment. Then, connect to a **dbserver** deployment pod and verify the content of the **/home/sharedfiles/insertdata.sql** file.
11. Connect to the database server and execute the **/home/sharedfiles/insertdata.sql** file to add data to the **world\_x** database. You can execute the file by using the following command:

```
mysql -u$MYSQL_USER -p$MYSQL_PASSWORD world_x </home/sharedfiles/insertdata.sql
```

Then, confirm connectivity between the web application and database server by accessing the **file-sharing** route in a web browser.

## 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 storage-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 storage-review
```

## ► Solution

# Manage Storage for Application Configuration and Data

Deploy a web application and its database that share database credentials from a secret. The database should use the default storage for the cluster. Also deploy a file-sharing application that runs with multiple replicas and shares its storage volume with a file uploader application. The file sharing and file uploader applications take configuration files from a config map and should use NFS file storage for shareability. The database should use local storage for increased and performance.

### Outcomes

- Deploy a database server.
- Deploy a web application.
- Create a secret that contains the database server credentials.
- Create a configuration map that contains an SQL file.
- Add and remove a volume on the database server and the web application.
- Expose the database server and the web application.
- Scale up the web application.
- Mount the configuration map as a volume.

### 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 accessible and that all exercise resources are available. It also creates the **storage-review** project, and it creates files that this lab uses, in the **/home/student/D0180/labs/storage-review** directory.

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

## Instructions

The API URL of your OpenShift cluster is <https://api.ocp4.example.com:6443>, and the **oc** command is already installed on your **workstation** machine.

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

Use the **storage-review** project for your work.

1. Log in to the OpenShift cluster and change to the **storage-review** project.
  - 1.1. Log in to the OpenShift cluster.

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

- 1.2. Change to the storage-review project.

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

2. Create a secret named world-cred that contains the following data:

Field	Value
user	redhat
password	redhat123
database	world_x

- 2.1. Create a secret that contains the database credentials.

```
[student@workstation]$ oc create secret generic world-cred \
--from-literal user=redhat \
--from-literal password=redhat123 \
--from-literal database=world_x
secret/world-cred created
```

- 2.2. Confirm the creation of the secret.

```
[student@workstation ~]$ oc get secrets world-cred
NAME      TYPE      DATA   AGE
world-cred  Opaque    3      2m34s
```

3. Create a configuration map named dbfiles by using the ~/D0180/labs/storage-review/insertdata.sql file.

- 3.1. Create a configuration map named dbfiles by using the insertdata.sql file in the ~/D0180/labs/storage-review directory.

```
[student@workstation ~]$ oc create configmap dbfiles \
--from-file ~/D0180/labs/storage-review/insertdata.sql
configmap/dbfiles created
```

- 3.2. Verify the creation of the configuration map.

```
[student@workstation]$ oc get configmaps
NAME  DATA  AGE
dbfiles  1      11s
...output omitted...
```

4. Create a database server deployment named dbserver by using the registry.ocp4.example.com:8443/redhattraining/mysql-app:v1 container image. Then, set the missing environment variables by using the world-cred secret.

- 4.1. Create the database server deployment.

```
[student@workstation ~]$ oc create deployment dbserver \
--image registry.ocp4.example.com:8443/redhattraining/mysql-app:v1
deployment.apps/dbserver created
```

- 4.2. Set the missing environment variables.

```
[student@workstation ~]$ oc set env deployment/dbserver --from secret/world-cred \
--prefix MYSQL_
deployment.apps/dbserver updated
```

- 4.3. Verify that the dbserver pod is in the RUNNING state. The pod name might differ in your output.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    ...
dbserver-6d5bf5d86c-ptrb2   1/1     Running ...
```

5. Add a volume to the dbserver deployment by using the following information:

Field	Value
name	dbserver-lvm
type	persistentVolumeClaim
claim mode	rwo
claim size	1Gi
mount path	/var/lib/mysql
claim class	lvms-vg1
claim name	dbserver-lvm-pvc

- 5.1. Add a volume to the dbserver deployment.

```
[student@workstation ~]$ oc set volume deployment/dbserver \
--add --name dbserver-lvm --type persistentVolumeClaim \
--claim-mode rwo --claim-size 1Gi --mount-path /var/lib/mysql \
--claim-class lvms-vg1 --claim-name dbserver-lvm-pvc
deployment.apps/dbserver volume updated
```

- 5.2. Verify the deployment status.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    ...
dbserver-5bc6bd5d7b-7z7lv  1/1     Running  ...
```

5.3. Verify the volume status.

```
[student@workstation ~]$ oc get pvc
NAME           STATUS  VOLUME          CAPACITY ...output omitted...
dbserver-lvm-pvc  Bound   pvc-2cb85025-...  1Gi      ...output omitted...
```

6. Create a service for the dbserver deployment by using the following information:

Field	Value
Name	mysql-service
Port	3306
Target port	3306

6.1. Expose the dbserver deployment.

```
[student@workstation ~]$ oc expose deployment dbserver --name mysql-service \
--port 3306 --target-port 3306
service/mysql-service exposed
```

6.2. Verify the service configuration. The endpoint IP address might differ in your output.

```
[student@workstation ~]$ oc get services
NAME        TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)      ...
mysql-service  ClusterIP  172.30.240.100  <none>        3306/TCP    ...
```

```
[student@workstation ~]$ oc get endpoints
NAME        ENDPOINTS    ...
mysql-service  10.8.1.36:3306 ...
```

7. Create a web application deployment named file-sharing by using the registry.ocp4.example.com:8443/redhattraining/php-webapp-mysql:v1 container image. Scale the deployment to two replicas. Then, expose the deployment by using the following information:

Field	Value
Name	file-sharing
Port	8080
Target port	8080

Create a route named **file-sharing** to expose the **file-sharing** web application to external access. Access the **file-sharing** route in a web browser to test the connection between the web application and the database server.

- 7.1. Create a web application deployment.

```
[student@workstation ~]$ oc create deployment file-sharing \
--image registry.ocp4.example.com:8443/redhattraining/php-webapp-mysql:v1
deployment.apps/file-sharing created
```

- 7.2. Verify the deployment status. Verify that the **file-sharing** application pod is in the **RUNNING** state. The pod names might differ on your system.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    ...
dbserver-5bc6bd5d7b-7z7lv   1/1     Running   ...
file-sharing-789c5948c8-gdrlz   1/1     Running   ...
```

- 7.3. Scale the deployment to two replicas.

```
[student@workstation ~]$ oc scale deployment file-sharing --replicas 2
deployment.apps/file-sharing scaled
```

- 7.4. Verify the replica status and retrieve the pod name. The pod names might differ on your system.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    ...
dbserver-5bc6bd5d7b-7z7lv   1/1     Running   ...
file-sharing-789c5948c8-62j9s   1/1     Running   ...
file-sharing-789c5948c8-gdrlz   1/1     Running   ...
```

- 7.5. Expose the **file-sharing** deployment.

```
[student@workstation ~]$ oc expose deployment file-sharing --name file-sharing \
--port 8080 --target-port 8080
service/file-sharing exposed
```

- 7.6. Verify the service configuration. The endpoint IP address might differ in your output.

```
[student@workstation ~]$ oc get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      ...
file-sharing   ClusterIP   172.30.139.210   <none>        8080/TCP      ...
mysql-service   ClusterIP   172.30.240.100   <none>        3306/TCP      ...
```

```
[student@workstation ~]$ oc get endpoints
NAME      ENDPOINTS
file-sharing   10.8.1.37:8080,10.8.1.38:8080      ...
mysql-service   10.8.1.36:3306      ...
```

## 7.7. Expose the file-sharing service.

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

```
[student@workstation ~]$ oc get routes
NAME          HOST/PORT           ...  SERVICES
...
file-sharing   file-sharing-storage-review.apps.ocp4.example.com  ...  file-
sharing ...
```

- 7.8. Test the connectivity between the web application and the database server. In a web browser, navigate to <http://file-sharing-storage-review.apps.ocp4.example.com>, and verify that a Connected successfully message is displayed.
8. Mount the dbfiles configuration map to the file-sharing deployment as a volume named config-map-pvc. Set the mount path to the /home/database-files directory. Then, verify the content of the insertdata.sql file.
- 8.1. Mount the dbfiles configuration map to the file-sharing deployment.

```
[student@workstation ~]$ oc set volume deployment/file-sharing \
--add --name config-map-pvc --type configmap \
--configmap-name dbfiles \
--mount-path /home/database-files
deployment.apps/file-sharing volume updated
```

## 8.2. Verify the deployment status.

```
[student@workstation ~]$ oc get pods
NAME        READY   STATUS    ...
dbserver-5bc6bd5d7b-7z7lv     1/1     Running   ...
file-sharing-7f77855b7f-949lg  1/1     Running   ...
file-sharing-7f77855b7f-9zvwq  1/1     Running   ...
```

## 8.3. Verify the content of the /home/database-files/insertdata.sql file.

```
[student@workstation ~]$ oc exec -it pod/file-sharing-7f77855b7f-949lg -- \
head /home/database-files/insertdata.sql
-- MySQL dump 10.13 Distrib 8.0.19, for osx10.14 (x86_64)
--
-- Host: 127.0.0.1 Database: world_x
-----
-- Server version 8.0.19-debug
...output omitted...
```

9. Add a shared volume to the file-sharing deployment. Use the following information to create the volume:

Field	Value
Name	shared-volume
Type	persistentVolumeClaim
Claim mode	rwo
Claim size	1Gi
Mount path	/home/sharedfiles
Claim class	nfs-storage
Claim name	shared-pvc

Next, connect to a file-sharing deployment pod and then use the cp command to copy the /home/database-files/insertdata.sql file to the /home/sharedfiles directory. Then, remove the config-map-pvc volume from the file-sharing deployment.

#### 9.1. Add the shared-volume volume to the file-sharing deployment.

```
[student@workstation ~]$ oc set volume deployment/file-sharing \
--add --name shared-volume --type persistentVolumeClaim \
--claim-mode rwo --claim-size 1Gi --mount-path /home/sharedfiles \
--claim-class nfs-storage --claim-name shared-pvc
deployment.apps/file-sharing volume updated
```

#### 9.2. Verify the deployment status. Your pod names might differ on your system.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    ...
dbserver-5bc6bd5d7b-7z7lv   1/1     Running ...
file-sharing-65884f75bb-92fxf   1/1     Running ...
file-sharing-65884f75bb-gsghk   1/1     Running ...
```

#### 9.3. Verify the volume status.

```
[student@workstation ~]$ oc get pvc
NAME      STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS
dbserver-lvm-pvc Bound    pvc-2cb...  1Gi        RWO          lvms-vg1 ...
shared-pvc   Bound    pvc-cf2...  1Gi        RWO          nfs-storage ...
```

#### 9.4. Copy the /home/database-files/insertdata.sql file to the /home/sharedfiles path.

```
[student@workstation ~]$ oc exec -it pod/file-sharing-65884f75bb-92fxf -- \
cp /home/database-files/insertdata.sql /home/sharedfiles/
```

```
[student@workstation ~]$ oc exec -it pod/file-sharing-65884f75bb-92fxf -- \
ls /home/sharedfiles/
insertdata.sql
```

9.5. Remove the config-map-pvc volume from the file-sharing deployment.

```
[student@workstation ~]$ oc set volume deployment/file-sharing \
--remove --name=config-map-pvc
deployment.apps/file-sharing volume updated
```

10. Add the shared-volume PVC to the dbserver deployment. Then, connect to a dbserver deployment pod and verify the content of the /home/sharedfiles/insertdata.sql file.

10.1. Add the shared-volume volume to the dbserver deployment.

```
[student@workstation ~]$ oc set volume deployment/dbserver \
--add --name shared-volume \
--claim-name shared-pvc \
--mount-path /home/sharedfiles
deployment.apps/dbserver volume updated
```

10.2. Verify the deployment status. The pod names might differ on your system.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    ...
dbserver-6676fbf5fc-n9hpk   1/1     Running   ...
file-sharing-5fdb44cf57-2hhwj  1/1     Running   ...
file-sharing-5fdb44cf57-z4n7g  1/1     Running   ...
```

10.3. Verify the content of the /home/sharedfiles/insertdata.sql file.

```
[student@workstation ~]$ oc exec -it pod/dbserver-6676fbf5fc-n9hpk -- \
head /home/sharedfiles/insertdata.sql
-- MySQL dump 10.13 Distrib 8.0.19, for osx10.14 (x86_64)
--
-- Host: 127.0.0.1 Database: world_x
--
-- Server version 8.0.19-debug
...output omitted...
```

11. Connect to the database server and execute the /home/sharedfiles/insertdata.sql file to add data to the world\_x database. You can execute the file by using the following command:

```
mysql -u$MYSQL_USER -p$MYSQL_PASSWORD world_x </home/sharedfiles/insertdata.sql
```

Then, confirm connectivity between the web application and database server by accessing the file-sharing route in a web browser.

- 11.1. Connect to the database server and execute the /home/sharedfiles/insertdata.sql file. Then, exit the database server.

```
[student@workstation ~]$ oc rsh dbserver-6676fbf5fc-n9hpk
```

```
sh-4.4$ mysql -u$MYSQL_USER -p$MYSQL_PASSWORD world_x
</home/sharedfiles/insertdata.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
sh-4.4$ exit
exit
```

- 11.2. Test the connectivity between the web application and the database server. In a web browser, navigate to <http://file-sharing-storage-review.apps.ocp4.example.com>, and verify that the application retrieves data from the `world_x` database.

Connected successfully		
1	Kabul	AFG Kabul {"Population": 1780000}
2	Qandahar	AFG Qandahar {"Population": 237500}
3	Herat	AFG Herat {"Population": 186800}
4	Mazar-e-Sharif	AFG Balkh {"Population": 127800}
5	Amsterdam	NLD Noord-Holland {"Population": 731200}
6	Rotterdam	NLD Zuid-Holland {"Population": 593321}

## 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 storage-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 storage-review
```

# Summary

---

- Configuration maps are objects that provide mechanisms to inject configuration data into containers.
- The values in secrets are always encoded (not encrypted).
- A persistent volume claim (PVC) resource represents a request from an application for storage, and specifies the minimal storage characteristics, such as the capacity and access mode.
- Kubernetes supports two volume modes for persistent volumes: `Filesystem` and `Block`.
- Storage classes are a way to describe types of storage for the cluster and to provision dynamic storage on demand.
- A reclaim policy determines what happens to the data on a PVC after the PVC is deleted.
- A storage class with block volume mode support can improve performance for applications that can use raw block devices.
- A stateful set is the representation of a set of pods with consistent identities.
- Stateful set pods are assigned individual persistent volumes.



# Configure Applications for Reliability

### Goal

Configure applications to work with Kubernetes for high availability and resilience.

### Objectives

- Describe how Kubernetes tries to keep applications running after failures.
- Describe how Kubernetes uses health probes during deployment, scaling, and failover of applications.
- Configure an application with resource requests so Kubernetes can make scheduling decisions.
- Configure an application with resource limits so Kubernetes can protect other applications from it.
- Configure a horizontal pod autoscaler for an application.

### Sections

- Application High Availability with Kubernetes (and Guided Exercise)
- Application Health Probes (and Guided Exercise)
- Reserve Compute Capacity for Applications (and Guided Exercise)
- Limit Compute Capacity for Applications (and Guided Exercise)
- Application Autoscaling (and Guided Exercise)

### Lab

- Configure Applications for Reliability

# Application High Availability with Kubernetes

---

## Objectives

- Describe how Kubernetes tries to keep applications running after failures.

## Concepts of Deploying Highly Available Applications

*High availability* (HA) is a goal of making applications more robust and resistant to runtime failures. Implementing HA techniques decreases the likelihood that an application is completely unavailable to users.

In general, HA can protect an application from failures in the following contexts:

- From itself in the form of application bugs
- From its environment, such as networking issues
- From other applications that exhaust cluster resources

Additionally, HA practices can protect the cluster from applications, such as one with a memory leak.

## Writing Reliable Applications

At its core, cluster-level HA tooling mitigates worst-case scenarios. HA is not a substitute for fixing application-level issues, but augments developer mitigations. Although required for reliability, application security is a separate concern.

Applications must work with the cluster so that Kubernetes can best handle failure scenarios. Kubernetes expects the following behaviors from applications:

- Tolerates restarts
- Responds to health probes, such as the startup, readiness, and liveness probes
- Supports multiple simultaneous instances
- Has well-defined and well-behaved resource usage
- Operates with restricted privileges

Although the cluster can run applications that lack the preceding behaviors, applications with these behaviors better use the reliability and HA features that Kubernetes provides.

Most HTTP-based applications provide an endpoint to verify application health. The cluster can be configured to observe this endpoint and mitigate potential issues for the application.

The application is responsible for providing such an endpoint. Developers must decide how the application determines its state.

For example, if an application depends on a database connection, then the application might respond with a healthy status only when the database is reachable. However, not all applications

that make database connections need such a check. This decision is at the discretion of the developers.

## Kubernetes Application Reliability

If an application pod crashes, then it cannot respond to requests. Depending on the configuration, the cluster can automatically restart the pod. If the application fails without crashing the pod, then the pod does not receive requests. However, the cluster can do so only with the appropriate health probes.

Kubernetes uses the following HA techniques to improve application reliability:

- Restarting pods: By configuring a restart policy on a pod, the cluster restarts misbehaving instances of an application.
- Probing: By using health probes, the cluster knows when applications cannot respond to requests, and can automatically act to mitigate the issue.
- Horizontal scaling: When the application load changes, the cluster can scale the number of replicas to match the load.

These techniques are explored throughout this chapter.

## ► Guided Exercise

# Application High Availability with Kubernetes

Simulate different types of application failures and observe how Kubernetes handles them.

### Outcomes

- Explore how the `restartPolicy` attribute affects crashing pods.
- Observe the behavior of a slow-starting application that has no configured probes.
- Use a deployment to scale the application, and observe the behavior of a broken 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 following conditions are true:

- The `reliability-ha` project exists.
- The resource files are available in the course directory.
- The classroom registry has the `long-load` container image.

The `long-load` container image contains an application with utility endpoints. These endpoints perform such tasks as crashing the process and toggling the server's health status.

```
[student@workstation ~]$ lab start reliability-ha
```

### Instructions

- 1. As the `developer` user, create a pod from a YAML manifest in the `reliability-ha` project.

1.1. Log in as the `developer` user with the `developer` password.

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

1.2. Select the `reliability-ha` project.

```
[student@workstation ~]$ oc project reliability-ha
Now using project "reliability-ha" on server "https://api.ocp4.example.com:6443".
```

1.3. Navigate to the lab materials directory and view the contents of the pod definition. In particular, `restartPolicy` is set to `Always`.

```
[student@workstation ~]$ cd DO180/labs/reliability-ha
```

```
[student@workstation reliability-ha]$ cat long-load.yaml
apiVersion: v1
kind: Pod
metadata:
  name: long-load
spec:
  containers:
    - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
      name: long-load
      securityContext:
        allowPrivilegeEscalation: false
  restartPolicy: Always
```

- 1.4. Create a pod by using the `oc apply` command.

```
[student@workstation reliability-ha]$ oc apply -f long-load.yaml
pod/long-load created
```

- 1.5. Send a request to the pod to confirm that it is running and responding.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
curl -s localhost:3000/health
Ok
```

- ▶ 2. Trigger the pod to crash, and observe that the `restartPolicy` instructs the cluster to re-create the pod.

- 2.1. Observe that the pod is running and has not restarted.

```
[student@workstation reliability-ha]$ oc get pods

NAME      READY   STATUS    RESTARTS   AGE
long-load  1/1     Running   0          1m
```

- 2.2. Send a request to the `/destruct` endpoint in the application. This request triggers the process to crash.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
curl -s localhost:3000/destruct
command terminated with exit code 52
```

- 2.3. Observe that the pod is running and restarted one time.

```
[student@workstation reliability-ha]$ oc get pods

NAME      READY   STATUS    RESTARTS   AGE
long-load  1/1     Running   1 (34s ago)  4m16s
```

- 2.4. Delete the `long-load` pod.

```
[student@workstation reliability-ha]$ oc delete pod long-load  
pod "long-load" deleted
```

The pod is not re-created, because it was created manually, and not via a workload resource such as a deployment.

- 3. Use a restart policy of **Never** to create the pod, and observe that it is not re-created on crashing.

3.1. Modify the `long-load.yaml` file so that the `restartPolicy` is set to **Never**.

```
...output omitted...  
restartPolicy: Never
```

3.2. Create the pod with the updated YAML file.

```
[student@workstation reliability-ha]$ oc apply -f long-load.yaml  
pod/long-load created
```

3.3. Send a request to the pod to confirm that the pod is running and that the application is responding.

```
[student@workstation reliability-ha]$ oc exec long-load -- \  
curl -s localhost:3000/health  
Ok
```

3.4. Send a request to the `/destruct` endpoint in the application to crash it.

```
[student@workstation reliability-ha]$ oc exec long-load -- \  
curl -s localhost:3000/destruct  
command terminated with exit code 52
```

3.5. Observe that the pod is not restarted and is in an error state.

```
[student@workstation reliability-ha]$ oc get pods  
NAME      READY   STATUS    RESTARTS   AGE  
long-load  0/1     Error    0          2m36s
```

3.6. Delete the `long-load` pod.

```
[student@workstation reliability-ha]$ oc delete pod long-load  
pod "long-load" deleted
```

- 4. Because the cluster does not know when the application inside the pod is ready to receive requests, you must add a startup delay to the application. Adding this capability by using probes is covered in a later exercise.

4.1. Update the `long-load.yaml` file by adding a startup delay and use a restart policy of **Always**. Set the `START_DELAY` variable to 60,000 milliseconds (one minute) so that the file looks like the following excerpt:

```
...output omitted...
spec:
  containers:
    - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
      imagePullPolicy: Always
      securityContext:
        allowPrivilegeEscalation: false
      name: long-load
      env:
        - name: START_DELAY
          value: "60000"
    restartPolicy: Always
```

**Note**

Although numbers are a valid YAML type, environment variables must be passed as strings. YAML syntax is also indentation-sensitive.

For these reasons, ensure that your file appears *exactly* as the preceding example.

- 4.2. Apply the YAML file to create the pod and proceed within one minute to the next step.

```
[student@workstation reliability-ha]$ oc apply -f long-load.yaml
pod/long-load created
```

- 4.3. Within a minute of pod creation, verify the status of the pod. The status shows as ready even though it is not. Try to send a request to the application, and observe that it fails.

```
[student@workstation reliability-ha]$ oc get pods
NAME      READY   STATUS    RESTARTS   AGE
long-load  1/1     Running   0          16s
```

```
[student@workstation reliability-ha]$ oc exec long-load -- \
curl -s localhost:3000/health
app is still starting
```

- 4.4. After waiting a minute for the application to start, send another a request to the pod to confirm that it is running and responding.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
curl -s localhost:3000/health
Ok
```

5. Use a deployment to scale up the number of deployed pods. Observe that deleting the pods causes service outages, even though the deployment handles re-creating the pods.
- 5.1. Review the `long-load-deploy.yaml` file, which defines a deployment, service, and route. The deployment creates three replicas of the application pod.

In each pod, a START\_DELAY environment variable is set to 15,000 milliseconds (15 seconds). In each pod, the application responds that it is not ready until after the delay.

```
[student@workstation reliability-ha]$ cat long-load-deploy.yaml
...output omitted...
spec:
  containers:
    - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
      imagePullPolicy: Always
      name: long-load
      env:
        - name: START_DELAY
          value: "15000"
...output omitted...
```

- 5.2. Start the load test script, which sends a request to the /health API endpoint of the application every two seconds. Leave the script running in a visible terminal window.

```
[student@workstation reliability-ha]$ ./load-test.sh
...output omitted...
```

- 5.3. In a new terminal window, apply the ~/D0180/labs/reliability-ha/long-load-deploy.yaml file.

```
[student@workstation ~]$ oc apply -f \
~/D0180/labs/reliability-ha/long-load-deploy.yaml
deployment.apps/long-load created
service/long-load created
route.route.openshift.io/long-load created
```

- 5.4. Watch the output of the load test script as the pods and the application instances start. After a delay, the requests succeed.

```
...output omitted...
Ok
Ok
Ok
...output omitted...
```

- 5.5. By using the /togglesick API endpoint of the application, put one of the three pods into a broken state.

```
[student@workstation ~]$ curl \
long-load-reliability-ha.apps.ocp4.example.com/togglesick
no output expected
```

- 5.6. Watch the output of the load test script as some requests start failing. Because of the load balancer, the exact order of the output is random.

```
...output omitted...
Ok
app is unhealthy
app is unhealthy
Ok
Ok
...output omitted...
```

Press **Ctrl+C** to end the load test script.

- 5.7. Return to the `/home/student/` directory.

```
[student@workstation reliability-ha]$ cd /home/student/
[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 reliability-ha
```

# Application Health Probes

---

## Objectives

- Describe how Kubernetes uses health probes during deployment, scaling, and failover of applications.

## Kubernetes Probes

Health probes are an important part of maintaining a robust cluster. *Probes* enable the cluster to determine the status of an application by repeatedly probing it for a response.

A set of health probes affect a cluster's ability to do the following tasks:

- Crash mitigation by automatically attempting to restart failing pods
- Failover and load balancing by sending requests only to healthy pods
- Monitoring by determining whether and when pods are failing
- Scaling by determining when a new replica is ready to receive requests

## Authoring Probe Endpoints

Application developers are expected to code health probe endpoints during application development. These endpoints determine the health and status of the application. For example, a data-driven application might report a successful health probe only if it can connect to the database.

Because the cluster calls them often, health probe endpoints should be quick to perform. Endpoints should not perform complicated database queries or many network calls.

## Probe Types

Kubernetes provides the following types of probes: startup, readiness, and liveness. Depending on the application, you might configure one or more of these types.

### Readiness Probes

A *readiness probe* determines whether the application is ready to serve requests. If the readiness probe fails, then Kubernetes prevents client traffic from reaching the application by removing the pod's IP address from the service resource.

Readiness probes help to detect temporary issues that might affect your applications. For example, the application might be temporarily unavailable when it starts, because it must establish initial network connections, load files in a cache, or perform initial tasks that take time to complete. The application might occasionally need to run long batch jobs, which make it temporarily unavailable to clients.

Kubernetes continues to run the probe even after the application fails. If the probe succeeds again, then Kubernetes adds back the pod's IP address to the service resource, and requests are sent to the pod again.

In such cases, the readiness probe addresses a temporary issue and improves application availability.

## Liveness Probes

Like a readiness probe, a *liveness probe* is called throughout the lifetime of the application. Liveness probes determine whether the application container is in a healthy state. If an application fails its liveness probe enough times, then the cluster restarts the pod according to its `restartPolicy` policy.

Unlike a startup probe, liveness probes are called after the application's initial start process. Usually, this mitigation is by restarting or re-creating the pod.

## Startup Probes

A *startup probe* determines when an application's startup is completed. Unlike a liveness probe, a startup probe is not called after the probe succeeds. If the startup probe does not succeed after a configurable timeout, then the pod is restarted based on its `restartPolicy` value.

Consider adding a startup probe to applications with a long start time. By using a startup probe, the liveness probe can remain short and responsive.

## Types of Tests

When defining a probe, you must specify one of the following types of test to perform:

### HTTP GET

Each time that the probe runs, the cluster sends a request to the specified HTTP endpoint. The test is considered a success if the request responds with an HTTP response code between 200 and 399. Other responses cause the test to fail.

### Container command

Each time that the probe runs, the cluster runs the specified command in the container. If the command exits with a status code of 0, then the test succeeds. Other status codes cause the test to fail.

### TCP socket

Each time that the probe runs, the cluster attempts to open a socket to the container. The test succeeds only if the connection is established.

## Timings and Thresholds

All the types of probes include timing variables. The `period seconds` variable defines how often the probe runs. The `failure threshold` defines how many failed attempts are required before the probe itself fails.

For example, a probe with a failure threshold of 3 and period seconds of 5 can fail up to three times before the overall probe fails. Using this probe configuration means that the issue can exist for 10 seconds before it is mitigated. However, running probes too often can waste resources. Consider these values when setting probes.

## Adding Probes via YAML

Because probes are defined on a pod template, probes can be added to workload resources such as deployments. To add a probe to an existing deployment, update and apply the YAML file or use the `oc edit` command. For example, the following YAML excerpt defines a deployment pod template with a probe:

```

apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
...output omitted...
  template:
    spec:
      containers:
        - name: web-server
          ...output omitted...
        livenessProbe: ①
          failureThreshold: 6 ②
          periodSeconds: 10 ③
          httpGet: ④
            path: /health ⑤
            port: 3000 ⑥

```

- ① Defines a liveness probe.
- ② Specifies how many times the probe must fail before mitigating.
- ③ Defines how often the probe runs.
- ④ Sets the probe as an HTTP request and defines the request port and path.
- ⑤ Specifies the HTTP path to send the request to.
- ⑥ Specifies the port to send the HTTP request over.

## Adding Probes via the CLI

The `oc set probe` command adds or modifies a probe on a deployment. For example, the following command adds a readiness probe to a deployment called `front-end`:

```

[user@host ~]$ oc set probe deployment/front-end \
--readiness \ ①
--failure-threshold 6 \ ②
--period-seconds 10 \ ③
--get-url http://:8080/healthz ④

```

- ① Defines a readiness probe.
- ② Sets how many times the probe must fail before mitigating.
- ③ Sets how often the probe runs.
- ④ Sets the probe as an HTTP request, and defines the request port and path.

## Adding Probes via the Web Console

To add or modify a probe on a deployment from the web console, navigate to the **Workloads > Deployments** menu and select a deployment.

Click Actions and then click Add Health Checks.

Click Edit probe to specify the readiness type, the HTTP headers, the path, the port, and more.



### Note

The set probe command is exclusive to RHOCP and oc.



### References

#### Configure Liveness, Readiness and Startup Probes

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes>

For more information about health probes, refer to the *Monitoring Application Health by Using Health Checks* chapter in the Red Hat OpenShift Container Platform 4.14 *Building Applications* documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#application-health](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#application-health)

## ► Guided Exercise

# Application Health Probes

Configure health probes in a deployment and verify that network clients are insulated from application failures.

### Outcomes

- Observe potential issues with an application that is not configured with health probes.
- Configure startup, liveness, and readiness probes for the application.

### Before You Begin

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

This command ensures that the following conditions are true:

- The `reliability-probes` project exists.
- The resource files are available in the course directory.
- The classroom registry has the `long-load` container image.

The `registry.ocp4.example.com:8443/redhattraining/long-load:v1` container image contains an application with utility endpoints. These endpoints perform such tasks as crashing the process and toggling the server's health status.

```
[student@workstation ~]$ lab start reliability-probes
```

## Instructions

- 1. As the `developer` user, deploy the `long-load` application in the `reliability-probes` project.
- 1.1. Log in 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. Set the `reliability-probes` project as the active project.

```
[student@workstation ~]$ oc project reliability-probes
Now using project reliability-probes on server "https://
api.ocp4.example.com:6443".
```

- 1.3. Apply the `long-load-deploy.yaml` file to create the pod. Move to the next step within one minute.

```
[student@workstation ~]$ oc apply -f \
~/D0180/labs/reliability-probes/long-load-deploy.yaml
deployment.apps/long-load created
service/long-load created
route.route.openshift.io/long-load created
```

- 1.4. Verify that the pods take several minutes to start by sending a request to a pod in the deployment.

```
[student@workstation ~]$ oc exec deploy/long-load -- \
curl -s localhost:3000/health
app is still starting
```

- 1.5. Observe that the pods are listed as ready even though the application is not ready.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
long-load-8564d998cc-579nx  1/1     Running   0          30s
long-load-8564d998cc-ttqpg  1/1     Running   0          30s
long-load-8564d998cc-wjtfw  1/1     Running   0          30s
```

- 2. Add a startup probe to the pods so that the cluster knows when the pods are ready.

- 2.1. Modify the ~/D0180/labs/reliability-probes/long-load-deploy.yaml YAML file by defining a startup probe. The probe runs every three seconds and triggers a pod as failed after 30 failed attempts. The file should match the following excerpt:

```
...output omitted...
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      containers:
        - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
          imagePullPolicy: Always
          name: long-load
          startupProbe:
            failureThreshold: 30
            periodSeconds: 3
            httpGet:
              path: /health
              port: 3000
          env:
...output omitted...
```

- 2.2. Scale down the deployment to zero replicas.

```
[student@workstation ~]$ oc scale deploy/long-load --replicas 0
deployment.apps/long-load scaled
```

**Note**

Red Hat recommends scaling down an application to zero replicas before deleting or changing a deployment. Scaling down to zero replicas stops any new pods from being created while enabling existing pods to terminate gracefully after finishing all current requests. For multiple deployment updates, scaling down to zero prevents a noisy event log from Kubernetes responding to every change, and conserves cluster resources.

- 2.3. Apply the updated `long-load-deploy.yaml` file. Because the YAML file specifies the number of replicas, the deployment is scaled up. Move to the next step within one minute.

```
[student@workstation ~] oc apply -f \
~/D0180/labs/reliability-probes/long-load-deploy.yaml
deployment.apps/long-load configured
service/long-load unchanged
route.route.openshift.io/long-load configured
```

- 2.4. Observe that the pods do not show as ready until the application is ready and the startup probe succeeds. Wait for the three pods to reach the ready state. Press `Ctrl+C` to stop the watch command.

```
[student@workstation ~]$ watch oc get pods
Every 2.0s: oc get pods

NAME          READY   STATUS    RESTARTS   AGE
long-load-785b5b4fc8-7x5ln  1/1     Running   0          90s
long-load-785b5b4fc8-f7pdk  1/1     Running   0          90s
long-load-785b5b4fc8-r2nqj  1/1     Running   0          90s
```

- ▶ 3. Add a liveness probe so that broken instances of the application are restarted.

- 3.1. Start the load test script.

```
[student@workstation ~]$ ~/D0180/labs/reliability-probes/load-test.sh
Ok
Ok
Ok
...output omitted...
```

Keep the script running in a visible window.

- 3.2. In a new terminal window, use the `/togglesick` endpoint to make one of the pods unhealthy. Move to the next step within one minute.

```
[student@workstation ~]$ oc exec \
deploy/long-load -- curl -s localhost:3000/togglesick
no output expected
```

The load test window begins to show `app is unhealthy`. Because only one pod is unhealthy, the remaining pods still respond with `Ok`.

- 3.3. Update the `~/DO180/labs/reliability-probes/long-load-deploy.yaml` file to add a liveness probe. The probe runs every three seconds and triggers the pod as failed after three failed attempts. Modify the `spec.template.spec.containers` object in the file to match the following excerpt.

```
spec:
  ...
  template:
    ...
    spec:
      containers:
        - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
          ...
          startupProbe:
            failureThreshold: 30
            periodSeconds: 3
            httpGet:
              path: /health
              port: 3000
            livenessProbe:
              failureThreshold: 3
              periodSeconds: 3
              httpGet:
                path: /health
                port: 3000
      env:
        ...

```

- 3.4. Scale down the deployment to zero replicas.

```
[student@workstation ~]$ oc scale deploy/long-load --replicas 0
deployment.apps/long-load scaled
```

The load test script shows that the application is not available.

- 3.5. Apply the updated `long-load-deploy.yaml` file to update the deployment, which triggers the deployment to re-create its pods.

```
[student@workstation ~]$ oc apply -f \
~/DO180/labs/reliability-probes/long-load-deploy.yaml
deployment.apps/long-load configured
service/long-load unchanged
route.route.openshift.io/long-load configured
```

- 3.6. Wait for the three new pods to reach the ready state. Press `Ctrl+c` to stop the watch command.

```
[student@workstation ~]$ watch oc get pods
Every 2.0s: oc get pods

NAME          READY   STATUS    RESTARTS   AGE
long-load-785b5b4fc8-8x5ln  1/1     Running   0          70s
long-load-785b5b4fc8-f8pdk  1/1     Running   0          70s
long-load-785b5b4fc8-r9nqj  1/1     Running   0          70s
```

- 3.7. Wait for the load test window to show Ok for all responses, and then toggle one of the pods to be unhealthy.

```
[student@workstation ~]$ oc exec \
  deploy/long-load -- curl -s localhost:3000/togglesick
no output expected
```

The load test window might show app is unhealthy a number of times before the pod is restarted.

- 3.8. Observe that the unhealthy pod is restarted after the liveness probe fails. After the pod is restarted, the load test window shows only Ok.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
long-load-fbb7468d9-8xm8j  1/1     Running   0          9m42s
long-load-fbb7468d9-k66dm  1/1     Running   0          8m38s
long-load-fbb7468d9-ncxkh  0/1     Running   1 (11s ago)  10m
```

- ▶ 4. Add a readiness probe so that traffic goes only to pods that are ready and healthy.

- 4.1. Scale down the deployment to zero replicas.

```
[student@workstation ~]$ oc scale deploy/long-load --replicas 0
deployment.apps/long-load scaled
```

- 4.2. Use the oc set probe command to add the readiness probe.

```
[student@workstation ~]$ oc set probe deploy/long-load \
  --readiness --failure-threshold 1 --period-seconds 3 \
  --get-url http://:3000/health
deployment.apps/long-load probes updated
```

- 4.3. Scale up the deployment to three replicas.

```
[student@workstation ~]$ oc scale deploy/long-load --replicas 3
deployment.apps/long-load scaled
```

- 4.4. Observe the status of the pods by using a watch command.

```
[student@workstation ~]$ watch oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
long-load-d5794d744-8hqlh  0/1     Running   0          48s
long-load-d5794d744-hphgb  0/1     Running   0          48s
long-load-d5794d744-lgkns  0/1     Running   0          48s
```

The command does not immediately finish, but continues to show updates to the pods' status. Leave this command running in a visible window.

- 4.5. Wait for the pods to show as ready. Then, in a new terminal window, make one of the pods unhealthy for five seconds by using the /hiccup endpoint.

```
[student@workstation ~]$ oc exec \
  deploy/long-load -- curl -s localhost:3000/hiccup?time=5
no output expected
```

The pod status window shows that one of the pods is no longer ready. After five seconds, the pod is healthy again and shows as ready.

The load test window might show app is unhealthy one time before the pod is set as not ready. After the cluster determines that the pod is no longer ready, it stops sending traffic to the pod until either the pod is fixed or the liveness probe fails. Because the pod is sick only for five seconds, it is enough time for the readiness probe to fail, but not the liveness probe.



### Note

Optionally, repeat this step and observe as the temporarily sick pod's status changes.

- 4.6. Stop the load test and status commands by pressing **Ctrl+c** in their respective windows.

## 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 reliability-probes
```

# Reserve Compute Capacity for Applications

---

## Objectives

- Configure an application with resource requests so Kubernetes can make scheduling decisions.

## Kubernetes Pod Scheduling

The Red Hat OpenShift Container Platform (RHOC) pod scheduler determines the placement of new pods onto nodes in the cluster. The pod scheduler algorithm follows a three-step process:

### Filtering nodes

A pod can define a node selector that matches the labels in the cluster nodes. Only labels that match are eligible.

Additionally, the scheduler filters the list of running nodes by evaluating each node against a set of predicates. A pod can define *resource requests* for compute resources such as CPU, memory, and storage. Only nodes with enough available computer resources are eligible.

The filtering step reduces the list of eligible nodes. In some cases, the pod could run on any of the nodes. In other cases, all the nodes are filtered out, so the pod cannot be scheduled until a node with the appropriate prerequisites becomes available.

If all nodes are filtered out, then a `FailedScheduling` event is generated for the pod.

### Prioritizing the filtered list of nodes

By using multiple priority criteria, the scheduler determines a weighted score for each node. Nodes with higher scores are better candidates to run the pod.

### Selecting the best fit node

The candidate list is sorted according to these scores, and the node with the highest score is selected to host the pod. If multiple nodes have the same high score, then one node is selected in a round-robin fashion. After a host is selected, then a `Scheduled` event is generated for the pod.

The scheduler is flexible and can be customized for advanced scheduling situations. Customizing the scheduler is outside the scope of this course.

## Compute Resource Requests

For such applications that require a specific amount of compute resources, you can define a resource request in the pod definition of your application deployment. The resource requests assign hardware resources for the application deployment.

Resource requests specify the minimum required compute resources necessary to schedule a pod. The scheduler tries to find a node with enough compute resources to satisfy the pod requests.

In Kubernetes, memory resources are measured in bytes, and CPU resources are measured in CPU units. CPU units are allocated by using millicore units. A millcore is a CPU core, either virtual or physical, that is split into 1000 units. A request value of "1000 m" allocates an entire CPU core to a pod. You can also use fractional values to allocate CPU resources. For example, you can set the

CPU resource request to a `0.1` value, which represents 100 millicores (`100 m`). Likewise, a CPU resource request with a `1.0` value represents an entire CPU or 1000 millicores (`1000 m`).

You can define resource requests for each container in either a deployment or a deployment configuration resource. If resources are not defined, then the container specification shows a `resources: {}` line.

In your deployment, modify the `resources: {}` line to specify the chosen requests. The following example defines a resource request of 100 millicores (`100 m`) of CPU and one gigabyte (`1 Gi`) of memory.

```
...output omitted...
spec:
  containers:
    - image: quay.io/redhattraining/hello-world-nginx:v1.0
      name: hello-world-nginx
      resources:
        requests:
          cpu: "100m"
          memory: "1Gi"
```

If you use the `edit` command to modify a deployment or a deployment configuration, then ensure that you use the correct indentation. Indentation mistakes can result in the editor refusing to save changes. Alternatively, use the `set resources` command that the `kubectl` and `oc` commands provide, to specify resource requests. The following command sets the same requests as the preceding example:

```
[user@host ~]$ oc set resources deployment hello-world-nginx \
--requests cpu=10m,memory=1gi
```

The `set resource` command works with any resource that includes a pod template, such as the deployments and job resources.

## Inspecting Cluster Compute Resources

Cluster administrators can view the available and used compute resources of a node. For example, as a cluster administrator, you can use the `describe node` command to determine the compute resource capacity of a node. The command shows the capacity of the node and how much of that capacity is allocatable. It also displays the amount of allocated and requested resources on the node.

```
[user@host ~]$ oc describe node master01
Name:           master01
Roles:          control-plane,master,worker
...output omitted...
Capacity:
  cpu:            8
  ephemeral-storage: 125293548Ki
  hugepages-1Gi:   0
  hugepages-2Mi:   0
  memory:         20531668Ki
  pods:           250
Allocatable:
  cpu:            7500m
```

```

ephemeral-storage: 114396791822
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 19389692Ki
pods: 250
...output omitted...
Non-terminated Pods: (88 in total)
  ... Name          CPU Requests  CPU Limits  Memory Requests  Memory Limits ...
  ... --           -----       -----      -----        -----
  ... controller-... 10m (0%)    0 (0%)     20Mi (0%)    0 (0%)      ...
  ... metallb-....  50m (0%)    0 (0%)     20Mi (0%)    0 (0%)      ...
  ... metallb-....  0 (0%)      0 (0%)     0 (0%)      0 (0%)      ...
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource      Requests         Limits
-----          -----         -----
cpu            3183m (42%)    1202m (16%)
memory        12717Mi (67%)  1350Mi (7%)
...output omitted...

```

RHOCP cluster administrators can also use the `oc adm top pods` command. This command shows the compute resource usage for each pod in a project. You must include the `--namespace` or `-n` options to specify a project. Otherwise, the command returns the resource usage for pods in the currently selected project.

The following command displays the resource usage for pods in the `openshift-dns` project:

```
[user@host ~]$ oc adm top pods -n openshift-dns
NAME              CPU(cores)  MEMORY(bytes)
dns-default-5kpn5   1m          33Mi
node-resolver-6kdxp  0m          2Mi
```

Additionally, cluster administrators can use the `oc adm top node` command to view the resource usage of a cluster node. Include the node name to view the resource usage of a particular node.

```
[user@host ~]$ oc adm top node master01
NAME      CPU(cores)  CPU%    MEMORY(bytes)  MEMORY%
master01  1250m      16%    10268Mi       54%
```



## References

For more information about pod scheduling, refer to the *Controlling Pod Placement onto Nodes (Scheduling)* chapter in the Red Hat OpenShift Container Platform 4.14 *Nodes* documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#controlling-pod-placement-onto-nodes-scheduling)

## ► Guided Exercise

# Reserve Compute Capacity for Applications

Configure an application with compute resource requests that allow and prevent successful scheduling and scaling of its pods.

### Outcomes

- Observe that memory resource requests allocate cluster node memory.
- Explore how adjusting resource requests impacts the number of replicas that can be scheduled on a node.

### 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 following conditions are true:

- The `reliability-requests` project exists.
- The resource files are available in the course directory.
- The classroom registry has the `registry.ocp4.example.com:8443/redhattraining/long-load:v1` container image.

The `registry.ocp4.example.com:8443/redhattraining/long-load:v1` container image contains an application with utility endpoints. These endpoints perform such tasks as crashing the process and toggling the server's health status.

```
[student@workstation ~]$ lab start reliability-requests
```

### Instructions

- 1. As the `admin` user, deploy the `long-load` application by applying the `long-load-deploy.yaml` file in the `reliability-requests` project.
- 1.1. 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...
```

**Note**

In general, use accounts with the least required privileges to perform a task. In the classroom environment, this account is the `developer` user. However, cluster administrator privileges are required to view the cluster node metrics in this exercise.

- 1.2. View the total memory request allocation for the node.

```
[student@workstation ~]$ oc describe node master01
...output omitted...
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource Requests Limits
-----
cpu 3158m (42%) 980m (13%)
memory 12667Mi (66%) 1250Mi (6%)
...output omitted...
```

The command output shows that the pods that are currently running on the node requested a total of 12667 MiB of memory. That value might be slightly different on your system.

**Important**

Projects and objects from previous exercises can cause the memory usage from this exercise to mismatch the intended results. Delete any unrelated projects before continuing.

If you still experience issues, re-create your classroom environment and try this exercise again.

- 1.3. Select the `reliability-requests` project.

```
[student@workstation ~]$ oc project reliability-requests
Now using project "reliability-requests" on server "https://
api.ocp4.example.com:6443".
```

- 1.4. Navigate to the `~/D0180/labs/reliability-requests` directory. Create a deployment, service, and route by using the `oc apply` command and the `long-load-deploy.yaml` file.

```
[student@workstation ~]$ cd D0180/labs/reliability-requests
[student@workstation reliability-requests]$ oc apply -f long-load-deploy.yaml
deployment.apps/long-load created
service/long-load created
route.route.openshift.io/long-load created
```

- ▶ 2. Add a resource request to the pod definition and scale the deployment beyond the cluster's capacity.

- 2.1. Modify the `long-load-deploy.yaml` file by adding a resource request. The request allocates one gibibyte (1 GiB) to each of the application pods.

```
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      containers:
        - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
          resources:
            requests:
              memory: 1Gi
...output omitted...
```

- 2.2. Apply the YAML file to modify the deployment with the resource request.

```
[student@workstation reliability-requests]$ oc apply -f long-load-deploy.yaml
deployment.apps/long-load configured
service/long-load unchanged
route.route.openshift.io/long-load unchanged
```

- 2.3. Scale the deployment to have 10 replicas.

```
[student@workstation reliability-requests]$ oc scale deploy/long-load \
--replicas 10
deployment.apps/long-load scaled
```

- 2.4. Observe that the cluster cannot schedule all pods on the single node. The pods with a Pending status cannot be scheduled.

```
[student@workstation reliability-requests]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
...output omitted...
long-load-86bb4b79f8-44zwd   0/1     Pending   0          58s
...output omitted...
```

- 2.5. Retrieve the cluster event log, and observe that insufficient memory is the cause of the failed scheduling.

```
[student@workstation reliability-requests]$ oc get events \
--field-selector reason="FailedScheduling"
...output omitted... pod/long-load-86bb4b79f8-44zwd   0/1 nodes are available:
1 Insufficient memory. ...output omitted...
```

- 2.6. Alternatively, view the events for a pending pod to see the reason. In the following command, replace the pod name with one of the pending pods in your classroom.

```
[student@workstation reliability-requests]$ oc describe \
pod/long-load-86bb4b79f8-44zwd
...output omitted...
Events:
...output omitted... 0/1 nodes are available: 1 Insufficient memory. ...output
omitted...
```

2.7. Observe that the node's requested memory usage is high.

```
[student@workstation reliability-requests]$ oc describe node master01
...output omitted...
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource Requests Limits
-----
cpu 3158m (42%) 980m (13%)
memory 18811Mi (99%) 1250Mi (6%)
...output omitted...
```

The command output shows that the pods from the long-load deployment requested most of the remaining memory from the node. However, not enough memory is available to accommodate the 10 replicas.

► 3. Reduce the requested memory per pod so that the replicas can run on the node.

3.1. Manually set the resource request to 250Mi.

```
[student@workstation reliability-requests]$ oc set resources deploy/long-load \
--requests memory=250Mi
deployment.apps/long-load resource requirements updated
```

3.2. Delete the pods so that they are re-created with the new resource request.

```
[student@workstation reliability-requests]$ oc delete pod -l app=long-load
pod "long-load-557b4d94f5-29brx" deleted
...output omitted...
```

3.3. Observe that all pods can start with the lowered memory request. Within a minute, the pods are marked as Ready and in a Running state, with no pods in a Pending status.

```
[student@workstation reliability-requests]$ oc get pods
NAME READY STATUS RESTARTS AGE
long-load-557b4d94f5-68hbb 1/1 Running 0 3m14s
long-load-557b4d94f5-bfk7c 1/1 Running 0 3m21s
long-load-557b4d94f5-bnpzh 1/1 Running 0 3m21s
long-load-557b4d94f5-chtv9 1/1 Running 0 3m21s
long-load-557b4d94f5-drg2p 1/1 Running 0 3m14s
long-load-557b4d94f5-hwsz6 1/1 Running 0 3m12s
long-load-557b4d94f5-k5vqj 1/1 Running 0 3m21s
```

```
long-load-557b4d94f5-lgstq  1/1    Running  0      3m21s
long-load-557b4d94f5-r8hq4  1/1    Running  0      3m21s
long-load-557b4d94f5-xrg7c  1/1    Running  0      3m21s
```

3.4. Observe that the memory usage of the node is lower.

```
[student@workstation reliability-requests]$ oc describe node master01
...output omitted...
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource          Requests          Limits
-----
cpu                3158m (42%)       980m (13%)
memory            15167Mi (80%)     1250Mi (6%)
...output omitted...
```

3.5. Return to the /home/student/ directory.

```
[student@workstation reliability-requests]$ cd /home/student/
[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 reliability-requests
```

# Limit Compute Capacity for Applications

---

## Objectives

- Configure an application with resource limits so Kubernetes can protect other applications from it.

Memory and CPU requests that you define for containers help Red Hat OpenShift Container Platform (RHOC) to select a compute node to run your pods. However, these resource requests do not restrict the memory and CPU that the containers can use. For example, setting a memory request at 1 GiB does not prevent the container from consuming more memory.

Red Hat recommends that you set the memory and CPU requests to the peak usage of your application. In contrast, by setting lower values, you overcommit the node resources. If all the applications that are running on the node start to use resources above the values that they request, then the compute nodes might run out of memory and CPU.

In addition to requests, you can set memory and CPU *limits* to prevent your applications from consuming too many resources.

## Setting Memory Limits

A memory limit specifies the amount of memory that a container can use across all its processes.

As soon as the container reaches the limit, the compute node selects and then kills a process in the container. When that event occurs, RHOC detects that the application is not working any more, because the main container process is missing, or because the health probes report an error. RHOC then restarts the container according to the pod `restartPolicy` attribute, which defaults to `Always`.

RHOC relies on Linux kernel features to implement resource limits, and to kill processes in containers that reach their memory limits:

### Control groups (cgroups)

RHOC uses control groups to implement resource limits. Control groups are a Linux kernel mechanism for controlling and monitoring system resources, such as CPU and memory.

### Out-of-Memory killer (OOM killer)

When a container reaches its memory limit, the Linux kernel triggers the OOM killer subsystem to select and then kill a process.

You must set a memory limit when the application has a memory usage pattern that you must mitigate, such as when the application has a memory leak. A memory leak is a bug in the application, which occurs when the application uses some memory but does not free it after use. If the leak appears in an infinite service loop, then the application uses more and more memory over time, and can end up consuming all the available memory on the system. For these applications, setting a memory limit prevents them from consuming all the node's memory. The memory limit also enables OpenShift to regularly restart applications to free up their memory when they reach the limit.

To set a memory limit for the container in a pod, use the `oc set resources` command:

```
[user@host ~]$ oc set resources deployment/hello --limits memory=1Gi
```

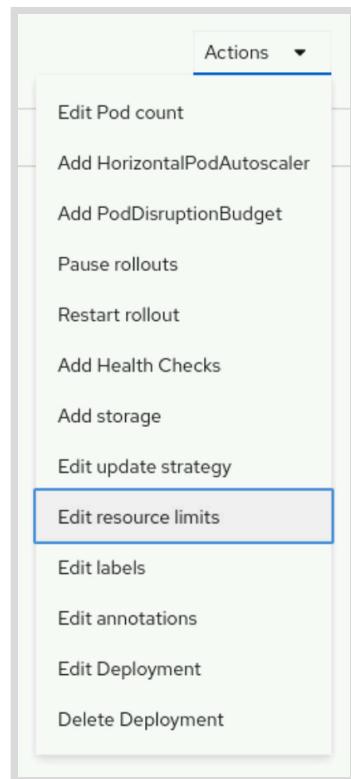
In addition to the `oc set resources` command, you can define resource limits from a file in the YAML format:

```
apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
  containers:
    - image: registry.access.redhat.com/ubi9/nginx-120:1-86
      name: hello
      resources:
        requests:
          cpu: 100m
          memory: 500Mi
        limits:
          cpu: 200m
          memory: 1Gi
```

When RHOCUP restarts a pod because of an OOM event, it updates the pod's `lastState` attribute, and sets the reason to `OOMKilled`:

```
[user@host ~]$ oc get pod hello-67645f4865-vvr42 -o yaml
...output omitted...
status:
  ...output omitted...
  containerStatuses:
    - containerID: cri-o://806b...9fe7
      image: registry.access.redhat.com/ubi9/nginx-120:1-86
      imageID: registry.access.redhat.com/ubi9/nginx-120:1-86@sha256:1403...fd34
  lastState:
    terminated:
      containerID: cri-o://bbc4...9eb2
      exitCode: 137
      finishedAt: "2023-03-08T07:56:06Z"
      reason: OOMKilled
      startedAt: "2023-03-08T07:51:43Z"
    name: hello
    ready: true
    restartCount: 1
  ...output omitted...
```

To set a memory limit for the container in a pod from the web console, select a deployment, and click **Actions > Edit resource limits**.



Set memory limits by increasing or decreasing the memory on the **Limit** section.

**Memory**

**Request**  
The minimum amount of Memory the Container is guaranteed.

**Limit**  
The maximum amount of Memory the Container is allowed to use when running.

**Cancel** **Save**

The form allows setting memory limits for a container. The "Request" section specifies the guaranteed minimum memory, and the "Limit" section specifies the maximum allowed memory. Both sections include input fields with increment (+) and decrement (-) buttons, and a unit selector (Mi). Below each section is a descriptive text explaining its purpose. At the bottom right are "Cancel" and "Save" buttons.

## Setting CPU Limits

CPU limits work differently from memory limits. When the container reaches the CPU limit, RHOCP inhibits the container's processes, even if the node has available CPU cycles. The application continues to work, but at a slower pace.

In contrast, if you do not set a CPU limit, then the container can consume as much CPU as is available on the node. If the node's CPU is under pressure, for example because several containers are running CPU-intensive tasks, then the Linux kernel shares the CPU resource between all these containers, according to the CPU requests value for the containers.

You must set a CPU limit when you require a consistent application behavior across clusters and nodes. For example, if the application runs on a node where the CPU is available, then the application can execute at full speed. On the other hand, if the application runs on a node with CPU pressure, then the application executes at a slower pace.

The same behavior can occur between your development and production clusters. Because the two environments might have different node configurations, the application might run differently when you move it from development to production.



### Note

Clusters can have differences in hardware configuration beyond what limits observe. For example, two clusters' nodes might have CPUs with equal core count and unequal clock speeds.

Requests and limits do not account for these hardware differences. If your clusters differ in such a way, take care that requests and limits are appropriate for both configurations.

By setting a CPU limit, you mitigate the differences between the configuration of the nodes, and you experience a more consistent behavior.

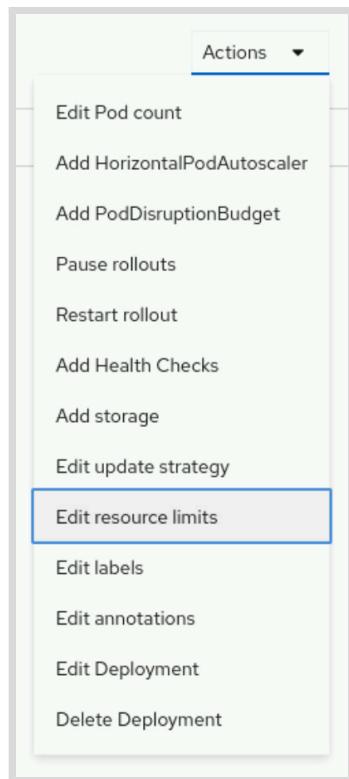
To set a CPU limit for the container in a pod, use the `oc set resources` command:

```
[user@host ~]$ oc set resources deployment/hello --limits cpu=200m
```

You can also define CPU limits from a file in the YAML format:

```
apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
  containers:
    - image: registry.access.redhat.com/ubi9/nginx-120:1-86
      name: hello
      resources:
        requests:
          cpu: 100m
          memory: 500Mi
        limits:
          cpu: 200m
          memory: 1Gi
```

To set a CPU limit for the container in a pod from the web console, select a deployment, and click **Actions > Edit resource limits**.



Set CPU limits by increasing or decreasing the CPU on the **Limit** section.

The screenshot shows the 'CPU' section of a resource configuration interface. It includes two sections: 'Request' and 'Limit'. Each section has a numeric input field with a minus sign, a value of '0', a plus sign, and a dropdown menu set to 'cores'. Below each section is a descriptive text: 'The minimum amount of CPU the Container is guaranteed.' for the Request section, and 'The maximum amount of CPU the Container is allowed to use when running.' for the Limit section.

## Viewing Requests, Limits, and Actual Usage

By using the RHOCP command-line interface, cluster administrators can view compute usage information on individual nodes. The `oc describe node` command displays detailed information about a node, including information about the pods that are running on the node. For each pod, it shows CPU requests and limits, as well as memory requests and limits. If you do not specify a request or limit, then the pod shows a zero for that column. The command also displays a summary of all the resource requests and limits.

```
[user@host ~]$ oc describe node master01
Name:           master01
Roles:          control-plane,master,worker
...output omitted...
Non-terminated Pods:              (88 in total)
  ... Name            CPU Requests  CPU Limits  Memory Requests  Memory Limits ...
  ... ----
  ... controller-...  10m (0%)    0 (0%)    20Mi (0%)    0 (0%)    ...
  ... metallb-...    50m (0%)    0 (0%)    20Mi (0%)    0 (0%)    ...
  ... metallb-...    0 (0%)     0 (0%)    0 (0%)     0 (0%)    ...
...output omitted...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource      Requests      Limits
  -----        -----
  cpu           3183m (42%)   1202m (16%)
  memory        12717Mi (67%)  1350Mi (7%)
...output omitted...
```

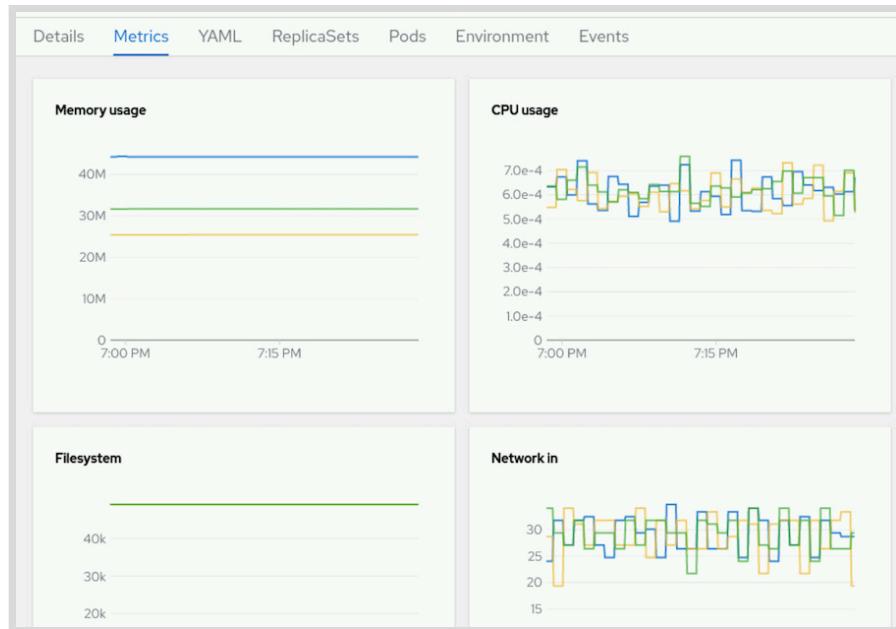
The `oc describe node` command displays requests and limits. The `oc adm top` command shows resource usage. The `oc adm top nodes` command shows the resource usage for nodes in the cluster. You must run this command as the cluster administrator.

The `oc adm top pods` command shows the resource usage for each pod in a project.

The following command displays the resource usage for the pods in the current project:

```
[user@host ~]$ oc adm top pods -n openshift-console
NAME                  CPU(cores)  MEMORY(bytes)
hello-67645f4865-vvr42  121m       309Mi
intradb-6f8d7cfffb-fz55b  0m         440Mi
```

To visualize the consumption of resources from the web console, select a deployment, and click the **Metrics** tab. From this tab, you can view the usage for memory, CPU, the file system, and incoming and outgoing traffic.



## References

cgroups(7) man page

For more information about resource limits, refer to the *Configuring Cluster Memory to Meet Container Memory and Risk Requirements* section in the *Working with Clusters* chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-cluster-resource-configure](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-cluster-resource-configure)

## ► Guided Exercise

# Limit Compute Capacity for Applications

Configure an application with compute resource limits that allow and prevent successful execution of its pods.

### Outcomes

You should be able to monitor the memory usage of an application, and set a memory limit for 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 all resources are available for this exercise. It also creates the `reliability-limits` project and the `/home/student/D0180/labs/reliability-limits/resources.txt` file. The `resources.txt` file contains some commands that you use during the exercise. You can use the file to copy and paste these commands.

```
[student@workstation ~]$ lab start reliability-limits
```

### Instructions

- 1. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Use the `reliability-limits` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `reliability-limits` project as the active project.

```
[student@workstation ~]$ oc project reliability-limits
...output omitted...
```

- 2. Create the `leakapp` deployment from the `~/D0180/labs/reliability-limits/leakapp.yml` file that the `lab` command prepared. The application has a bug, and leaks 1 MiB of memory every second.

- 2.1. Review the `~/D0180/labs/reliability-limits/leakapp.yml` resource file. The memory limit is set to 35 MiB. Do not change the file.

```
...output omitted...
resources:
requests:
  memory: 20Mi
limits:
  memory: 35Mi
```

- 2.2. Use the `oc apply` command to create the application. Ignore the warning message.

```
[student@workstation ~]$ oc apply -f \
~/DO180/labs/reliability-limits/leakapp.yml
Warning: would violate PodSecurity "restricted:v1.24":
...output omitted...
deployment.apps/leakapp created
```

- 2.3. Wait for the pod to start. You might have to rerun the command several times for the pod to report a `Running` status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
leakapp-99bb64c8d-hk26k   1/1     Running   0          12s
```

- 3. Watch the pod. OpenShift restarts the pod after 30 seconds.

- 3.1. Use the `watch` command to monitor the `oc get pods` command. Wait for OpenShift to restart the pod, and then press `Ctrl+C` to quit the `watch` command.

```
[student@workstation ~]$ watch oc get pods
Every 2.0s: oc get pods                                     workstation: Wed Mar  8 07:27:45 2023
NAME           READY   STATUS    RESTARTS   AGE
leakapp-99bb64c8d-hk26k   1/1     Running   1 (15s ago)  48s
```

- 3.2. Retrieve the container status to verify that OpenShift restarted the pod due to an Out-Of-Memory (OOM) event.

```
[student@workstation ~]$ oc get pods leakapp-99bb64c8d-hk26k \
-o jsonpath='{.status.containerStatuses[0].lastState}' | jq .
{
  "terminated": {
    "containerID": "cri-o://5800...1d04",
    "exitCode": 137,
    "finishedAt": "2023-03-08T12:29:24Z",
    "reason": "OOMKilled",
    "startedAt": "2023-03-08T12:28:53Z"
  }
}
```

- ▶ 4. Observe the pod status for a few minutes, until the `CrashLoopBackOff` status is displayed. During this period, OpenShift restarts the pod several times because of the memory leak.

Between each restart, OpenShift sets the pod status to `CrashLoopBackOff`, waits an increasing amount of time between retries, and then restarts the pod. The delay between restarts gives the operator the opportunity to fix the issue.

After various retries, OpenShift finally sets the `CrashLoopBackOff` wait timer to five minutes. During this wait time, the application is not available to your customers.

```
[student@workstation ~]$ watch oc get pods
Every 2.0s: oc get pods   workstation: Wed Mar  8 07:33:15 2023

NAME                  READY   STATUS            RESTARTS   AGE
leakapp-99bb64c8d-hk26k   0/1    CrashLoopBackoff   4 (82s ago)  5m25s
```

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

- ▶ 5. Fixing the memory leak would resolve the issue. However, it might take some time for the developers to fix the bug. In the meantime, set the memory limit to 600 MiB. With this setting, the pod can run for ten minutes before the application reaches the limit.
- 5.1. Use the `oc set resources` command to set the new limit. Ignore the warning message.

```
[student@workstation ~]$ oc set resources deployment/leakapp \
--limits memory=600Mi
Warning: would violate PodSecurity "restricted:v1.24":
...output omitted...
deployment.apps/leakapp resource requirements updated
```

- 5.2. Wait for the pod to start. You might have to rerun the command several times for the pod to report a `Running` status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS            RESTARTS   AGE
leakapp-6bc64dfcd-86fpc   1/1    Running          0           12s
```

- 5.3. Wait two minutes to verify that OpenShift no longer restarts the pod every 30 seconds.

```
[student@workstation ~]$ watch oc get pods
Every 2.0s: oc get pods   workstation: Wed Mar  8 07:38:15 2023

NAME                  READY   STATUS            RESTARTS   AGE
leakapp-6bc64dfcd-86fpc   1/1    Running          0           3m12s
```

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

- ▶ 6. Review the memory that the pod consumes. You might have to rerun the command several times for the metrics to be available. The memory usage on your system probably differs.

```
[student@workstation ~]$ oc adm top pods
NAME                  CPU(cores)   MEMORY(bytes)
leakapp-6bc64dfcd-86fpc  0m          174Mi
```

- 7. **Optional.** Wait about 10 minutes from the creation time until the application reaches the out of memory error. After this period, OpenShift restarts the pod, because it reached the 600 MiB memory limit.

- 7.1. Open a new terminal window, and then run the `watch` command to monitor the `oc adm top pods` command.

```
[student@workstation ~]$ watch oc adm top pods
Every 2.0s: oc adm top pods                         workstation: Wed Mar  8 07:38:55 2023
NAME                  CPU(cores)   MEMORY(bytes)
leakapp-6bc64dfcd-86fpc  0m          176Mi
```

Leave the command running and do not interrupt it.



### Note

You might see a message that metrics are not yet available. If so, wait some time and try again.

- 7.2. In the first terminal, run the `watch` command to monitor the `oc get pods` command. Watch the output of the `oc adm top pods` command in the second terminal. When the memory usage reaches 600 MiB, the OOM subsystem kills the process inside the container, and OpenShift restarts the pod.

```
[student@workstation ~]$ watch oc get pods
Every 2.0s: oc get pods                           workstation: Wed Mar  8 07:46:35 2023
NAME        READY   STATUS    RESTARTS   AGE
leakapp-6bc64dfcd-86fpc  1/1     Running   1 (3s ago)  9m58s
```

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

- 7.3. Press `Ctrl+C` to quit the `watch` command in the second terminal. Close this second terminal when 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 reliability-limits
```

# Application Autoscaling

## Objectives

- Configure a horizontal pod autoscaler for an application.

Kubernetes can autoscale a deployment based on current load on the application pods, by means of a `HorizontalPodAutoscaler` (HPA) resource type.

A horizontal pod autoscaler resource uses performance metrics that the OpenShift Metrics subsystem collects. The Metrics subsystem comes preinstalled in OpenShift. To autoscale a deployment, you must specify resource requests for pods so that the horizontal pod autoscaler can calculate the percentage of usage.

The autoscaler works in a loop. Every 15 seconds by default, it performs the following steps:

- The autoscaler retrieves the details of the metric for scaling from the HPA resource.
- For each pod that the HPA resource targets, the autoscaler collects the metric from the metric subsystem.
- For each targeted pod, the autoscaler computes the usage percentage, from the collected metric and from the pod resource requests.
- The autoscaler computes the average usage and the average resource requests across all the targeted pods. It establishes a usage ratio from these values, and then uses the ratio for its scaling decision.

The simplest way to create a horizontal pod autoscaler resource is by using the `oc autoscale` command, for example:

```
[user@host ~]$ oc autoscale deployment/hello --min 1 --max 10 --cpu-percent 80
```

The previous command creates a horizontal pod autoscaler resource that changes the number of replicas on the `hello` deployment to keep its pods under 80% of their total requested CPU usage.

The `oc autoscale` command creates a horizontal pod autoscaler resource by using the name of the deployment as an argument (`hello` in the previous example).

The maximum and minimum values for the horizontal pod autoscaler resource accommodate bursts of load and avoid overloading the OpenShift cluster. If the load on the application changes too quickly, then it might help to keep several spare pods to cope with sudden bursts of user requests. Conversely, too many pods can use up all cluster capacity and impact other applications that use the same OpenShift cluster.

To get information about horizontal pod autoscaler resources in the current project, use the `oc get` command. For example:

```
[user@host ~]$ oc get hpa
NAME      REFERENCE          TARGETS      MINPODS   MAXPODS   REPLICAS   ...
hello     Deployment/hello  <unknown>/80%    1          10         1           ...
scale     Deployment/scale   60%/80%       2          10         2           ...
```



### Important

The horizontal pod autoscaler initially has a value of <unknown> in the TARGETS column. It might take up to five minutes before <unknown> changes to display a percentage for current usage.

A persistent value of <unknown> in the TARGETS column might indicate that the deployment does not define resource requests for the metric. The horizontal pod autoscaler does not scale these pods.

Pods that are created by using the `oc create deployment` command do not define resource requests. Using the OpenShift autoscaler might therefore require editing the deployment resources, creating custom YAML or JSON resource files for your application, or adding limit range resources to your project that define default resource requests.

In addition to the `oc autoscale` command, you can create a horizontal pod autoscaler resource from a file in the YAML format.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hello
spec:
  minReplicas: 1 ①
  maxReplicas: 10 ②
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 80 ③
        type: Utilization
      type: Resource
  scaleTargetRef: ④
    apiVersion: apps/v1
    kind: Deployment
    name: hello
```

- ① Minimum number of pods.
- ② Maximum number of pods.
- ③ Ideal average CPU usage for each pod. If the global average CPU usage is above that value, then the horizontal pod autoscaler starts new pods. If the global average CPU usage is below that value, then the horizontal pod autoscaler deletes pods.
- ④ Reference to the name of the deployment resource.

Use the `oc apply -f hello-hpa.yaml` command to create the resource from the file.

The preceding example creates a horizontal pod autoscaler resource that scales based on CPU usage. Alternatively, it can scale based on memory usage by setting the resource name to `memory`, as in the following example:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hello
spec:
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - resource:
        name: memory
        target:
          averageUtilization: 80
...output omitted...
```

To create a horizontal pod autoscaler resource from the web console, click the **Workloads > HorizontalPodAutoscalers** menu. Click **Create HorizontalPodAutoscaler** and customize the YAML manifest.



### Note

If an application uses more overall memory as the number of replicas increases, then it cannot be used with memory-based autoscaling.



### References

For more information, refer to the *Automatically Scaling Pods with the Horizontal Pod Autoscaler* section in the *Working with Pods* chapter in the Red Hat OpenShift Container Platform 4.14 Nodes documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-pods-autoscaling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-pods-autoscaling)

## ► Guided Exercise

# Application Autoscaling

Configure an autoscaler for an application and then load test that application to observe scaling up.

### Outcomes

You should be able to manually scale up a deployment, configure a horizontal pod autoscaler resource, and monitor the autoscaler.

### 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. It also creates the `reliability-autoscaling` project.

```
[student@workstation ~]$ lab start reliability-autoscaling
```

### Instructions

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

Use the `reliability-autoscaling` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `reliability-autoscaling` project as the active project.

```
[student@workstation ~]$ oc project reliability-autoscaling
...output omitted...
```

- 2. Create the `loadtest` deployment, service, and route. The deployment uses the `registry.ocp4.example.com:8443/redhat-training/loadtest:v1.0` container

image that provides a web application. The web application exposes an API endpoint that creates a CPU-intensive task when queried.

- 2.1. Review the `~/DO180/labs/reliability-autoscaling/loadtest.yml` resource file that the `lab` command prepared. The container specification does not include the `resources` section that you use to specify CPU requests and limits. You configure that section in another step. Do not change the file for now.

```

apiVersion: v1
kind: List
metadata: {}
items:
  - apiVersion: apps/v1
    kind: Deployment
    ...output omitted...
    spec:
      containers:
        - image: registry.ocp4.example.com:8443/redhattraining/loadtest:v1.0
          name: loadtest
          readinessProbe:
            failureThreshold: 3
            httpGet:
              path: /api/loadtest/v1/healthz
              port: 8080
              scheme: HTTP
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 1

        - apiVersion: v1
          kind: Service
        ...output omitted...

      - apiVersion: route.openshift.io/v1
        kind: Route
      ...output omitted...

```

2.2. Use the `oc apply` command to create the application.

```

[student@workstation ~]$ oc apply -f \
~/DO180/labs/reliability-autoscaling/loadtest.yml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created

```

2.3. Wait for the pod to start. You might have to rerun the command several times for the pod to report a `Running` status. The name of the pod on your system probably differs.

```

[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
loadtest-65c55b7dc-r4s4s   1/1     Running   0          49s

```

- 3. Configure a horizontal pod autoscaler resource for the `loadtest` deployment. Set the minimum number of replicas to 2 and the maximum to 20. Set the average CPU usage to 50% of the CPU requests attribute.

The horizontal pod autoscaler does not work, because the `loadtest` deployment does not specify `requests` for CPU usage.

3.1. Use the `oc autoscale` command to create the horizontal pod autoscaler resource.

```
[student@workstation ~]$ oc autoscale deployment/loadtest --min 2 --max 20 \
--cpu-percent 50
horizontalpodautoscaler.autoscaling/loadtest autoscaled
```

- 3.2. Retrieve the status of the `loadtest` horizontal pod autoscaler resource. The `unknown` value in the `TARGETS` column indicates that OpenShift cannot compute the current CPU usage of the `loadtest` deployment. The deployment must include the `CPU requests` attribute for OpenShift to be able to compute the CPU usage.

```
[student@workstation ~]$ oc get hpa loadtest
NAME      REFERENCE      TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
loadtest  Deployment/loadtest  <unknown>/50%    2          20        2          74s
```

- 3.3. Get more details about the resource status. You might have to rerun the command several times. Wait three minutes for the command to report the warning message.

```
[student@workstation ~]$ oc describe hpa loadtest
Name:                                loadtest
Namespace:                            reliability-autoscaling
...
Conditions:
  Type        Status  Reason           Message
  ----        -----  -----           -----
  AbleToScale  True    SucceededGetScale  the HPA controller was able to
  get the target's current scale
  ScalingActive False   FailedGetResourceMetric  the HPA was unable to compute
  the replica count: failed to get cpu utilization: missing request for cpu
Events:
  Type    ... Message
  ----    ...
...
Warning ... failed to get cpu utilization: missing request for cpu
...

```

- 3.4. Delete the horizontal pod autoscaler resource. You re-create the resource in another step, after you fix the `loadtest` deployment.

```
[student@workstation ~]$ oc delete hpa loadtest
horizontalpodautoscaler.autoscaling "loadtest" deleted
```

- 3.5. Delete the `loadtest` application.

```
[student@workstation ~]$ oc delete -f \
~/D0180/labs/reliability-autoscaling/loadtest.yml
deployment.apps "loadtest" deleted
service "loadtest" deleted
route.route.openshift.io "loadtest" deleted
```

- 4. Add a CPU resource section to the `~/D0180/labs/reliability-autoscaling/loadtest.yml` file. Redeploy the application from the file.

- 4.1. Edit the `~/D0180/labs/reliability-autoscaling/loadtest.yml` file, and configure the CPU limits and requests for the `loadtest` deployment. The pods need 25 millicores to operate, and must not consume more than 100 millicores.

You can compare your work with the completed `~/D0180/solutions/reliability-autoscaling/loadtest.yml` file that the `lab` command prepared.

```
...output omitted...
spec:
  containers:
    - image: registry.ocp4.example.com:8443/redhattraining/loadtest:v1.0
      name: loadtest
      readinessProbe:
        failureThreshold: 3
        httpGet:
          path: /api/loadtest/v1/healthz
          port: 8080
          scheme: HTTP
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 1
      resources:
        requests:
          cpu: 25m
        limits:
          cpu: 100m
...output omitted...
```

- 4.2. Use the `oc apply` command to deploy the application from the file.

```
[student@workstation ~]$ oc apply -f \
~/D0180/labs/reliability-autoscaling/loadtest.yml
deployment.apps/loadtest created
service/loadtest created
route.route.openshift.io/loadtest created
```

- 4.3. Wait for the pod to start. You might have to rerun the command several times for the pod to report a `Running` status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
loadtest-667bcdcdc99-vhc9x  1/1     Running   0          36s
```

- 5. Manually scale the `loadtest` deployment by first increasing and then decreasing the number of running pods.

- 5.1. Scale up the `loadtest` deployment to five pods.

```
[student@workstation ~]$ oc scale deployment/loadtest --replicas 5
deployment.apps/loadtest scaled
```

- 5.2. Confirm that all five application pods are running. You might have to rerun the command several times for all the pods to report a **Running** status. The name of the pods on your system probably differ.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
loadtest-667bdcdc99-5fcvh  1/1     Running   0          43s
loadtest-667bdcdc99-dpspr  1/1     Running   0          42s
loadtest-667bdcdc99-hkssk  1/1     Running   0          43s
loadtest-667bdcdc99-vhc9x  1/1     Running   0          8m11s
loadtest-667bdcdc99-z5n9q  1/1     Running   0          43s
```

- 5.3. Scale down the **loadtest** deployment back to one pod.

```
[student@workstation ~]$ oc scale deployment/loadtest --replicas 1
deployment.apps/loadtest scaled
```

- 5.4. Confirm that only one application pod is running. You might have to rerun the command several times for the pods to terminate.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
loadtest-667bdcdc99-vhc9x  1/1     Running   0          11m
```

- 6. Configure a horizontal pod autoscaler resource for the **loadtest** deployment. Set the minimum number of replicas to 2 and the maximum to 20. Set the average CPU usage to 50% of the CPU request attribute.

- 6.1. Use the **oc autoscale** command to create the horizontal pod autoscaler resource.

```
[student@workstation ~]$ oc autoscale deployment/loadtest --min 2 --max 20 \
--cpu-percent 50
horizontalpodautoscaler.autoscaling/loadtest autoscaled
```

- 6.2. Open a new terminal window and run the **watch** command to monitor the **oc get hpa loadtest** command. Wait five minutes for the **loadtest** horizontal pod autoscaler to report usage in the **TARGETS** column.

Notice that the horizontal pod autoscaler scales up the deployment to two replicas, to conform with the minimum number of pods that you configured.

```
[student@workstation ~]$ watch oc get hpa loadtest
Every 2.0s: oc get hpa loadtest                         workstation: Fri Mar  3 06:26:24 2023
NAME      REFERENCE           TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
loadtest  Deployment/loadtest  0%/50%       2          20         2          52s
```

Leave the command running, and do not interrupt it.

- 7. Increase the CPU usage by sending requests to the **loadtest** application API.

- 7.1. Use the **oc get route** command to retrieve the URL of the application.

```
[student@workstation ~]$ oc get route loadtest
NAME      HOST/PORT
loadtest  loadtest-reliability-autoscaling.apps.ocp4.example.com  ...
...
```

- 7.2. Send a request to the application API to simulate additional CPU pressure on the container. Do not wait for the `curl` command to complete, and continue with the exercise. After a minute, the command reports a timeout error that you can ignore.

```
[student@workstation ~]$ curl \
loadtest-reliability-autoscaling.apps.ocp4.example.com/api/loadtest/v1/cpu/1
<html><body><h1>504 Gateway Time-out</h1>
The server didn't respond in time.
</body></html>
```

- 7.3. Watch the output of the `oc get hpa loadtest` command in the second terminal. After a minute, the horizontal pod autoscaler detects an increase in the CPU usage and deploys additional pods.



### Note

The increased activity of the application does not immediately trigger the autoscaler. Wait a few moments if you do not see any changes to the number of replicas.

You might need to run the `curl` command multiple times before the application uses enough CPU to trigger the autoscaler.

The CPU usage and the number of replicas on your system probably differ.

Every 2.0s: oc get hpa loadtest				workstation: Fri Mar 3 07:20:19 2023			
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE	
loadtest	Deployment/loadtest	220%/50%	2	20	9	16m	

- 7.4. Wait five minutes after the `curl` command completes. The `oc get hpa loadtest` command shows that the CPU load decreases.



### Note

Although the horizontal pod autoscaler resource can be quick to scale up, it is slower to scale down.

Every 2.0s: oc get hpa loadtest				workstation: Fri Mar 3 07:23:11 2023			
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE	
loadtest	Deployment/loadtest	0%/50%	2	20	9	18m	

- 7.5. **Optional:** Wait for the `loadtest` application to scale down. It takes five additional minutes for the horizontal pod autoscaler to scale down to two replicas.

```
Every 2.0s: oc get hpa loadtest           workstation: Fri Mar  3 07:29:12 2023
NAME      REFERENCE      TARGETS      MINPODS      MAXPODS      REPLICAS      AGE
loadtest  Deployment/loadtest  0%/50%        2            20          2            24m
```

7.6. Press **Ctrl+C** to quit the **watch** command. Close that second terminal when 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 reliability-autoscaling
```

## ► Lab

# Configure Applications for Reliability

Deploy and troubleshoot a reliable application that defines health probes, compute resource requests, and compute resource limits so it can run N instances per node; and configure a horizontal pod autoscaler that will scale to a maximum of N instances.

## Outcomes

You should be able to add resource requests to a `Deployment` object, configure probes, and create a horizontal pod autoscaler resource.

## 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. It also creates the `reliability-review` project and deploys the `longload` application in that project.

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

## Instructions

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your `workstation` machine.

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

Use the `reliability-review` project for your work.

1. The `longload` application in the `reliability-review` project fails to start. Diagnose and then fix the issue. The application needs 512 MiB of memory to work.

After you fix the issue, you can confirm that the application works by running the `~/D0180/labs/reliability-review/curl_loop.sh` script that the `lab` command prepared. The script sends requests to the application in a loop. For each request, the script displays the pod name and the application status. Press `Ctrl+C` to quit the script.

2. When the application scales up, your customers complain that some requests fail. To replicate the issue, manually scale up the `longload` application to three replicas, and run the `~/D0180/labs/reliability-review/curl_loop.sh` script at the same time.

The application takes seven seconds to initialize. The application exposes the `/health` API endpoint on HTTP port 3000. Configure the `longload` deployment to use this endpoint, to ensure that the application is ready before serving client requests.

3. Configure the application so that it automatically scales up when the average memory usage is above 60% of the memory requests value, and scales down when the usage is below this percentage. The minimum number of replicas must be one, and the maximum must be three. The resource that you create for scaling the application must be named `longload`.

The `lab` command provides the `~/D0180/labs/reliability-review/hpa.yml` resource file as an example. Use the `oc explain` command to learn the valid parameters for

the `hpa.spec.metrics.resource.target` attribute. Because the file is incomplete, you must update it first if you choose to use it.

To test your work, use the `oc exec deploy/longload –curl localhost:3000/leak` command to sends an HTTP request to the application /leak API endpoint. Each request consumes an additional 480 MiB of memory. To free this memory, you can use the `~/DO180/labs/reliability-review/free.sh` script.

## 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 reliability-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 reliability-review
```

## ► Solution

# Configure Applications for Reliability

Deploy and troubleshoot a reliable application that defines health probes, compute resource requests, and compute resource limits so it can run N instances per node; and configure a horizontal pod autoscaler that will scale to a maximum of N instances.

## Outcomes

You should be able to add resource requests to a `Deployment` object, configure probes, and create a horizontal pod autoscaler resource.

## 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. It also creates the `reliability-review` project and deploys the `longload` application in that project.

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

## Instructions

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your `workstation` machine.

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

Use the `reliability-review` project for your work.

1. The `longload` application in the `reliability-review` project fails to start. Diagnose and then fix the issue. The application needs 512 MiB of memory to work.

After you fix the issue, you can confirm that the application works by running the `~/D0180/labs/reliability-review/curl_loop.sh` script that the `lab` command prepared. The script sends requests to the application in a loop. For each request, the script displays the pod name and the application status. Press `Ctrl+C` to quit the script.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `reliability-review` project as the active project.

```
[student@workstation ~]$ oc project reliability-review
...output omitted...
```

- 1.3. List the pods in the project. The pod is in the Pending status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
longload-64bf8dd776-b6rkz   0/1     Pending   0          8m1s
```

- 1.4. Retrieve the events for the pod. No compute node has enough memory to accommodate the pod.

```
[student@workstation ~]$ oc describe pod longload-64bf8dd776-b6rkz
Name:           longload-64bf8dd776-b6rkz
Namespace:      reliability-review
...output omitted...
Events:
  Type     Reason          Age   From            Message
  ----   -----          ----  ----
  Warning FailedScheduling 8m    default-scheduler 0/1 nodes are available:
  1 Insufficient memory.  preemption: 0/1 nodes are available: 1 No preemption
  victims found for incoming pod.
```

- 1.5. Review the resource requests for memory. The longload deployment requests 8 GiB of memory.

```
[student@workstation ~]$ oc get deployment longload -o \
  jsonpath='{.spec.template.spec.containers[0].resources.requests.memory}{"\n"}'
8Gi
```

- 1.6. Set the memory requests to 512 MiB. Ignore the warning message.

```
[student@workstation ~]$ oc set resources deployment/longload \
  --requests memory=512Mi
deployment.apps/longload resource requirements updated
```

- 1.7. Wait for the pod to start. You might have to rerun the command several times for the pod to report a Running status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
longload-5897c9558f-cx4gt   1/1     Running   0          86s
```

- 1.8. Run the ~/D0180/labs/reliability-review/curl\_loop.sh script to confirm that the application works.

```
[student@workstation ~]$ ~/D0180/labs/reliability-review/curl_loop.sh
1 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection refused
2 longload-5897c9558f-cx4gt: app is still starting
3 longload-5897c9558f-cx4gt: app is still starting
4 longload-5897c9558f-cx4gt: app is still starting
5 longload-5897c9558f-cx4gt: Ok
6 longload-5897c9558f-cx4gt: Ok
```

```
7 longload-5897c9558f-cx4gt: ok
8 longload-5897c9558f-cx4gt: ok
...output omitted...
```

Press **Ctrl+C** to quit the script.

2. When the application scales up, your customers complain that some requests fail. To replicate the issue, manually scale up the `longload` application to three replicas, and run the `~/D0180/labs/reliability-review/curl_loop.sh` script at the same time.

The application takes seven seconds to initialize. The application exposes the `/health` API endpoint on HTTP port 3000. Configure the `longload` deployment to use this endpoint, to ensure that the application is ready before serving client requests.

- 2.1. Open a new terminal window and run the `~/D0180/labs/reliability-review/curl_loop.sh` script.

```
[student@workstation ~]$ ~/D0180/labs/reliability-review/curl_loop.sh
1 longload-5897c9558f-cx4gt: ok
2 longload-5897c9558f-cx4gt: ok
3 longload-5897c9558f-cx4gt: ok
4 longload-5897c9558f-cx4gt: ok
...output omitted...
```

Leave the script running and do not interrupt it.

- 2.2. Scale up the application to three replicas.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 3
deployment.apps/longload scaled
```

- 2.3. Watch the output of the `curl_loop.sh` script in the second terminal. Some requests fail because OpenShift sends requests to the new pods before the application is ready.

```
...output omitted...
22 longload-5897c9558f-cx4gt: ok
23 longload-5897c9558f-cx4gt: ok
24 longload-5897c9558f-cx4gt: ok
25 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection refused
26 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection refused
27 longload-5897c9558f-cx4gt: ok
28 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection refused
29 longload-5897c9558f-cx4gt: ok
30 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection refused
31 longload-5897c9558f-tpssf: app is still starting
32 longload-5897c9558f-kkvm5: app is still starting
33 longload-5897c9558f-cx4gt: ok
34 longload-5897c9558f-tpssf: app is still starting
35 longload-5897c9558f-tpssf: app is still starting
36 longload-5897c9558f-tpssf: app is still starting
37 longload-5897c9558f-cx4gt: ok
```

```
38 longload-5897c9558f-tpssf: app is still starting
39 longload-5897c9558f-cx4gt: Ok
40 longload-5897c9558f-cx4gt: Ok
...output omitted...
```

Leave the script running and do not interrupt it.

- 2.4. Add a readiness probe to the `longload` deployment. Ignore the warning message.

```
[student@workstation ~]$ oc set probe deployment/longload --readiness \
--initial-delay-seconds 7 \
--get-url http://:3000/health
deployment.apps/longload probes updated
```

- 2.5. Scale down the application back to one pod.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 1
deployment.apps/longload scaled
```

If scaling down breaks the `curl_loop.sh` script, then press `Ctrl+C` to stop the script in the second terminal. Then, restart the script.

- 2.6. To test your work, scale up the application to three replicas again.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 3
deployment.apps/longload scaled
```

- 2.7. Watch the output of the `curl_loop.sh` script in the second terminal. No request fails.

```
...output omitted...
92 longload-7ddcc9b7fd-72dtm: Ok
93 longload-7ddcc9b7fd-72dtm: Ok
94 longload-7ddcc9b7fd-72dtm: Ok
95 longload-7ddcc9b7fd-qln95: Ok
96 longload-7ddcc9b7fd-wrxrb: Ok
97 longload-7ddcc9b7fd-qln95: Ok
98 longload-7ddcc9b7fd-wrxrb: Ok
99 longload-7ddcc9b7fd-72dtm: Ok
...output omitted...
```

Press `Ctrl+C` to quit the script.

3. Configure the application so that it automatically scales up when the average memory usage is above 60% of the memory requests value, and scales down when the usage is below this percentage. The minimum number of replicas must be one, and the maximum must be three. The resource that you create for scaling the application must be named `longload`.

The `lab` command provides the `~/D0180/labs/reliability-review/hpa.yml` resource file as an example. Use the `oc explain` command to learn the valid parameters for the `hpa.spec.metrics.resource.target` attribute. Because the file is incomplete, you must update it first if you choose to use it.

To test your work, use the `oc exec deploy/longload-curl localhost:3000/leak` command to send an HTTP request to the application / leak API endpoint. Each request consumes an additional 480 MiB of memory. To free this memory, you can use the `~/D0180/labs/reliability-review/free.sh` script.

- 3.1. Before you create the horizontal pod autoscaler resource, scale down the application to one pod.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 1
deployment.apps/longload scaled
```

- 3.2. Edit the ~/DO180/labs/reliability-review/hpa.yaml resource file. You can retrieve the parameters for the `resource` attribute by using the `oc explain hpa.spec.metrics.resource` and `oc explain hpa.spec.metrics.resource.target` commands.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: longload
  labels:
    app: longload
spec:
  maxReplicas: 3
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: longload
  metrics:
    - type: Resource
      resource:
        name: memory
      target:
        type: Utilization
        averageUtilization: 60
```

- 3.3. Use the `oc apply` command to deploy the horizontal pod autoscaler.

```
[student@workstation ~]$ oc apply -f ~/DO180/labs/reliability-review/hpa.yaml
horizontalpodautoscaler.autoscaling/longload created
```

- 3.4. In the second terminal, run the `watch` command to monitor the `oc get hpa longload` command. Wait for the `longload` horizontal pod autoscaler to report usage in the `TARGETS` column. The percentage on your system probably differs.

```
[student@workstation ~]$ watch oc get hpa longload
Every 2.0s: oc get hpa longload                                     workstation: Fri Mar 10 05:15:34 2023

NAME      REFERENCE          TARGETS      MINPODS     MAXPODS     REPLICAS     AGE
longload   Deployment/longload   13%/60%     1           3           1           75s
```

Leave the command running and do not interrupt it.

- 3.5. To test your work, run the `oc exec deploy/longload –curl localhost:3000/leak` command in the first terminal for the application to allocate 480 MiB of memory.

```
[student@workstation ~]$ oc exec deploy/longload -- curl -s localhost:3000/leak  
longload-7ddcc9b7fd-72dtm: consuming memory!
```

- 3.6. In the second terminal, after two minutes, the `oc get hpa longload` command shows the memory increase. The horizontal pod autoscaler scales up the application to more than one replica. The percentage on your system probably differs.

```
Every 2.0s: oc get hpa longload          workstation: Fri Mar 10 05:19:44 2023  
  
NAME      REFERENCE          TARGETS      MINPODS      MAXPODS      REPLICAS      AGE  
longload  Deployment/longload  145%/60%    1            3            2            5m18s
```

- 3.7. To test your work, run the `~/D0180/labs/reliability-review/free.sh` script in the first terminal for the application to release the memory. Ensure that the pod that frees the memory is the same pod that was consuming memory. Execute the `free.sh` script several times if necessary.

```
[student@workstation ~]$ ~/D0180/labs/reliability-review/free.sh  
longload-7ddcc9b7fd-72dtm: releasing memory!
```

- 3.8. In the second terminal, after ten minutes, the `oc get hpa longload` command shows the memory decrease. The horizontal pod autoscaler scales down the application to one replica. The percentage on your system probably differs.

```
Every 2.0s: oc get hpa longload          workstation: Fri Mar 10 05:19:44 2023  
  
NAME      REFERENCE          TARGETS      MINPODS      MAXPODS      REPLICAS      AGE  
longload  Deployment/longload  12%/60%     1            3            1            15m28s
```

Press `Ctrl+C` to quit the `watch` command. Close that second terminal when done.

## 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 reliability-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 reliability-review
```

## ► Quiz

# Configure Applications for Reliability

Choose the correct answers to the following questions:

- ▶ 1. **In the previous lab, given your node capacity and the application's initial memory request, how many pods can OpenShift schedule?**
  - a. The request is set to 8 GiB; therefore, no pod can be scheduled.
  - b. The request is set to 8 GiB; therefore, only one pod can be scheduled.
  - c. The request is set to 512 MiB; therefore, no pod can be scheduled.
  - d. The request is set to 512 MiB; therefore, three pods can be scheduled.
  
- ▶ 2. **In the previous lab, which deployment setting is incorrect and prevents the application from starting?**
  - a. The readiness probe is misconfigured and always fails.
  - b. The liveness probe is misconfigured and always fails.
  - c. The memory request is too high. No compute nodes have enough memory to accommodate the application.
  - d. The memory limit is too low. The OOM kernel subsystem keeps killing the container processes.
  
- ▶ 3. **In the previous lab, which setting is initially absent from the deployment, but does not prevent the application from running?**
  - a. The liveness probe is not defined.
  - b. The readiness probe is not defined.
  - c. The memory limit is not set.
  - d. The CPU limit is not set.

## ► Solution

# Configure Applications for Reliability

Choose the correct answers to the following questions:

► 1. **In the previous lab, given your node capacity and the application's initial memory request, how many pods can OpenShift schedule?**

- a. The request is set to 8 GiB; therefore, no pod can be scheduled.
- b. The request is set to 8 GiB; therefore, only one pod can be scheduled.
- c. The request is set to 512 MiB; therefore, no pod can be scheduled.
- d. The request is set to 512 MiB; therefore, three pods can be scheduled.

► 2. **In the previous lab, which deployment setting is incorrect and prevents the application from starting?**

- a. The readiness probe is misconfigured and always fails.
- b. The liveness probe is misconfigured and always fails.
- c. The memory request is too high. No compute nodes have enough memory to accommodate the application.
- d. The memory limit is too low. The OOM kernel subsystem keeps killing the container processes.

► 3. **In the previous lab, which setting is initially absent from the deployment, but does not prevent the application from running?**

- a. The liveness probe is not defined.
- b. The readiness probe is not defined.
- c. The memory limit is not set.
- d. The CPU limit is not set.

# Summary

---

- A highly available application is resistant to scenarios that might otherwise make it unavailable.
- Kubernetes and RHOCUP provide HA features, such as health probes, that help the cluster to route traffic only to working pods.
- Resource requests and limits help to keep cluster node resource usage balanced.
- Horizontal pod autoscalers automatically add or remove replicas based on current resource usage and specified parameters.



# Manage Application Updates

### Goal

Manage reproducible application updates and rollbacks of code and configurations.

### Objectives

- Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes.
- Update applications with minimal downtime by using deployment strategies.
- Ensure reproducibility of application deployments by using image streams and short image names.
- Ensure automatic update of application pods by using image streams with Kubernetes workload resources.

### Sections

- Container Image Identity and Tags (and Guided Exercise)
- Update Application Image and Settings (and Guided Exercise)
- Reproducible Deployments with OpenShift Image Streams (and Guided Exercise)
- Automatic Image Updates with OpenShift Image Change Triggers (and Guided Exercise)

### Lab

- Manage Application Updates

# Container Image Identity and Tags

---

## Objectives

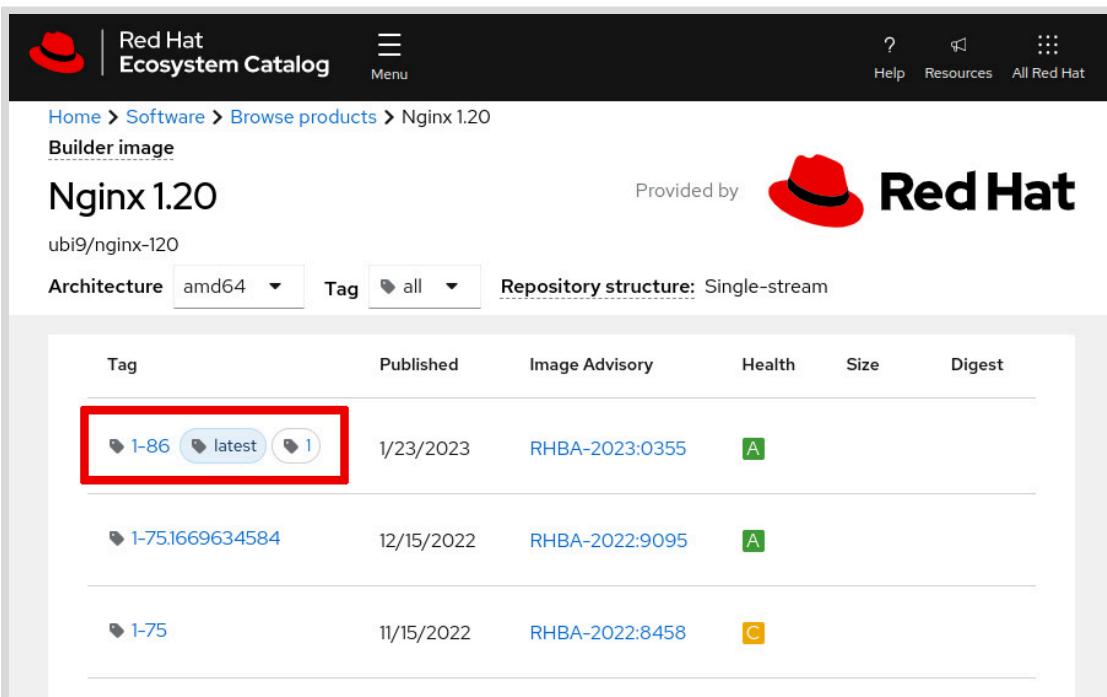
- Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes.

## Kubernetes Image Tags

The full name of a container image is composed of several parts. For example, you can decompose the `registry.access.redhat.com/ubi9/nginx-120:1-86` image name into the following elements:

- The registry server is `registry.access.redhat.com`.
- The namespace is `ubi9`.
- The name is `nginx-120`. In this example, the name of the image includes the version of the software, Nginx version 1.20.
- The tag, which points to a specific version of the image, is `1-86`. If you omit the tag, then most container tools use the `latest` tag by default.

Multiple tags can refer to the same image version. The following screen capture of the Red Hat Ecosystem Catalog at <https://catalog.redhat.com/software/containers/explore> lists the tags for the `ubi9/nginx-120` image:



The screenshot shows the Red Hat Ecosystem Catalog interface. At the top, there's a navigation bar with a Red Hat logo, the text "Red Hat Ecosystem Catalog", a "Menu" icon, and links for "Help", "Resources", and "All Red Hat". Below the navigation, the URL "Home > Software > Browse products > Nginx 1.20" is displayed, along with a link to "Builder image". The main content area shows "Nginx 1.20" provided by Red Hat. It includes filters for "Architecture" (amd64), "Tag" (all), and "Repository structure" (Single-stream). A table lists three tags: "1-86" (highlighted with a red box), "1-75.1669634584", and "1-75". Each row includes columns for Tag, Published date, Image Advisory, Health, Size, and Digest.

Tag	Published	Image Advisory	Health	Size	Digest
<a href="#">1-86</a> <a href="#">latest</a> <a href="#">1</a>	1/23/2023	RHBA-2023:0355	A		
<a href="#">1-75.1669634584</a>	12/15/2022	RHBA-2022:9095	A		
<a href="#">1-75</a>	11/15/2022	RHBA-2022:8458	C		

In this case, the `1.86`, `latest`, and `1` tags point to the same image version. You can use any of these tags to refer to that version.

The `latest` and `1` tags are *floating tags*, because they can point to different image versions over time. For example, when developers publish a new version of the image, they change the `latest` tag to point to that new version. They also update the `1` tag to point to the latest release of that version, such as `1-87` or `1-88`.

As a user of the image, by specifying a floating tag, you ensure that you always consume the up-to-date image version that corresponds to the tag.

## Floating Tag Issues

Vendors, organizations, and developers who publish images manage their tags and establish their own lifecycle for floating tags. They can reassign a floating tag to a new image version without notice.

As a user of the image, you might not notice that the tag that you were using now points to a different image version.

Suppose that you deploy an application on OpenShift and use the `latest` tag for the image. The following series of events might occur:

1. When OpenShift deploys the container, it pulls the image with the `latest` tag from the container registry.
2. Later, the image developer pushes a new version of the image, and reassigns the `latest` tag to that new version.
3. OpenShift relocates the pod to a different cluster node, for example because the original node fails.
4. On that new node, OpenShift pulls the image with the `latest` tag, and thereby retrieves the new image version.
5. Now the OpenShift deployment runs with a new version of the application, without your awareness of that version update.

A similar issue is that when you scale up your deployment, OpenShift starts new pods. On the nodes, OpenShift pulls the `latest` image version for these new pods. As a result, if a new version is available, then your deployment runs with containers that use different versions of the image. Application inconsistencies and unexpected behavior might occur.

To prevent these issues, select an image that is guaranteed not to change over time. You thus gain control over the lifecycle of your application: you can choose when and how OpenShift deploys a new image version.

You can select a static image version in several ways:

- Use a tag that does not change, instead of relying on floating tags.
- Use OpenShift image streams for tight control over the image versions. Another section in this course discusses image streams further.
- Use the *SHA (Secure Hash Algorithm)* image ID instead of a tag when referencing an image version.

The distinction between a floating and non-floating tag is not a technical one, but a convention. Although it is discouraged, there is no mechanism to prevent a developer from pushing a different image to an existing tag. Thus, you must specify the SHA image ID to guarantee that the referenced container image does not change.

## Using SHA Image ID

Developers assign tags to images. In contrast, an SHA image ID, or *digest*, is a unique identifier that the container registry computes and assigns to images. The SHA ID is an immutable string that refers to a specific image version. Using the SHA ID for identifying an image is the most secure approach.

To refer to an image by its SHA ID, replace *name : tag* with *name@SHA-ID* in the image name. The following example uses the SHA image ID instead of a tag.

```
registry.access.redhat.com/ubi9/nginx-120@sha256:1be2006abd21735e7684eb4cc6eb62...
```

To retrieve the SHA image ID from the tag, use the `oc image info` command.



### Note

A multi-architecture image references images for several CPU architectures. Multi-architecture images include an index that points to the images for different platforms and CPU architectures.

For these images, the `oc image info` command requires you to select an architecture by using the `--filter-by-os` option:

```
[user@host ~]$ oc image info registry.access.redhat.com/ubi9/nginx-120:1-86
error: the image is a manifest list and contains multiple images - use --
filter-by-os to select from:
```

OS	DIGEST
linux/amd64	sha256:1be2006abd21735e7684eb4cc6eb6295346a89411a187e37cd4...
linux/arm64	sha256:d765193e823bb89b878d2d2cb8be0e0073839a6c19073a21485...
linux/ppc64le	sha256:0dd0036620f525b3ba9a46f9f1c52ac70414f939446b2ba3a07...
linux/s390x	sha256:d8d95cc17764b82b19977bc7ef2f60ff56a3944b3c7c14071dd...

The following example displays the SHA ID for the image that the 1-86 tag currently points to.

```
[user@host ~]$ oc image info --filter-by-os linux/amd64 \
registry.access.redhat.com/ubi9/nginx-120:1-86
Name:      registry.access.redhat.com/ubi9/nginx-120:1-86
Digest:    sha256:1be2006abd21735e7684eb4cc6eb6295346a89411a187e37cd4a3aa2f1bd13a5
Manifest List: sha256:5bc635dc946fedb4ba391470e8f84f9860e06a1709e30206a95ed9955...
Media Type: application/vnd.docker.distribution.manifest.v2+json
...output omitted...
```

You can also use the `skopeo inspect` command. The output format differs from the `oc image info` command, although both commands report similar data.

If you use the `oc debug node/node-name` command to connect to a compute node, then you can list the locally available images by running the `crlctl images --digests --no-trunc` command. The `--digests` option instructs the command to display the SHA image IDs, and the `--no-trunc` option instructs the command to display the full SHA string; otherwise, the command displays only the first characters.

```
[user@host ~]$ oc debug node/node-name
Temporary namespace openshift-debug-csn2p is created for debugging node...
Starting pod/node-name-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-4.4# crictl images --digests --no-trunc \
registry.access.redhat.com/ubi9/nginx-120:1-86
IMAGE          TAG   DIGEST           IMAGE ID    ...
registry.access.redhat.com/ubi9/nginx-120 1-86 sha256:1be2...13a5  2e68...949e ...
```

The IMAGE\_ID column displays the local image identifier that the container engine assigns to the image. This identifier is not related to the SHA ID.

The container image format relies on SHA-256 hashes to identify several image components, such as the image layers or the image metadata. Because some commands also report these SHA-256 strings, ensure that you use the SHA-256 hash that corresponds to the SHA image ID. Commands often refer to the SHA image ID as the image digest.

## Selecting a Pull Policy

When you deploy an application, OpenShift selects a compute node to run the pod. On that node, OpenShift pulls the image and then starts the container.

By setting the `imagePullPolicy` attribute in the deployment resource, you can control how OpenShift pulls the image.

The following example shows the `myapp` deployment resource. The pull policy is set to `IfNotPresent`.

```
[user@host ~]$ oc get deployment myapp -o yaml
apiVersion: apps/v1
kind: Deployment
...output omitted...
template:
  metadata:
    creationTimestamp: null
    labels:
      app: myapp
  spec:
    containers:
      - image: registry.access.redhat.com/ubi9/nginx-120:1-86
        imagePullPolicy: IfNotPresent
        name: nginx-120
...output omitted...
```

The `imagePullPolicy` attribute can take the following values:

### IfNotPresent

If the image is already on the compute node, because another container is using it or because OpenShift pulled the image during a preceding pod run, then OpenShift uses that local image. Otherwise, OpenShift pulls the image from the container registry.

If you use a floating tag in your deployment, and the image with that tag is already on the node, then OpenShift does not pull the image again, even if the floating tag might point to a newer image in the source container registry.

OpenShift sets the `imagePullPolicy` attribute to `IfNotPresent` by default when you use a tag or the SHA ID to identify the image.

#### Always

OpenShift always verifies whether an updated version of the image is available on the source container registry. To do so, OpenShift retrieves the SHA ID of the image from the registry. If a local image with that same SHA ID is already on the compute node, then OpenShift uses that image. Otherwise, OpenShift pulls the image.

If you use a floating tag in your deployment, and an image with that tag is already on the node, then OpenShift queries the registry anyway to ensure that the tag still points to the same image version. However, if the developer pushed a new version of the image and updated the floating tag, then OpenShift retrieves that new image version.

OpenShift sets the `imagePullPolicy` attribute to `Always` by default when you use the `latest` tag, or when you do not specify a tag.

#### Never

OpenShift does not pull the image, and expects the image to be already available on the node. Otherwise, the deployment fails.

To use this option, you must prepopulate your compute nodes with the images that you plan to use. You use this mechanism to improve speed or to avoid relying on a container registry for these images.

## Pruning Images from Cluster Nodes

When OpenShift deletes a pod from a compute node, it does not remove the associated image. OpenShift can reuse the images without having to pull them again from the remote registry.

Because the images consume disk space on the compute nodes, OpenShift needs to remove, or *prune*, the unused images when disk space becomes sparse. The `kubelet` process, which runs on the compute nodes, includes a garbage collector that runs every five minutes. If the usage of the file system that stores the images is above 85%, then the garbage collector removes the oldest unused images. Garbage collection stops when the file system usage drops below 80%.

The reference documentation at the end of this lecture includes instructions to adjust these default thresholds.

From a compute node, you can run the `crlctl imagefsinfo` command to retrieve the name of the file system that stores the images:

```
[user@host ~]$ oc debug node/node-name
Temporary namespace openshift-debug-csn2p is created for debugging node...
Starting pod/node-name-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-4.4# crictl imagefsinfo
{
  "status": {
    "timestamp": "1674465624446958511",
```

```
"fsId": {  
    "mountpoint": "/var/lib/containers/storage/overlay-images"  
},  
"usedBytes": {  
    "value": "1318560"  
},  
"inodesUsed": {  
    "value": "446"  
}  
}  
}
```

From the preceding command output, the file system that stores the images is `/var/lib/containers/storage/overlay-images`. The images consume 1318560 bytes of disk space.

From the compute node, you can use the `crlctl rmi` to remove an unused image. However, pruning objects by using the `crlctl` command might interfere with the garbage collector and the `kubelet` process.

It is recommended that you rely on the garbage collector to prune unused objects, images, and containers from the compute nodes. The garbage collector is configurable to better fulfill custom needs that you might have.



## References

`skopeo-inspect(1)` and `podman-system-prune(1)` man pages

For more information about image names, refer to the *Overview of Images* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#overview-of-images](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#overview-of-images)

For more information about pull policies, refer to the *Image Pull Policy* section in the *Managing Images* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#image-pull-policy](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#image-pull-policy)

For more information about garbage collection, refer to the *Understanding How Terminated Containers Are Removed Through Garbage Collection* section in the *Working with Nodes* chapter in the Red Hat OpenShift Container Platform 4.14 *Nodes* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/nodes/index#nodes-nodes-garbage-collection-containers\\_nodes-nodes-configuring](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/nodes/index#nodes-nodes-garbage-collection-containers_nodes-nodes-configuring)

## ► Guided Exercise

# Container Image Identity and Tags

Update an application by changing its deployment to reference a newer image tag, and find the hashes of the old and new application images.

## Outcomes

You should be able to inspect container images, list images of containers that run on compute nodes, and deploy applications by using image tags or SHA IDs.

## 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. It also creates the `updates-ids` project and the `/home/student/D0180/labs/updates-ids/resources.txt` file. The `resources.txt` file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

```
[student@workstation ~]$ lab start updates-ids
```

## Instructions

- 1. Log in to the OpenShift cluster as the developer user with the developer password. Use the `updates-ids` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `updates-ids` project as the active project.

```
[student@workstation ~]$ oc project updates-ids
...output omitted...
```

- 2. Inspect the two versions of the `registry.ocp4.example.com:8443/ubi8/httpd-24` image from the classroom container registry. The classroom setup copied that image from the Red Hat Ecosystem Catalog. The original image is `registry.access.redhat.com/ubi8/httpd-24`.

- 2.1. Use the `oc image info` command to inspect the image version that the `1-209` tag references. Notice the unique SHA ID that identifies the image version.

**Note**

To improve readability, the instructions truncate the SHA-256 strings.

On your system, the commands return the full SHA-256 strings. Also, you must type the full SHA-256 string, to provide such a parameter to a command.

```
[student@workstation ~]$ oc image info \
registry.ocp4.example.com:8443/ubi8/httpd-24:1-209
Name: registry.ocp4.example.com:8443/ubi8/httpd-24:1-209
Digest: sha256:b1e3...f876
...output omitted...
```

- 2.2. Inspect the image version that the 1-215 tag references. Notice that the SHA ID, or digest, differs from the preceding image version.

```
[student@workstation ~]$ oc image info \
registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
Name: registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
Digest: sha256:91ad...fd83
...output omitted...
```

- 2.3. For inspecting images, you can also use the `skopeo inspect` command. The output format differs from the `oc image info` command, although both commands report similar data.

Log in to the registry as the developer user with the developer password by using the `skopeo login` command. Then, use the `skopeo inspect` command to inspect the 1-215 image tag.

```
[student@workstation ~]$ skopeo login registry.ocp4.example.com:8443 -u developer
Password: developer
Login Succeeded!
```

```
[student@workstation ~]$ skopeo inspect \
docker://registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
{
  "Name": "registry.ocp4.example.com:8443/ubi8/httpd-24",
  "Digest": "sha256:91ad...fd83",
  "RepoTags": [
    "1-209",
    "1-215"
  ],
  ...output omitted...
}
```

The `skopeo inspect` command also shows other existing image tags.

- ▶ 3. Deploy an application from the image version that the 1-209 tag references.

- 3.1. Use the `oc create deployment` command to deploy the application. Set the name of the deployment to `httpd1`.

```
[student@workstation ~]$ oc create deployment httpd1 \
--image registry.ocp4.example.com:8443/ubi8/httpd-24:1-209
deployment.apps/httpd1 created
```

- 3.2. Wait for the pod to start, and then retrieve the name of the cluster node that runs it. You might have to rerun the command several times for the pod to report a **Running** status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
httpd1-6dff796d99-pm2x6  1/1     Running   0          19s   10.8.0.104  master01 ...
```

- 3.3. Retrieve the name of the container that is running inside the pod. The `crlctl ps` command that you run in a following step takes the container name as an argument.

```
[student@workstation ~]$ oc get deployment httpd1 -o wide
NAME      READY   UP-TO-DATE   AVAILABLE   AGE      CONTAINERS   ...
httpd1    1/1     1           1           1m10s   httpd-24    ...
```

► **4.** Access the cluster node and then retrieve the image that the container is using.

- 4.1. You must log in as the `admin` user to access the cluster node. Use the `redhatocp` password.

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

- 4.2. Use the `oc debug node` command to access the cluster node.

```
[student@workstation ~]$ oc debug node/master01
Temporary namespace openshift-debug-flz4d is created for debugging node...
Starting pod/master01-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
```

- 4.3. In the remote shell, run the `chroot /host` command.

```
sh-4.4# chroot /host
sh-4.4#
```

- 4.4. Use the `crlctl ps` command to confirm that the `httpd-24` container is running. Add the `-o yaml` option to display the container details in YAML format.

```
sh-4.4# crictl ps --name httpd-24 -o yaml
containers:
- annotations:
```

```
...output omitted...
image:
  annotations: {}
  image: registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876
imageRef: registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876
  labels:
...output omitted...
state: CONTAINER_RUNNING
```

Notice that the command refers to the image by its SHA ID, and not by the tag that you specified when you created the deployment resource.

- 4.5. Use the `crlctl images` command to list the locally available images on the node. The `registry.ocp4.example.com:8443/ubi8/httpd-24:1-209` is in that list, because the local container engine pulled it when you deployed the `httpd1` application.



### Note

The `IMAGE ID` column displays the local image identifier that the container engine assigns to the image. This identifier is not related to the SHA image ID that the container registry assigned to the image.

Most `crlctl` commands, such as `crlctl images` or `crlctl rmi`, accept a local image identifier instead of the full image name. For example, you can run the `crlctl images 8ee59251acc93` command as a short version of the `crlctl images registry.ocp4.example.com:8443/ubi8/httpd-24:1-209` command.

### sh-4.4# `crlctl images`

IMAGE	TAG	IMAGE ID	SIZE
quay.io/openshift-release-dev/ocp-release	<none>	d52324cb88017	444MB
quay.io/openshift-release-dev/ocp-v4.0-art-dev	<none>	22e6e45df32af	468MB
quay.io/openshift-release-dev/ocp-v4.0-art-dev	<none>	e798432938c49	503MB
quay.io/openshift-release-dev/ocp-v4.0-art-dev	<none>	3ca084e53b321	873MB
...output omitted...			
<b>registry.ocp4.example.com:8443/ubi8/httpd-24</b>	<b>1-209</b>	<b>8ee59251acc93</b>	461MB
...output omitted...			

- 4.6. The preceding `crlctl images` command does not display the SHA image IDs by default. Rerun the command and add the `--digests` option to display the SHA IDs. Also add the local image ID to the command to limit the output to the `registry.ocp4.example.com:8443/ubi8/httpd-24:1-209` image.

The command reports only the first characters of the SHA image ID. These characters match the SHA ID of the image that the `httpd-24` container is using. Therefore, the `httpd-24` container is using the expected image.

### sh-4.4# `crlctl images --digests 8ee59251acc93`

IMAGE	TAG	DIGEST	IMAGE ID	...
registry.ocp4.example.com:8443/ubi8/httpd-24	1-209	<b>b1e3c572516d1</b>	8ee59251acc93	...

- 4.7. Disconnect from the cluster node.

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

Removing debug pod ...
Temporary namespace openshift-debug-flz4d was removed.
[student@workstation ~]$
```

- 5. Log in as the developer user and then deploy another application by using the SHA ID of the image as the digest.

- 5.1. Log in to the OpenShift cluster as the developer user.

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

- 5.2. Rerun the `oc image info` command to retrieve the SHA ID of the image version that the `1-209` tag references. Specify the JSON format for the command output. Parse the JSON output with the `jq -r` command to retrieve the value of the `.digest` object. Export the SHA ID as the `$IMAGE` environment variable.

```
[student@workstation ~]$ oc image info \
  registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 -o json | \
  jq -r .digest
sha256:b1e3...f876
```

```
[student@workstation ~]$ IMAGE=sha256:b1e3...f876
```

- 5.3. Use the `oc create deployment` command to deploy the application. Set the name of the deployment to `httpd2`.

```
[student@workstation ~]$ oc create deployment httpd2 \
  --image registry.ocp4.example.com:8443/ubi8/httpd-24@$IMAGE
deployment.apps/httpd2 created
```

- 5.4. Confirm that the new deployment refers to the image version by its SHA ID.

```
[student@workstation ~]$ oc get deployment httpd2 -o wide
NAME      READY  ...  CONTAINERS   IMAGES ...
httpd2    1/1    ...  httpd-24    registry.../ubi8/httpd-24@sha256:b1e3...f876 ...
```

- 6. Update the `httpd2` application by using a more recent image version.

- 6.1. In the `httpd2` deployment, update the `httpd-24` container to use the image version that the `1-215` tag references.

```
[student@workstation ~]$ oc set image deployment/httpd2 \
httpd-24=registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
deployment.apps/httpd2 image updated
```

- 6.2. Confirm that the deployment refers to the new image version.

```
[student@workstation ~]$ oc get deployment httpd2 -o wide
NAME      READY    ...   IMAGES ...
httpd2   1/1     ...   registry.ocp4.example.com:8443/ubi8/httpd-24:1-215 ...
```

- 6.3. Confirm that the deployment finished redeploying the pod. You might have to rerun the command several times for the pod to report a **Running** status. The pod names probably differ on your system.

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
httpd1-6dff796d99-pm2x6   1/1     Running   0          118m
httpd2-998d9b9b9-5859j   1/1     Running   0          21s
```

- 6.4. Inspect the pod to confirm that the container is using the new image. Replace the pod name with your own from the previous step.

```
[student@workstation ~]$ oc get pod httpd2-998d9b9b9-5859j \
-o jsonpath='{.spec.containers[0].image}{"\n"}'
registry.ocp4.example.com:8443/ubi8/httpd-24:1-215
```

- 7. Add the **latest** tag to the image version that the **1-209** tag already references. Deploy an application from the image with the **latest** tag.

- 7.1. Use the `skopeo login` command to log in to the classroom container registry as the `developer` user. Use `developer` for the password.

```
[student@workstation ~]$ skopeo login -u developer -p developer \
registry.ocp4.example.com:8443
Login Succeeded!
```

- 7.2. Use the `skopeo copy` command to add the **latest** tag to the image.

```
[student@workstation ~]$ skopeo copy \
docker://registry.ocp4.example.com:8443/ubi8/httpd-24:1-209 \
docker://registry.ocp4.example.com:8443/ubi8/httpd-24:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

- 7.3. Use the `oc image info` command to confirm that both tags refer to the same image. The two commands report the same SHA image ID, which indicates that the tags point to the same image version.

```
[student@workstation ~]$ oc image info \
  registry.ocp4.example.com:8443/ubi8/httpd-24:1-209
Name:          registry.ocp4.example.com:8443/ubi8/httpd-24:1-209
Digest:        sha256:b1e3...f876
...output omitted...
```

```
[student@workstation ~]$ oc image info \
  registry.ocp4.example.com:8443/ubi8/httpd-24:latest
Name:          registry.ocp4.example.com:8443/ubi8/httpd-24:latest
Digest:        sha256:b1e3...f876
...output omitted...
```

- 7.4. Use the `oc create deployment` command to deploy another application. Set the name of the deployment to `httpd3`. To confirm that by default the command selects the `latest` tag, do not provide the tag part in the image name.

```
[student@workstation ~]$ oc create deployment httpd3 \
  --image registry.ocp4.example.com:8443/ubi8/httpd-24
deployment.apps/httpd3 created
```

- 7.5. Confirm that the pod is running. You might have to rerun the command several times for the pod to report a `Running` status. The pod names probably differ on your system.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
httpd1-6dff796d99-pm2x6   1/1     Running   0          150m
httpd2-998d9b9b9-5859j   1/1     Running   0          32m
httpd3-85b978d758-fvqdr   1/1     Running   0          42s
```

- 7.6. Confirm that the pod is using the expected image. Notice that the SHA image ID corresponds to the image that the `1-209` tag references. You retrieved that SHA image ID in a preceding step when you ran the `oc image info` command.

```
[student@workstation ~]$ oc describe pod httpd3-85b978d758-fvqdr
...output omitted...
Containers:
  httpd-24:
    Container ID: cri-o://2cee...3a68
    Image:        registry.ocp4.example.com:8443/ubi8/httpd-24
    Image ID:      registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876
...output omitted...
```

- 8. Assign the `latest` tag to a different image version. This operation simulates a developer who pushes a new version of an image and assigns the `latest` tag to that new image version.

- 8.1. Use the `skopeo copy` command to add the `latest` tag to the image version that the `1-215` tag already references. The command automatically removes the `latest` tag from the earlier image.

```
[student@workstation ~]$ skopeo copy \
    docker://registry.ocp4.example.com:8443/ubi8/httpd-24:1-215 \
    docker://registry.ocp4.example.com:8443/ubi8/httpd-24:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

8.2. Log out of the classroom container registry.

```
[student@workstation ~]$ skopeo logout registry.ocp4.example.com:8443
Removed login credentials for registry.ocp4.example.com:8443
```



### Note

The `skopeo logout` command logs out of a specified registry server by deleting the cached credentials that are stored in the `$(XDG_RUNTIME_DIR)/containers/auth.json` file.

Red Hat recommends removing cached credentials that are no longer required.

8.3. Even though the `latest` tag is now referencing a different image version, OpenShift does not redeploy the pods that are running with the previous image version.

Rerun the `oc describe pod` command to confirm that the pod still uses the preceding image.

```
[student@workstation ~]$ oc describe pod httpd3-85b978d758-fvqdr
...output omitted...
Containers:
  httpd-24:
    Container ID: cri-o://2cee...3a68
    Image:         registry.ocp4.example.com:8443/ubi8/httpd-24
    Image ID:      registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876
    ...output omitted...
```

► 9. Scale the `httpd3` deployment to two pods.

9.1. Use the `oc scale` command to add a new pod to the deployment.

```
[student@workstation ~]$ oc scale deployment/httpd3 --replicas 2
deployment.apps/httpd3 scaled
```

9.2. List the pods to confirm that two pods are running for the `httpd3` deployment. The pod names probably differ on your system.

```
[student@workstation ~]$ oc get pods
httpd1-6dff796d99-pm2x6  1/1     Running   0          75m
httpd2-998d9b9b9-5859j    1/1     Running   0          30m
httpd3-85b978d758-f98jh   1/1     Running   0          54s
httpd3-85b978d758-fvqdr   1/1     Running   0          11m
```

- 9.3. Retrieve the SHA image ID for the pod that the deployment initially created. The ID did not change. The container is still using the original image version.

```
[student@workstation ~]$ oc describe pod httpd3-85b978d758-fvqdr
...output omitted...
Containers:
  httpd-24:
    Container ID: cri-o://2cee...3a68
    Image:         registry.ocp4.example.com:8443/ubi8/httpd-24
    Image ID:      registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:b1e3...f876
...output omitted...
```

- 9.4. Retrieve the SHA image ID for the additional pod. Notice that the ID is different. The additional pod is using the image that the `latest` tag is currently referencing.

```
[student@workstation ~]$ oc describe pod httpd3-85b978d758-f98jh
...output omitted...
Containers:
  httpd-24:
    Container ID: cri-o://d254...c893
    Image:         registry.ocp4.example.com:8443/ubi8/httpd-24
    Image ID:      registry.ocp4.example.com:8443/ubi8/httpd-24@sha256:91ad...fd83
...output omitted...
```

The state of the deployment is inconsistent. The two replicated pods use a different image version. Consequently, the scaled application might not behave correctly. Red Hat recommends that you use a less volatile tag than `latest` in production environments, or that you tightly control the tag assignments in your container registry.

## 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 updates-ids
```

# Update Application Image and Settings

## Objectives

- Update applications with minimal downtime by using deployment strategies.

## Application Code, Configuration, and Data

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

OpenShift provides configuration map, secret, and volume resources to store the application configuration and data. The application code is available through container images.

Because OpenShift deploys applications from container images, developers must build a new version of the image when they update the code of their application. Organizations usually use a continuous integration and continuous delivery (CI/CD) pipeline to automatically build the image from the application source code, and then to push the resulting image to a container registry.

You use OpenShift resources, such as configuration maps and secrets, to update the configuration of the application. To control the deployment process of a new image version, you use a `Deployment` object.

## Deployment Strategies

Deploying functional application changes or new versions to users is a significant phase of the CI/CD pipelines, where you add value to the development process.

Introducing application changes carries risks, such as downtime during the deployment, bugs, or reduced application performance. You can reduce or mitigate some risks with testing and validation stages in your pipelines.

Application or service downtime can result in lost business, disruption to other services that depend on yours, and violations of service level agreements, among others. To reduce downtime and minimize risks in deployments, use a *deployment strategy*. A deployment strategy changes or upgrades an application in a way that minimizes the impact of those changes.

In OpenShift, you use `Deployment` objects to define deployments and deployment strategies. The `RollingUpdate` and the `Recreate` strategies are the main OpenShift deployment strategies.

To select the `RollingUpdate` or `Recreate` strategies, you set the `.spec.strategy.type` property of the `Deployment` object. The following snippet shows a `Deployment` object that uses the `Recreate` strategy:

```
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
```

```

spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: myapp2
  strategy:
    type: Recreate
  template:
...output omitted...

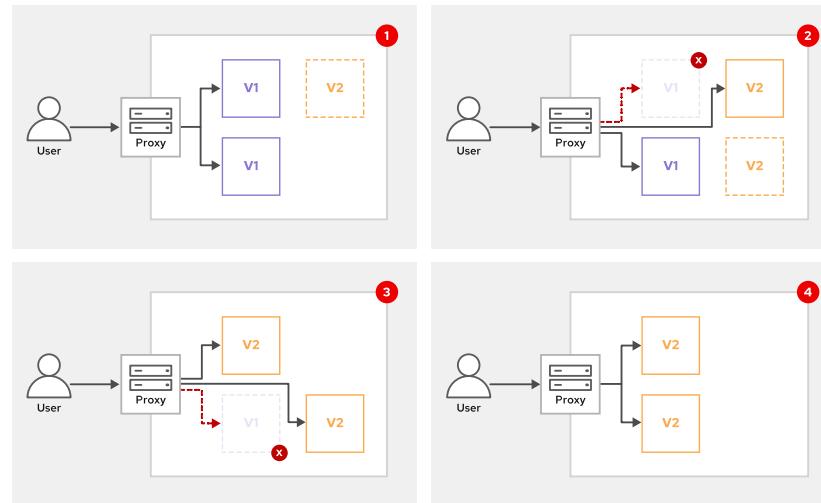
```

## Rolling Update Strategy

The `RollingUpdate` strategy consists of updating a version of an application in stages. It replaces one instance after another until all instances are replaced.

In this strategy, both versions of the application run simultaneously, and it scales down instances of the previous version only when the new version is ready. The main drawback is that this strategy requires compatibility between the versions in the deployment.

The following graphic shows the deployment of a new version of an application by using the `RollingUpdate` strategy:



1. Some application instances run a code version that needs updating (v1). OpenShift scales up a new instance with the updated application version (v2). Because the new instance with version v2 is not ready, only the version v1 instances fulfill customer requests.
2. The instance with v2 is ready and accepts customer requests. OpenShift scales down an instance with version v1, and scales up a new instance with version v2. Both versions of the application fulfill customer requests.
3. The new instance with v2 is ready and accepts customer requests. OpenShift scales down the remaining instance with version v1.
4. No instances remain to replace. The application update was successful, and without downtime.

The `RollingUpdate` strategy supports continuous deployment, and eliminates application downtime during deployments. You can use this strategy if the different versions of your application can run at the same time.

**Note**

The `RollingUpdate` strategy is the default strategy if you do not specify a strategy on the `Deployment` objects.

The following snippet shows a `Deployment` object that uses the `RollingUpdate` strategy:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...output omitted...
spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: myapp2
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 50%
    type: RollingUpdate
  template:
  ...output omitted...
```

Out of many parameters to configure the `RollingUpdate` strategy, the preceding snippet shows the `maxSurge` and `maxUnavailable` parameters.

During a rolling update, the number of pods for the application varies, because OpenShift starts new pods for the new revision, and removes pods from the previous revision. The `maxSurge` parameter indicates how many pods OpenShift can create above the normal number of replicas. The `maxUnavailable` parameter indicates how many pods OpenShift can remove below the normal number of replicas. You can express these parameters as percentages or as a number of pods.

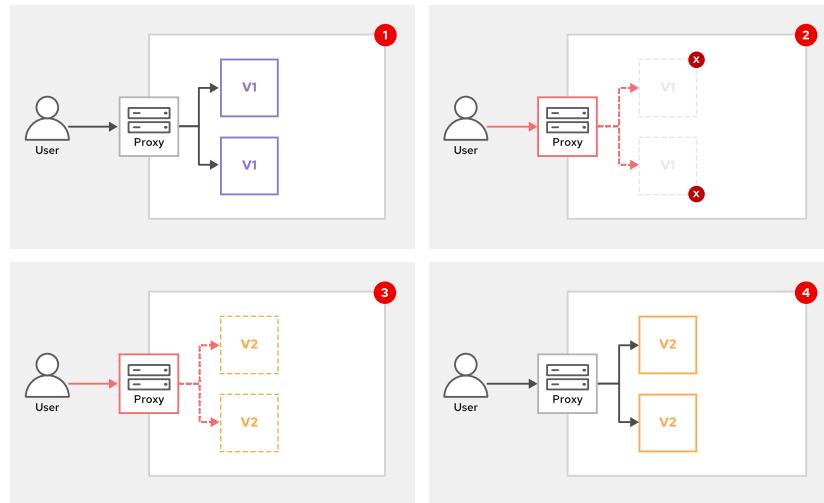
If you do not configure a readiness probe for your deployment, then during a rolling update, OpenShift starts sending client traffic to new pods as soon as they are running. However, the application inside a container might not be immediately ready to accept client requests. The application might have to load files to cache, establish a network connection to a database, or perform initial tasks that might take time to complete. Consequently, OpenShift redirects client requests to a container that is not yet ready, and these requests fail.

Adding a readiness probe to your deployment prevents OpenShift from sending traffic to new pods that are not ready.

## Recreate Strategy

In this strategy, all the instances of an application are killed first, and are then replaced with new ones. The major drawback of this strategy is that it causes a downtime in your services. For a period, no application instances are available to fulfill requests.

The following graphic shows the deployment of a new version of an application that uses the Recreate strategy:



1. The application has some instances that run a code version to update (v1).
2. OpenShift scales down the running instances to zero. This action causes application downtime, because no instances are available to fulfill requests.
3. OpenShift scales up new instances with a new version of the application (v2). When the new instances are booting, the downtime continues.
4. The new instances finished booting, and are ready to fulfill requests. This step is the last step of the Recreate strategy, and it resolves the application outage.

You can use this strategy when your application cannot have different simultaneously running code versions. You might also use it to execute data migrations or data transformations before the new code starts. This strategy is not recommended for applications that need high availability, for example, medical systems.

## Rolling out Applications

When you update a `Deployment` object, OpenShift automatically rolls out the application. If you apply several modifications in a row, such as modifying the image version, updating environment variables, and configuring the readiness probe, then OpenShift rolls out the application for each modification.

To prevent these multiple deployments, pause the rollout, apply all your modifications to the `Deployment` object, and then resume the rollout. OpenShift then performs a single rollout to apply all your modifications:

- Use the `oc rollout pause` command to pause the rollout of the `myapp` deployment:

```
[user@host ~]$ oc rollout pause deployment/myapp
```

- Apply all your modifications to the Deployment object. The following example modifies the image, an environment variable, and the readiness probe.

```
[user@host ~]$ oc set image deployment/myapp \
nginx-120=registry.access.redhat.com/ubi9/nginx-120:1-86
[user@host ~]$ oc set env deployment/myapp NGINX_LOG_TO_VOLUME=1
[user@host ~]$ oc set probe deployment/myapp --readiness --get-url http://:8080
```

- Resume the rollout:

```
[user@host ~]$ oc rollout resume deployment/myapp
```

OpenShift rolls out the application to apply all your modifications to the Deployment object.

You can follow a similar process when you create and configure a new deployment:

- Create the deployment, and set the number of replicas to zero. This way, OpenShift does not roll out your application, and no pods are running.

```
[user@host ~]$ oc create deployment myapp2 \
--image registry.access.redhat.com/ubi9/nginx-120:1-86 --replicas 0
[user@host ~]$ oc get deployment/myapp2
NAME      READY     UP-TO-DATE   AVAILABLE   AGE
myapp2    0/0       0           0           9s
```

- Apply the configuration to the Deployment object. The following example adds a readiness probe.

```
[user@host ~]$ oc set probe deployment/myapp2 --readiness --get-url http://:8080
```

- Scale up the deployment. OpenShift rolls out the application.

```
[user@host ~]$ oc scale deployment/myapp2 --replicas 10
[user@host ~]$ oc get deployment/myapp2
NAME      READY     UP-TO-DATE   AVAILABLE   AGE
myapp2    10/10    10          10          18s
```

## Monitoring Replica Sets

Whenever OpenShift rolls out an application from a Deployment object, it creates a ReplicaSet object. Replica sets are responsible for creating and monitoring the pods. If a pod fails, then the ReplicaSet object deploys a new one.

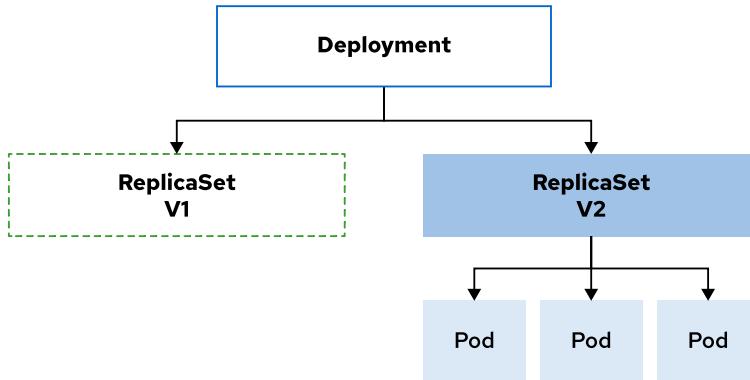
To deploy pods, replica sets use the pod template definition from the Deployment object. OpenShift copies the template definition from the Deployment object when it creates the ReplicaSet object.

When you update the Deployment object, OpenShift does not update the existing ReplicaSet object. Instead, it creates another ReplicaSet object with the new pod template definition. Then, OpenShift rolls out the application according to the update strategy.

Thus, several ReplicaSet objects for a deployment can exist at the same time on your system. During a rolling update, the old and the new ReplicaSet objects coexist and coordinate the

rollout of the new application version. After the rollout completes, OpenShift keeps the old `ReplicaSet` object so that you can roll back if the new application version does not operate correctly.

The following graphic shows a `Deployment` object and two `ReplicaSet` objects. The old `ReplicaSet` object for version 1 of the application does not run any pods. The current `ReplicaSet` object for version 2 of the application manages three replicated pods.



Do not directly change or delete `ReplicaSet` objects, because OpenShift manages them through the associated `Deployment` objects. The `.spec.revisionHistoryLimit` attribute in `Deployment` objects specifies how many `ReplicaSet` objects OpenShift keeps. OpenShift automatically deletes the extra `ReplicaSet` objects. Also, when you delete a `Deployment` object, OpenShift deletes all the associated `ReplicaSet` objects.

Run the `oc get replicaset` command to list the `ReplicaSet` objects. OpenShift uses the `Deployment` object name as a prefix for the `ReplicaSet` objects.

```
[user@host ~]$ oc get replicaset
NAME      DESIRED   CURRENT   READY   AGE
myapp2-574968dd59  0          0          0      3m27s
myapp2-76679885b9  10         10         10     22s
myapp2-786cbf9bc8  0          0          0      114s
```

The preceding output shows three `ReplicaSet` objects for the `myapp2` deployment. Whenever you modified the `myapp2` deployment, OpenShift created a `ReplicaSet` object. The second object in the list is active and monitors 10 pods. The other `ReplicaSet` objects do not manage any pods. They represent the previous versions of the `Deployment` object.

During a rolling update, two `ReplicaSet` objects are active. The old `ReplicaSet` object is scaling down, and at the same time the new object is scaling up:

```
[user@host ~]$ oc get replicaset
NAME      DESIRED   CURRENT   READY   AGE
myapp2-574968dd59  0          0          0      13m
myapp2-5fb5766df5  4          4          2      21s  ①
myapp2-76679885b9  8          8          8      10m  ②
myapp2-786cbf9bc8  0          0          0      11m
```

- ➊ The new ReplicaSet object already started four pods, but the READY column shows that the readiness probe succeeded for only two pods so far. These two pods are likely to receive client traffic.
- ➋ The ReplicaSet object already scaled down from 10 to 8 pods.

## Managing Rollout

Because OpenShift preserves ReplicaSet objects from earlier deployment versions, you can roll back if you notice that the new version of the application does not work.

Use the `oc rollout undo` command to roll back to the preceding deployment version. The command uses the existing ReplicaSet object for that version to roll back the pods. The command also reverts the Deployment object to the preceding version.

```
[user@host ~]$ oc rollout undo deployment/myapp2
```

Use the `oc rollout status` command to control the rollout process:

```
[user@host ~]$ oc rollout status deployment/myapp2
deployment "myapp2" successfully rolled out
```

If the rollout operation fails, because you specify a wrong container image name or the readiness probe fails, then OpenShift does not automatically roll back your deployment. In this case, run the `oc rollout undo` command to revert to the preceding working configuration.

In contrast to the Deployment resource, from Kubernetes, the DeploymentConfig resource, from OpenShift, supports an automated rollback. In addition, the `oc rollout cancel` command works only with DeploymentConfig objects, and supports canceling a stuck rollout operation. Another command, `oc rollback`, also works only with OpenShift DeploymentConfig resources. Red Hat recommends that you use the `oc rollout` command instead, which provides similar functionalities.

By default, the `oc rollout undo` command rolls back to the preceding deployment version. If you need to roll back to an earlier revision, then list the available revisions and add the `--to-revision rev` option to the `oc rollout undo` command.

- Use the `oc rollout history` command to list the available revisions:

```
[user@host ~]$ oc rollout history deployment/myapp2
deployment.apps/myapp2
REVISION  CHANGE-CAUSE
1          <none>
3          <none>
4          <none>
5          <none>
```

**Note**

The CHANGE-CAUSE column provides a user-defined message that describes the revision. You can store the message in the kubernetes.io/change-cause deployment annotation after every rollout:

```
[user@host ~]$ oc annotate deployment/myapp2 \
    kubernetes.io/change-cause="Image updated to 1-86"
deployment.apps/myapp2 annotated
[user@host ~]$ oc rollout history deployment/myapp2
deployment.apps/myapp2
REVISION  CHANGE-CAUSE
1          <none>
3          <none>
4          <none>
5          Image updated to 1-86
```

- Add the --revision option to the `oc rollout history` command for more details about a specific revision:

```
[user@host ~]$ oc rollout history deployment/myapp2 --revision 1
deployment.apps/myapp2 with revision #1
Pod Template:
Labels: app=myapp2
       pod-template-hash=574968dd59
Containers:
nginx-120:
  Image:      registry.access.redhat.com/ubi9/nginx-120:1-86
  Port:       <none>
  Host Port:  <none>
  Environment: <none>
  Mounts:     <none>
Volumes: <none>
```

The `pod-template-hash` attribute is the suffix of the associated `ReplicaSet` object. You can inspect that `ReplicaSet` object for more details by using the `oc describe replicaset myapp2-574968dd59` command, for example.

- Roll back to a specific revision by adding the `--to-revision` option to the `oc rollout undo` command:

```
[user@host ~]$ oc rollout undo deployment/myapp2 --to-revision 1
```

If you use floating tags to refer to container image versions in deployments, then the resulting image when you roll back a deployment might have changed in the container registry. Thus, the image that you run after the rollback might not be the original one that you used.

To prevent this issue, use OpenShift image streams for referencing images instead of floating tags. Another section in this course discusses image streams further.



## References

For more information about deployment strategies, 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://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#deployment-strategies](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#deployment-strategies)

For more information about readiness probes, refer to the *Monitoring Application Health by Using Health Checks* chapter in the Red Hat OpenShift Container

Platform 4.14 *Building Applications* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#application-health](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#application-health)

For more information about replication sets, refer to the *Understanding Deployment and DeploymentConfig Objects* section in the *Deployments* chapter in the Red Hat OpenShift Container Platform 4.14 *Building Applications* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/building\\_applications/index#what-deployments-are](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/building_applications/index#what-deployments-are)

## ► Guided Exercise

# Update Application Image and Settings

Update the manifests of a database and a web application while minimizing interruption of service to their users.

### Outcomes

You should be able to pause, update, and resume a deployment, and roll back a failing application.

### Before You Begin

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

This command ensures that all resources are available for this exercise. It also creates the `updates-rollout-db` project and deploys a MySQL database in that project. It creates the `updates-rollout-web` project and then deploys a web application with 10 replicas.

The command creates the `/home/student/D0180/labs/updates-rollout/resources.txt` file. The `resources.txt` file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

```
[student@workstation ~]$ lab start updates-rollout
```

### Instructions

- 1. Log in to the OpenShift cluster as the developer user with the developer password. Use the `updates-rollout-db` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `updates-rollout-db` project as the active project.

```
[student@workstation ~]$ oc project updates-rollout-db
...output omitted...
```

- 2. Review the resources that the `lab` command created. Confirm that you can connect to the database. The MySQL database uses ephemeral storage.

- 2.1. List the Deployment object and confirm that the pod is available. Retrieve the name of the container. You use that information when you update the container image in another step.

```
[student@workstation ~]$ oc get deployment -o wide
NAME      READY     UP-TO-DATE   AVAILABLE   AGE      CONTAINERS ...
mydb      1/1       0           1           17m     mysql-80    ...
```

- 2.2. List the pods and confirm that the pod is running. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pod
NAME                  READY   STATUS    RESTARTS   AGE
mydb-5c79866d48-5xzkk 1/1     Running   0          18m
```

- 2.3. Retrieve the name of the image that the pod is using. The pod is using the rhel9/mysql-80 image version 1-224. Replace the pod name with your own from the previous step.

```
[student@workstation ~]$ oc get pod mydb-5c79866d48-5xzkk \
-o jsonpath='{.spec.containers[0].image}{"\n"}'
registry.ocp4.example.com:8443/rhel9/mysql-80:1-224
```

The classroom setup copied that image from the Red Hat Ecosystem Catalog. The original image is `registry.redhat.io/rhel9/mysql-80`.

- 2.4. Confirm that you can connect to the database system by listing the available databases. Run the `mysql` command from inside the pod and connect as the `operator1` user by using `test` as the password.

```
[student@workstation ~]$ oc rsh mydb-5c79866d48-5xzkk \
mysql --user=operator1 --password=test -e "SHOW DATABASES"
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| Database      |
+-----+
| information_schema |
| performance_schema |
| quotes          |
+-----+
```

- 3. You must implement several updates to the Deployment object. Pause the deployment to prevent OpenShift from rolling out the application for each modification that you make. After you pause the deployment, change the password for the `operator1` database user, update the container image, and then resume the deployment.

- 3.1. Pause the `mydb` deployment.

```
[student@workstation ~]$ oc rollout pause deployment/mydb
deployment.apps/mydb paused
```

- 3.2. Change the password of the `operator1` database user to `redhat123`. To change the password, update the `MYSQL_PASSWORD` environment variable in the pod template of the Deployment object.

```
[student@workstation ~]$ oc set env deployment/mydb MYSQL_PASSWORD=redhat123
deployment.apps/mydb updated
```

- 3.3. Because the Deployment object is paused, confirm that the new password is not yet active. To do so, rerun the `mysql` command by using the current password. The database connection succeeds.

```
[student@workstation ~]$ oc rsh mydb-5c79866d48-5xzkk \
  mysql --user=operator1 --password=test -e "SHOW DATABASES"
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| Database      |
+-----+
| information_schema |
| performance_schema |
| quotes          |
+-----+
```

- 3.4. Update the MySQL container image to the 1-228 version.

```
[student@workstation ~]$ oc set image deployment/mydb \
  mysql-80=registry.ocp4.example.com:8443/rhel9/mysql-80:1-228
deployment.apps/mydb image updated
```

- 3.5. Because the Deployment object is paused, confirm that the pod still uses the 1-224 image version.

```
[student@workstation ~]$ oc get pod mydb-5c79866d48-5xzkk \
  -o jsonpath='{.spec.containers[0].image}'"
registry.ocp4.example.com:8443/rhel9/mysql-80:1-224
```

- 3.6. Resume the `mydb` deployment.

```
[student@workstation ~]$ oc rollout resume deployment/mydb
deployment.apps/mydb resumed
```

- 3.7. Confirm that the new rollout completes by waiting for the new pod to be running. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
mydb-dd5dcbdbb-rmf85  1/1     Running   0          2m2s
```

- 4. Verify that OpenShift applied all your modifications to the Deployment object.

- 4.1. Retrieve the name of the image that the new pod is using. In the following command, use the name of the new pod as a parameter to the `oc get pod` command. The pod is now using the 1-228 image version.

```
[student@workstation ~]$ oc get pod mydb-dd5dcbdb-rmf85 \
-o jsonpath='{.spec.containers[0].image}{"\n"}'
registry.ocp4.example.com:8443/rhel9/mysql-80:1-228
```

- 4.2. Confirm that you can connect to the database system by using the new password, `redhat123`, for the `operator1` database user.

```
[student@workstation ~]$ oc rsh mydb-dd5dcbdb-rmf85 \
mysql --user=operator1 --password=redhat123 -e "SHOW DATABASES"
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+
| Database      |
+-----+
| information_schema |
| performance_schema |
| quotes          |
+-----+
```

- 5. In the second part of the exercise, you perform a rolling update of a replicated web application. Use the `updates-rollout-web` project and review the resources that the `lab` command created.

- 5.1. Set the `updates-rollout-web` project as the active project.

```
[student@workstation ~]$ oc project updates-rollout-web
...output omitted...
```

- 5.2. List the `Deployment` object and confirm that the pods are available. Retrieve the name of the containers. You use that information when you update the container image in another step.

```
[student@workstation ~]$ oc get deployment -o wide
NAME      READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   ...
version   10/10   10           10          32m   versioned-hello ...
```

- 5.3. List the `ReplicaSet` objects. Because OpenShift did not yet perform rolling updates, only one `ReplicaSet` object exists. The name of the `ReplicaSet` object on your system probably differs.

```
[student@workstation ~]$ oc get replicaset
NAME        DESIRED   CURRENT   READY   AGE
version-7bfff6b5b4  10        10        10      11m
```

- 5.4. Retrieve the name and version of the image that the `ReplicaSet` object uses to deploy the pods. The pods are using the `redhattraining/versioned-hello` image version v1.0.

```
[student@workstation ~]$ oc get replicaset version-7bfff6b5b4 \
-o jsonpath='{.spec.template.spec.containers[0].image}{"\n"}'
registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0
```

- 5.5. Confirm that the **version** deployment includes a readiness probe. The probe performs an HTTP GET request on port 8080.

```
[student@workstation ~]$ oc get deployment version \
-o jsonpath='{.spec.template.spec.containers[0].readinessProbe}' | jq .
{
  "failureThreshold": 3,
  "httpGet": {
    "path": "/",
    "port": 8080,
    "scheme": "HTTP"
  },
  "initialDelaySeconds": 3,
  "periodSeconds": 10,
  "successThreshold": 1,
  "timeoutSeconds": 1
}
```

- 6. To watch the rolling update that you cause in a following step, open a new terminal window and then run the `~/D0180/labs/updates-rollout/curl_loop.sh` script that the `lab` command prepared. The script sends web requests to the application in a loop.

- 6.1. Open a new terminal.
- 6.2. Run the `/home/student/D0180/labs/updates-rollout/curl_loop.sh` script. Leave the script running and do not interrupt it.

```
[student@workstation ~]$ /home/student/D0180/labs/updates-rollout/curl_loop.sh
Hi!
Hi!
Hi!
Hi!
...output omitted...
```

- 7. Change the container image of the **version** deployment. The new application version creates a web page with a different message.

- 7.1. Switch back to the first terminal window, and then use the `oc set image` command to update the deployment.

```
[student@workstation ~]$ oc set image deployment/version \
versioned-hello=registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.1
deployment.apps/version image updated
```

- 7.2. Changing the image caused a rolling update. Watch the output of the `curl_loop.sh` script in the second terminal. If the output of the `curl_loop.sh` is not updating, then press `Ctrl+c` to stop the script in the second terminal. Then, restart the script.

Before the update, only pods that run the v1.0 version of the application reply. During the rolling updates, both old and new pods are responding. After the update, only pods that run the v1.1 version of the application reply. The following output probably differs on your system.

```
...output omitted...
Hi!
Hi!
Hi!
Hi!
Hi! v1.1
Hi! v1.1
Hi!
Hi! v1.1
Hi!
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi! v1.1
...output omitted...
```

Do not stop the script.

- ▶ 8. Confirm that the rollout process is successful. List the ReplicaSet objects and verify that the new object uses the new image version.
  - 8.1. Use the `oc rollout status` command to confirm that the rollout process is successful.

```
[student@workstation ~]$ oc rollout status deployment/version
deployment "version" successfully rolled out
```

- 8.2. List the ReplicaSet objects. The initial object scaled down to zero pods. The new ReplicaSet object scaled up to 10 pods. The names of the ReplicaSet objects on your system probably differ.

```
[student@workstation ~]$ oc get replicaset
NAME          DESIRED   CURRENT   READY   AGE
version-7bfff6b5b4    0          0          0     28m
version-b7fddfc8c    10         10         10    3m40s
```

- 8.3. Retrieve the name and version of the image that the new ReplicaSet object uses. This image provides the new version of the application.

```
[student@workstation ~]$ oc get replicaset version-b7fddfc8c \
-o jsonpath='{.spec.template.spec.containers[0].image}{"\n"}'
registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.1
```

- ▶ 9. Roll back the `version` deployment.
  - 9.1. Use the `oc rollout undo` command to roll back to the initial application version. Ignore the warning message.

```
[student@workstation ~]$ oc rollout undo deployment/version  
deployment.apps/version rolled back
```

- 9.2. Watch the output of the `curl_loop.sh` script in the second terminal. The pods that run the v1.0 version of the application are responding again. The following output probably differs on your system.

```
...output omitted...  
Hi! v1.1  
Hi! v1.1  
Hi! v1.1  
Hi! v1.1  
Hi!  
Hi! v1.1  
Hi!  
Hi! v1.1  
Hi! v1.1  
Hi!  
Hi!  
Hi!  
...output omitted...
```

Press `Ctrl+C` to quit the script. Close that second terminal when done.

- 9.3. List the `ReplicaSet` objects. The initial object scaled up to 10 pods. The object for the new application version scaled down to zero pods.

```
[student@workstation ~]$ oc get replicaset  
NAME          DESIRED   CURRENT   READY   AGE  
version-7bfff6b5b4  10        10        10      52m  
version-b7fddfc8c  0         0         0       27m
```

## 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 updates-rollout
```

# Reproducible Deployments with OpenShift Image Streams

---

## Objectives

- Ensure reproducibility of application deployments by using image streams and short image names.

## Image Streams

*Image streams* are one of the main differentiators between OpenShift and upstream Kubernetes. Kubernetes resources reference container images directly, but OpenShift resources, such as deployment configurations and build configurations, reference image streams. OpenShift also extends Kubernetes resources, such as Kubernetes Deployments, with annotations that make them work with OpenShift image streams.

With image streams, OpenShift can ensure reproducible, stable deployments of containerized applications and also rollbacks of deployments to their latest known-good state.

Image streams provide a stable, short name to reference a container image that is independent of any registry server and container runtime configuration.

As an example, an organization could start by downloading container images directly from the Red Hat public registry and later set up an enterprise registry as a mirror of those images to save bandwidth. OpenShift users would not notice any change, because they still refer to these images by using the same image stream name. Users of the RHEL container tools would notice the change, because they would be required either to change the registry names in their commands, or to change their container engine configurations to search for the local mirror first.

In other scenarios, the indirection that an image stream provides can be helpful. Suppose that you start with a database container image that has security issues, and the vendor takes too long to update the image with fixes. Later, you find an alternative vendor who provides an alternative container image for the same database, with those security issues already fixed, and even better, with a track record of providing timely updates to them. If those container images are compatible regarding configuration of environment variables and volumes, you could change your image stream to point to the image from the alternative vendor.

Red Hat provides hardened, supported container images that work mostly as drop-in replacements of container images from some popular open source projects, such as the MariaDB database.

## Image Stream Tags

An image stream represents one or more sets of container images. Each set, or stream, is identified by an *image stream tag*. Unlike container images in a registry server, which have multiple tags from the same image repository (or user or organization), an image stream can have multiple image stream tags that reference container images from different registry servers and from different image repositories.

An image stream provides default configurations for a set of image stream tags. Each image stream tag references one stream of container images, and can override most configurations from its associated image stream.

An image stream tag stores a copy of the metadata about its current container image. Storing metadata supports faster search and inspection of container images, because you do not need to reach its source registry server.

You can also configure an image stream tag to store the source image layers in the OpenShift internal container registry, which acts as a local image cache. Storing image layers locally avoids the need to fetch these layers from their source registry server. Consumers of the cached image, such as pods and deployments, just reference the internal registry as the source registry of the image.

For some other OpenShift resource types that relate to image streams, you can usually dismiss them as implementation details of the internal registry, and focus only on image streams and image stream tags.

To better visualize the relationship between image streams and image stream tags, you can explore the `openshift` project that is pre-created in all OpenShift clusters. You can see many image streams in that project, including the `php` image stream:

```
[user@host ~]$ oc get is -n openshift -o name
...output omitted...
imagestream.image.openshift.io/nodejs
imagestream.image.openshift.io/perl
imagestream.image.openshift.io/php
imagestream.image.openshift.io/postgresql
imagestream.image.openshift.io/python
...output omitted...
```

Several tags exist for the `php` image stream, and an image stream tag resource exists for each tag:

```
[user@host ~]$ oc get istag -n openshift | grep php
8.0-ubi9      image-registry ...   6 days ago
8.0-ubi8      image-registry ...   6 days ago
7.4-ubi8      image-registry ...   6 days ago
7.3-ubi7      image-registry ...   6 days ago
```

The `oc describe` command on an image stream shows information from both the image stream and its image stream tags:

```
[user@host ~]$ oc describe is php -n openshift
Name:                  php
Namespace:             openshift
...output omitted...
Tags:                  5

8.0-ubi9
tagged from registry.access.redhat.com/ubi9/php-80:latest
...output omitted...

8.0-ubi8 (latest)
tagged from registry.access.redhat.com/ubi8/php-80:latest
...output omitted...

7.4-ubi8
tagged from registry.access.redhat.com/ubi8/php-74:latest
```

```
...output omitted...

7.3-ubi7
tagged from registry.access.redhat.com/ubi7/php-73:latest
...output omitted...
```

In the previous example, each of the `php` image stream tags refers to a different image name.

## Image Names, Tags, and IDs

The textual name of a container image is a string. This name is sometimes interpreted as being made of multiple components, such as `registry-host-name/repository-or-organization-or-user-name/image-name:tag-name`, but splitting the image name into its components is a matter of convention, not of structure.

A SHA image ID is a SHA-256 hash that uniquely identifies an immutable container image. You cannot modify a container image. Instead, you create a container image that has a new ID. When you push a new container image to a registry server, the server associates the existing textual name with the new image ID.

When you start a container from an image name, you download the image that is currently associated with that image name. The image ID behind that name might change at any moment, and the next container that you start might have a different image ID. If the image that is associated with an image name has any issues, and you know only the image name, then you cannot roll back to an earlier image.

OpenShift image stream tags keep a history of the latest image IDs that they fetched from a registry server. The history of image IDs is the stream of images from an image stream tag. You can use the history inside an image stream tag to roll back to a previous image, if for example a new container image causes a deployment error.

Updating a container image in an external registry does not automatically update an image stream tag. The image stream tag keeps the reference to the last image ID that it fetched. This behavior is crucial to scaling applications, because it isolates OpenShift from changes that happen at a registry server.

Suppose that you deploy an application from an external registry, and after a few days of testing with a few users, you decide to scale its deployment to enable a larger user population. In the meantime, your vendor updates the container image on the external registry. If OpenShift had no image stream tags, then the new pods would get the new container image, which is different from the image on the original pod. Depending on the changes, this new image could cause your application to fail. Because OpenShift stores the image ID of the original image in an image stream tag, it can create new pods by using the same image ID and avoid any incompatibility between the original and updated image.

OpenShift keeps the image ID of the first pod, and ensures that new pods use the same image ID. OpenShift ensures that all pods use the same image.

To better visualize the relationship between an image stream, an image stream tag, an image name, and an image ID, refer to the following `oc describe is` command, which shows the source image and current image ID for each image stream tag:

```
[user@host ~]$ oc describe is php -n openshift
Name:                      php
Namespace:                  openshift
...output omitted...
```

```
8.0-ubi9
tagged from registry.access.redhat.com/ubi9/php-80:latest
...output omitted...
* registry.access.redhat.com/ubi9/php-80@sha256:2b82...f544
  2 days ago

8.0-ubi8 (latest)
tagged from registry.access.redhat.com/ubi8/php-80:latest
* registry.access.redhat.com/ubi8/php-80@sha256:2c74...5ef4
  2 days ago
...output omitted...
```

If your OpenShift cluster administrator already updated the `php:8.0-ubi9` image stream tag, the `oc describe is` command shows multiple image IDs for that tag:

```
[user@host ~]$ oc describe is php -n openshift
Name:          php
Namespace:     openshift
...output omitted...

8.0-ubi9
tagged from registry.access.redhat.com/ubi9/php-80:latest
...output omitted...
* registry.access.redhat.com/ubi9/php-80@sha256:2b82...f544
  2 days ago
registry.access.redhat.com/ubi9/php-80@sha256:8840...94f0
  5 days ago
registry.access.redhat.com/ubi9/php-80@sha256:506c...5d90
  9 days ago
```

In the previous example, the asterisk (\*) shows which image ID is the current one for each image stream tag. It is usually the latest one to be imported, and the first one that is listed.

When an OpenShift image stream tag references a container image from an external registry, you must explicitly update the image stream tag to get new image IDs from the external registry. By default, OpenShift does not monitor external registries for changes to the image ID that is associated with an image name.

You can configure an image stream tag to check the external registry for updates on a defined schedule. By default, new image stream tags do not check for updated images.

## Creating Image Streams and Tags

In addition to the image streams in the `openshift` project, you can create image streams in your project so that the resources in that project, such as `Deployment` objects, can use them.

Use the `oc create is` command to create image streams in the current project. The following example creates an image stream named `keycloak`:

```
[user@host ~]$ oc create is keycloak
```

After you create the image stream, use the `oc create istag` command to add image stream tags. The following example adds the `20.0` tag to the `keycloak` image stream. In this example,

the image stream tag refers to the quay.io/keycloak/keycloak:20.0.2 image from the Quay.io public repository.

```
[user@host ~]$ oc create istag keycloak:20.0 \
--from-image quay.io/keycloak/keycloak:20.0.2
```

Repeat the preceding command if you need more image stream tags:

```
[user@host ~]$ oc create istag keycloak:19.0 \
--from-image quay.io/keycloak/keycloak:19.0
```

Use the `oc tag SOURCE-IMAGE IMAGE-STREAM-TAG` command to update an image stream tag with a new source image reference. The following example changes the keycloak:20.0 image stream tag to point to the quay.io/keycloak/keycloak:20.0.3 image:

```
[user@host ~]$ oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0
```

Use the `oc describe is` command to verify that the image stream tag points to the SHA ID of the source image:

```
[user@host ~]$ oc describe is keycloak
Name:           keycloak
Namespace:      myproject
Created:        5 minutes ago
Labels:          <none>
Annotations:    openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z
Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak
Image Lookup:   local=false
Unique Images:  3
Tags:           2

20.0
tagged from quay.io/keycloak/keycloak:20.0.3

* quay.io/keycloak/keycloak@sha256:c167...62e9
  47 seconds ago
quay.io/keycloak/keycloak@sha256:5569...b311
  5 minutes ago

19.0
tagged from quay.io/keycloak/keycloak:19.0

* quay.io/keycloak/keycloak@sha256:40cc...ffde
  5 minutes ago
```

## Importing Image Stream Tags Periodically

When you create an image stream tag, OpenShift configures it with the SHA ID of the source image that you specify. After that initial creation, the image stream tag does not change, even if the developer pushes a new version of the source image.

By using image stream tags, you are in control of the images that your applications are using. If you want to use a new image version, then you manually need to update the image stream tag to point to that new version.

However, for some container registries that you trust, or for some specific images, you might prefer the image stream tags to automatically refresh.

For example, Red Hat regularly updates the images from the Red Hat Ecosystem Catalog with bug and security fixes. To benefit from these updates as soon as Red Hat releases them, you can configure your image stream tags to regularly refresh.

OpenShift can periodically verify whether a new image version is available. When it detects a new version, it automatically updates the image stream tag. To activate that periodic refresh, add the `--scheduled` option to the `oc tag` command.

```
[user@host ~]$ oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 --scheduled
```

By default, OpenShift verifies the image every 15 minutes. This period is a setting that your cluster administrators can adapt.

## Configuring Image Pull-through

When OpenShift starts a pod that uses an image stream tag, it pulls the corresponding image from the source container registry.

When the image comes from a registry on the internet, pulling the image can take time, or even fail in case of a network outage. Some public registries have bandwidth throttling rules that can slow down your downloads further.

To mitigate these issues, you can configure your image stream tags to cache the images in the OpenShift internal container registry. The first time that OpenShift pulls the image, it downloads the image from the source repository and then stores the image in its internal registry. After that initial pull, OpenShift retrieves the image from the internal registry.

To activate image pull-through, add the `--reference-policy local` option to the `oc tag` command.

```
[user@host ~]$ oc tag quay.io/keycloak/keycloak:20.0.3 keycloak:20.0 \
--reference-policy local
```

## Using Image Streams in Deployments

When you create a `Deployment` object, you can specify an image stream instead of a container image from a registry. Using an image stream in Kubernetes workload resources, such as deployments, requires preparation:

- Create the image stream object in the same project as the `Deployment` object.
- Enable the local lookup policy in the image stream object.
- In the `Deployment` object, reference the image stream tag by its name, such as `keycloak:20.0`, and not by the full image name from the source registry.

## Enabling the Local Lookup Policy

When you use an image stream in a Deployment object, OpenShift looks for that image stream in the current project. However, OpenShift searches only the image streams that you enabled the local lookup policy for.

Use the `oc set image-lookup` command to enable the local lookup policy for an image stream:

```
[user@host ~]$ oc set image-lookup keycloak
```

Use the `oc describe is` command to verify that the policy is active:

```
[user@host ~]$ oc describe is keycloak
Name:          keycloak
Namespace:     myproject
Created:       3 hours ago
Labels:        <none>
Annotations:   openshift.io/image.dockerRepositoryCheck=2023-01-31T11:12:44Z
Image Repository: image-registry.openshift-image-registry.svc:5000/.../keycloak
Image Lookup:  local=true
Unique Images: 3
Tags:          2
...output omitted...
```

You can also retrieve the local lookup policy status for all the image streams in the current project by running the `oc set image-lookup` command without parameters:

```
[user@host ~]$ oc set image-lookup
NAME      LOCAL
keycloak  true
zabbix-agent false
nagios    false
```

To disable the local lookup policy, add the `--enabled=false` option to the `oc set image-lookup` command:

```
[user@host ~]$ oc set image-lookup keycloak --enabled=false
```

## Configuring Image Streams in Deployments

When you create a Deployment object by using the `oc create deployment` command, use the `--image` option to specify the image stream tag:

```
[user@host ~]$ oc create deployment mykeycloak --image keycloak:20.0
```

When you use a short name, OpenShift looks for a matching image stream in the current project. OpenShift considers only the image streams that you enabled the local lookup policy for. If it does not find an image stream, then OpenShift looks for a regular container image in the allowed container registries. The reference documentation at the end of this lecture describes how to configure these allowed registries.

You can also use image streams with other Kubernetes workload resources:

- Job objects that you can create by using the following command:

```
[user@host ~]$ oc create job NAME --image IMAGE-STREAM-TAG -- COMMAND
```

- CronJob objects that you can create by using the following command:

```
[user@host ~]$ oc create cronjob NAME --image IMAGE-STREAM-TAG \
--schedule CRON-SYNTAX -- COMMAND
```

- Pod objects that you can create by using the following command:

```
[user@host ~]$ oc run NAME --image IMAGE-STREAM-TAG
```

Another section in this course discusses how changing an image stream tag can automatically roll out the associated deployments.



## References

For more information about using image streams, refer to the *Managing Image Streams* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#managing-image-streams](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#managing-image-streams)

For more information about using image streams with deployments, refer to the *Using Image Streams with Kubernetes Resources* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#using-imagestreams-with-kube-resources](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#using-imagestreams-with-kube-resources)

For more information on how to configure container registries in OpenShift, refer to the *Image Configuration Resources* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#image-configuration](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#image-configuration)

### How to Simplify Container Image Management in Kubernetes with OpenShift Image Streams

<https://cloud.redhat.com/blog/image-streams-faq>

## ► Guided Exercise

# Reproducible Deployments with OpenShift Image Streams

Deploy an application that references container images indirectly by using image streams.

### Outcomes

You should be able to create image streams and image stream tags, and deploy applications that use image stream tags.

### 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. It also creates the `updates-imagestreams` project and the `/home/student/D0180/labs/updates-imagestreams/resources.txt` file. The `resources.txt` file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

```
[student@workstation ~]$ lab start updates-imagestreams
```

### Instructions

- 1. Log in to the OpenShift cluster as the `developer` user with the `developer` password. Use the `updates-imagestreams` project.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `updates-imagestreams` project as the active project.

```
[student@workstation ~]$ oc project updates-imagestreams
...output omitted...
```

- 2. Create the `versioned-hello` image stream and the `v1.0` image stream tag from the `registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0` image.

- 2.1. Use the `oc create is` command to create the image stream.

```
[student@workstation ~]$ oc create is versioned-hello  
imagestream.image.openshift.io/versioned-hello created
```

- 2.2. Use the `oc create istag` command to create the image stream tag.

```
[student@workstation ~]$ oc create istag versioned-hello:v1.0 \  
--from-image registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0  
imagestreamtag.image.openshift.io/versioned-hello:v1.0 created
```

- 3. Enable image stream resolution for the `versioned-hello` image stream so that Kubernetes resources in the current project can use it.

- 3.1. Use the `oc set image-lookup` command to enable image lookup resolution.

```
[student@workstation ~]$ oc set image-lookup versioned-hello  
imagestream.image.openshift.io/versioned-hello image lookup updated
```

- 3.2. Run the `oc set image-lookup` command without any arguments to verify your work.

```
[student@workstation ~]$ oc set image-lookup  
NAME          LOCAL  
versioned-hello true
```

- 4. Review the image stream and confirm that the image stream tag refers to the source image by its SHA ID. Verify that the source image in the `registry.ocp4.example.com:8443` registry has the same SHA ID.

- 4.1. Retrieve the details of the `versioned-hello` image stream.



**Note**

To improve readability, the instructions truncate the SHA-256 strings.

```
[student@workstation ~]$ oc describe is versioned-hello  
Name:           versioned-hello  
Namespace:      updates-imagestreams  
Created:        7 minutes ago  
...output omitted...  
  
v1.0  
tagged from registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0  
  
* registry.ocp4.example.com:8443/.../versioned-hello@sha256:66e0...105e  
    7 minutes ago
```

- 4.2. Use the `oc image info` command to query the image from the classroom container registry. The SHA image ID is the same as the one from the image stream tag.

```
[student@workstation ~]$ oc image info \
  registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0
Name:      registry.ocp4.example.com:8443/redhattraining/versioned-hello:v1.0
Digest:    sha256:66e0...105e
Media Type: application/vnd.docker.distribution.manifest.v2+json
...output omitted...
```

- 5. Create a deployment named `version` that uses the `versioned-hello:v1.0` image stream tag.

- 5.1. Use the `oc create deployment` command to create the object.

```
[student@workstation ~]$ oc create deployment version --image versioned-hello:v1.0
deployment.apps/version created
```

- 5.2. Wait for the pod to start. You might have to rerun the command several times for the pod to report a `Running` status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
version-744bf7694b-bzhd2   1/1     Running   0          2m11s
```

- 6. Confirm that both the deployment and the pod refer to the image by its SHA ID.

- 6.1. Retrieve the image that the deployment uses. The deployment refers to the image from the source registry by its SHA ID. The `v1.0` image stream tag also points to that SHA image ID.

```
[student@workstation ~]$ oc get deployment -o wide
... IMAGES ...
... registry.ocp4.example.com:8443/.../versioned-hello@sha256:66e0...105e ...
```

- 6.2. Retrieve the image that the pod is using. The pod is also referring to the image by its SHA ID.

```
[student@workstation ~]$ oc get pod version-744bf7694b-bzhd2 \
-o jsonpath='{.spec.containers[0].image}{"\n"}'
registry.ocp4.example.com:8443/redhattraining/versioned-hello@sha256:66e0...105e
```

## 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 updates-imagestreams
```

# Automatic Image Updates with OpenShift Image Change Triggers

## Objectives

- Ensure automatic update of application pods by using image streams with Kubernetes workload resources.

## Using Triggers to Manage Images

Image stream tags record the SHA ID of the source container image. Thus, an image stream tag always points to an immutable image.

If a new version of the source image becomes available, then you can change the image stream tag to point to that new image. However, a Deployment object that uses the image stream tag does not roll out automatically. For an automatic rollout, you must configure the Deployment object with an image trigger.

If you update an image stream tag to point to a new image version, and you notice that this version does not work as expected, then you can revert the image stream tag. Deployment objects for which you configured a trigger automatically roll back to that previous image.

Other Kubernetes workloads also support image triggers, such as Pod, CronJob, and Job objects.

## Configuring Image Trigger for Deployments

Before you can configure image triggers for a Deployment object, ensure that the Deployment object is using image stream tags for its containers:

- Create the image stream object in the same project as the Deployment object.
- Enable the local lookup policy in the image stream object by using the `oc set image-lookup` command.
- In the Deployment object, reference the image stream tags by their names, such as `keycloak:20`, and not by the full image names from the source registry.

Image triggers apply at the container level. If your Deployment object includes several containers, then you can specify a trigger for each one. Before you can set triggers, retrieve the container names:

```
[user@host ~]$ oc get deployment mykeycloak -o wide
NAME      READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS ...
mykeycloak  0/1     1           0           6s    keycloak ...
```

Use the `oc set triggers` command to configure an image trigger for the container inside the Deployment object. Use the `--from-image` option to specify the image stream tag to watch.

```
[user@host ~]$ oc set triggers deployment/mykeycloak --from-image keycloak:20 \
--containers keycloak
```

OpenShift DeploymentConfig objects natively support image streams, and support automatic rollout on image change. DeploymentConfig resources have attributes to support image streams and triggers.



### Note

As of Red Hat OpenShift Container Platform 4.14, DeploymentConfig objects are deprecated. DeploymentConfig objects are still supported, but are not recommended for new installations.

Instead, use Deployment objects or another alternative to provide declarative updates for pods.

In contrast, Kubernetes Deployment resources do not natively support image streams, and do not have attributes to store the related configuration. To provide automatic image rollout for Deployment objects, OpenShift adds the `image.openshift.io/triggers` annotation to store the configuration in JSON format.

The following example retrieves the content of the `image.openshift.io/triggers` annotation, and then uses the `jq` command to display the configuration in a more readable format:

```
[user@host ~]$ oc get deployment mykeycloak \
-o jsonpath='{.metadata.annotations.image\.openshift\.io/triggers}' | jq .
[
  {
    "from": {
      "kind": "ImageStreamTag",
      "name": "keycloak:20"
    },
    "fieldPath": "spec.template.spec.containers[?(.name=="keycloak")].image"
  }
]
```

The `fieldPath` attribute is a JSONPath expression that OpenShift uses to locate the attribute that stores the container image name. OpenShift updates that attribute with the new image name and SHA ID whenever the image stream tag changes.

For a more concise view, use the `oc set triggers` command with the name of the Deployment object as an argument:

```
[user@host ~]$ oc set triggers deployment/mykeycloak
NAME          TYPE      VALUE           AUTO
deployments/mykeycloak config   true  ①
deployments/mykeycloak image    keycloak:20 (keycloak) true ②
```

- ① OpenShift uses the configuration trigger to roll out the deployment whenever you change its configuration, such as to update environment variables or to configure the readiness probe.
- ② OpenShift watches the `keycloak:20` image stream tag that the `keycloak` container uses.

The `true` value under the `AUTO` column indicates that the trigger is enabled.

You can disable the configuration trigger by using the `oc rollout pause` command, and you can re-enable it by using the `oc rollout resume` command.

You can disable the image trigger by adding the `--manual` option to the `oc set triggers` command:

```
[user@host ~]$ oc set triggers deployment/mykeycloak --manual \
--from-image keycloak:20 --containers keycloak
```

You re-enable the trigger by using the `--auto` option:

```
[user@host ~]$ oc set triggers deployment/mykeycloak --auto \
--from-image keycloak:20 --containers keycloak
```

You can remove the triggers from all the containers in the Deployment object by adding the `--remove-all` option to the command:

```
[user@host ~]$ oc set triggers deployment/mykeycloak --remove-all
```

## Rolling out Deployments

A Deployment object with an image trigger automatically rolls out when the image stream tag changes.

The image stream tag might change because you manually update it to point to a new version of the source image. The image stream tag might also change automatically if you configure it for periodic refresh, by adding the `--scheduled` option to the `oc tag` command. When the image stream tag automatically changes, all the Deployment objects with a trigger that refers to that image stream tag also roll out.

## Rolling Back Deployments

OpenShift DeploymentConfig objects and Kubernetes Deployment objects behave differently when you need to roll back because of a malfunctioning image.

For OpenShift DeploymentConfig objects, use the `oc rollout undo` command. The command rolls back the object and disables the image trigger. The command disables the trigger, because otherwise, after rolling back, OpenShift would notice that a new image is available, the malfunctioning image, and would then roll out. You must manually re-enable the image trigger after you fix the issue.

For Kubernetes Deployment objects, you cannot use the `oc rollout undo` command in the event of malfunctioning images, because the command does not disable the image triggers. If you use the command, then OpenShift notices that the deployment is already using the image that the trigger points to, and therefore does not roll back the application.

For Kubernetes Deployment objects, instead of rolling back the deployment, you revert the image stream tag. By reverting the image stream tag, OpenShift rolls out the Deployment object to use the previous image that the image stream tag is pointing to again.

## Managing Image Stream Tags

You can create image streams and image stream tags in several ways. The following commands perform the same operation. They all create the `keycloak` image stream if it does not exist, and then create the `keycloak:20.0.2` image stream tag:

```
[user@host ~]$ oc create istag keycloak:20.0.2 \
--from-image quay.io/keycloak/keycloak:20.0.2

[user@host ~]$ oc import-image keycloak:20.0.2 \
--from quay.io/keycloak/keycloak:20.0.2 --confirm

[user@host ~]$ oc tag quay.io/keycloak/keycloak:20.0.2 keycloak:20.0.2
```

You can rerun the `oc import-image` and `oc tag` commands to update the image stream tag from the source image. If the source image changes, then the commands update the image stream tag to point to that new version. However, you can use the `oc create istag` command only to initially create the image stream tag. You cannot update tags by using that command.

Use the `--help` option for more details about the commands.

You can create several image stream tags that point to the same image. The following command creates the `keycloak:20` image stream tag, which points to the same image as the `keycloak:20.0.2` image stream tag. In other words, the `keycloak:20` image stream tag is an alias for the `keycloak:20.0.2` image stream tag.

```
[user@host ~]$ oc tag --alias keycloak:20.0.2 keycloak:20
```

The `oc describe is` command reports that both tags point to the same image:

```
[user@host ~]$ oc describe is keycloak
Name:      keycloak
Namespace: myproject
...output omitted...

20.0.2 (20)
tagged from quay.io/keycloak/keycloak:20.0.2

* quay.io/keycloak/keycloak@sha256:5569...b311
  3 minutes ago
```

Using aliases is a similar concept to floating tags for container images. Suppose that a new image version is available in the Quay.io repository. You could create an image stream tag for that new image:

```
[user@host ~]$ oc create istag keycloak:20.0.3 \
--from-image quay.io/keycloak/keycloak:20.0.3
imagestreamtag.image.openshift.io/keycloak:20.0.3 created
[user@host ~]$ oc describe is keycloak
Name:      keycloak
Namespace: myproject
...output omitted...

20.0.3
tagged from quay.io/keycloak/keycloak:20.0.3

* quay.io/keycloak/keycloak@sha256:c167...62e9
  36 seconds ago
```

```
20.0.2 (20)
tagged from quay.io/keycloak/keycloak:20.0.2

* quay.io/keycloak/keycloak@sha256:5569...b311
  About an hour ago
```

The `keycloak:20` image stream tag does not change. Therefore, the `Deployment` objects that use that tag do not roll out.

After testing the new image, you can move the `keycloak:20` tag to point to the new image stream tag:

```
[user@host ~]$ oc tag --alias keycloak:20.0.3 keycloak:20
Tag keycloak:20 set up to track keycloak:20.0.3.
[user@host ~]$ oc describe is keycloak
Name:      keycloak
Namespace: myproject
...output omitted...

20.0.3 (20)
tagged from quay.io/keycloak/keycloak:20.0.3

* quay.io/keycloak/keycloak@sha256:c167...62e9
  10 minutes ago

20.0.2
tagged from quay.io/keycloak/keycloak:20.0.2

* quay.io/keycloak/keycloak@sha256:5569...b311
  About an hour ago
```

Because the `keycloak:20` image stream tag points to a new image, OpenShift rolls out all the `Deployment` objects that use that tag.

If the new application does not work as expected, then you can roll back the deployments by resetting the `keycloak:20` tag to the previous image stream tag:

```
[user@host ~]$ oc tag --alias keycloak:20.0.2 keycloak:20
```

By providing a level of indirection, image streams give you control over managing the container images that you use in your OpenShift cluster.



## References

For more information about image triggers, refer to the *Triggering Updates on Image Stream Changes* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#triggering-updates-on-imagestream-changes](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#triggering-updates-on-imagestream-changes)

For more information about image stream tags, refer to the *Tagging Images* section in the *Managing Images* chapter in the Red Hat OpenShift Container Platform 4.14 *Images* documentation at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.14/html-single/images/index#tagging-images](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.14/html-single/images/index#tagging-images)

### **Using Red Hat OpenShift Image Streams with Kubernetes Deployments**

<https://developers.redhat.com/blog/2019/09/20/using-red-hat-openshift-image-streams-with-kubernetes-deployments>

## ► Guided Exercise

# Automatic Image Updates with OpenShift Image Change Triggers

Update an application that references container images indirectly through image streams.

### Outcomes

- Add an image trigger to a deployment.
- Modify an image stream tag to point to a new image.
- Watch the rollout of the application.
- Roll back a deployment to the previous 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 all resources are available for this exercise. It also creates the `updates-triggers` project and deploys a web application with 10 replicas.

The command creates the `/home/student/D0180/labs/updates-triggers/resources.txt` file. The `resources.txt` file contains the name of the images and some commands that you use during the exercise. You can use the file to copy and paste these image names and commands.

```
[student@workstation ~]$ lab start updates-triggers
```

### Instructions

- 1. Log in to the OpenShift cluster as the developer user with the developer password. Use the `updates-triggers` project.
- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `updates-triggers` project as the active project.

```
[student@workstation ~]$ oc project updates-triggers
...output omitted...
```

- 2. Inspect the `versioned-hello` image stream that the `lab` command created.

- 2.1. Verify that the `lab` command enabled the local lookup policy for the `versioned-hello` image stream.

```
[student@workstation ~]$ oc set image-lookup
NAME      LOCAL
versioned-hello  true
```

- 2.2. Verify that the `lab` command created the `versioned-hello:1` image stream tag. The image stream tag refers to the image in the classroom registry by its SHA ID.



### Note

To improve readability, the instructions truncate the SHA-256 strings.

On your system, the commands return the full SHA-256 strings. Also, you must type the full SHA-256 string, to provide such a parameter to a command.

```
[student@workstation ~]$ oc get istag
NAME      IMAGE REFERENCE ...
versioned-hello:1  ...:8443/redhattraining/versioned-hello@sha256:66e0...105e ...
```

- 2.3. Verify that the `lab` command created the `versioned-hello:1` image stream tag from the `registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-123` image.

```
[student@workstation ~]$ oc get istag versioned-hello:1 \
-o jsonpath='{.tag.from.name}{"\n"}'
registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-123
```

- 3. Inspect the `Deployment` object that the `lab` command created. Verify that the application is available from outside the cluster.

- 3.1. List the `Deployment` objects. The `version` deployment retrieved the SHA image ID from the `versioned-hello:1` image stream tag. The `Deployment` object includes a container named `versioned-hello`. You use that information in a later step, when you configure the trigger.

```
[student@workstation ~]$ oc get deployment -o wide
NAME      READY ... CONTAINERS      IMAGES ...
version  10/10 ... versioned-hello  .../versioned-hello@sha256:66e0...105e ...
```

- 3.2. Open a new terminal.
- 3.3. Run the `/home/student/D0180/labs/updates-triggers/curl_loop.sh` script that the `lab` command prepared. The script sends web requests to the application in a loop. Leave the script running and do not interrupt it.

```
[student@workstation ~]$ /home/student/D0180/labs/updates-triggers/curl_loop.sh
Hi!
Hi!
Hi!
Hi!
...output omitted...
```

▶ **4.** Add an image trigger to the Deployment object.

- 4.1. Switch back to the first terminal window, and then use the `oc set triggers` command to add the trigger for the `versioned-hello:1` image stream tag to the `versioned-hello` container.

```
[student@workstation ~]$ oc set triggers deployment/version \
--from-image versioned-hello:1 --containers versioned-hello
deployment.apps/version triggers updated
```

- 4.2. Review the definition of the trigger from the `image.openshift.io/triggers` annotation.

```
[student@workstation ~]$ oc get deployment version \
-o jsonpath='{.metadata.annotations.image\.openshift\.io/triggers}' | jq .
[
{
  "from": {
    "kind": "ImageStreamTag",
    "name": "versioned-hello:1"
  },
  "fieldPath": "spec.template.spec.containers[?(.name==\"versioned-hello
\")]".image"
}
]
```

▶ **5.** Update the `versioned-hello:1` image stream tag to point to the `1-125` tag of the `registry.ocp4.example.com:8443/redhattraining/versioned-hello` image. Watch the output of the `curl_loop.sh` script to verify that the Deployment object automatically rolls out.

- 5.1. Use the `oc tag` command to update the `versioned-hello:1` tag.

```
[student@workstation ~]$ oc tag \
  registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-125 \
  versioned-hello:1
Tag versioned-hello:1 set to registry.ocp4.example.com:8443/redhattraining/
versioned-hello:1-125.
```

- 5.2. Changing the image stream tag triggered a rolling update. Watch the output of the `curl_loop.sh` script in the second terminal.

Before the update, only pods that use the earlier version of the image reply. During the rolling updates, both old and new pods respond. After the update, only pods that

run the latest version of the image reply. The following output probably differs on your system.

```
...output omitted...
Hi!
Hi!
Hi!
Hi!
Hi! v1.1
Hi! v1.1
Hi!
Hi! v1.1
Hi!
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi! v1.1
...output omitted...
```

Do not stop the script.

- ▶ 6. Inspect the Deployment object and the image stream.
- 6.1. List the `version` deployment and notice that the image changed.
- ```
[student@workstation ~]$ oc get deployment version -o wide
NAME      READY ... CONTAINERS      IMAGES ...
version  10/10 ... versioned-hello  .../versioned-hello@sha256:834d...fcb4 ...
```
- 6.2. Display the details of the `versioned-hello` image stream. The `versioned-hello:1` image stream tag points to the image with the same SHA ID as in the Deployment object.  
Notice that the preceding image is still available. In the following step, you roll back to that image by specifying its SHA ID.
- ```
[student@workstation ~]$ oc describe is versioned-hello
Name:          versioned-hello
Namespace:     updates-triggers
...output omitted...

1
tagged from registry.ocp4.example.com:8443/redhattraining/versioned-hello:1-125

* registry.ocp4.example.com:8443/.../versioned-hello@sha256:834d...fcb4
  6 minutes ago
registry.ocp4.example.com:8443/.../versioned-hello@sha256:66e0...105e
  37 minutes ago
```
- ▶ 7. Roll back the Deployment object by reverting the `versioned-hello:1` image stream tag.
- 7.1. Use the `oc tag` command. For the source image, copy and paste the old image name and the SHA ID from the output of the preceding command.

```
[student@workstation ~]$ oc tag \
registry.ocp4.example.com:8443/redhattraining/versioned-hello@sha256:66e0...105e \
versioned-hello:1
Tag versioned-hello:1 set to registry.ocp4.example.com:8443/redhattraining/
versioned-hello@sha256:66e0...105e.
```

- 7.2. Watch the output of the `curl_loop.sh` script in the second terminal. The pods that run the v1.0 version of the application are responding again. The following output probably differs on your system.

```
...output omitted...
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi!
Hi! v1.1
Hi! v1.1
Hi! v1.1
Hi!
Hi!
Hi!
Hi!
...output omitted...
```

Press `Ctrl+C` to quit the script. Close that second terminal when 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 updates-triggers
```

## ► Lab

# Manage Application Updates

Update two live applications to their latest releases as identified by non-floating tags.

## Outcomes

You should be able to configure Deployment objects with images and triggers, and configure image stream tags and aliases.

## 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. It also creates the `updates-review` project and deploys two applications, `app1` and `app2`, in that project.

The command creates the `/home/student/D0180/labs/updates-review/resources.txt` file. The `resources.txt` file contains the name of the images that you use during the exercise. You can use the file to copy and paste these image names.

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

## Instructions

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your workstation machine.

Log in to the OpenShift cluster as the developer user with the developer password. Use the `updates-review` project for your work.

1. Your team created the `app1` deployment in the `updates-review` project from the `registry.ocp4.example.com:8443/redhattraining/php-ssl:latest` container image. Recently, a developer in your organization pushed a new version of the image and then reassigned the `latest` tag to that version.  
Reconfigure the `app1` deployment to use the `1-222` static tag instead of the `latest` floating tag, to prevent accidental redeployment of your application with untested image versions that your developers can publish at any time.
2. The `app2` deployment is using the `php-ssl:1` image stream tag, which is an alias for the `php-ssl:1-222` image stream tag.  
Enable image triggering for the `app2` deployment, so that whenever the `php-ssl:1` image stream tag changes, OpenShift rolls out the application. You test your configuration in a later step, when you reassign the `php-ssl:1` alias to a new image stream tag.
3. A new image version, `registry.ocp4.example.com:8443/redhattraining/php-ssl:1-234`, is available in the container registry. Your QA team tested and approved that version. It is ready for production.

Create the `php-ssl:1-234` image stream tag that points to the new image. Move the `php-ssl:1` image stream tag alias to the new `php-ssl:1-234` image stream tag. Verify that the `app2` application redeploys.

## 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 updates-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 updates-review
```

## ► Solution

# Manage Application Updates

Update two live applications to their latest releases as identified by non-floating tags.

### Outcomes

You should be able to configure Deployment objects with images and triggers, and configure image stream tags and aliases.

### 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. It also creates the `updates-review` project and deploys two applications, `app1` and `app2`, in that project.

The command creates the `/home/student/D0180/labs/updates-review/resources.txt` file. The `resources.txt` file contains the name of the images that you use during the exercise. You can use the file to copy and paste these image names.

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

### Instructions

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your workstation machine.

Log in to the OpenShift cluster as the developer user with the developer password. Use the `updates-review` project for your work.

1. Your team created the `app1` deployment in the `updates-review` project from the `registry.ocp4.example.com:8443/redhattraining/php-ssl:latest` container image. Recently, a developer in your organization pushed a new version of the image and then reassigned the `latest` tag to that version.  
Reconfigure the `app1` deployment to use the `1-222` static tag instead of the `latest` floating tag, to prevent accidental redeployment of your application with untested image versions that your developers can publish at any time.

- 1.1. Log in to the OpenShift cluster.

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

- 1.2. Set the `updates-review` project as the active project.

```
[student@workstation ~]$ oc project updates-review  
...output omitted...
```

- 1.3. Verify that the app1 deployment uses the latest tag. Retrieve the container name.

```
[student@workstation ~]$ oc get deployment/app1 -o wide  
NAME READY ... CONTAINERS IMAGES ...  
app1 1/1 ... php-ssl registry...:8443/redhattraining/php-ssl:latest ...
```

- 1.4. In the Deployment object, change the image to registry.ocp4.example.com:8443/redhattraining/php-ssl:1-222.

```
[student@workstation ~]$ oc set image deployment/app1 \  
php-ssl=registry.ocp4.example.com:8443/redhattraining/php-ssl:1-222  
deployment.apps/app1 image updated
```

- 1.5. Verify your work.

```
[student@workstation ~]$ oc get deployment/app1 -o wide  
NAME READY ... CONTAINERS IMAGES ...  
app1 1/1 ... php-ssl registry...:8443/redhattraining/php-ssl:1-222 ...
```

2. The app2 deployment is using the php-ssl:1 image stream tag, which is an alias for the php-ssl:1-222 image stream tag.

Enable image triggering for the app2 deployment, so that whenever the php-ssl:1 image stream tag changes, OpenShift rolls out the application. You test your configuration in a later step, when you reassign the php-ssl:1 alias to a new image stream tag.

- 2.1. Retrieve the container name from the Deployment object.

```
[student@workstation ~]$ oc get deployment/app2 -o wide  
NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS ...  
app2 1/1 1 1 21m php-ssl ...
```

- 2.2. Add the image trigger to the Deployment object.

```
[student@workstation ~]$ oc set triggers deployment/app2 \  
--from-image php-ssl:1 --containers php-ssl  
deployment.apps/app2 triggers updated
```

- 2.3. Verify your work.

```
[student@workstation ~]$ oc set triggers deployment/app2  
NAME TYPE VALUE AUTO  
deployments/app2 config true  
deployments/app2 image php-ssl:1 (php-ssl) true
```

3. A new image version, registry.ocp4.example.com:8443/redhattraining/php-ssl:1-234, is available in the container registry. Your QA team tested and approved that version. It is ready for production.

Create the `php-ssl:1-234` image stream tag that points to the new image. Move the `php-ssl:1` image stream tag alias to the new `php-ssl:1-234` image stream tag. Verify that the `app2` application redeploys.

- 3.1. Create the `php-ssl:1-234` image stream tag.

```
[student@workstation ~]$ oc create istag php-ssl:1-234 \
--from-image registry.ocp4.example.com:8443/redhattraining/php-ssl:1-234
imagestreamtag.image.openshift.io/php-ssl:1-234 created
```

- 3.2. Move the `php-ssl:1` alias to the new `php-ssl:1-234` image stream tag.

```
[student@workstation ~]$ oc tag --alias php-ssl:1-234 php-ssl:1
Tag php-ssl:1 set up to track php-ssl:1-234.
```

- 3.3. Verify that the `app2` application rolls out. The names of the replica sets on your system probably differ.

```
[student@workstation ~]$ oc describe deployment/app2
Name:                      app2
Namespace:                  updates-review
...output omitted...
Events:
  Type    ...  Age            Message
  ----  ...
  Normal  ...  33m           ... Scaled up replica set app2-7dd589f6d5 to 1
  Normal  ...  3m30s         ... Scaled up replica set app2-7bf5b7787 to 1
  Normal  ...  3m28s         ... Scaled down replica set app2-7dd589f6d5 to 0 from 1
```

## 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 updates-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 updates-review
```

# Summary

---

- You reference container images by tags or by SHA IDs. Image developers assign tags to images. Container registries compute and assign SHA IDs to images.
- Deployment objects have an `imagePullPolicy` attribute that specifies how compute nodes pull the image from the registry.
- A deployment strategy is a way of changing or upgrading your application to minimize the impact of those changes.
- Deployment objects support the rolling update and the re-create deployment strategies.
- OpenShift image stream and image stream tag resources provide stable references to container images.
- Kubernetes workload resources, such as Deployments and Jobs, can use image streams. You must create the image streams in the same project as the Kubernetes resources, and you must enable the local lookup policy in the image streams to use them.
- You can configure image monitoring in deployments so that OpenShift rolls out the application whenever the image stream tag changes.

## Chapter 8

# Comprehensive Review

### Goal

Review tasks from *Red Hat OpenShift Administration I: Operating a Production Cluster*.

### Sections

- Comprehensive Review

### Lab

- Deploy Web Applications
- Troubleshoot and Scale 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 I: Operating a Production Cluster*.

## Reviewing Red Hat OpenShift Administration I: Operating 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, Introduction to Kubernetes and OpenShift**

Identify the main Kubernetes cluster services and OpenShift platform services and monitor them by using the web console.

- Describe the main characteristics of containers and Kubernetes.
- Describe the relationship between OpenShift, Kubernetes, and other Open Source projects, and list key features of Red Hat OpenShift products and editions.
- Navigate the OpenShift web console to identify running applications and cluster services.
- Navigate the Events, Compute, and Observe panels of the OpenShift web console to assess the overall state of a cluster.

### **Chapter 2, Kubernetes and OpenShift Command-line Interfaces and APIs**

Access an OpenShift cluster by using the command line and query its Kubernetes API resources to assess the health of a cluster.

- Access an OpenShift cluster by using the Kubernetes and OpenShift command-line interfaces.
- Query, format, and filter attributes of Kubernetes resources.
- Query the health of essential cluster services and components.

### **Chapter 3, Run Applications as Containers and Pods**

Run and troubleshoot containerized applications as unmanaged Kubernetes pods.

- Run containers inside pods and identify the host OS processes and namespaces that the containers use.
- Find containerized applications in container registries and get information about the runtime parameters of supported and community container images.
- Troubleshoot a pod by starting additional processes on its containers, changing their ephemeral file systems, and opening short-lived network tunnels.

## **Chapter 4, Deploy Managed and Networked Applications on Kubernetes**

Deploy applications and expose them to network access from inside and outside a Kubernetes cluster.

- Identify the main resources and settings that Kubernetes uses to manage long-lived applications and demonstrate how OpenShift simplifies common application deployment workflows.
- Deploy containerized applications as pods that Kubernetes workload resources manage.
- Interconnect applications pods inside the same cluster by using Kubernetes services.
- Expose applications to clients outside the cluster by using Kubernetes ingress and OpenShift routes.

## **Chapter 5, Manage Storage for Application Configuration and Data**

Externalize application configurations in Kubernetes resources and provision storage volumes for persistent data files.

- Configure applications by using Kubernetes secrets and configuration maps to initialize environment variables and to provide text and binary configuration files.
- Provide applications with persistent storage volumes for block and file-based data.
- Match applications with storage classes that provide storage services to satisfy application requirements.
- Deploy applications that scale without sharing storage.

## **Chapter 6, Configure Applications for Reliability**

Configure applications to work with Kubernetes for high availability and resilience.

- Describe how Kubernetes tries to keep applications running after failures.
- Describe how Kubernetes uses health probes during deployment, scaling, and failover of applications.
- Configure an application with resource requests so Kubernetes can make scheduling decisions.
- Configure an application with resource limits so Kubernetes can protect other applications from it.
- Configure a horizontal pod autoscaler for an application.

## **Chapter 7, Manage Application Updates**

Manage reproducible application updates and rollbacks of code and configurations.

- Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes.
- Update applications with minimal downtime by using deployment strategies.
- Ensure reproducibility of application deployments by using image streams and short image names.

- Ensure automatic update of application pods by using image streams with Kubernetes workload resources.

## ▶ Lab

# Deploy Web Applications

Use image streams with Kubernetes workload resources to ensure reproducibility of application deployments.

Configure applications by using Kubernetes secrets to initialize environment variables.

Provide applications with persistent storage volumes.

Expose applications to clients outside the cluster.

## Outcomes

You should be able to create and configure OpenShift and Kubernetes resources, such as projects, secrets, deployments, persistent volumes, services, and routes.

## 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. The command also creates the `/home/student/D0180/labs/compreview-deploy/resources.txt` file. The `resources.txt` file contains the URLs of your OpenShift cluster and the image names that you use in the exercise. You can use the file to copy and paste these URLs and image names.

```
[student@workstation ~]$ lab start compreview-deploy
```

## Specifications

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your workstation machine.

The URL of the OpenShift web console is `https://console-openshift-console.apps.ocp4.example.com`. When you access the web console, select **Red Hat Identity Management** as the authentication mechanism.

Log in to the OpenShift cluster as the developer user with the developer password. The password for the `admin` user is `redhatocp`, although you do not need administrator privileges to complete the exercise.

In this exercise, you deploy a web application and its database for testing purposes. The resulting configuration is not ready for production, because you do not configure probes and resource limits, which are required for production. Another comprehensive review exercise covers these subjects.

Perform the following tasks to complete the exercise:

- Create a project named `review` to store your work.

- Configure your project so that its workloads refer to the database image by the `mysql8:1` short name.
  - The short name must point to the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-228` container image. The database image name and its source registry are expected to change in the near future, and you want to isolate your workloads from that change.

The classroom setup copied the image from the Red Hat Ecosystem Catalog. The original image is `registry.redhat.io/rhel9/mysql-80:1-228`.

- Ensure that the workload resources in the `review` project can use the `mysql8:1` resource. You create these workload resources in a later step.
- Create the `dbparams` secret to store the MySQL database parameters. Both the database and the front-end deployment need these parameters. The `dbparams` secret must include the following variables:

Name	Value
user	operator1
password	redhat123
database	quotesdb

- Create the `quotesdb` deployment and configure it as follows:
    - Use the `mysql8:1` image for the deployment.
    - The database must automatically roll out whenever the source container in the `mysql8:1` resource changes.
- To test your configuration, you can change the `mysql8:1` image to point to the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237` container image that the classroom provides, and then verify that the `quotesdb` deployment rolls out. Remember to reset the `mysql8:1` image to the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-228` container image before grading your work.
- Define the following environment variables in the deployment from the keys in the `dbparams` secret:

Environment variable	dbparams secret key
<code>MYSQL_USER</code>	user
<code>MYSQL_PASSWORD</code>	password
<code>MYSQL_DATABASE</code>	database

- Ensure that OpenShift preserves the database data between pod restarts. This data does not consume more than 2 GiB of disk space. The MySQL database stores its data under the `/var/lib/mysql` directory. Use the `lvms-vg1` storage class for the volume.
- Create a `quotesdb` service to make the database available to the front-end web application. The database service is listening on port 3306.

- Create the **frontend** deployment and configure it as follows:
  - Use the `registry.ocp4.example.com:8443/redhattraining/famous-quotes:2-42` image. For this deployment, you refer to the image by its full name, because your organization develops the image and controls its release process.
  - Define the following environment variables in the deployment:

Environment variable name	Value
<code>QUOTES_USER</code>	The user key from the <code>dbparams</code> secret
<code>QUOTES_PASSWORD</code>	The password key from the <code>dbparams</code> secret
<code>QUOTES_DATABASE</code>	The database key from the <code>dbparams</code> secret
<code>QUOTES_HOSTNAME</code>	<code>quotesdb</code>

- You cannot yet test the application from outside the cluster. Expose the **frontend** deployment so that the application can be reached at `http://frontend-review.apps.ocp4.example.com`.

The **frontend** deployment is listening to port 8000.

When you access the `http://frontend-review.apps.ocp4.example.com` URL, the application returns a list of quotations from famous authors.

## 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-deploy
```

## 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-deploy
```

## ► Solution

# Deploy Web Applications

Use image streams with Kubernetes workload resources to ensure reproducibility of application deployments.

Configure applications by using Kubernetes secrets to initialize environment variables.

Provide applications with persistent storage volumes.

Expose applications to clients outside the cluster.

## Outcomes

You should be able to create and configure OpenShift and Kubernetes resources, such as projects, secrets, deployments, persistent volumes, services, and routes.

## 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. The command also creates the `/home/student/D0180/labs/comprevew-deploy/resources.txt` file. The `resources.txt` file contains the URLs of your OpenShift cluster and the image names that you use in the exercise. You can use the file to copy and paste these URLs and image names.

```
[student@workstation ~]$ lab start comprevew-deploy
```

1. Log in to the OpenShift cluster from the command line, and then create the `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 `review` project.

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

2. Create the `mysql8:1` image stream tag from the `registry.ocp4.example.com:8443/rhel9/mysql-80:1-228` image. Enable image stream resolution for the `mysql8` image stream so that Kubernetes resources in the current project can use it.

- 2.1. Use the `oc create istag` command to create the image stream and the image stream tag.

```
[student@workstation ~]$ oc create istag mysql8:1 \
--from-image registry.ocp4.example.com:8443/rhel9/mysql-80:1-228
imagestreamtag.image.openshift.io/mysql8:1 created
```

- 2.2. Use the `oc set image-lookup` command to enable image lookup resolution.

```
[student@workstation ~]$ oc set image-lookup mysql8
imagestream.image.openshift.io/mysql8 image lookup updated
```

- 2.3. Run the `oc set image-lookup` command without any arguments to verify your work.

```
[student@workstation ~]$ oc set image-lookup
NAME      LOCAL
mysql8   true
```

3. Create the dbparams secret.

```
[student@workstation ~]$ oc create secret generic dbparams \
--from-literal user=operator1 --from-literal password=redhat123 \
--from-literal database=quotesdb
secret/dbparams created
```

4. Create the quotesdb deployment from the mysql8:1 image stream tag. Set the number of replicas to zero, to prevent OpenShift from deploying the database before you finish its configuration.

```
[student@workstation ~]$ oc create deployment quotesdb --image mysql8:1 \
--replicas 0
deployment.apps/quotesdb created
```

5. Add an image trigger to the quotesdb deployment.

- 5.1. Retrieve the name of the container from the quotesdb deployment.

```
[student@workstation ~]$ oc get deployment quotesdb -o wide
NAME      READY     UP-TO-DATE   AVAILABLE   AGE      CONTAINERS ...
quotesdb  0/0       0           0           11s     mysql8     ...
```

- 5.2. Use the `oc set triggers` command to add the trigger for the mysql8:1 image stream tag to the mysql8 container.

```
[student@workstation ~]$ oc set triggers deployment/quotesdb \
--from-image mysql8:1 --containers mysql8
deployment.apps/quotesdb triggers updated
```

6. Add environment variables to the quotesdb deployment from the dbparams secret. Add the MySQL\_ prefix to each variable name.

```
[student@workstation ~]$ oc set env deployment/quotesdb \
--from secret/dbparams --prefix MYSQL_
deployment.apps/quotesdb updated
```

7. Add a 2 GiB persistent volume to the quotesdb deployment. Use the lvms-vg1 storage class. Inside the pods, mount the volume under the /var/lib/mysql directory.

```
[student@workstation ~]$ oc set volumes deployment/quotesdb --add \
--claim-class lvms-vg1 --claim-size 2Gi --mount-path /var/lib/mysql
info: Generated volume name: volume-n7xpd
deployment.apps/quotesdb volume updated
```

8. Start the database by scaling up the quotesdb deployment to one replica.

- 8.1. Scale up the deployment.

```
[student@workstation ~]$ oc scale deployment/quotesdb --replicas 1
deployment.apps/quotesdb scaled
```

- 8.2. Wait for the pod to start. You might have to rerun the command several times for the pod to report a Running status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
quotesdb-99f9b4ff8-ggs7z   1/1     Running   0          4s
```

9. Create the quotesdb service for the quotesdb deployment. The database server is listening on port 3306.

- 9.1. Use the `oc expose deployment` command to create the service.

```
[student@workstation ~]$ oc expose deployment quotesdb --port 3306
service/quotesdb exposed
```

- 9.2. Verify that OpenShift associates the IP address of the MySQL server with the endpoint. The endpoint IP address on your system probably differs.

```
[student@workstation ~]$ oc describe service quotesdb
Name:           quotesdb
Namespace:      review
...output omitted...
TargetPort:     3306/TCP
Endpoints:      10.8.0.123:3306
Session Affinity: None
Events:         <none>
```

10. Create the frontend deployment from the `registry.ocp4.example.com:8443/redhattraining/famous-quotes:2-42` image. Set the number of replicas to zero, to prevent OpenShift from deploying the application before you finish its configuration.

```
[student@workstation ~]$ oc create deployment frontend \
--image registry.ocp4.example.com:8443/redhattraining/famous-quotes:2-42 \
--replicas 0
deployment.apps/frontend created
```

11. Add environment variables to the `frontend` deployment from the `dbparams` secret, and add the `QUOTES_HOSTNAME` variable with the `quotesdb` value.
  - 11.1. Add the variables from the `dbparams` secret. Add the `QUOTES_` prefix to each variable name.

```
[student@workstation ~]$ oc set env deployment/frontend \
--from secret/dbparams --prefix QUOTES_
deployment.apps/frontend updated
```

- 11.2. Declare the `QUOTES_HOSTNAME` variable.

```
[student@workstation ~]$ oc set env deployment/frontend QUOTES_HOSTNAME=quotesdb
deployment.apps/frontend updated
```

12. Start the application by scaling up the `frontend` deployment to one replica.

- 12.1. Scale up the deployment.

```
[student@workstation ~]$ oc scale deployment/frontend --replicas 1
deployment.apps/frontend scaled
```

- 12.2. Wait for the pod to start. You might have to rerun the command several times for the pod to report a `Running` status. The name of the pod on your system probably differs.

NAME	READY	STATUS	RESTARTS	AGE
<code>frontend-86cdd7c7bf-hpnwz</code>	1/1	<code>Running</code>	0	44s
<code>quotesdb-99f9b4ff8-ggs7z</code>	1/1	Running	0	2m11s

13. Expose the `frontend` deployment so that the application is accessible from outside the cluster. The web application is listening on port 8000.

- 13.1. Create the `frontend` service for the `frontend` deployment.

```
[student@workstation ~]$ oc expose deployment frontend --port 8000
service/frontend exposed
```

- 13.2. Create the route.

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

- 13.3. Retrieve the application URL from the route.

```
[student@workstation ~]$ oc get route
NAME      HOST/PORT          PATH  SERVICES ...
frontend  frontend-review.apps.ocp4.example.com    frontend ...
```

13.4. Use the curl command to test the application.

```
[student@workstation ~]$ curl http://frontend-review.apps.ocp4.example.com
<html>
<head>
  <title>Quotes</title>
</head>
<body>

  <h1>Quote List</h1>

  <ul>

    <li>1: When words fail, music speaks.
- William Shakespeare
  </li>
  ...output omitted...
```

## 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-deploy
```

## 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-deploy
```

## ► Lab

# Troubleshoot and Scale Applications

Navigate the OpenShift web console to identify CPU-consuming workloads.

Troubleshoot and fix a failed MySQL pod.

Manually scale an application.

Configure health probes.

## Outcomes

You should be able to troubleshoot malfunctioning workloads, configure deployments, and scale applications.

## 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. The command also creates the `comprevew-scale` project and deploys some applications in that project.

The command creates the `/home/student/D0180/labs/comprevew-scale/resources.txt` file. The `resources.txt` file contains the URLs of your OpenShift cluster and the name of the images that you use during the exercise. You can use the file to copy and paste these URLs and image names.

```
[student@workstation ~]$ lab start comprevew-scale
```

## Specifications

The API URL of your OpenShift cluster is `https://api.ocp4.example.com:6443`, and the `oc` command is already installed on your **workstation** machine.

The URL of the OpenShift web console is `https://console-openshift-console.apps.ocp4.example.com`. When you access the web console, select **Red Hat Identity Management** as the authentication mechanism.

Log in to the OpenShift cluster as the **developer** user with the **developer** password. The password for the **admin** user is **redhatocp**.

Perform the following tasks to complete the comprehensive review:

- A pod in the cluster is consuming excessive CPU and is interfering with other tasks. Identify the pod and remove its workload.
- The `comprevew-scale` project already includes a web application at `http://frontend-comprevew-scale.apps.ocp4.example.com`. When you access this URL, the application returns a list of quotations from famous authors. The application is broken for now, and is missing some configuration to be ready for production.

The application uses two Kubernetes Deployment objects. The **frontend** deployment provides the application web pages, and relies on the **quotesdb** deployment that runs a MySQL database. The `lab` command already created the services and routes that connect the application components and that make the application available from outside the cluster.

Fix the application and make it ready for production:

- The **quotesdb** deployment in the `comprevew-scale` project starts a MySQL server, but the database is failing. Review the logs of the pod to identify and then fix the issue.

Use the following parameters for the database:

Name	Value
Username	operator1
Password	redhat123
Database name	quotes

- You security team validated a new version of the MySQL container image that fixes a security issue. The new container image is `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237`.

Update the **quotesdb** deployment to use this image. Ensure that the database redeploys.

The classroom setup copied the image from the Red Hat Ecosystem Catalog. The original image is `registry.redhat.io/rhel9/mysql-80:1-237`.

- Add a probe to the **quotesdb** deployment so that OpenShift can detect when the database is ready to accept requests. Use the `mysqladmin ping` command for the probe.
- Add a second probe that regularly verifies the status of the database. Use the `mysqladmin ping` command as well.
- Configure CPU and memory usage for the **quotesdb** deployment. The deployment needs 200 millicores of CPU and 256 MiB of memory to run, and you must restrict its CPU usage to 500 millicores and its memory usage to 1 GiB.
- Add a probe to the **frontend** deployment so that OpenShift can detect when the web application is ready to accept requests. The application is ready when an HTTP request on port 8000 to the `/status` path is successful.
- Add a second probe that regularly verifies the status of the web front end. The front end works as expected when an HTTP request on port 8000 to the `/env` path is successful.
- Configure CPU and memory usage for the **frontend** deployment. The deployment needs 200 millicores of CPU and 256 MiB of memory to run, and you must restrict its CPU usage to 500 millicores and its memory usage to 512 MiB.
- Scale the **frontend** application to three pods to accommodate for the estimated production load.
- To verify your work, access the `http://frontend-comprevew-scale.apps.ocp4.example.com` URL. The application returns a list of quotations from famous authors.

## 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-scale
```

## 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-scale
```

## ► Solution

# Troubleshoot and Scale Applications

Navigate the OpenShift web console to identify CPU-consuming workloads.

Troubleshoot and fix a failed MySQL pod.

Manually scale an application.

Configure health probes.

### Outcomes

You should be able to troubleshoot malfunctioning workloads, configure deployments, and scale applications.

### 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. The command also creates the `comprevew-scale` project and deploys some applications in that project.

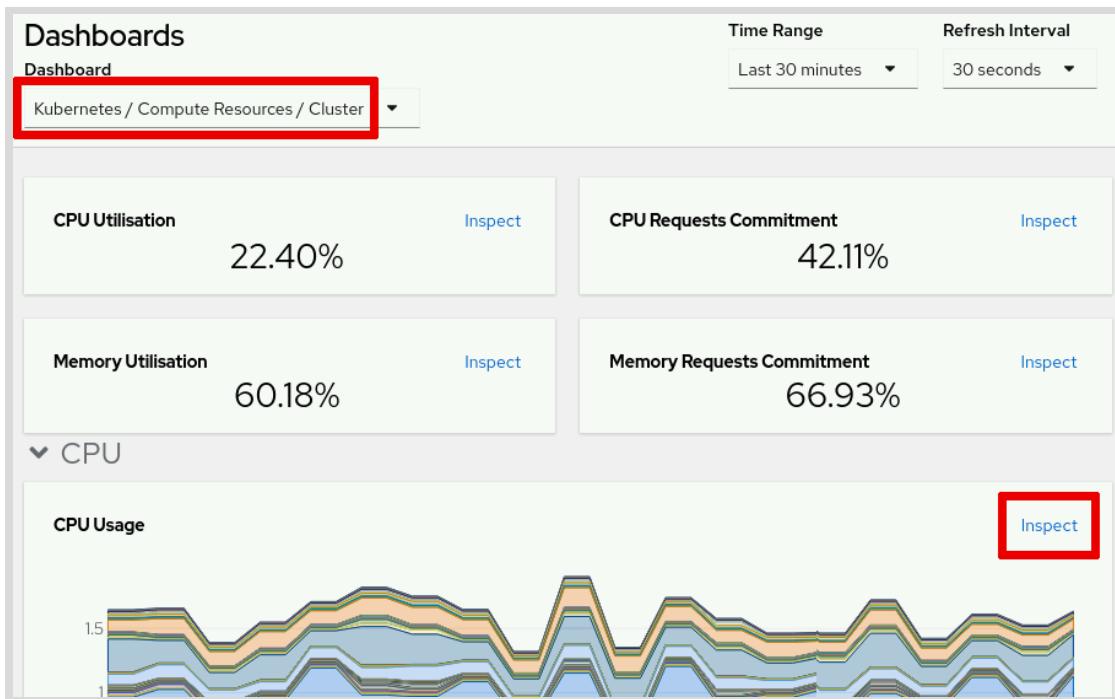
The command creates the `/home/student/D0180/labs/comprevew-scale/resources.txt` file. The `resources.txt` file contains the URLs of your OpenShift cluster and the name of the images that you use during the exercise. You can use the file to copy and paste these URLs and image names.

```
[student@workstation ~]$ lab start comprevew-scale
```

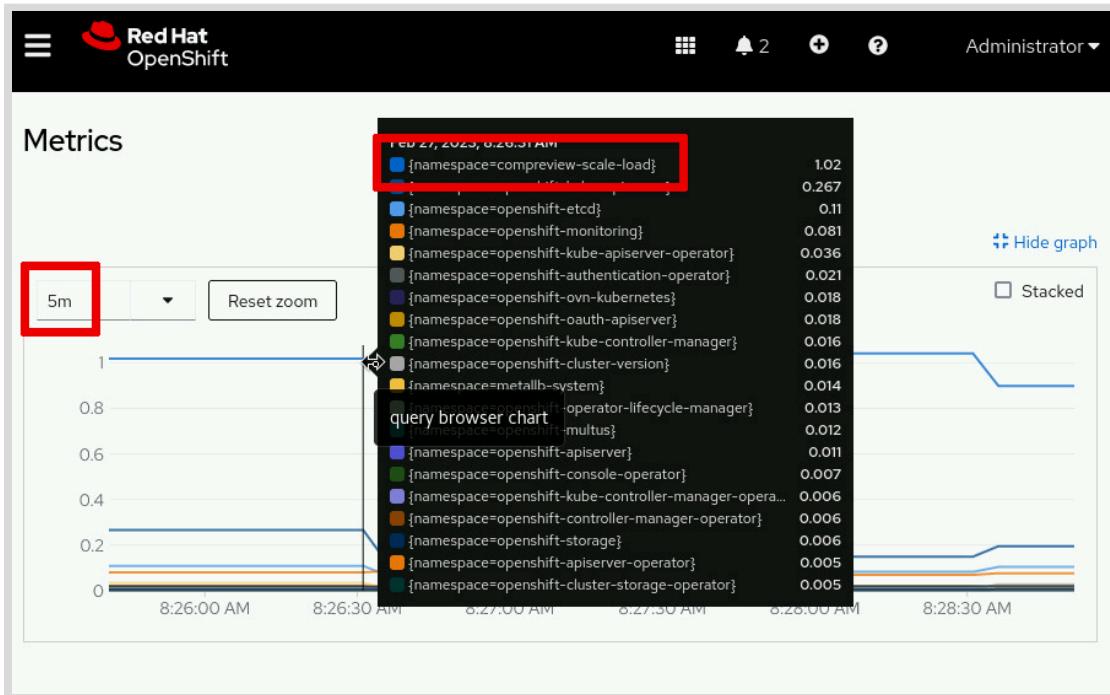
1. Use the OpenShift web console to identify and then delete the pod that consumes excessive CPU.
  - 1.1. Use a web browser to access the `https://console-openshift-console.apps.ocp4.example.com` URL.
  - 1.2. Select **Red Hat Identity Management**, and then log in as the **admin** user with the `redhatocp` password. Click **Skip tour** if the **Welcome to the Developer Perspective** message is displayed.
  - 1.3. Switch to the **Administrator** perspective and then navigate to **Observe > Dashboards**.

The screenshot shows the Red Hat OpenShift web interface. The left sidebar is expanded to show the 'Administrator' dropdown and the 'Dashboards' option, which is highlighted with a red box. The main dashboard area shows a line chart titled 'API Request Duration by Verb - 99th Percentile' for the 'kube-apiserver' over the last 30 minutes. The chart displays four data series: APPLY (blue), DELETE (orange), GET (green), and LIST (purple). The LIST series shows the highest peaks, reaching approximately 1.0 at around 7:40 AM.

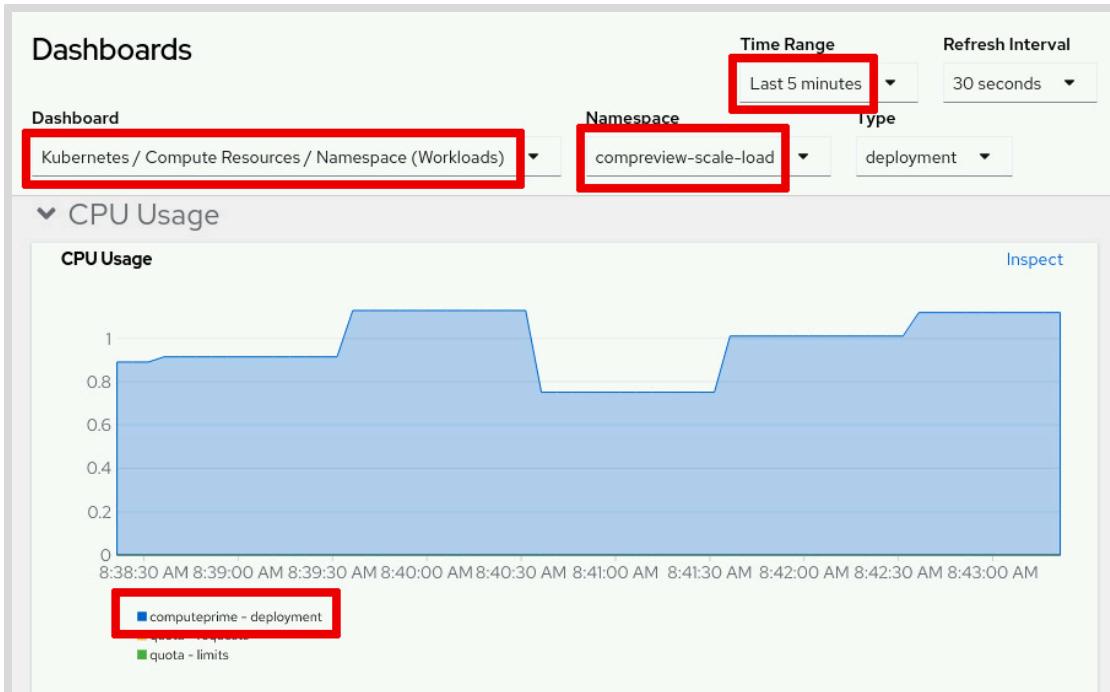
- 1.4. Select the **Kubernetes / Compute Resources / Cluster** dashboard, and then click **Inspect** in the CPU Usage graph.



- Set the zoom to five minutes and then hover over the graph. Notice that the interface lists the `comprevew-scale-load` namespace in the first position, which indicated that this namespace is the first CPU consumer.



- Navigate to **Observe > Dashboards** and then select the **Kubernetes / Compute Resources / Namespace (Workloads)** dashboard. Select the `comprevew-scale-load` namespace and then set the time range to the last five minutes. The `computeprime` deployment is the workload that consumes excessive CPU.



- 1.7. Navigate to **Workloads > Deployments** and then select the **comprevew-scale-load** project. Select the menu for the **computeprime** deployment and then click **Delete Deployment**. Click **Delete** to confirm the operation.

The screenshot shows the OpenShift web interface. On the left, there's a sidebar with 'Administrator' at the top, followed by 'Home', 'Operators', 'Workloads' (with 'Pods' and 'Deployments' under it, both highlighted with red boxes), 'DeploymentConfigs', 'StatefulSets', 'Secrets', 'ConfigMaps', 'CronJobs', 'Jobs', and 'Resource Quotas'. The main area shows a 'Project: comprevew-scale-load' dropdown (also with a red box) and a 'Deployments' table. The table has a single row for 'computeprime' with '1 of 1 pods'. To the right of the table is a context menu with options like 'Edit Pod count', 'Add HorizontalPodAutoscaler', etc., ending with 'Delete Deployment' which is also highlighted with a red box.

2. Review the logs of the pod that is failing for the **quotesdb** deployment. Set the missing environment variables in the **quotesdb** deployment.

- 2.1. Log in to the OpenShift cluster from the command line.

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

- 2.2. Set the **comprevew-scale** project as the active project.

```
[student@workstation ~]$ oc project comprevew-scale
...output omitted...
```

- 2.3. List the pods to identify the failing pod from the **quotesdb** deployment. The names of the pods on your system probably differ.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS        RESTARTS   AGE
frontend-5fb85b4c75-5s7xr  0/1    CrashLoopBackOff  14 (2m52s ago)  50m
quotesdb-9b9776479-4z4g9  0/1    CrashLoopBackOff  14 (3m4s ago)  50m
```

- 2.4. Retrieve the logs for the failing pod. Some environment variables are missing.

```
[student@workstation ~]$ oc logs quotesdb-9b9776479-4z4g9
=> sourcing 20-validate-variables.sh ...
You must either specify the following environment variables:
  MYSQL_USER (regex: '^[_a-zA-Z0-9]+$')
  MYSQL_PASSWORD (regex: '[_a-zA-Z0-9~!@#$%^&*()=-<>,._?;:|]+$')
  MYSQL_DATABASE (regex: '^[_a-zA-Z0-9]+$')
...output omitted...
```

- 2.5. Add the missing environment variables to the quotesdb deployment.

```
[student@workstation ~]$ oc set env deployment/quotesdb \
  MYSQL_USER=operator1 MYSQL_PASSWORD=redhat123 MYSQL_DATABASE=quotes
deployment.apps/quotesdb updated
```

3. Update the MySQL container image for the quotesdb deployment.

- 3.1. Retrieve the name of the container that is running inside the pod. You need the container name to update its image.

```
[student@workstation ~]$ oc get deployment/quotesdb -o wide
NAME      READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS ...
quotesdb  1/1     1           1           59m   mysql-80   ...
```

- 3.2. Set the image to `registry.ocp4.example.com:8443/rhel9/mysql-80:1-237`.

```
[student@workstation ~]$ oc set image deployment/quotesdb \
  mysql-80=registry.ocp4.example.com:8443/rhel9/mysql-80:1-237
deployment.apps/quotesdb image updated
```

- 3.3. Verify your work.

```
[student@workstation ~]$ oc get deployment/quotesdb -o wide
NAME      ... CONTAINERS   IMAGES
quotesdb  ... mysql-80    registry.ocp4.example.com:8443/rhel9/mysql-80:1-237
```

- 3.4. Wait for the deployment to roll out. You might have to rerun the command several times for the pod to report a Running status. The name of the pod on your system probably differs.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS            RESTARTS   AGE
frontend-5fb85b4c75-5s7xr  0/1    CrashLoopBackOff  15 (3m39s ago)  56m
quotesdb-54d64749c4-chhq6  1/1    Running          0          106s
```

4. Add a readiness and a liveness probe to the quotesdb deployment that runs the `mysqladmin ping` command.

- 4.1. Use the `oc set probe` command with the `--readiness` option to add the readiness probe.

```
[student@workstation ~]$ oc set probe deployment/quotesdb \
--readiness -- mysqladmin ping
deployment.apps/quotesdb probes updated
```

- 4.2. Use the `oc set probe` command with the `--liveness` option to add the liveness probe.

```
[student@workstation ~]$ oc set probe deployment/quotesdb \
--liveness -- mysqladmin ping
deployment.apps/quotesdb probes updated
```

5. Define resource limits for the quotesdb deployment. Set the CPU request to 200 millicores and the memory request to 256 MiB. Set the CPU limit to 500 millicores and the memory limit to 1 GiB.

```
[student@workstation ~]$ oc set resources deployment/quotesdb \
--requests cpu=200m,memory=256Mi --limits cpu=500m,memory=1Gi
deployment.apps/quotesdb resource requirements updated
```

6. Add a readiness and a liveness probe to the frontend deployment.

- 6.1. Use the `oc set probe` command with the `--readiness` option to add the readiness probe that tests the `/status` path on HTTP port 8000.

```
[student@workstation ~]$ oc set probe deployment/frontend --readiness \
--get-url http://:8000/status
deployment.apps/frontend probes updated
```

- 6.2. Use the `oc set probe` command with the `--liveness` option to add the liveness probe that tests the `/env` path on HTTP port 8000.

```
[student@workstation ~]$ oc set probe deployment/frontend --liveness \
--get-url http://:8000/env
deployment.apps/frontend probes updated
```

7. Define resource limits for the frontend deployment. Set the CPU request to 200 millicores and the memory request to 256 MiB. Set the CPU limit to 500 millicores and the memory limit to 512 MiB.

```
[student@workstation ~]$ oc set resources deployment/frontend \
--requests cpu=200m,memory=256Mi --limits cpu=500m,memory=512Mi
deployment.apps/frontend resource requirements updated
```

8. Scale the frontend deployment to three pods.

- 8.1. Scale the deployment.

```
[student@workstation ~]$ oc scale deployment/frontend --replicas 3
deployment.apps/frontend scaled
```

- 8.2. Wait for the deployment to scale up. You might have to rerun the command several times for the pods to report a **Running** status. The names of the pods on your system probably differ.

```
[student@workstation ~]$ oc get pods
NAME                  READY   STATUS    RESTARTS   AGE
frontend-86cdd7c7bf-8vrrs  1/1     Running   0          3m10s
frontend-86cdd7c7bf-ds79w  1/1     Running   0          44s
frontend-86cdd7c7bf-hpnwz  1/1     Running   0          44s
quotesdb-66ff98b88c-fhwbs 1/1     Running   0          12m
```

9. Verify that the application responds to web requests.

- 9.1. Retrieve the URL of the application.

```
[student@workstation ~]$ oc get route
NAME      HOST/PORT           PATH      SERVICES ...
frontend  frontend-comprevie...  ...        frontend ...
```

- 9.2. Use the `curl` command to test the application.

```
[student@workstation ~]$ curl \
  http://frontend-comprevie...apps.ocp4.example.com
<html>
<head>
  <title>Quotes</title>
</head>
<body>

  <h1>Quote List</h1>

  <ul>

    <li>1: When words fail, music speaks.
    - William Shakespeare
  </li>
  ...output omitted...
```

## 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 comprevie-scale
```

## 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 comprevie-scale
```