





Join a community dedicated to learning open source

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



Network with tens of thousands of community members



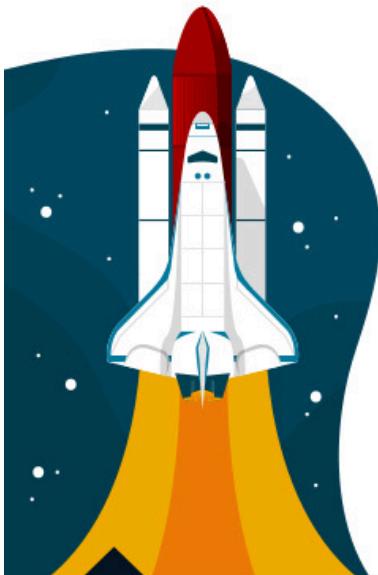
Engage in thousands of active conversations and posts



Join and interact with hundreds of certified training instructors



Unlock badges as you participate and accomplish new goals



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

Access free Red Hat training videos

Discover the latest Red Hat Training and Certification news

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

Join peers as you explore Red Hat products

Join the conversation learn.redhat.com



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

Red Hat OpenShift Administration III: Scaling Kubernetes Deployments in the Enterprise



Openshift Container Platform 4.6 DO380
Red Hat OpenShift Administration III : Scaling Kubernetes
Deployments in the Enterprise
Edition 2 20211223
Publication date 20211223

Authors: Alejandro Coma, Alex Corcoles, Ivan Chavero, Federico Fapitalle, Andres Hernandez, James Mighion, Joel Birchler, Michael Phillips, Christopher Caillouet, Harpal Singh, Razique Mahroua, Dan Kolepp
Editor: Nicole Muller

Copyright © 2020 Red Hat, Inc.

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

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

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

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

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, 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 and/or its affiliates.

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

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

Node.js® is an official 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 OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Chris Tusa, David Sacco, Eliezer Campos Laux, Fernando Lozano, Jim Rigsbee, Nicolette Lucas

Document Conventions	xi
Introduction	xiii
Red Hat OpenShift Administration III : Scaling Kubernetes Deployments in the Enterprise	xiii
Orientation to the Classroom Environment	xiv
Performing Lab Exercises	xxiii
1. Moving From Kubernetes to OpenShift	1
Deploying Kubernetes Applications on OpenShift	2
Guided Exercise: Deploying Kubernetes Applications on OpenShift	6
Optimizing Kubernetes Applications for OpenShift	9
Guided Exercise: Optimizing Kubernetes Applications for OpenShift	12
Summary	17
2. Introducing Automation with OpenShift	19
Querying OpenShift Resources	20
Guided Exercise: Querying OpenShift Resources	26
Deploying Scripts on OpenShift	31
Guided Exercise: Deploying Scripts on OpenShift	36
Navigating the OpenShift REST API	40
Guided Exercise: Navigating the OpenShift REST API	44
Writing Ansible Playbooks to Manage OpenShift Resources	49
Guided Exercise: Writing Ansible Playbooks to Manage OpenShift Resources	52
Summary	57
3. Managing OpenShift Operators	59
Describing Operators	60
Quiz: Describing Operators Quiz	62
Installing Operators	64
Guided Exercise: Installing Operators	70
Managing Cluster Operators	73
Guided Exercise: Managing Cluster Operators	75
Lab: Managing Operators	77
Summary	82
4. Implementing GitOps with Jenkins	83
Introducing Jenkins Declarative Pipelines	84
Quiz: Introducing Jenkins Declarative Pipelines Quiz	92
Deploying Jenkins on OpenShift	96
Guided Exercise: Deploying Jenkins on OpenShift	102
Configuring OpenShift Resources Using a Declarative GitOps Workflow	113
Guided Exercise: Configuring OpenShift Resources Using a Declarative GitOps Workflow	118
Configuring OpenShift using GitOps and Jenkins	123
Guided Exercise: Configuring OpenShift using GitOps and Jenkins	132
Lab: Implementing GitOps with Jenkins	144
Summary	160
5. Configuring Enterprise Authentication	161
Configuring the LDAP Identity Provider	162
Guided Exercise: Configuring the LDAP Identity Provider	166
Synchronizing OpenShift Groups with LDAP	169
Guided Exercise: Synchronizing OpenShift Groups with LDAP	174
Lab: Configuring OpenShift Enterprise Authentication	180
Summary	190
6. Configuring Trusted TLS Certificates	191

Integrating OpenShift with an Enterprise Certificate Authority	192
Guided Exercise: Integrating OpenShift with an Enterprise Certificate Authority	199
Configuring Applications to Trust the Enterprise Certificate Authority	205
Guided Exercise: Configuring Applications to Trust the Enterprise Certificate Authority ..	209
Troubleshooting OpenShift Certificates	215
Guided Exercise: Troubleshooting OpenShift Certificates	220
Lab: Configuring Trusted TLS Certificates	229
Summary	237
7. Configuring Dedicated Node Pools	239
Adding Compute Nodes	240
Guided Exercise: Adding Compute Nodes	244
Creating Custom Machine Config Pools	249
Guided Exercise: Creating Custom Machine Config Pools	255
Summary	261
8. Configuring Persistent Storage	263
Describing the OpenShift Storage Architecture	264
Quiz: Describing the OpenShift Storage Architecture	268
Provisioning Shared Storage for Applications	270
Guided Exercise: Provisioning Shared Storage for Applications	278
Provisioning Block Storage for Databases	285
Guided Exercise: Provisioning Block Storage for Databases	290
Provisioning Local Block Storage	302
Guided Exercise: Installing the Local Storage Operator	305
Lab: Configuring Persistent Storage	315
Summary	322
9. Managing Cluster Monitoring and Metrics	323
Introducing the Cluster Monitoring Stack	324
Quiz: Introducing the Cluster Monitoring Stack	329
Managing OpenShift Alerts	331
Guided Exercise: Managing OpenShift Alerts	335
Troubleshooting Using the Cluster Monitoring Stack	340
Guided Exercise: Troubleshooting Using the Cluster Monitoring Stack	346
Configuring Storage for the Cluster Monitoring Stack	349
Guided Exercise: Configuring Storage for the Cluster Monitoring Stack	352
Lab: Managing Cluster Monitoring and Metrics	357
Summary	363
10. Provisioning and Inspecting Cluster Logging	365
Deploying Cluster Logging	366
Guided Exercise: Deploying Cluster Logging	374
Querying Cluster Logs with Kibana	379
Guided Exercise: Querying Cluster Logs with Kibana	383
Visualizing Cluster Logs with Kibana	385
Guided Exercise: Visualizing Cluster Logs with Kibana	391
Diagnosing Cluster Logging Problems	394
Guided Exercise: Diagnosing Cluster Logging Problems	398
Lab: Provisioning and Inspecting Cluster Logging	402
Summary	412
11. Recovering Failed Worker Nodes	413
Profiling Degraded Worker Nodes	414
Guided Exercise: Profiling Degraded Worker Nodes	417
Troubleshooting Worker Node Capacity Issues	420
Guided Exercise: Troubleshooting Worker Node Capacity Issues	422

Lab: Recovering Failed Worker Nodes	426
Summary	432
A. Creating a Quay Account	433
Creating a Quay Account	434
Repositories Visibility	436
B. Creating a GitHub Account	439
Creating a GitHub Account	440

Document Conventions



References

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



Note

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



Important

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



Warning

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

Introduction

Red Hat OpenShift Administration III : Scaling Kubernetes Deployments in the Enterprise

This course teaches the skills required to manage OpenShift clusters at scale to productively support a growing number of stakeholders, applications, and users. This course enables organizations to innovate faster by adopting containers as part of Digital Transformation initiatives.

Course Objectives

- Automating day 2 tasks necessary to establish production clusters with higher performance and availability requirements.
- Integrating OpenShift software with enterprise infrastructure such as authentication and storage.
- Troubleshooting issues related to cluster operators, resource availability, and capacity.

Audience

- Senior System Administrators focused in planning, designing, and implementing production-grade OpenShift clusters.
- Site Reliability Engineers (SREs) focused in keeping OpenShift clusters and applications running without disruption.

Prerequisites

- Red Hat Certified Specialist in OpenShift Administration (EX280) certification or equivalent knowledge for the role of an OpenShift Cluster Administrator.
- Red Hat Certified Systems Administrator (EX200) certification or equivalent knowledge of the Linux shell and managing Linux servers.
- Ability to write and run simple Ansible Playbooks.

Orientation to the Classroom Environment

The Workstation Machine

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

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

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

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

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



Note

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

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

Log in on OpenShift from the Shell

To access your OpenShift cluster from the **workstation** machine, use `https://api.ocp4.example.com:6443` as the API URL, for example:

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

Besides the **admin** user, who has cluster administrator privileges, your OpenShift cluster also provides a **developer** user, with the password **developer**, who has no special privileges.



Note

If you get the "Use insecure connections" prompt, then answer "yes" because your cluster uses a TLS certificate that was generated by the installer and is not trusted by your **workstation** machine.

Accessing the OpenShift Web Console

If you prefer to use the OpenShift web console, open a Firefox web browser on your **workstation** machine and access the following URL:

<https://console-openshift-console.apps.ocp4.example.com>

Click `htpasswd_provider` and provide the log in credentials for either the **admin** or **developer** user.



Note

The first time you try to access the web console, your browser will not trust its TLS certificate. As with the API TLS certificate, it is generated by the OpenShift installer and not trusted by the **workstation** machine. Click **Advanced** in the Firefox warning page and follow the steps required to accept the risk and trust the TLS certificate.

The first time you access the web console you must also trust the TLS certificate of the OpenShift OAuth authentication server.

The Classroom Environment

Every student gets a complete remote classroom environment. As part of that environment, every student gets a dedicated OpenShift cluster to perform administration tasks.

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

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

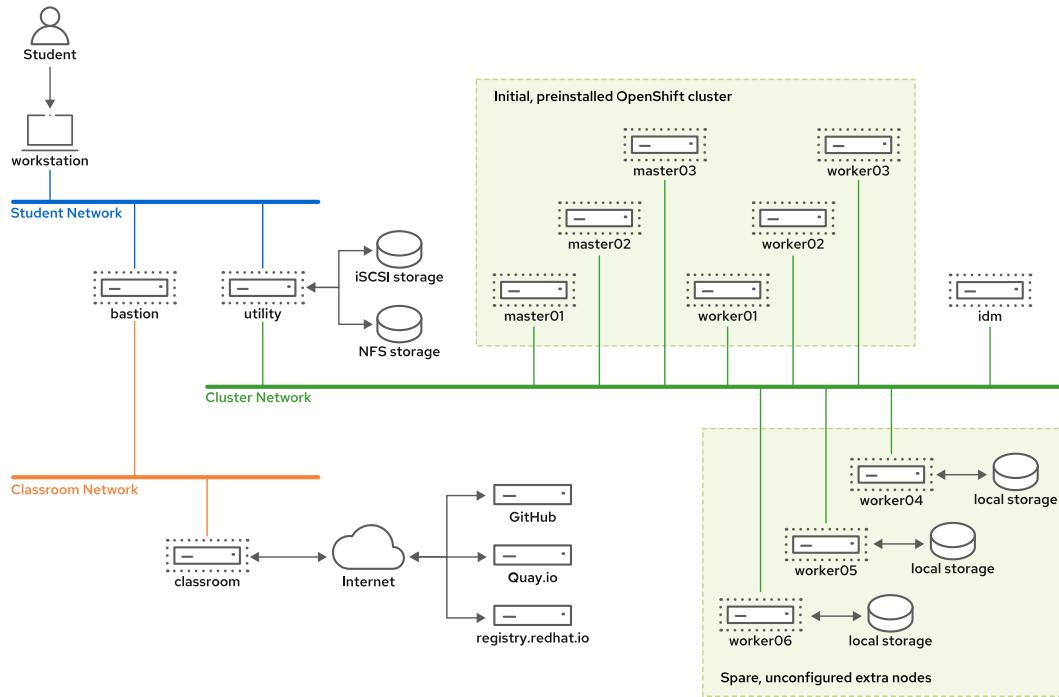


Figure 0.1: Classroom Environment.

Introduction

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

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

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

For the few exercises that require you to access a system other than **workstation**, primarily the **utility** system, you receive explicit instructions and necessary connection information as part of the exercise.

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

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

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

Classroom Machines

Machine name	IP addresses	ROLE
<code>workstation.lab.example.com</code>	172.25.250.9	Graphical workstation used for system administration.
<code>classroom.example.com</code>	172.25.254.254	Router linking the Classroom Network to the Internet.
<code>bastion.lab.example.com</code>	172.25.250.254	Router linking the Student Network to the Classroom Network.
<code>utility.lab.example.com</code>	172.25.250.253	Router linking the Student Network to the Cluster Network and also storage server.
<code>idm.ocp4.example.com</code>	192.168.50.30	RHEL Identity Management (IdM) server.
<code>master01.ocp4.example.com</code>	192.168.50.10	Control plane node
<code>master02.ocp4.example.com</code>	192.168.50.11	Control plane node
<code>master03.ocp4.example.com</code>	192.168.50.12	Control plane node
<code>worker01.ocp4.example.com</code>	172.25.250.13	Compute node
<code>worker02.ocp4.example.com</code>	172.25.250.14	Compute node
<code>worker03.ocp4.example.com</code>	172.25.250.15	Compute node
<code>worker04.ocp4.example.com</code>	172.25.250.16	Unconfigured compute node

Machine name	IP addresses	ROLE
worker05.ocp4.example.com	172.25.250.17	Unconfigured compute node
worker06.ocp4.example.com	172.25.250.18	Unconfigured compute node

Dependencies on Internet Services

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

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

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

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

A few exercises in DO380 also require that students have personal, free accounts on:

- GitHub (<https://github.com>)
- Quay.io (<https://quay.io>)

If either GitHub or Quay.io is not available, the exercises that require them cannot be completed.



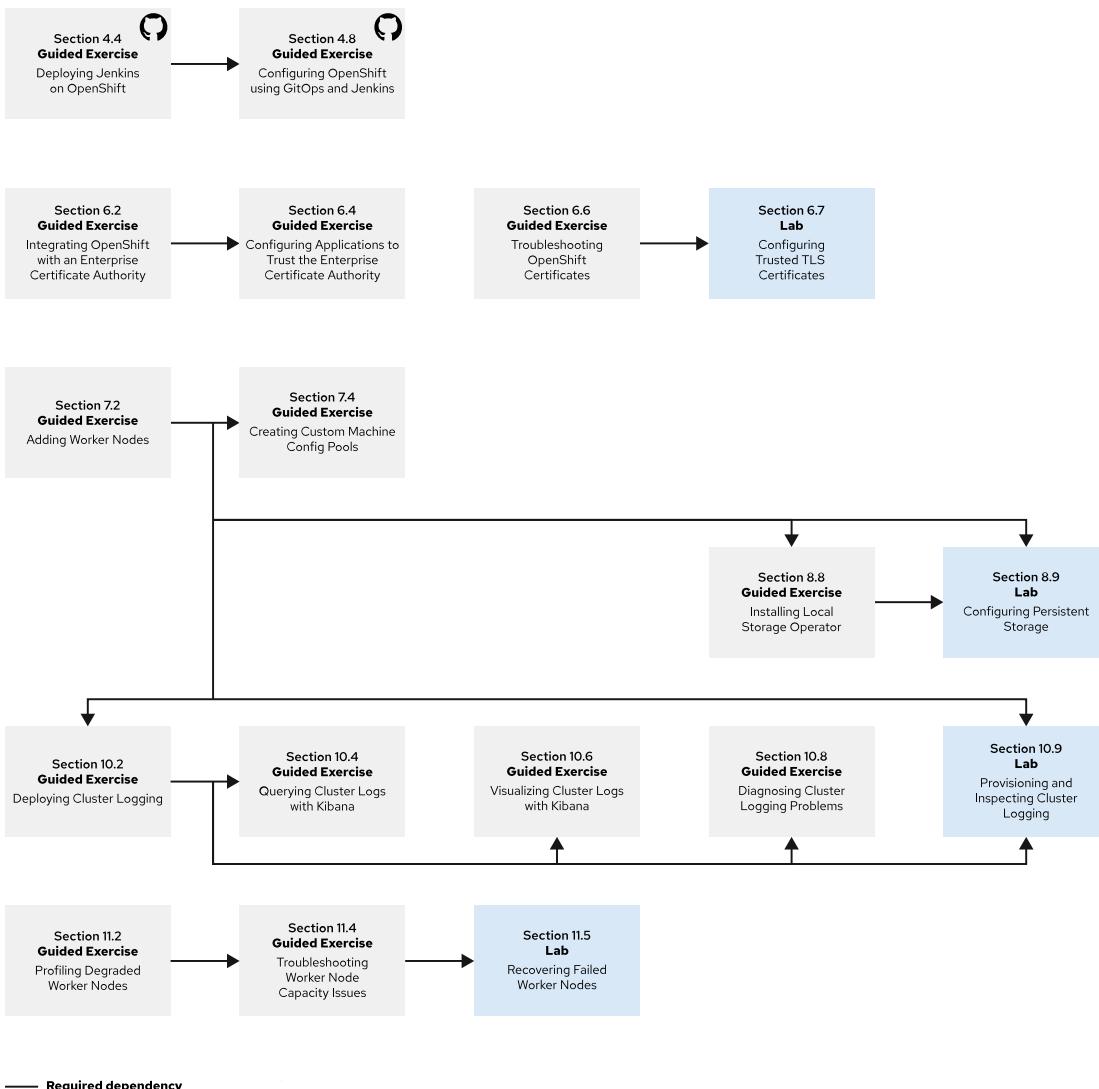
Note

Students are expected to sign in to GitHub and Quay.io and create their accounts, if they do not already have them (see Appendix A), and verify their access by signing in to these services before starting the class.

Dependencies on Course Guided Exercises

In this course, there are guided exercises that requires that another one is completed before starting it. This requirements are show in the next diagram.

Introduction

**Figure 0.2: Guided exercise dependencies**

The Dedicated OpenShift Cluster

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

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

Three of the compute nodes are not initially configured and you will add them to your cluster during this course. These unconfigured compute nodes also provide an extra local disk device to be used as local storage.

Your OpenShift cluster is at the state left by running the OpenShift installer with default configurations, excepting a few day-2 customizations:

- The cluster provides a default storage class backed by a Network File System (NFS) storage provider. This means that applications requiring persistent storage volumes perform comparably to running on a cluster installed using the Full-stack Automation installation method.

Introduction

- There is an HTPasswd Identity Provider (IdP) preconfigured with two users: `admin` and `developer`.

Two additional machines are in the Cluster Network but are not cluster nodes:

- The `idm` machine provides a RHEL IdM server that handles LDAP authentication services.
- The `utility` machine provides NFS and iSCSI storage servers backed by LVM volumes.

It is necessary to access `utility` for exercises that add nodes to the cluster, because the `utility` machine also runs the DHCP server that provides PXE boot to cluster nodes.

The section *Troubleshooting Access to your OpenShift Cluster* provides information about how to access the `utility` machine.

Restoring Access to your OpenShift Cluster

If you suspect that you cannot log in to your OpenShift cluster as the `admin` user anymore because you incorrectly changed your cluster authentication settings, then run the `lab finish` command from your current exercise and restart the exercise by running its `lab start` command.

The `lab` command for all exercises resets cluster authentication settings and restores the default passwords of the `admin` and `developer` users.

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

Troubleshooting Access to your OpenShift Cluster

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

Logging in to the `utility` server is rarely required to perform exercises. If it looks like your OpenShift cluster is taking too long to start, or is in a degraded state, then you can log in on the `utility` machine as the `lab` user to troubleshoot your classroom environment.

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

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

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

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

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

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

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

Approving Node Certificates on your OpenShift Cluster

Red Hat OpenShift Container Platform clusters are designed to run continuously, 24x7, until they are decommissioned. Unlike a production cluster, the classroom environment contains a cluster that was stopped after installation and will be stopped and restarted several times before you finish this course. This scenario requires special handling that would not be required by a production cluster.

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

The OpenShift installer handles creating and approving TLS certificate signing requests (CSRs) for the Full-stack Automation installation method. The system administrator manually approves these CSRs for the Preexisting Infrastructure installation method.

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

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

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

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

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

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

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

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

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

**Note**

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

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

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

Controlling Your Systems

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

Controlling the Virtual Machines

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

Machine States

Virtual Machine State	Description
active	The virtual machine is running and available (or, when booting, soon will be).
stopped	The virtual machine is completely shut down.
building	The initial creation of the virtual machine is being performed.

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

Classroom/Machine Actions

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

Button or Action	Description
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. Students can log in directly to the virtual machine and run commands. In most cases, students should log in to the workstation virtual machine and use <code>ssh</code> to connect to the other virtual machines.
ACTION → Start	Start (power on) the virtual machine.
ACTION → Shutdown	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION → Power Off	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION → Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. Caution: Any work generated on the disk is lost.

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

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

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



Warning

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

The Autostop Timer

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

To adjust the timer, click + to add one hour to the timer. Note that there is a maximum time of twelve hours.

Performing Lab Exercises

Run the `lab` command from `workstation` to prepare your environment before each hands-on exercise, and again to clean up after an exercise. Each hands-on exercise has a unique name within a course. The exercise is prepended with `lab-` as its file name in `/usr/local/lib`. For example, the `instances-cli` exercise has the file name `/usr/local/lib/lab-instances-cli`. To list the available exercises, use tab completion in the `lab` command:

```
[student@workstation ~]$ lab Tab Tab
administer-users  deploy-overcloud-lab  prep-deploy-ips      stacks-autoscale
analyze-metrics   instances-cli        prep-deploy-router  stacks-deploy
assign-roles       manage-interfaces  public-instance-deploy verify-overcloud
```

There are two types of exercises. The first type, a *guided exercise*, is a practice exercise that follows a course narrative. If a narrative is followed by a quiz, this usually indicates that the topic did not have an achievable practice exercise. The second type, an *end-of-chapter lab*, is a gradable exercise to help verify your learning. When a course includes a comprehensive review, the review exercises are structured as gradable labs. The syntax for running an exercise script is:

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

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`. Older courses might still use `setup` and `cleanup` instead of the current `start` and `finish` actions.

start

Formerly `setup`. A script's start logic verifies the resources required to begin an exercise. This may include configuring settings, creating resources, checking prerequisite services, and verifying necessary outcomes from previous exercises. Exercise start logic allows you to perform any exercise at any time, even if prerequisite exercises have not been performed.

grade

End-of-chapter labs help verify what you have learned, after practicing with earlier guided exercises. The `grade` action directs the `lab` command to display 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 re-run the `grade` action.

finish

Formerly `cleanup`. A script's finish logic deletes exercise resources that are no longer necessary. Cleanup logic allows you to repeatedly perform an exercise, and benefits course performance by ensuring that unneeded objects release their resources.

Troubleshooting Lab Scripts

Exercise scripts do not exist on `workstation` until each is first run. When you run the `lab` command with a valid exercise and action, the script named `lab-exercise` is downloaded from the `classroom` server content share to `/usr/local/lib` on `workstation`. The `lab` command creates two log files in `/var/tmp/labs`, plus the directory if it does not exist. One file, named `exercise`, captures standard output messages that normally display on your terminal. The other file, named `exercise.err`, captures error messages.

```
[student@workstation ~]$ ls -l /usr/local/lib
-rwxr-xr-x. 1 root root 4131 May  9 23:38 lab-instances-cli
-rwxr-xr-x. 1 root root 93461 May  9 23:38 labtool.cl110.shlib
-rwxr-xr-x. 1 root root 10372 May  9 23:38 labtool.shlib

[student@workstation ~]$ ls -l /var/tmp/labs
-rw-r--r--. 1 root root 113 May  9 23:38 instances-cli
-rw-r--r--. 1 root root 113 May  9 23:38 instances-cli.err
```



Note

Scripts download from the `http://content.example.com/courses/course/release/grading-scripts` share, but only if the script does not yet exist on `workstation`. When you need to download a script again, such as when a script on the share is modified, manually delete the current exercise script from `/usr/local/lib` on `workstation`, then run the `lab` command for the exercise again. The newer exercise script then downloads from the `grading-scripts` share.

To delete all current exercise scripts on `workstation`, use the `lab` command's `--refresh` option. A refresh deletes all scripts in `/usr/local/lib` but does not delete the log files.

```
[student@workstation ~]$ lab --refresh
[student@workstation ~]$ ls -l /usr/local/lib

[student@workstation ~]$ ls -l /var/tmp/labs
-rw-r--r--. 1 root root 113 May  9 23:38 instances-cli
-rw-r--r--. 1 root root 113 May  9 23:38 instances-cli.err
```

Interpreting the Exercise Log Files

Exercise scripts send output to log files, even when the scripts are successful. Step header text is added between steps and additional date and time headers are added at the start of each script run. The exercise log normally contains messages that indicate successful completion of command steps. Therefore, the exercise output log is useful for observing messages that are expected if no problems occur, but offers no additional help when failures occur.

Instead, the exercise error log is more useful for troubleshooting. Even when the scripts succeed, messages are still sent to the exercise error log. For example, a script that verifies that an object already exists before attempting to create it should cause an *object not found* message when the object does not exist yet. In this scenario, that message is expected and does not indicate a failure. Actual failure messages are typically more verbose, and experienced system administrators should recognize common log message entries.

Although exercise scripts are always run from `workstation`, they perform tasks on other systems in the course environment. Many course environments, including OpenStack and OpenShift, use a command-line interface (CLI) invoked from `workstation` to communicate with server systems using API calls. Because script actions typically distribute tasks to multiple systems, additional troubleshooting is necessary to determine where a failed task occurred. Log in to those other systems and use Linux diagnostic skills to read local system log files and determine the root cause of the lab script failure.

Chapter 1

Moving From Kubernetes to OpenShift

Goal

Demonstrate that OpenShift is Kubernetes by deploying Kubernetes-native applications on OpenShift.

Objectives

- Deploy applications on OpenShift using standard Kubernetes resources.
- Enhance Kubernetes applications using OpenShift extensions from Kubernetes.

Sections

- Deploying Kubernetes Applications on OpenShift (and Guided Exercise)
- Optimizing Kubernetes Applications for OpenShift (and Guided Exercise)

Deploying Kubernetes Applications on OpenShift

Objectives

After completing this section, you should be able to deploy applications on OpenShift using standard Kubernetes resources.

Describing How OpenShift is Enterprise Kubernetes

Red Hat OpenShift Container Platform is a Kubernetes-based container platform for building and running applications on-premise or in cloud environments. Kubernetes provides features to run containers on a cluster of nodes. OpenShift builds on Kubernetes and its extensibility, adding features such as logging, monitoring, and authentication.

Kubernetes provides the `kubectl` command-line tool to interact with Kubernetes clusters.

OpenShift provides the `oc` and `kubectl` command-line tools. The `oc` command adds OpenShift features to the `kubectl` command.



Note

`kubectl` and `oc` in OpenShift are the same binary. The `oc` binary embeds `kubectl` code. Execute the binary using the `kubectl` name to invoke the `kubectl` code and features. Execute the binary using the `oc` name to invoke OpenShift code and features.

Authenticating to OpenShift

The `kubectl` command-line tool connects to a Kubernetes cluster using the Kubernetes HTTP API. The `kubeconfig` files contain the information required to connect to a Kubernetes cluster. Specify the path to the `kubeconfig` file using the `--kubeconfig` option. The default `kubeconfig` path is `~/.kube/config`.

A `kubeconfig` file contains clusters, users, and contexts.

A cluster describes a Kubernetes API endpoint, including its URL.

A user describes the credentials to authenticate with a cluster, such as user names and tokens.

A context is a combination of a cluster, a user, and a default namespace.

The OpenShift CLI provides the `oc login` command to handle the process of logging in to an OpenShift cluster and managing `kubeconfig` files. Run the `oc login` command to log in to an OpenShift cluster.

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

...output omitted...
```

The `oc login` command creates or updates `~/.kube/config` as in the following example:

```
apiVersion: v1
clusters:
- cluster:
  insecure-skip-tls-verify: true
  server: https://api.ocp4.example.com:6443
  name: api-ocp4-example-com:6443
contexts:
- context:
  cluster: api-ocp4-example-com:6443
  namespace: default
  user: admin
  name: default/api-ocp4-example-com:6443/admin
current-context: default/api-ocp4-example-com:6443/admin
kind: Config
preferences: {}
users:
- name: admin
  user:
    token: ErPG5n4aT6CuDQ8mJPSwS5WpvYD_N6Fhj8Yd603l7Ns
```

Run the `oc login` command to create a new context using the `developer` user.

```
[user@host ~]$ oc login -u developer -p developer
Login successful.

...output omitted...
```

The `oc login` command updates the `~/.kube/config` file with the newly created context, which is comprised of the `developer` user and the existing cluster:

```
...output omitted...
contexts:
- context:
  cluster: api-ocp4-example-com:6443
  user: developer
  name: /api-ocp4-example-com:6443/developer
- context:
  cluster: api-ocp4-example-com:6443
  namespace: default
  user: admin
  name: default/api-ocp4-example-com:6443/admin
...output omitted...
- name: admin
  user:
    token: ErPG5n4aT6CuDQ8mJPSwS5WpvYD_N6Fhj8Yd603l7Ns
- name: developer
  user:
    token: 6KGBB0WjZpGpE3MJr3KaGsCSY0k4MJQpb9dWFb1p_M
```

The `oc config get-contexts` command lists the contexts in the `kubeconfig` file.

```
[user@host ~]$ oc config get-contexts
CURRENT NAME CLUSTER AUTHINFO NAMESPACE
* /api-ocp4-...:6443/developer api-ocp... developer
  default/api-ocp4-...:6443/admin api-ocp... admin default
```

The `oc config use-context` command changes the current context.

```
[user@host ~]$ oc config use-context \
  default/api-ocp4-example-com:6443/admin
Switched to context "default/api-ocp4-example-com:6443/admin".
```

The `oc config set-context` command updates a context.

```
[user@host ~]$ oc config set-context \
  /api-ocp4-example-com:6443/developer \
  --namespace=namespace*
Context "/api-ocp4-example-com:6443/developer" modified.
```

Deploying Native Kubernetes Applications on OpenShift

OpenShift Container Platform is a certified Kubernetes distribution. Kubernetes workloads can be deployed in OpenShift using the same procedures used in any other Kubernetes distribution. Scripts using the `kubectl` command-line tool work unmodified on OpenShift. Use the `kubectl create -f <file name>` command to create a Kubernetes workload made of resources contained in a file. Use the `kubectl get type name` command to display a resource.

Customizing Resources with Kubernetes Kustomize

The `kubectl` command integrates the `kustomization` tool. A `kustomization` is a directory containing a `kustomization.yaml` file.

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

The `resources` field is a list of files containing Kubernetes resources.

A `kustomization` can contain fields that describe modifications to be made to the `kustomization` resources.

The `images` field describes modifications to images in the `kustomization`.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - deployment.yaml
images:
  - name: image
    newName: new-image
    newTag: new-tag
```

The image *image* will be replaced by *new-image* with the tag *new-tag*.

`kustomization.yaml` can contain a `bases` field. The `bases` field is a list of other kustomization directories.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
bases:
  - path-to-kustomization
```

A kustomization with a `bases` field is an overlay. A kustomization without a `bases` field is a base. An overlay includes all resources in its bases. `Kustomization` applies modifications described in an overlay to the resources described in the overlay and its bases.

Use the `kubectl apply -k directory_name` command to apply a kustomization.

OpenShift templates are an alternative to kustomizations. OpenShift templates are Kubernetes parameterized resource definitions. Users provide parameters for templates and deploy customized resources using the web console.



References

For more information about the `oc` and `kubectl` commands, refer to the *Usage of oc and kubectl commands* section of the *OpenShift CLI (oc)* chapter in the Red Hat OpenShift Container Platform 4.6 *CLI Tools* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/cli_tools/index#usage-oc-kubectl

Organizing Cluster Access Using kubeconfig Files

<https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/>

Declarative Management of Kubernetes Objects Using Kustomize

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

► Guided Exercise

Deploying Kubernetes Applications on OpenShift

In this exercise, you will deploy an application on OpenShift using standard Kubernetes resources.

Outcomes

You should be able to:

- Interact with OpenShift using the Kubernetes `kubectl` CLI.
- Deploy Kubernetes resources using either the `kubectl` or `oc` commands.

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 k8s-deploy start
```

Instructions

- 1. As the `admin` user, configure a namespace and rolebinding for the developer user.

- 1.1. Use the `oc login` command to log in as the administrator and change to the `~/D0380/labs/k8s-deploy/` directory.

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

Login successful.

...output omitted...

```
[student@workstation ~]$ cd ~/D0380/labs/k8s-deploy/
```

- 1.2. Using the `kubectl` CLI and the provided manifest, create a namespace and provide the developer user with appropriate access.

```
[student@workstation k8s-deploy]$ kubectl create -f ns-and-rbac.yml
namespace/k8s-deploy created
rolebinding.rbac.authorization.k8s.io/hello-dev created
```

- 2. As the developer user, deploy the Kubernetes resources.

- 2.1. Use the `oc login` command to log in as the developer user.

```
[student@workstation k8s-deploy]$ oc login -u developer -p developer
Login successful.

You have one project on this server: "k8s-deploy"

Using project "k8s-deploy".
```

- 2.2. Create a plain Kubernetes deployment, service, and ingress using the `kubectl` CLI and the provided manifest.

```
[student@workstation k8s-deploy]$ kubectl apply -f hello.yml
deployment.apps/hello created
service/hello created
ingress.networking.k8s.io/hello created
```

- 2.3. Verify that the deployment is successful.

```
[student@workstation k8s-deploy]$ kubectl get ingresses.v1.networking.k8s.io
NAME      CLASS      HOSTS          ADDRESS      PORTS      AGE
hello     <none>    hello.apps.ocp4.example.com        80         7s

[student@workstation k8s-deploy]$ curl hello.apps.ocp4.example.com
<html>
  <body>
    <h1>Hello, world from nginx!</h1>
  </body>
</html>
```

3. Deploy using the Kubernetes-native configuration system: Kustomize.

- 3.1. As the `admin` user, create the `k8s-deploy-prod` namespace.

```
[student@workstation k8s-deploy]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation k8s-deploy]$ kubectl create namespace k8s-deploy-prod
namespace/k8s-deploy-prod created
```

- 3.2. Review the provided files in the base and prod overlay Kustomize directories located at `~/D0380/labs/k8s-deploy/base/` and `~/D0380/labs/k8s-deploy/overlays/prod/`.

- 3.3. Deploy the base and overlay with Kustomize.

```
[student@workstation k8s-deploy]$ kubectl apply -k overlays/prod
service/hello created
deployment.apps/hello created
ingress.networking.k8s.io/hello created
```

- 3.4. Verify that the deployment is successful.

```
[student@workstation k8s-deploy]$ kubectl get ingresses.v1.networking.k8s.io \
-n k8s-deploy-prod
NAME      CLASS      HOSTS                           ADDRESS      PORTS      AGE
hello     <none>    deploying-practice.apps.ocp4.example.com        80          7s

[student@workstation k8s-deploy]$ curl deploying-practice.apps.ocp4.example.com
Hi!
```

**Note**

The container name is required to merge the change to the container image.

- 3.5. Update the ~/D0380/labs/k8s-deploy/overlays/prod/kustomization.yml file to change the tag of the image from the base. The file should look similar to the following:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: prod
bases:
- ../../base
images:
- name: quay.io/redhattraining/versioned-hello
  newTag: v1.1
```

- 3.6. Optionally, compare your changes to the solution file located at ~/D0380/solutions/k8s-deploy/overlays/prod/kustomization.yml.

- 3.7. Deploy the patch and verify that the deployment is successful.

```
[student@workstation k8s-deploy]$ kubectl apply -k overlays/prod
service/hello unchanged
deployment.apps/hello configured
ingress.networking.k8s.io/hello unchanged

[student@workstation k8s-deploy]$ curl deploying-practice.apps.ocp4.example.com
Hi! v1.1
```

- 4. Change to the /home/student/ directory.

```
[student@workstation k8s-deploy]$ cd ~
```

Finish

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

```
[student@workstation ~]$ lab k8s-deploy finish
```

Optimizing Kubernetes Applications for OpenShift

Objectives

After completing this section, you should be able to enhance Kubernetes applications using OpenShift extensions from Kubernetes.

Describing Kubernetes Resources on OpenShift

OpenShift adds enterprise features to Kubernetes that support multitenancy and improved developer workflows.

Kubernetes	OpenShift
Namespaces	Projects
<ul style="list-style-type: none"> Provides a scoping mechanism for resources. 	<ul style="list-style-type: none"> Supports increased permissions control, creation requests, and default templates for improved multitenancy workflows. Adds additional display name and description fields.
Ingresses	Routes
<ul style="list-style-type: none"> In a plain Kubernetes cluster, using ingress resources requires installation of an ingress controller. Different ingress controllers will provide different features. 	<ul style="list-style-type: none"> Supports TLS termination and re-encryption. Supports HAProxy annotation for extended features, such as enforcing HTTPS, selecting a load balancing algorithm, and enabling rate limiting.
Deployments	DeploymentConfigs
<ul style="list-style-type: none"> Emphasizes availability over consistency. Uses ReplicaSets that support set-based match selectors. Supports pausing rollouts. Red Hat recommends using Deployments unless you need a specific DeploymentConfigs feature. 	<ul style="list-style-type: none"> Emphasizes consistency over availability. Supports automatic rollbacks. Changes to the pod template automatically trigger rollouts. Supports custom deployment strategies and lifecycle hooks.
Kustomize	Templates

Kubernetes	OpenShift
<ul style="list-style-type: none"> Kubernetes kustomize manages configurations using file overlays without templates. Typically, manifests and kustomize overlays are stored in version control. 	<ul style="list-style-type: none"> OpenShift Template resources are parameterized specification resources. Create resources from templates using the <code>oc process</code> command or the web console.

Managing Image Streams

Image streams allow administrators to use tags for referencing specific versions of container images. This way, you can configure `Builds` and `Deployments` to use a fixed, known good image version until the image stream tag is updated. Using image streams, you can also configure `Builds` and `Deployments` to react upon changes to the original container image and trigger a new build or deployment in your cluster using the updated image version.

Source images are stored in image registries, such as `registry.redhat.io`, or the OpenShift integrated registry. On the contrary, image streams are virtual references to source images, and their metadata is stored in `etcd`.

Image streams use a unique SHA256 identifier instead of a mutable image tag. This approach is more reliable than standard container image tags, such as `:latest` or `:v2.1`, where the tagged image can change without notice.

You can check for image updates on a periodic basis by importing the image using the `--scheduled` flag, for example, `oc import-image IMAGE --confirm --scheduled`. By default, `--scheduled` checks the image registry for updated image versions every fifteen minutes.

Using image streams provides the following advantages:

- Tagging images without pushing using the command line.
- Configuring `Builds` and `Deployments` for automatic redeployment upon image updates.
- Fine-grained access control and image sharing across teams.
- Improving security by configuring view and use permissions on the image stream object.
- Avoiding application downtime that can occur when using untested image versions.
- Users without permissions on the source images can interact with image streams.

Annotating Deployments with Image Stream Triggers

Enhance Kubernetes deployments with OpenShift image stream tags by adding the following metadata annotation.

```
{
  "image.openshift.io/triggers": "[
    {
      \"from\": {
        \"kind\": \"ImageStreamTag\",
        \"name\": \"versioned-hello:latest\"
      },
      ...
    }
  ]"
}
```

```

        \\"fieldPath\\":\\"spec.template.spec.containers[?(@.name==\\\\\"hello\\\\
        \")].image\\"
    }
]
}

```

Notice that the JSON `image.openshift.io/triggers` value is a string. Set the trigger annotation on the deployment using the command `oc set trigger deploy/DEPLOYMENT_NAME --from-image IMAGE_STREAM_TAG -c CONTAINER_NAME`.



Note

In a GitOps workflow, the state of the resource specifications should be stored in a version control system. Annotating deployments with image streams may cause an unexpected container image version mismatch between the YAML specification stored in version control and the deployment resource specification in the cluster.



References

For more information on managing image streams, refer to the *Managing imagestreams* chapter in the Red Hat OpenShift Container Platform 4.6 *Images* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#managing-image-streams

For more information on managing Deployments and DeploymentConfigs, refer to the *Understanding Deployments and DeploymentConfigs* section in the *Deployments* chapter in the Red Hat OpenShift Container Platform 4.6 *Applications* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/applications/index#what-deployments-are

For more information on templates, refer to the *Using Templates* chapter in the Red Hat OpenShift Container Platform 4.6 *Images* documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#using-templates

Kubernetes Ingress vs OpenShift Route

<https://www.openshift.com/blog/kubernetes-ingress-vs-openshift-route>

► Guided Exercise

Optimizing Kubernetes Applications for OpenShift

In this exercise, you will enhance a Kubernetes deployment to use OpenShift image streams.

Outcomes

You should be able to annotate a Kubernetes deployment to automatically perform a rollout when an image is updated.

Before You Begin

To perform this exercise, ensure you have a personal, free Quay.io account. If you need to register with Quay.io, see the instructions in the *Creating a Quay Account* section of Appendix A.

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise. This command provides a basic Kubernetes namespace, deployment, service, and ingress.

```
[student@workstation ~]$ lab k8s-optimize start
```

Instructions

- 1. Switch to the `k8s-optimize` project and verify that the resources are deployed.

1.1. Use the `oc login` command to log in as the administrator.

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

1.2. Switch to the `k8s-optimize` project.

```
[student@workstation ~]$ oc project k8s-optimize
Now using project "k8s-optimize" on server "https://api.ocp4.example.com:6443".
```

1.3. Verify that the provided resources are deployed.

<code>oc get deployments,pods,services</code>				
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/hello	2/2	2	2	28s
NAME	READY	STATUS	RESTARTS	AGE
pod/hello-67cc4b8f87-77xln	1/1	Running	0	27s
pod/hello-67cc4b8f87-pqs64	1/1	Running	0	27s

```

NAME          TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/hello  ClusterIP  172.30.19.75  <none>           8080/TCP    28s

[student@workstation ~]$ oc get ingresses.v1.networking.k8s.io
NAME          CLASS      HOSTS          ADDRESS      PORTS      AGE
ingress.extensions/hello  <none>    hello...com        80          28s

```

- ▶ 2. Identify the container image specified by the deployment.

```
[student@workstation ~]$ oc get deployment/hello -o json | \
jq '.spec.template.spec.containers[0].image'
"quay.io/redhattraining/versioned-hello:v1.0"
```

**Note**

Alternatively, use a jsonpath expression.

```
[student@workstation ~]$ oc get deployment/hello \
-o jsonpath='{.spec.template.spec.containers[].image}{"\n"}'
quay.io/redhattraining/versioned-hello:v1.0
```

- ▶ 3. Run the provided watch script to observe changes in cluster resources and poll the example server.

```
[student@workstation ~]$ ~/D0380/labs/k8s-optimize/watch.sh
Every 2.0s: oc get dep...  workstation.lab.example.com: Thu Jul  2 15:42:37 2021

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello  2/2     2           2           115s

NAME          READY   STATUS    RESTARTS   AGE
pod/hello-7b6d8fbf49-bdqkt  1/1     Running   0           115s
pod/hello-7b6d8fbf49-svj9v  1/1     Running   0           115s

"quay.io/redhattraining/versioned-hello:v1.0"

Hi!
```

Leave the current terminal running. Open a new terminal to execute the rest of the steps in the exercise.

- ▶ 4. Using the skopeo command, copy the `versioned-hello:v1.0` image from the RedHatTraining Quay.io repository to an image tagged `latest` in a public personal repository.

- 4.1. Log in to `quay.io`.

```
[student@workstation ~]$ podman login quay.io
Username: your_username
Password: your_password
Login Succeeded!
```

- 4.2. Use the `skopeo copy SOURCE DESTINATION` command to copy the container image to a repository in your account.

```
[student@workstation ~]$ skopeo copy \
    docker://quay.io/redhattraining/versioned-hello:v1.0 \
    docker://quay.io/your_account/versioned-hello:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

**Note**

If the `skopeo copy` commands fails, retry it. There have been cases of intermittent authentication issues copying to personal repositories.

- 4.3. Navigate to the new repository page on `quay.io`. Change the **Repository Visibility** setting to public.

**Important**

Refer to the *Repositories Visibility* section of Appendix A to read details about how to change repository visibility.

- 5. Import the image from the remote repository into your OpenShift cluster, and then enable periodically scheduled imports.

```
[student@workstation ~]$ oc import-image \
    quay.io/your_account/versioned-hello:latest --confirm --scheduled
imagestream.image.openshift.io/versioned-hello imported
...output omitted...
```

Observe the change in the terminal window running the `watch` command. The `istag` lists the `versioned-hello` image. Notice that the image stream tag refers to an immutable image digest sha.

```
...output omitted...
NAME          IMAGE REPOSITORY           TAGS      UPDATED
.../versioned-hello  .../k8s-optimize/versioned-hello  latest   8 minutes ago

NAME          IMAGE REFERENCE
.../versioned-hello:latest  quay.io/your_account/versioned-hello@sha256:66...5e
...output omitted...
```

- 6. Annotate the deployment with the image stream trigger.

```
[student@workstation ~]$ oc set triggers deployment/hello \
--from-image versioned-hello:latest -c hello
deployment.apps/hello triggers updated

[student@workstation ~]$ oc get deployment/hello -o json | \
jq '.metadata.annotations'
{
  "deployment.kubernetes.io/revision": "2",
  "image.openshift.io/triggers": "[{\\"from\\":{\\\"kind\\\":\\\"ImageStreamTag\\\", \\\"name\\\":\\\"versioned-hello:latest\\\"}, \\\"fieldPath\\\":\\\"spec.template.spec.containers[?(@.name==\\\"hello\\\")].image\\\"}]]"
}
```

Observe the change in the terminal window running the `watch` command. The image used in the deployment is now the same as the image stream.

```
...output omitted...
NAME          READY   STATUS    RESTARTS   AGE
pod/hello-77c8b88f4f-qczp2  1/1     Running   0          25s
pod/hello-77c8b88f4f-r9zqx  1/1     Running   0          28s
...output omitted...
"quay.io/your_account/versioned-hello@sha256:66e0a9c7341e5...acf3ecb5a105e"
...output omitted...
```

- ▶ 7. Copy `versioned-hello:v1.1` from the RedHatTraining Quay.io repository to the image tagged `latest` in your repository.

```
[student@workstation ~]$ skopeo copy \
docker://quay.io/redhattraining/versioned-hello:v1.1 \
docker://quay.io/your_account/versioned-hello:latest
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

- ▶ 8. Import manually the image.

The deployment will update and roll out new pods with the `v1.1` container image when the new image is imported. The default scheduled update imports every fifteen minutes.

8.1. To manually import the image:

```
[student@workstation ~]$ oc import-image \
quay.io/your_account/versioned-hello:latest
imagestream.image.openshift.io/versioned-hello imported
...output omitted...
```

The `curl` command from the `watch` terminal displays a message ending with `v1.1` when the deployment has completed successfully

```
...output omitted...
NAME READY STATUS RESTARTS AGE
pod/hello-85fc4486f7-7j9b7 1/1 Running 0 19s
pod/hello-85fc4486f7-dtvh4 1/1 Running 0 16s

NAME IMAGE REPOSITORY TAGS UPDATED
.../versioned-hello .../k8s-optimize/versioned-hello latest 19 seconds ago

... IMAGE REFERENCE UPDATED
... quay.io/your_account/versioned-hello@sha256:83...b4 19 seconds ago

"quay.io/your_account/versioned-hello@sha256:834d5eb861cf5...88e7c0a08fcb4"
Hi! v1.1
```

8.2. Confirm that the deployment.kubernetes.io/revision is now three.

```
[student@workstation ~]$ oc get deployment/hello -o json | \
  jq '.metadata.annotations'
{
  "deployment.kubernetes.io/revision": "3",
  ...output omitted...
}
```

▶ 9. Terminate the watch script in the other terminal using **Ctrl+C**.

Finish

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

```
[student@workstation ~]$ lab k8s-optimize finish
```

Summary

In this chapter, you learned:

- OpenShift supports native Kubernetes applications.
- How to use Kubernetes Kustomize.
- How to optimize Kubernetes resources with OpenShift features.

Chapter 2

Introducing Automation with OpenShift

Goal

Automate OpenShift using scripts and Ansible Playbooks.

Objectives

- Extract resource information from OpenShift using JSONPath.
- Execute cluster operation scripts from inside and outside of OpenShift.
- Navigate the OpenShift REST API.
- Automate creation and modification of OpenShift resources using the Ansible k8s modules.

Sections

- Querying OpenShift Resources (and Guided Exercise)
- Deploying Scripts on OpenShift (and Guided Exercise)
- Navigating the OpenShift REST API (and Guided Exercise)
- Writing Ansible Playbooks to Manage OpenShift Resources (and Guided Exercise)

Querying OpenShift Resources

Objectives

After completing this section, you should be able to extract resource information from OpenShift using JSONPath.

Describing Kubernetes Resources

The desired persistent state of a Kubernetes cluster is encoded using resources, such as pods and services. Resources typically have a `spec` (desired status) and a `status` (current status). Resources are exposed uniformly by the API.

Extracting Information from Resources

The following example command demonstrates how to list resources of a specific type located within a namespace using the command `oc get type -n namespace -o json`:

```
[user@host ~]$ oc get deployment -n openshift-cluster-samples-operator -o json
{
  "apiVersion": "v1",
  "items": [
    {
      "apiVersion": "apps/v1",
      "kind": "Deployment",
      "metadata": {
        ...output omitted...
      },
      "spec": {
        ...output omitted...
      },
      "status": {
        ...output omitted...
      }
    }
  ],
  "kind": "List",
  "metadata": {
    ...output omitted...
  }
}
```

The JSON output is a list of `items`. Each item has `metadata`, `spec`, and `status` fields.

The following command demonstrates how to get information about the "schema" of an object or its properties:

```
[user@host ~]$ oc explain deployment.status.replicas
KIND:     Deployment
VERSION:  apps/v1

FIELD:    replicas <integer>

DESCRIPTION:
  Total number of non-terminated pods targeted by this deployment (their
  labels match the selector).
```

Extracting Information from a Single Resource

Use the command `oc get type name -n namespace` to retrieve a single object in the specified namespace. Additionally, use JSONPath templates to extract and format the contents of the queried information.

```
[user@host ~]$ oc get deployment -n openshift-cluster-samples-operator \
cluster-samples-operator -o jsonpath='{.status.availableReplicas}'
...output omitted...
```

- {} signals JSONPath code to be evaluated.
- .status.availableReplicas accesses properties of an object.

Iterate over lists in the resource using a wildcard index [*]:

```
[user@host ~]$ oc get deployment -n openshift-cluster-samples-operator \
cluster-samples-operator -o jsonpath='{.status.conditions[*].type}'
Available Progressing
```

The command `oc get -o jsonpath` prints results in a single line, separated by spaces. To simplify processing by other tools, put one item per line parsing the output with `tr " " "\n"` or a similar command.

Get a specific item in a list using the indexing notation [index]:

```
[user@host ~]$ oc get deployment -n openshift-cluster-samples-operator \
cluster-samples-operator -o jsonpath='{.spec.template.spec.containers[0].name}'
cluster-samples-operator
```

Filter items in a list using a filter expression [?(condition)]:

```
[user@host ~]$ oc get deployment -n openshift-cluster-samples-operator \
cluster-samples-operator \
-o jsonpath='{.status.conditions[?(@.type=="Available")].status}'
```

Extracting a Single Property from Multiple Resources

The command `oc get` returns a JSON object containing an `items` object, which is a list of all the objects queried.

The following is an example of how to list a single property from many objects:

```
[user@host ~]$ oc get route -n openshift-monitoring \
-o jsonpath='{.items[*].spec.host}'
alertmanager-main.openshift-monitoring.apps.ocp4.example.com
...output omitted...
```

Use `oc get -o=custom-columns` to print specific properties in a tabular format:

```
[user@host ~]$ oc get pod --all-namespaces \
-o=custom-columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spec.nodeName
NAME                               STATUS   NODE
nfs-client-provisioner-865c67f466-r7txk   Running  worker03
openshift-apiserver-operator-7f87667d89-6bcsj   Running  master01
apiserver-69b8c6f494-9hflm      Running  master01
apiserver-69b8c6f494-kkfjj      Running  master02
apiserver-69b8c6f494-xbvmn      Running  master03
authentication-operator-559d87b6-t5ttm    Running  master01
oauth-openshift-6786dd496c-ghb6s      Running  master01
oauth-openshift-6786dd496c-hnbsj      Running  master02
...output omitted...
```

Extracting a Single Property with Multiple Nesting from Resources

The command `oc get pods` returns a list of pods. Supplying `.items[*]` iterates over each pod. Each pod contains a nested `spec.containers` object, which is a list of the containers in the pod. Using `[*]` iterates over all containers in all pods.

```
[user@host ~]$ oc get pods -A \
-o jsonpath='{.items[*].spec.containers[*].image}'
quay.io/external_storage/nfs-client-provisioner:latest
...output omitted...
```

Extracting Multiple Properties at Different Levels of Nesting from Resources

Use the `{range}` statement to perform more complex iterations. The statement `{range x}...template...{end}` executes `template` for each item in `x` until reaching the `end`.

Use `{range}` as an alternative to `.items[*].property`.

```
[user@host ~]$ oc get pods -A \
-o jsonpath='{range .items[*]}' \
'{.metadata.namespace} {.metadata.creationTimestamp}{\n}'
nfs-client-provisioner 2021-06-22T16:50:34Z
openshift-apiserver-operator 2021-06-22T16:38:09Z
...output omitted...
```

Use nested ranges to print nested information in lists of items.

Print a line for every pod. Print all pod conditions on each line.

```
[user@host ~]$ oc get pods -A -o jsonpath='{range.items[*]}'\
'{"\n"}{.metadata.namespace}{"\t"}{.metadata.name}{"\t"}'\
'{range.status.conditions[*]}{.type}-{.status},{end}{end}"\n"'

nfs-client-provisioner nfs-client-provisioner-865c67f466-spfv6 Initialized-
True,Ready-True,ContainersReady-True,PodScheduled-True,
openshift-apiserver-operator openshift-apiserver-operator-54b67788df-7k9k9
Initialized-True,Ready-True,ContainersReady-True,PodScheduled-True,
openshift-apiserver apiserver-587b85b7db-47nqv Initialized-True,Ready-
True,ContainersReady-True,PodScheduled-True,
...output omitted...
```

It is often helpful to use the argument `-o jsonpath-file` to store JSONPath reports in files for readability.

Create `not_ready_nodes.jsonpath`:

```
{range .items[*]}
{.metadata.name}
{range .status.conditions[?(@.status=="False")]}
{.type}{"="}{.status} {.message}
{end}
{end}
```

Execute the report:

```
[user@host ~]$ oc get nodes -o jsonpath-file=not_ready_nodes.jsonpath

master01

    MemoryPressure=False kubelet has sufficient memory available

    DiskPressure=False kubelet has no disk pressure

    PIDPressure=False kubelet has sufficient PID available

master02

...output omitted...
```

Note that spacing in JSONPath is significant, so the output contains the indentation and line breaks in the template.

Using Label Filtering

Use `--show-labels` to view labels with `oc get`.

```
[user@host ~]$ oc get deployment -n openshift-cluster-storage-operator \
--show-labels
NAME                           ... LABELS
cluster-storage-operator        ... <none>
csi-snapshot-controller        ... <none>
csi-snapshot-controller-operator ... app=csi-snapshot-controller-operator
```

Use the argument `-l label` to filter by label:

```
[user@host ~]$ oc get deployment -n openshift-cluster-storage-operator \
-l app=csi-snapshot-controller-operator -o name
deployment.apps/csi-snapshot-controller-operator
```

Introducing Alternative Methods for Extracting Resource Information

The command `oc get -o name` is also useful for many automation cases. JSONPath might not cover all operations that you want to perform on resource information. Because `oc get` can return a JSON output, any tool that understands JSON can process OpenShift resource information. Lastly, the `jq` command line processor provides a domain-specific language to process JSON information, enabling users to perform more powerful transformations.

Describing Automation Challenges with Resource Updates

In general, automation executes a series of defined steps. Operators execute steps that change an OpenShift resource specification in the background, following the controller pattern. Steps depending on a previous step that is not ready yet can fail, breaking the automation.

Waiting for the controllers to execute the changes prevents these issues. Waiting for a predefined amount of time after a change can work in some situations, but this is a fragile and inefficient approach. Some automation tools provide features to solve this problem robustly and efficiently:

- `oc rollout status` waits until a Deployment is rolled out.
- The Ansible `k8s` module can wait until resources it manages are in the desired state.

However, sometimes waiting must be implemented ad hoc and can provide the only available option to complete some automation approaches.

Waiting on Triggered Deployment Updates

The authentication Cluster Operator uses a Deployment to run the processes that implement authentication. When using the HTPasswd identity provider, user credentials are stored in a Secret. Changing the contents of the Secret triggers a Deployment update. This update occurs in the background, so after replacing the secret changes are not be executed immediately.

To perform a change and wait until it is executed:

- Retrieve the `observedGeneration` of the Deployment before applying the change that triggers the update.

```
deployment_generation=$(oc get \
    -n openshift-authentication deployment/oauth-openshift \
    -o jsonpath='{.status.observedGeneration}')
```

- Apply the change
- Wait until the `observedGeneration` increases.

```
while
    new_generation=$(oc get \
        -n openshift-authentication deployment/oauth-openshift \
        -o jsonpath='{.status.observedGeneration}')
    [ $new_generation -eq $deployment_generation ]
do
    sleep 3
done
```

- Execute the `oc rollout status` command specifying the new revision. Note that this command does not successfully execute until the preceding changes are complete.

```
oc rollout status -n openshift-authentication deployment/oauth-openshift \
    --revision=$new_generation
```

Executing `oc rollout status` immediately after triggering the update does not work, as the update does not start immediately.

- Wait until there are no pods pending deletion.

```
while [ -n "$($oc get pod -n openshift-authentication -o \
    jsonpath='{.items[?(@.metadata.deletionTimestamp != "")].metadata.name}')" ]
do
    sleep 3
done
```



References

Understanding Kubernetes Objects

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

kubectl CLI

<https://kubernetes.io/docs/reference/kubectl/#output-options>

JSONPath Support

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

► Guided Exercise

Querying OpenShift Resources

In this exercise you will query OpenShift resources using the `oc get` command and JSONPath.

Outcomes

You should be able to:

- Extract attributes from OpenShift resources using JSONPath expressions and filters.
- Write shell scripts that extract and modify data from OpenShift resources reliably.

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 automation-resources start
```

Instructions

► 1. Get the expiry date for the web console certificate.

- 1.1. From the `workstation` machine, log in to OpenShift as the `admin` user.

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

- 1.2. Store the host name of the web console in a variable.

```
[student@workstation ~]$ console=$(oc get route -n openshift-console console \
-o jsonpath='{.spec.host}')
[student@workstation ~]$ echo $console
console-openshift-console.apps.ocp4.example.com
```

- 1.3. Use the `curl` command to display the expiry date of the OpenShift Router TLS certificate.

```
[student@workstation ~]$ curl https://$console -k -v 2>&1 | grep 'expire date'
* expire date: Jun 22 16:43:53 2022 GMT
```

► 2. Check that all routes respond.

- 2.1. Get the host names for all routes and store them in a variable.

```
[student@workstation ~]$ hosts=$(oc get route -A \
-o jsonpath='{.items[*].spec.host}'')
```

- 2.2. Use curl to get the HTTP status for each route.

```
[student@workstation ~]$ for host in $hosts ; do \
curl https://$host -k -w "%{url_effective} %{http_code}\n" -o /dev/null -s ; \
done
https://oauth-openshift.apps.ocp4.example.com/ 403
https://console-openshift-console.apps.ocp4.example.com/ 200
https://downloads-openshift-console.apps.ocp4.example.com/ 200
https://alertmanager-main-openshift-monitoring.apps.ocp4.example.com/ 403
https://grafana-openshift-monitoring.apps.ocp4.example.com/ 403
https://prometheus-k8s-openshift-monitoring.apps.ocp4.example.com/ 403
https://thanos-querier-openshift-monitoring.apps.ocp4.example.com/ 403
```

- 3. Create a script to list the users configured in the HTPasswd identity provider.

- 3.1. Change to the ~/D0380/labs/automation-resources directory.

```
[student@workstation ~]$ cd ~/D0380/labs/automation-resources
```

- 3.2. Show the OAuth configuration in JSON format to locate the name of the secret that contains the users.

```
[student@workstation automation-resources]$ oc get oauth cluster -o json
{
    ...output omitted...
},
"spec": {
    "identityProviders": [
        {
            "htpasswd": {
                "fileData": {
                    "name": "htpasswd-secret"
                }
            },
            "mappingMethod": "claim",
            "name": "htpasswd_provider",
            "type": "HTPasswd"
        }
    ]
}
```

- 3.3. Use JSONPath with a filter to extract the secret name from the identity provider named htpasswd_provider.

```
[student@workstation automation-resources]$ filter='?(.name=="htpasswd_provider")'  
[student@workstation automation-resources]$ oc get oauth cluster \  
-o jsonpath=".spec.identityProviders[$filter].htpasswd.fileData.name}{'\n'}"  
htpasswd-secret
```

- 3.4. Create the `get-users.sh` script with contents from the following listing.

You can compare your script with the solution file in the `~/DO380/solutions/automation-resources` folder.

```
#!/bin/sh  
  
filter='?(.name=="htpasswd_provider")'  
  
secret_name=$(oc get oauth cluster \  
-o jsonpath=".spec.identityProviders[$filter].htpasswd.fileData.name")  
  
secret_file=$(oc extract secret/$secret_name -n openshift-config --confirm)  
  
cut -d : -f 1 <$secret_file  
rm $secret_file
```

- 3.5. Execute the script.

```
[student@workstation automation-resources]$ chmod +x get-users.sh  
[student@workstation automation-resources]$ ./get-users.sh  
admin  
developer
```

► 4. Fix a script that adds a user to the HTPasswd identity provider .

- 4.1. Examine the incomplete `add-user.sh` script. You will make changes at a later step, after you test the incomplete script.

The script finds the HTPasswd secret and adds a new user to the secret. Then, the script tests if the new user can log in on OpenShift without affecting the Kubernetes context of the current Linux user.

The following partial listing shows important commands inside the incomplete `add-user.sh` script.

```
...output omitted...  
  
secret=$(oc get oauth/cluster \  
-o jsonpath='{.spec.identityProviders[0].htpasswd.fileData.name}')  
tmpdir=$(mktemp -d)  
oc extract -n openshift-config secret/$secret \  
--keys htpasswd --to $tmpdir  
htpasswd -b $tmpdir/htpasswd $user $pass  
oc set data secret/$secret --from-file htpasswd=$tmpdir/htpasswd \  
-n openshift-config  
  
...output omitted...
```

```
oc login -u $user -p $pass --kubeconfig /dev/null \
--insecure-skip-tls-verify \
https://api.ocp4.example.com:6443
```

- 4.2. Try to add a user using the incomplete `add-user.sh` script. Expect the script to fail. If it does not fail, then try again, changing the user name until you see an error message similar to the following:

```
[student@workstation automation-resources]$ ./add-user.sh user1 user1
/tmp/tmp.cbAMhwTRob/htpasswd
Adding password for user user1
secret/htpasswd-secret data updated
Login failed (401 Unauthorized)
Verify you have provided correct credentials.
```

The script might fail because the OpenShift Authentication cluster operator takes time to react to the change in the HTPasswd secret and restart the OAuth pods.

If there is a log in attempt before the OAuth pods restart, then the new user is not recognized.

- 4.3. Log in as the new user. If you cannot log in, then wait a few seconds and try again. Repeat until you can log in successfully.

```
[student@workstation automation-resources]$ oc login -u user1 -p user1
Login successful.
...output omitted...
```

- 4.4. Log in as the `admin` user.

```
[student@workstation automation-resources]$ oc login -u admin -p redhat
```

- 4.5. Edit the `add-user.sh` script to wait until:

- The Authentication Operator notices the change in the secret.
- New OAuth pods are deployed.
- Old OAuth pods are removed.

The script must wait until old OAuth pods are removed because both old and new pods get requests from the OAuth service. If an authentication request reaches one of the old pods, then it does not accept the new user.

Replace the `FIXME` comments with the commands highlighted in the following partial listing. You can compare your edits with the solution located in the `~/D0380/solutions/automation-resources` folder.

```
...output omitted...

pass=$2

oldpods="$(oc get pod -n openshift-authentication -o name)"

secret=$(oc get oauth/cluster \
-o jsonpath='{.spec.identityProviders[0].htpasswd.fileData.name}' )
```

```
...output omitted...

rm -rf $tmpdir

oc wait co/authentication --for condition=Progressing \
--timeout 90s

oc rollout status -n openshift-authentication deployment/oauth-openshift \
--timeout 90s

oc wait $oldpods -n openshift-authentication --for delete \
--timeout 90s

oc login -u $user -p $pass --kubeconfig /dev/null \
--insecure-skip-tls-verify \
https://api.ocp4.example.com:6443
```

4.6. Add another user, and then note if log in succeeds.

If you see a "pod not found" error message, similar to the following sample output, then you can safely ignore it. The error occurs if OpenShift has finished deleting the pod before the `oc wait` command executes.

```
[student@workstation automation-resources]$ ./add-user.sh user2 user2
...output omitted...
Adding password for user user2
secret/htpasswd-secret data updated
clusteroperator.config.openshift.io/authentication condition met
...output omitted...
deployment "oauth-openshift" successfully rolled out
pod/oauth-openshift-7c9bd95b5d-klq5w condition met
Error from server (NotFound): pods "oauth-openshift-7c9bd95b5d-qvrwg" not found
Login successful.
...output omitted...
```

4.7. Change to the /home/student directory.

```
[student@workstation automation-resources]$ cd
```

Finish

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

```
[student@workstation ~]$ lab automation-resources finish
```

Deploying Scripts on OpenShift

Objectives

After completing this section, you should be able to:

- Execute cluster operation scripts from inside and outside of OpenShift.
- Schedule automation scripts using OpenShift CronJobs.

Automating Operations

Replacing repetitive manual work with software automation is a core DevOps practice. By minimizing hands-on operational tasks, Site Reliability Engineers (SREs) can focus on adding features and increasing system reliability.

Benefits of automation:

- Automated systems can scale beyond the linear capacity of human operators manually tending to servers. Without automation, growing the number or complexity of services requires adding increasingly more administrators to your operations group.
- Version control systems, such as Git, are a center of collaboration. Scripts and configuration files are easily shared across teams. Peer reviews and automated workflows replace manual, unreviewed system changes.
- Automated operation scripts are both testable and repeatable. Scripts can be unit tested using popular testing tools.

Discussing Service Accounts

Programs such as continuous integration applications, scripts, and operators provide identification in the form of a service account. User accounts are for real human users, whereas service accounts are for programs.

When a user authenticates with the OpenShift OAuth server using the `oc` client, an access token is retrieved and saved to a `kubeconfig` file. This token is used to provide proof of identity to the OpenShift API server.

Processes running inside an OpenShift cluster interact with the OpenShift API server using an access token for the service account. By default, OpenShift adds a file containing the service account access token to a running pod, located at `/var/run/secrets/kubernetes.io/serviceaccount/token`. Client binaries, such as `oc`, and client libraries use an access token to interact with the OpenShift API.

Creating a Custom Service Account

When running an operational script in OpenShift, create a custom service account to provide its identity.

A basic service account must only specify the name and namespace. Notice that unlike users, service accounts belong to a namespace.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: backup_sa
  namespace: database
```

**Note**

You can also create a service account manually using the `oc create serviceaccount` command. However, specifying resources in declarative YAML or JSON text files encourages the DevOps practices of version control and code review.

By default, the service account does not have permission to make requests to the OpenShift API server. For that, use roles and role bindings.

Defining Roles and Role Bindings

An OpenShift Role or ClusterRole specifies the actions and resources that a user or service account can perform. Roles are namespaced and only define rules within the scope of the namespace. However, cluster roles are not namespaced and their rules can be applied to both namespaced and non-namespaced resources.

OpenShift includes a useful set of predefined cluster roles, such as `basic-user`, `cluster-reader`, and `view`. Use the `oc describe clusterrole CLUSTER_ROLE` command to list the rules specified by the cluster role.

```
[user@host ~]$ oc describe clusterrole view
Name:           view
Labels:         kubernetes.io/bootstrapping=rbac-defaults
                rbac.authorization.k8s.io/aggregate-to-edit=true
Annotations:   rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources          ...    Verbs
  -----
  namespaces         ...    [get get list watch]
  packagemanifests.pac...coreos.com ...    [get list watch get list watch]
  appliedclusterresourcequotas ...    [get list watch]
  ...
  ...output omitted...
```

OpenShift associates accounts with roles using role bindings.

A `RoleBinding` resource is namespaced. It associates accounts with roles within its namespace. A namespaced role binding can also associate a cluster role with an account. In this case, the permissions defined by the cluster role only apply to the resources in the namespace.

A `ClusterRoleBinding` resource is not namespaced and applies to all resources in the cluster. For example, to grant a service account `cluster-reader` permissions, create a cluster role binding as follows:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
```

```
name: backup
namespace: database
subjects: ①
- kind: ServiceAccount
  name: backup_sa
  namespace: database
roleRef: ②
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-reader
```

- ①** Subjects are either users, groups, or service accounts.
- ②** The roleRef must be either a ClusterRole or Role resource.



Note

Alternatively, use the `oc policy add-role-to-user` command to create or modify role bindings.

Creating a Custom Role

Assign permissions to accounts with the minimum amount of privileges required to operate. Limiting access avoids unintended consequences and restricts potential attack vectors.

Roles define a list of policy rules. Each rule specifies the API groups, resources, and permitted verbs.

In some cases, the predefined cluster roles grant more permissions than are required by a script.

The following example demonstrates a role with a rule permitting the reading and creation of ConfigMaps.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: configurator
  namespace: database
rules:
- apiGroups: []
  resources: ["configmaps"]
  verbs: ["get", "list", "create"]
```

Creating Jobs and CronJobs

A Job creates and executes pods until they successfully complete their task. Jobs can be configured to run a single pod or many pods concurrently. Jobs are useful for automating one-off tasks that do not need to be constantly running, such as report generation or file clean up.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: backup
  namespace: database
```

```

spec:
  activeDeadlineSeconds: 600 1
  parallelism: 2 2
  template: 3
    metadata:
      name: backup
    spec:
      serviceAccountName: backup_sa 4
      containers:
        - name: backup
          image: example/backup_maker:v1.2.3
      restartPolicy: OnFailure
  
```

- 1** Optionally, provide a duration limit in seconds. The Job will attempt to terminate if it exceeds the deadline.
- 2** Specify the number of pods to run concurrently.
- 3** The spec includes a pod template.
- 4** Specify the name of the service account to associate with the pod. If you do not specify a service account, then the pod will use the `default` service account in the namespace.

Scheduling OpenShift CronJobs

CronJobs create jobs based on a given time schedule. Use CronJobs for running automation, such as backups or reports, which should be executed on a regular interval.

When using CronJobs, the execution schedule is specified in the standard cron format.

- This follows the format "*minute hour day-of-the-month month day-of-the-week*".
- The * symbol is a wildcard that matches any value. For example, "1 * 1 1 *" matches every hour of the first minute of January 1st.
- A slash / symbol denotes step values. For example, using the pattern "*/5 * * * *" means that the job should run every five minutes.

```

apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: backup-cron
  namespace: database
spec:
  schedule: "1 0 * * *" 1
  jobTemplate: 2
    spec:
      activeDeadlineSeconds: 600
      parallelism: 2
      template:
        metadata:
          name: backup
        spec:
          serviceAccountName: backup_sa
          containers:
  
```

```
- name: backup
  image: example/backup_maker:v1.2.3
  restartPolicy: OnFailure
```

- ➊ The example schedule executes the job at one minute past midnight.
- ➋ Specify a regular job in the CronJob's jobTemplate field.

Comparing Script Deployment Strategies

Strategy	Description
Container Command	Specify short Bash scripts as arguments to a container in the spec. This method is easy to deploy, but makes reviewing and automated tests more difficult.
Volume	Mount a ConfigMap or persistent storage as a volume.
Container Image	Package the script in a container image with all necessary dependencies. Use a GitOps pipeline with build, test, and deployment automation.



References

- For more information on roles and role bindings, refer to the *Using RBAC to define and apply permissions* chapter in the Red Hat OpenShift Container Platform 4.6 *Authentication and authorization* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/authentication_and_authorization/index#using-rbac
- For more information on Jobs and CronJobs, refer to the *Running tasks in pods using jobs* section in the *Using Jobs and DaemonSets* chapter in the Red Hat OpenShift Container Platform 4.6 Nodes documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#nodes-nodes-jobs

► Guided Exercise

Deploying Scripts on OpenShift

In this exercise, you will execute a cluster operation script from both inside and outside of OpenShift.

Outcomes

You should be able to:

- Run a cluster operation script from outside of OpenShift.
- Create a service account and custom role for unattended authentication.
- Deploy a cluster operation script to run as a CronJob.

Before You Begin

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

This command provides starter files used throughout the exercise and solutions to compare against.

```
[student@workstation ~]$ lab automation-scripts start
```

Instructions

As a cluster administrator, you are tasked with running an audit script, which the security team uses to list all container images used in the cluster. First, test the script by executing it from outside of the cluster. Then, create a CronJob so the script automatically runs in the cluster at regular intervals.

► 1. Review and execute the shell script.

- 1.1. Change to the `~/D0380/labs/automation-scripts` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/automation-scripts
```

- 1.2. Open `audit.sh` and review the contents.

The script fetches a list of all of the pods in the cluster and uses the `jsonpath` argument to filter the results to only the container images. `sed` replaces spaces with newlines. The script then removes duplicates and sorts the entries before printing the list.

```
#!/usr/bin/env bash
oc get pods --all-namespaces \
-o jsonpath="{.items[*].spec.containers[*].image}" \
| sed 's/ /\n/g' \
| sort \
| uniq
```

- 1.3. From the workstation machine, log in to OpenShift as the admin user.

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

- 1.4. Run the audit script locally.

```
[student@workstation automation-scripts]$ ./audit.sh
quay.io/external_storage/nfs-client-provisioner:latest
quay.io/openshift-release-dev/ocp-release@sha256:b51a...c035
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:c945...1c0d
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:ef5c...26af
...output omitted...
```

▶ 2. Create a new project named automation-scripts.

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

▶ 3. Create a service account with suitable minimal permissions to read pods from all namespaces.

Ensure that the file named `rbac.yml` specifies an `auditor` service account, a cluster role with read access to pods, and a cluster role binding.

- 3.1. Define an `auditor` service account for the job to use for API authentication, and then bind the `auditor` service account to the `auditor` cluster role with a cluster role binding. Edit the file and replace each `CHANGE_ME` entry for all `name` keys with the text `auditor`.
- 3.2. Restrict the cluster role to allow only `get` and `list` actions on pod resources by replacing each `CHANGE_ME` entry for the `verbs` key with the text `["get", "list"]`.
- 3.3. Verify the completed file by comparing it to the solution located at `/home/student/D0380/solutions/automation-scripts/rbac.yml`.
- 3.4. Deploy the OpenShift resources.

```
[student@workstation automation-scripts]$ oc create -f rbac.yml
serviceaccount/auditor created
clusterrole.rbac.authorization.k8s.io/auditor created
clusterrolebinding.rbac.authorization.k8s.io/auditor created
```

► 4. Deploy an OpenShift job that executes the shell script.

- 4.1. In the `job.yml` file, specify `auditor` as the service account by replacing `CHANGE_ME` for the `serviceAccountName` keys with the text `auditor`.
- 4.2. Verify the completed file by comparing it to the solution located at `/home/student/D0380/solutions/automation-scripts/job.yml`.
- 4.3. Deploy the job to OpenShift.

```
[student@workstation automation-scripts]$ oc create -f job.yml
job.batch/audit-sh created
```

4.4. Verify that the job completed successfully.

```
[student@workstation automation-scripts]$ oc get jobs,pods
NAME          COMPLETIONS  DURATION   AGE
job.batch/audit-sh  1/1        9s         16m

NAME           READY  STATUS    RESTARTS  AGE
pod/audit-sh-k2c4q  0/1    Completed  0          16m
```

4.5. Check the logs to review the audit report.

```
[student@workstation automation-scripts]$ oc logs job.batch/audit-sh
quay.io/external_storage/nfs-client-provisioner:latest
quay.io/openshift-release-dev/ocp-release@sha256:b51a...c035
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:c945...1c0d
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:ef5c...26af
...output omitted...
```

► 5. Create a CronJob to automatically run the audit script on a schedule. Edit the `cronjob.yml` file.

- 5.1. Schedule the job to run every two minutes by replacing `CHANGE_ME` for the `schedule` with the text `"*/2 * * * *"`.
- 5.2. Specify `auditor` as the service account by replacing `CHANGE_ME` for the `serviceAccountName` keys with the text `auditor`.
- 5.3. Verify the completed file by comparing it to the solution located at `/home/student/D0380/solutions/automation-scripts/cronjob.yml`.
- 5.4. Create the CronJob.

```
[student@workstation automation-scripts]$ oc create -f cronjob.yml
cronjob.batch/audit-cron created
```

5.5. Watch the CronJob, Job, and Pod resources until the `audit-cron` script completes.

```
[student@workstation automation-scripts]$ watch oc get cronjobs,jobs,pods
NAME          SCHEDULE  SUSPEND  ACTIVE  LAST SCHEDULE   AGE
cronjob.batch/audit-cron  */2 * * * *  False     0      45s        2m20s

NAME          COMPLETIONS  DURATION  AGE
job.batch/audit-cron-1594932720  1/1       13s      40s
job.batch/audit-sh                1/1       25s      3m21s

NAME          READY  STATUS    RESTARTS  AGE
pod/audit-cron-1594932720-kp9q7  0/1   Completed  0      40s
pod/audit-sh-9d7cd                0/1   Completed  0      3m21s
```

5.6. Verify the results of the script by viewing the logs.

```
[student@workstation automation-scripts]$ oc logs job.batch/audit-cron-1594932720
quay.io/external_storage/nfs-client-provisioner:latest
quay.io/openshift-release-dev/ocp-release@sha256:b51a...c035
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:c945...1c0d
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:ef5c...26af
...output omitted...
```

▶ 6. Change to the `/home/student` directory.

```
[student@workstation automation-scripts]$ cd
```

Finish

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

```
[student@workstation ~]$ lab automation-scripts finish
```

Navigating the OpenShift REST API

Objectives

After completing this section, you should be able to navigate the OpenShift REST API.

Discussing the OpenShift REST API

OpenShift provides a REST API, served by the API Server. Clients and automation tools can use the REST API to interact with the cluster. This way, both users and internal cluster components can perform operations with OpenShift without the need of using the `oc` or `kubectl` binaries.

Authenticating with the REST API

For interacting with the OpenShift API, two security concepts intervene:

- Authentication. Handled by the OpenShift OAuth server, its purpose is to validate that users are who they say they are.
- Authorization. Handled by the OpenShift API server, its purpose is to validate that users have access to the resources they are trying to interact with.

In order to interact with OpenShift resources via the REST API, you must retrieve a bearer token from the OpenShift OAuth server, and then include this token as a header in requests to the API server.

There are several methods for retrieving the token, including:

1. Request a token from the OAuth server path `/oauth/authorize?client_id=openshift-challenging-client&response_type=token`. The server responds with a `302 Redirect` to a location. Find the `access_token` parameter in the location query string.
2. Log in using the `oc login` command and inspect the `kubeconfig.yaml`. This is normally located at `~/.kube/config`. Find the token listed under your user entry.
3. Log in using the `oc login` command, and then run the `oc proxy` command to expose an API server proxy on your local machine. Any requests to the proxy server will identify as the user logged in with `oc`.
4. Log in using the `oc login` command, and then run the command `oc whoami -t`.



Warning

Protect access to the localhost proxy port when running `oc proxy`.

After that, you must include the bearer token as a header in requests to the API server as follows:

```
[user@host ~]$ curl -k \
--header "Authorization: Bearer qfyV5ULvv...i712kJT" \
-X GET https://api.example.com/api
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.136.182:6443"
    }
  ]
}
```

Finding REST API Paths

The OpenShift REST API paths can be explored with a HTTP GET request to retrieve a list of possible paths:

```
[user@host ~]$ curl -k \
--header "Authorization: Bearer qfyV5ULvv...i712kJT" \
-X GET https://api.example.com/oapi
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/admissionregistration.k8s.io",
    "/apis/admissionregistration.k8s.io/v1",
    "/apis/admissionregistration.k8s.io/v1beta1",
    "/apis/apiextensions.k8s.io",
    "/apis/apiextensions.k8s.io/v1",
    "/apis/apiextensions.k8s.io/v1beta1",
    ...output omitted...
  ]
}
```

REST APIs are organized by Kind and resource Version. From the output above, observe the line:

```
"/apis/admissionregistration.k8s.io/v1"
```

- The resource Kind is `admissionregistration.k8s.io`.
- The resource Version is `v1`.

API resources are grouped as follows:

- The core v1 API resources, such as pods, are placed in the `/api` path.
- Other resources, such as `deployments` and `routes`, are placed in the `/apis` path.

- Cluster-scoped resources are organized as: `/API_OR_APIS/API_VERSION/RESOURCE_TYPE`
- Namespaced resources are organized as: `/API_OR_APIS/API_VERSION/namespaces/NAMESPACE/RESOURCE_TYPE`

The REST API can also modify and delete resources in the cluster using verbs as shown in the following table:

API Verb	HTTP Request Command	Description
Create	POST	Create a new resource.
Get/List	GET	Retrieve a single resource or a list of resources.
Update	PUT	Update a resource by providing a full specification.
Patch	PATCH	Update a resource by providing a partial change described in a patch operation.
Delete	DELETE	Delete a resource or a list of resources.

Create Project Example

The following is an example of project creation:

```
[user@host ~]$ cat project.json
{
  "apiVersion": "project.openshift.io/v1",
  "kind": "Project",
  "metadata": {
    "name": "example"
  }
}

[user@host ~]$ curl -k --header "Authorization: Bearer qfyV5ULvv...i712kJT" \
--header 'Content-Type: application/json' -d "$(cat project.json)" \
-X POST https://api.example.com/apis/project.openshift.io/v1/projects
{
  "kind": "Project",
  "apiVersion": "project.openshift.io/v1",
  "metadata": {
    "name": "example",
    ...output omitted...
  },
  "spec": {
    "finalizers": [
      "openshift.io/origin",
      "kubernetes"
    ]
  },
  "status": {
    "phase": "Active"
  }
}
```

Inspecting API Resources from the Command Line

The `oc explain RESOURCE` command displays documentation for a given resource, including the API version and group necessary for forming a request path. For example, the `oc explain pvc` command outputs `PersistentVolumeClaim` as the `KIND` and `v1` as the `VERSION`. The API path for a PVC is `/api/v1/persistentvolumeclaims`. The `oc api-versions` command lists the API group and versions for all available APIs. The `oc api-resources` command retrieves a list of all of the available APIs, including the API group and whether or not the API resources are namespaced.

Filtering Output

When using the API, responses return in JSON format. Sometimes, the responses can be quite long or verbose, and you might only require a single field. You can use the `jq` application to filter the output. Piping a `curl` response to `jq` will format and color code the output.

Services Names Example

```
[user@host ~]$ curl -sk --header "Authorization: Bearer qfyV5ULvv...i712kJT" \
-X GET https://api.example.com/api/v1/services | jq '.items[].metadata.name' \
"kubernetes"
"openshift"
"kubelet"
"metrics"
...output omitted...
```



References

OpenShift Container Platform 4.6 REST APIs

https://docs.openshift.com/container-platform/4.6/rest_api/objects/index.html

jq application

<https://stedolan.github.io/jq/>

► Guided Exercise

Navigating the OpenShift REST API

In this exercise you will explore the OpenShift REST API using the `curl` command.

Outcomes

You should be able to:

- Authenticate with the OpenShift OAuth server to retrieve an access token.
- Find and inspect OpenShift resources using the `curl` command.

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 automation-rest start
```

Instructions

- 1. Attempt a GET request to the OpenShift API server without authentication.

```
[student@workstation ~]$ curl -k https://api.ocp4.example.com:6443/api
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/api\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
}
```

- 2. Authenticate with the OpenShift OAuth server to retrieve an access token.

- 2.1. Log in to OpenShift as the `admin` user.

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

...output omitted...
```

- 2.2. Find the route to the OpenShift OAuth server using the oc command, and then assign it to a variable.

```
[student@workstation ~]$ oc get route -n openshift-authentication
NAME          HOST/PORT           PATH      SERVICES
PORT  TERMINATION   WILDCARD
oauthOpenshift oauth.openshift.apps.ocp4.example.com      oauth-openshift
6443  passthrough/Redirect  None

[student@workstation ~]$ OAUTH_HOST=$(oc get route oauth-openshift \
-n openshift-authentication -o jsonpath='{.spec.host}')

[student@workstation ~]$ echo $OAUTH_HOST
oauth-openshift.apps.ocp4.example.com
```

- 2.3. Authenticate with the OAuth server using the curl command. OpenShift responds with an HTTP 302 Redirect response code, but there is no need to follow it. The token can be found as the access_token parameter in the Location URL.

```
[student@workstation ~]$ curl -u admin -kv "https://$OAUTH_HOST/oauth/\
authorize?client_id=openshift-challenging-client&response_type=token"
Enter host password for user 'admin': redhat
*   Trying 192.168.50.254...
*   TCP_NODELAY set
*   Connected to oauth-openshift.apps.ocp4.example.com (192.168.50.254) port 443
 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
  CApath: none
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
...output omitted...
* Server auth using Basic with user 'admin'
...output omitted...
< Location: https://oauth-openshift.apps...example.com/oauth/token/
implicit#access_token=sha256-xvZ8SsTiA3jRIiEX9QMUOLdaZRUPqubLy2AiQbQGDb0
&expires_in=86400&scope=user%3Afull&token_type=Bearer
...output omitted...
```

- 2.4. Save the token to a variable.

```
[student@workstation ~]$ TOKEN=sha256-xvZ8SsTiA3jRIiEX9QMUOLdaZRUPqubLy2AiQbQGDb0
```

- 3. Attempt a GET request to the OpenShift API server using the access token.

```
[student@workstation ~]$ HEADER="Authorization: Bearer $TOKEN"
[student@workstation ~]$ curl -k --header "$HEADER" \
-X GET https://api.ocp4.example.com:6443/api
{
  "kind": "APIVersions",
```

```
"versions": [
    "v1"
],
"serverAddressByClientCIDRs": [
    {
        "clientCIDR": "0.0.0.0/0",
        "serverAddress": "192.168.50.10:6443"
    }
]
```

► 4. Traverse the v1 API group using curl requests.

4.1. Fetch the list of API resources available in the v1 API group.

```
[student@workstation ~]$ curl -k --header "$HEADER" \
-X GET https://api.ocp4.example.com:6443/api/v1
{
    "kind": "APIResourceList",
    "groupVersion": "v1",
    "resources": [
        {
            "name": "bindings",
            "singularName": "",
            "namespaced": true,
            "kind": "Binding",
            "verbs": [
                "create"
            ],
            {
                "name": "componentstatuses",
                "singularName": "",
                "namespaced": false,
                "kind": "ComponentStatus",
                "verbs": [
                    "get",
                    "list"
                ],
                "shortNames": [
                    "cs"
                ],
            },
            ...
        ...output omitted...
    ]
}
```

4.2. Then, fetch a list of one of the resources, such as pods.

```
[student@workstation ~]$ curl -k --header "$HEADER" \
-X GET https://api.ocp4.example.com:6443/api/v1/pods
{
    "kind": "PodList",
    "apiVersion": "v1",
    "metadata": {
        "selfLink": "/api/v1/pods",
    }
}
```

```
"resourceVersion": "7148472"
},
"items": [
...output omitted...
```

- 4.3. Filter the results to list only the names that include jq.

```
[student@workstation ~]$ curl -sk --header "$HEADER" \
-X GET https://api.ocp4.example.com:6443/api/v1/pods/ \
| jq ".items[].metadata.name"
"nfs-client-provisioner-865c67f466-k2b98"
"nfs-client-provisioner-865c67f466-spfv6"
"openshift-apiserver-operator-54b67788df-7k9k9"
"apiserver-848cb9fb66-lmjf7"
"apiserver-848cb9fb66-ssl5w"
"apiserver-848cb9fb66-twmcq"
"authentication-operator-5b96cc9776-f8fcv"
"oauth-openshift-6dd57bd6b5-4krw9"
"oauth-openshift-6dd57bd6b5-5zdqk"
"cloud-credential-operator-594567c7b4-4xqwk"
...output omitted...
```

- 5. Locate the API endpoint for the Route resources using the oc command, and then fetch a list of all of the hosts using the curl command.

- 5.1. Use the oc explain command to discover the resource Kind and API Version.

```
[student@workstation ~]$ oc explain routes
KIND:     Route
VERSION:  route.openshift.io/v1
...output omitted...
```

- 5.2. List the available APIs for the route.openshift.io/v1 API version group.

```
[student@workstation ~]$ curl -k --header "$HEADER" \
-X GET https://api.ocp4.example.com:6443/apis/route.openshift.io/v1
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "route.openshift.io/v1",
  "resources": [
    {
      "name": "routes",
      "singularName": "",
      "namespaced": true,
      "kind": "Route",
      ...output omitted...
```

Notice that both routes and routes/status resources are located under the route.openshift.io/v1 API version group.

- 5.3. List the hosts for all routes using the curl and jq commands.

```
[student@workstation ~]$ curl -sk --header "$HEADER" \
-X GET https://api.ocp4.example.com:6443/apis/route.openshift.io/v1/routes \
| jq '.items[].spec.host'
"oauth-openshift.apps.ocp4.example.com"
"console-openshift-console.apps.ocp4.example.com"
"downloads-openshift-console.apps.ocp4.example.com"
"alertmanager-main-openshift-monitoring.apps.ocp4.example.com"
"grafana-openshift-monitoring.apps.ocp4.example.com"
"prometheus-k8s-openshift-monitoring.apps.ocp4.example.com"
"thanos-querier-openshift-monitoring.apps.ocp4.example.com"
```

Finish

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

```
[student@workstation ~]$ lab automation-rest finish
```

Writing Ansible Playbooks to Manage OpenShift Resources

Objectives

After completing this section, you should be able to automate creation and modification of OpenShift resources using the Ansible k8s modules.

Explaining the Kubernetes Ansible Modules

There are a handful of modules available to help manage and interact with Kubernetes. Those same modules also work with OpenShift because OpenShift is a distribution of Kubernetes. The shorthand name for Kubernetes, k8s, is used to refer to all of the Kubernetes modules and is the group name that prepends of all the Kubernetes modules.

k8s Modules in Ansible 2.9

Module name	Description
k8s	Manages Kubernetes objects. This module has access to all Kubernetes APIs, allowing you to create, update, or delete Kubernetes objects.
k8s_auth	Authenticates to Kubernetes clusters which require an explicit login step, like OpenShift. This module returns an <code>api_key</code> that the other modules can use for authentication.
k8s_info	Retrieves information about Kubernetes objects.
k8s_scale	Sets a new size for a Deployment, ReplicaSet, Replication Controller, or Job.
k8s_service	Manages Services on Kubernetes.

Most of these modules have common parameters like `api_key`, `context`, and `namespace`. Shared parameters can be set to default values using the `module_defaults` attribute.

module_defaults Example

```
- hosts: localhost
  module_defaults:
    group/k8s:
      api_key: "{{ auth_token }}"
      host: https://api.example.com/
      ca_cert: ca.pem
```

Example Playbook

```
- name: Log in to OpenShift
  hosts: localhost
  tasks:
```

```

- name: Log in (obtain access token)
  k8s_auth:
    host: https://api.ocp4.example.com:6443
    username: admin
    password: redhat
    validate_certs: false
  register: k8s_auth_results

- name: Demonstrate k8s modules
  hosts: localhost
  vars:
    namespace: dev
  module_defaults:
    group/k8s:
      namespace: "{{ namespace }}"
      api_key: "{{ hostvars['localhost']['k8s_auth_results']['k8s_auth']['api_key'] }}"
  tasks:
    - name: Create objects from the manifest
      k8s:
        state: present
        src: "{{ playbook_dir + '/files/manifest.yml' }}"

    - name: Get info about Pods that are web apps in dev or test
      k8s_info:
        kind: Pod
        label_selectors:
          - app = web
          - tier in (dev, test)

    - name: Scale deployment up
      k8s_scale:
        kind: Deployment
        name: example_deployment
        replicas: 3

    - name: Expose https port with ClusterIP
      k8s_service:
        state: present
        name: example_service
        ports:
          - port: 443
            protocol: TCP
        selector:
          key: value

```

**Note**

The apiVersion in the manifest might need to be more explicit for the `openshift` python package. Instead of putting `v1`, use the full endpoint, such as `authorization.openshift.io/v1`.

**Important**

The current implementation of the k8s modules does not respect role standards. When using `src` or `resource_definition` parameters, you must provide the full path. The modules will not check the standard `files` or `templates` directories in a role automatically.

Describing the K8s Modules Dependencies

Prerequisite python packages for the k8s, k8s_info, and k8s_scale modules are:

- `openshift` >= 0.6
- `PyYAML` >= 3.11

The k8s_service module does not require PyYAML and requires `openshift` >= 0.6.2.

The k8s_auth module requires:

- `urllib3`
- `requests`
- `requests-oauthlib`

Explaining the Module Authentication Options

Authenticate with the API by using either a `kubeconfig` file, HTTP Basic Auth, or authentication tokens.

If no connection options are provided, the `openshift` client attempts to load the default configuration file from `~/.kube/config`. The `kubeconfig` module parameter specifies a path to an existing Kubernetes config file. The `context` module parameter chooses a context from the `kubeconfig` file.

To authenticate via by HTTP Basic Auth, provide the `host`, `username`, and `password` module parameters. The optional `proxy` module parameter connects through an HTTP proxy.

The k8s_auth module provides an authentication token that can be used with the `api_key` module parameter to authenticate with the API.

Verify SSL certificates for the API server using the `ca_cert`, `client_cert`, and `client_key` module parameters. Toggle certificate validation with the `validate_certs` module parameter.

**References****K8S – Ansible Documentation**

https://docs.ansible.com/ansible/2.9/modules/list_of_clustering_modules.html#k8s

► Guided Exercise

Writing Ansible Playbooks to Manage OpenShift Resources

In this exercise you will create a playbook to automate the creation and modification of OpenShift resources.

Outcomes

You should be able to:

- Create and execute a playbook that uses the k8s modules.

Before You Begin

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

This command provides a working directory with a manifest file and a playbook with some boilerplate content.

```
[student@workstation ~]$ lab automation-ansible start
```

Instructions

- 1. Change to the `~/D0380/labs/automation-ansible` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/automation-ansible
```

- 2. Optionally, review the provided `hello.yaml` manifest. This manifest provides a `Hello world` application, a service, and a route.
- 3. Add tasks to the provided `k8s.yaml` playbook. The provided `k8s.yaml` playbook includes some boilerplate code.
- 3.1. Edit the `k8s.yaml` playbook with your preferred text editor. Add a task to create the OpenShift project. Use the provided `namespace` variable for the name of the project. Set the `namespace` parameter in the task to `""` to allow this task to use a value other than the `module_default` value.

```

- name: Demonstrate k8s modules
  hosts: localhost
  become: false
  ...output omitted...
  tasks:
    - name: Create the project
      k8s:
        api_version: project.openshift.io/v1
        kind: Project
        name: "{{ namespace }}"
        state: present
        namespace: ""

```

- 3.2. Add tasks to create the objects from the `hello.yml` manifest and view information about the new pods. Use the `src` or `resource_definition` parameter of the `k8s` module to read from the manifest file. The completed tasks should match the following:

```

...output omitted...
- name: Create objects from the manifest
  k8s:
    state: present
    src: "{{ playbook_dir + '/hello.yml' }}"

- name: Get a info about all of the pods in the namespace
  k8s_info:
    kind: Pod

```



Important

The `src` and `resource_definition` parameters of the `k8s` module do not behave the same as other modules with a `src` parameter, which automatically read from the playbook directory or the role's `files` directory. The absolute path is needed.

- 3.3. Add a task to scale the deployment up.

Scale the number of replicas to 3 using the `k8s_scale` module.

```

...output omitted...
- name: Scale deployment up
  k8s_scale:
    kind: Deployment
    name: hello
    replicas: 3

```

- 3.4. Add tasks to get information about the route, test the route, and then display the content from the application.

```

...output omitted...
- name: Get hostname of the route
k8s_info:
  kind: Route
  name: hello
  register: route

- name: Test access to the app
uri:
  url: "http://{{ route.resources[0].spec.host }}"
  return_content: yes
register: response
until: response.status == 200
retries: 10
delay: 5

- name: Display response of the application
debug:
  var: response.content

```

- 3.5. Optionally, compare your playbook to the solution file located at ~/DO380/solutions/automation-ansible/k8s.yml.

► 4. Run the k8s.yml playbook as the developer user.

- 4.1. Log in as the developer user.

```

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

```

- 4.2. Run the k8s.yml playbook using the ansible-playbook command with at least one level of verbosity to see the output of the tasks.

```

[student@workstation automation-ansible]$ ansible-playbook -v k8s.yml
Using /etc/ansible/ansible.cfg as config file

PLAY [Demonstrate k8s modules] ****
...output omitted...

PLAY RECAP ****
localhost                  : ok=7    changed=3    unreachable=0    failed=0
  skipped=0    rescued=0    ignored=0

```

**Important**

The lab script deploys a global Ansible configuration file to `/etc/ansible/ansible.cfg`. This global Ansible configuration file sets `become: true`. The `become: true` setting makes playbooks execute as the `root` user. The lab script runs as the `root` user and logs in to OpenShift as a user with administrator privileges.

The `k8s.yml` playbook specifies `become: false` to override the global configuration. This is required so the playbook executes as the `student` user, whose `kubeconfig` file is configured to interact with OpenShift as the `developer` user. Otherwise, the playbook tasks would execute as the `root` user and interact with OpenShift as a user with administrator privileges.

**Note**

The Ansible inventory file configures Ansible tasks executed against `localhost` to use a Python virtual environment. This Python virtual environment is in the `/home/student/.venv/openshift` directory. The virtual environment includes the dependencies required to run the Kubernetes Ansible modules.

▶ 5. Inspect the results of the playbook.

- 5.1. Use the `oc get all` command on the `automation-ansible` namespace to inspect the components created by the playbook.

```
[student@workstation automation-ansible]$ oc project automation-ansible
Now using project "automation-ansible" on server "https://api.ocp4.example.com:6443".

[student@workstation automation-ansible]$ oc get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/hello-76db5c69d5-ghrjj 1/1     Running   0          63s
pod/hello-76db5c69d5-wj4l5 1/1     Running   0          66s
pod/hello-76db5c69d5-xhtwm 1/1     Running   0          63s

NAME           TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/hello  ClusterIP  172.30.174.154  <none>          80/TCP       66s

NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello 3/3     3           3           66s

NAME                      DESIRED  CURRENT  READY   AGE
replicaset.apps/hello-76db5c69d5 3        3        3       66s

NAME          HOST/PORT
PATH  SERVICES  PORT  TERMINATION  WILDCARD
route.route.openshift.io/hello  hello-automation-ansible.apps.ocp4.example.com
                               hello     8080          None
```

- 5.2. Use the `curl` command to verify that the application is running.

```
[student@workstation automation-ansible]$ curl \
  hello-automation-ansible.apps.ocp4.example.com
<html>
  <body>
    <h1>Hello, world from nginx!</h1>
  </body>
</html>
```

- 6. Change to the /home/student directory.

```
[student@workstation automation-ansible]$ cd
```

Finish

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

```
[student@workstation ~]$ lab automation-ansible finish
```

Summary

In this chapter, you learned:

- How to use JSONPath to extract information from OpenShift resources.
- About service accounts and how to use them in Roles and RoleBindings.
- How to execute pods as Jobs and CronJobs.
- About the OpenShift REST API and how to navigate it.
- About the Ansible k8s modules and how to use them to automate OpenShift resources.

Chapter 3

Managing OpenShift Operators

Goal

Manage operators and OpenShift cluster operators.

Objectives

- Describe Kubernetes controllers and the operator pattern.
- Install and troubleshoot operators from the OperatorHub.
- Identify namespaces and resources of a cluster operator.

Sections

- Describing Operators (and Quiz)
- Installing Operators (and Guided Exercise)
- Managing Cluster Operators (and Guided Exercise)

Lab

Managing Operators

Describing Operators

Objectives

After completing this section, you should be able to describe Kubernetes controllers and the operator pattern.

Describing Operators

Kubernetes clusters desired persistent state is encoded using resources, such as pods and services. Resources are endpoints in the Kubernetes API, storing sets of API objects that represent the state of the cluster.

Controllers are responsible for maintaining the state of the cluster. By following a control loop in which the controller tracks one or more resource types, Controllers bring the current state of the objects closer to the desired state described within them.

The Kubernetes API is extended by adding Custom Resource Definitions (CRDs). CRDs integrate with the cluster as if they are native resources. In combination with Custom Controllers, CRDs define a true declarative API.

The operator pattern is a combination of CRDs and Custom Controllers that perform administrative tasks for API objects or the whole cluster.

Describing Operators in Kubernetes

The Kubernetes control plane runs:

- `cloud-controller-manager`
- `kube-controller-manager`

`cloud-controller-manager` controls standard Kubernetes resources that interact with a cloud provider. `cloud-controller-manager` includes:

Node controller

Manages Node resources.

Route controller

Manages cloud routes for node communication.

Service controller

Manages cloud load balancers to publish services.

`kube-controller-manager` controls standard Kubernetes resources that do not interact with a cloud provider.

Describing Operators in OpenShift

OpenShift operators implement OpenShift features such as self-healing, updates, and other administrative tasks, either on resources or cluster-wide actions. Operators package, deploy, and manage an OpenShift application.

OpenShift uses operators to manage the cluster by controlling tasks, such as upgrades and self-healing, among others. For example, the Cluster Version Operator manages cluster operators and their upgrades.

Additional operators can run applications or add features to OpenShift.

There are several operators in an OpenShift cluster, such as:

- The OperatorHub, which is a registry for OpenShift operators.
- The Operator Lifecycle Manager, which installs, updates, and manages operators.

Use either the CLI or the web console to install operators located in the OperatorHub.



References

For more information, refer to the *Understanding Operators* chapter in the Red Hat OpenShift Container Platform 4.6 Operators documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/operators/index#olm-what-operators-are

Controllers

<https://kubernetes.io/docs/concepts/architecture/controller/>

Operator pattern

<https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>

► Quiz

Describing Operators Quiz

Choose the correct answers to the following questions:

- ▶ 1. **Which of the following methods is recommended for installing operators in an OpenShift cluster?**
 - a. Helm files
 - b. Manual installation
 - c. OperatorHub

- ▶ 2. **Choose the correct statement about OpenShift operators.**
 - a. The operator pattern consists of creating custom resources.
 - b. Custom resources and a controller are used to create operators.
 - c. Kubernetes operators are humans that manage the cluster.

- ▶ 3. **Choose the correct statement about Kubernetes controllers.**
 - a. Kubernetes controllers maintain the cluster resources in a desired state.
 - b. Kubernetes controllers are not part of the cluster.
 - c. A controller never talks to the OpenShift API.

- ▶ 4. **Choose the correct statement about OpenShift operator custom resource definitions.**
 - a. Kubernetes ignores OpenShift operators custom resource definitions.
 - b. Custom resource definitions create pods.
 - c. OpenShift operators require the creation of custom resources based on the operator CRDs.

► Solution

Describing Operators Quiz

Choose the correct answers to the following questions:

- ▶ 1. **Which of the following methods is recommended for installing operators in an OpenShift cluster?**
 - a. Helm files
 - b. Manual installation
 - c. OperatorHub

- ▶ 2. **Choose the correct statement about OpenShift operators.**
 - a. The operator pattern consists of creating custom resources.
 - b. Custom resources and a controller are used to create operators.
 - c. Kubernetes operators are humans that manage the cluster.

- ▶ 3. **Choose the correct statement about Kubernetes controllers.**
 - a. Kubernetes controllers maintain the cluster resources in a desired state.
 - b. Kubernetes controllers are not part of the cluster.
 - c. A controller never talks to the OpenShift API.

- ▶ 4. **Choose the correct statement about OpenShift operator custom resource definitions.**
 - a. Kubernetes ignores OpenShift operators custom resource definitions.
 - b. Custom resource definitions create pods.
 - c. OpenShift operators require the creation of custom resources based on the operator CRDs.

Installing Operators

Objectives

After completing this section, you should be able to:

- Install and troubleshoot operators from the OperatorHub.
- List workload and custom resources from an operator.

Installing Operators from the OperatorHub

There are several ways to install operators in an OpenShift cluster, including: OperatorHub, Helm charts, and custom YAML files, among others. The recommended way to install operators is from the OperatorHub, using either the Web Console or the CLI. Operators that come with OpenShift (also called cluster operators) are managed by the Cluster Version Operator, and operators that are installed from the OperatorHub are managed by the Operator Lifecycle Manager (OLM).

Operators are installed to access one or more namespaces depending on the application or task performed by the operator.

Operators follow a maturity model that is divided in phases, that range from basic installation to full automation.

There are three initial settings that must be defined when installing an operator.

Installation mode

An operator can be installed on all namespaces or on an individual namespace, if supported.

Update Channel

If the operator is available through multiple channels, this option selects the channel to which the operator will subscribe.

Approval strategy

This option determines whether the operator is updated, either automatically or manually.

Installing an Operator from the OLM by Using the CLI

Check the list of available operators.

[user@host ~]\$ oc get packagemanifests			
ionir-operator	Certified Operators	28d	
aws-event-sources-operator-certified-rhmp	Red Hat Marketplace	28d	
jtrac-app-operator-rhmp	Red Hat Marketplace	28d	
seldon-operator-certified-rhmp	Red Hat Marketplace	28d	
ham-deploy	Community Operators	28d	
mtc-operator	Red Hat Operators	28d	
cortex-certifai-operator	Certified Operators	28d	
coralogix-operator-certified	Certified Operators	28d	
cert-manager-operator	Certified Operators	28d	

snyk-operator-marketplace-rhmp	Red Hat Marketplace	28d
nfs-provisioner-operator	Community Operators	28d
<i>...output omitted...</i>		

Inspect an operator. Each operator adds new resource types using Custom Resource Definitions (CRDs).

```
[user@host ~]$ oc describe packagemanifests file-integrity-operator
Name:           file-integrity-operator
Namespace:      openshift-marketplace
Labels:         catalog=redhat-operators
                catalog-namespace=openshift-marketplace
                operatorframework.io/arch.amd64=supported
                operatorframework.io/os.linux=supported
                provider=Red Hat
                provider-url=https://github.com/openshift/file-integrity-operator
...output omitted...
Customresourcedefinitions:
  Owned:
    Description: FileIntegrity is the Schema for the fileintegrities API
    Kind:        FileIntegrity
    Name:        fileintegrities.fileintegrity.openshift.io
    Version:     v1alpha1
    Description: FileIntegrityNodeStatus defines the status of a specific
node
    Kind:        FileIntegrityNodeStatus
    Name:        fileintegritynodestatuses.fileintegrity.openshift.io
    Version:     v1alpha1
...output omitted...
```

In the CLI, there is no direct way to get the operator channels so you need a JSONPath query.

```
[user@host ~]$ oc get packagemanifests file-integrity-operator \
--output='jsonpath={range .status.channels[*]}{.name}{"\n"}{end}'
4.6
release-0.1
```

Install an Operator

Create a namespace for the operator. This namespace is needed when the install mode is SingleNamespace.



Note

Operators need standard Kubernetes namespaces instead of OpenShift projects.

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: <namespace>
```

Create an operator group object. An operator group selects target namespaces in which to generate required RBAC access for all operators sharing the same namespace.

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup-name>
  namespace: <namespace>
spec:
  targetNamespaces:
    - <namespace>
```

Create a subscription object to subscribe a namespace to an operator.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription-name> ①
  namespace: <namespace> ②
spec:
  channel: "4.6"
  name: <operator-name> ③
  source: redhat-operators ④
  sourceNamespace: openshift-marketplace ⑤
```

- ① Name of the subscription
- ② Namespace in which the operator will run
- ③ Name of the operator to subscribe to
- ④ Catalog source that provides the operator
- ⑤ Namespace of the catalog source

The OLM creates the operator and then begins creating and running the proper resources.

Verifying Operator Status

There are several ways to check operator status. The remainder of this section provides additional information about inspecting event logs, subscription objects, and namespace information to verify the status of an operator.

Check the logs of the OLM pod to verify operator installation.

```
[user@host ~]$ oc logs pod/olm-operator-c5599dfd7-nknfx \
-n openshift-operator-lifecycle-manager
```

Inspect the subscription object to verify the operator status.

```
[user@host ~]$ oc describe sub <subscription-name> -n <namespace>
...output omitted...
```

List All Operators

List installed operators by checking cluster service versions. To list all the installed operators, get the cluster service versions in all namespaces.

```
[user@host ~]$ oc get csv -A
```

Operators can be subscribed to one namespace or to all namespaces. To list the operators managed by the OLM, list the active subscriptions.

```
[user@host ~]$ oc get subs -A
```

To view the status and events from custom resources related to a given operator, describe the operator deployment.

```
[user@host ~]$ oc describe deployment.apps/file-integrity-operator | \
grep -i kind
          olm.owner.kind=ClusterServiceVersion
          "kind": "FileIntegrity",

[user@host ~]$ oc get crd | grep -i fileintegrity
fileintegrities.fileintegrity.openshift.io           2021-08-25T19:33:33Z

[user@host ~]$ oc describe crd fileintegrities.fileintegrity.openshift.io | \
grep -w Kind
Kind:      CustomResourceDefinition
Kind:      FileIntegrity
List Kind: FileIntegrityList
```

Check the operator elements by getting all objects in the operator's namespace. If the operator is installed in all the namespaces, then make the query in the `openshift-operators` namespace and look for the name of the operator to discover the elements.

```
[user@host ~]$ oc get all -n openshift-file-integrity
NAME                           READY   STATUS    RESTARTS   AGE
pod/file-integrity-operator-796c967948-5rtc6   1/1     Running   0          12m

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)     AGE
service/file-integrity-operator-metrics   ClusterIP   172.30.32.169   <none>
8383/TCP, 8686/TCP   12m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
```

deployment.apps/file-integrity-operator	1/1	1	1	12m
NAME	DESIRED	CURRENT	READY	
AGE				

NAME	DESIRED	CURRENT	READY	
replicaset.apps/file-integrity-operator-796c967948	1	1	1	
12m				

Use the `oc logs` command to view Events and logs of an operator.

```
[user@host ~]$ oc logs deployment.apps/file-integrity-operator
{"level":"info","ts":1629920026.6320865,"logger":"cmd","msg":"Go Version:
go1.15.7"}
{"level":"info","ts":1629920026.6321144,"logger":"cmd","msg":"Go OS/Arch: linux/
amd64"}
 {"level":"info","ts":1629920026.6321173,"logger":"cmd","msg":"Version of operator-
sdk: v0.18.1"}
...output omitted...
 {"level":"info","ts":1629920039.4583714,"logger":"metrics","msg":"Metrics
Service object created","Service.Name":"file-integrity-operator-
metrics","Service.Namespace":"openshift-file-integrity"}
 {"level":"info","ts":1629920042.0481858,"logger":"cmd","msg":"Starting the Cmd."}
...output omitted...
```

Updating an Operator from the OLM Using the CLI

Modify the operator's YAML file and apply the desired changes to the subscription object.

```
[user@host ~]$ oc apply -f file-integrity-operator-subscription.yaml
```

If the update presents failures, check the operator logs for errors or warnings. Also, check the components and prerequisites because they might be the cause of the problem.

Deleting Operators

To remove an operator from the cluster, delete the subscription and cluster service version objects.

Check the current version of the subscribed operator in the `currentCSV` field.

```
[user@host ~]$ oc get sub <subscription-name> -o yaml | grep currentCSV
currentCSV: ...output omitted...
```

Delete the subscription object. Use the value obtained from the preceding command to delete the cluster service version object.

```
[user@host ~]$ oc delete sub <subscription-name>
[user@host ~]$ oc delete csv <currentCSV>
```



References

For more information about the OLM, refer to the *Understanding Operators* chapter in the Red Hat OpenShift Container Platform 4.6 Operators documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/operators/index#olm-what-operators-are

For more information about adding operators to an OpenShift Cluster, refer to the *Adding Operators to a cluster* chapter in the Red Hat OpenShift Container Platform 4.6 Operators documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/operators/index#olm-adding-operators-to-a-cluster

► Guided Exercise

Installing Operators

In this exercise you will install an operator from the OpenShift marketplace.

Outcomes

You should be able to install the OpenShift metering operator using the OLM.

Before You Begin

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

This command ensures that the cluster API is reachable and that needed resources are set up for the exercise.

```
[student@workstation ~]$ lab operators-install start
```

Instructions

- 1. Create a namespace for the operator.

The file integrity operator requires a namespace called `openshift-file-integrity`.

- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Create a file called `file-integrity-operator-namespace.yaml` with the following content:

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-file-integrity
```

- 1.3. Create the namespace.

```
[student@workstation ~]$ oc apply -f file-integrity-operator-namespace.yaml
namespace/openshift-file-integrity created
```

- 1.4. Enter the new namespace.

```
[student@workstation ~]$ oc project openshift-file-integrity
Now using project "openshift-file-integrity" on server "https://
api.ocp4.example.com:6443".
```

► 2. Create a group for the operator.

Operators that are installed using the OLM require a group that matches the namespace name.

- 2.1. Create a file called `file-integrity-operator-group.yaml` and add the following content:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  targetNamespaces:
    - openshift-file-integrity
```

- 2.2. Create the operator group.

```
[student@workstation ~]$ oc apply -f file-integrity-operator-group.yaml
operatorgroup.operator.coreos.com/file-integrity-operator created
```

► 3. Create the operator subscription.

The operator must be subscribed to a namespace.

- 3.1. Create the subscription object YAML file called `file-integrity-operator-subscription.yaml`.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: file-integrity-operator-sub
  namespace: openshift-file-integrity
spec:
  channel: "4.6"
  name: file-integrity-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

- 3.2. Create the subscription object.

```
[student@workstation ~]$ oc apply -f file-integrity-operator-subscription.yaml
subscription.operator.coreos.com/file-integrity-operator-sub created
```

► 4. Verify that the operator installed successfully.

```
[student@workstation ~]$ oc describe sub file-integrity-operator-sub
Name:           file-integrity-operator-sub
Namespace:      openshift-file-integrity
Labels:         operators.coreos.com/file-integrity-operator.openshift-file-
integrity=
Annotations:   <none>
API Version:   operators.coreos.com/v1alpha1
Kind:          Subscription
...output omitted...
Spec:
  Channel:      4.6
  Name:         file-integrity-operator
  Source:       redhat-operators
  Source Namespace: openshift-marketplace
...output omitted...
```

Finish

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

```
[student@workstation ~]$ lab operators-install finish
```

Managing Cluster Operators

Objectives

After completing this section, you should be able to identify and manage cluster operators and their components.

Describing Cluster Operators

OpenShift uses operators to perform internal cluster functions, such as authentication, DNS, and the web console. These operators are called cluster operators because of the internal cluster functions they provide. Cluster operators are similar to operators, but they are managed by the Cluster Version Operator (CVO) instead of the OLM.

Cluster operators were previously referred to as Platform operators.



Note

Not all cluster operators define `CustomResourceDefinitions` nor strictly follow the Controller pattern.

Use the following command to list the cluster operators and their status.

```
[user@host ~]$ oc get clusteroperator
NAME          VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
authentication  4.6.36   True       False        False     14d
cloud-credential 4.6.36   True       False        False     14d
cluster-autoscaler 4.6.36   True       False        False     14d
config-operator  4.6.36   True       False        False     14d
console         4.6.36   True       False        False     46h
csi-snapshot-controller 4.6.36   True       False        False     14d
...output omitted...
```

AVAILABLE

The cluster operator is working correctly.

PROGRESSING

The Cluster Version Operator is making changes to this operator.

DEGRADED

The cluster operator has detected a problem and it may not be working correctly.

The console also shows this information in **Administration** → **Cluster Settings** → **Cluster Operators**.

Describing the Cluster Version Operator

The Cluster Version Operator manages installation and upgrades of cluster operators. During installation and upgrades, the Cluster Version Operator scans a release image, locating cluster operators and applying their resources.

Examine the contents of the release image.

```
[user@host ~]$ VER=$(oc get clusterversion \
-o jsonpath='{.status.desired.image}' version)

[user@host ~]$ oc adm release extract --from=$VER --to=release-image

[user@host ~]$ grep -l "kind: ClusterOperator" release-image/*
...output omitted...
release-image/0000_50_cluster-samples-operator_07-clusteroperator.yaml
...output omitted...

[user@host ~]$ cat release-image/*-samples*clusterop*
apiVersion: config.openshift.io/v1
kind: ClusterOperator
metadata:
  name: openshift-samples
spec: {}
status:
  versions:
    - name: operator
      version: "4.6.36"
```



References

- For more information on the architecture of cluster operators, refer to the *Operators in OpenShift Container Platform* section in the *The OpenShift Container Platform control plane* chapter in the Red Hat OpenShift Container Platform 4.6 Architecture documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/architecture/index#operators-overview_control-plane
- For more information on the cluster operators in an OpenShift cluster, refer to the *Red Hat Operators* chapter in the Red Hat OpenShift Container Platform 4.6 *Operators* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/operators/red-hat-operators#red-hat-operators

► Guided Exercise

Managing Cluster Operators

In this exercise you will examine the Cluster Samples Operator.

Outcomes

You should be able to:

- View the Cluster Samples Operator components and associated logs.

Before You Begin

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

This command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab operators-cluster start
```

Instructions

- 1. Examine the Cluster Samples Operator components by extracting the manifests that define them from the release image.
- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Extract the release.

```
[student@workstation ~]$ VER="$(oc get clusterversion version \
-o jsonpath='{.status.desired.image}')"

[student@workstation ~]$ oc adm release extract --from=$VER --to=release-image
Extracted release payload created at 2021-07-28T05:51:05Z
```

- 1.3. View the Cluster Samples Operator manifests.

```
[student@workstation ~]$ ls release-image/*samples*
release-image/0000_10_samplesconfig.crd.yaml
release-image/0000_50_cluster-samples-operator_010-prometheus-rules.yaml
...output omitted...
```

- 1.4. Examine the Cluster Samples Operator deployment.

```
[student@workstation ~]$ less release-image/*samples*06-operator.yaml  
...output omitted...
```

- 2. Examine the Cluster Samples Operator components.

- 2.1. List the Cluster Samples Operator pods.

```
[student@workstation ~]$ oc get pods -n openshift-cluster-samples-operator  
NAME                               READY   STATUS    RESTARTS   AGE  
cluster-samples-operator-5d7d5b94b6-rw284   2/2     Running   0          4d15h
```

- 2.2. View logs for the Cluster Samples Operator pods.

```
[student@workstation ~]$ oc logs pod/cluster-samples-operator-5d7d5b94b6-rw284 \  
-n openshift-cluster-samples-operator --all-containers  
...output omitted...
```

Finish

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

```
[student@workstation ~]$ lab operators-cluster finish
```

► Lab

Managing Operators

In this lab, you will install and troubleshoot operators.

Outcomes

You should be able to:

- Install the OpenShift file-integrity operator from the OperatorHub.
- Update the channel of an operator.
- Verify the CVO restoration of a cluster operator.

Before You Begin

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

This command ensures that the environment for this lab is set up properly.

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

Instructions

1. Install the OpenShift file integrity operator by using the `file-integrity-operator.yaml` file located in the `/home/student/D0380/labs/operators-review` directory. This file has errors that must be fixed for a successful installation. Find these errors by using the `oc apply` command.
2. Update the file integrity operator channel to "**4.6**" by modifying the `channel` field in the `file-integrity-operator.yaml` file. Use the `oc apply` command for the update.
3. Delete the project `openshift-cluster-storage-operator` to confirm that the CVO restores the cluster storage operator.

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 operators-review grade
```

Finish

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

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


► Solution

Managing Operators

In this lab, you will install and troubleshoot operators.

Outcomes

You should be able to:

- Install the OpenShift file-integrity operator from the OperatorHub.
- Update the channel of an operator.
- Verify the CVO restoration of a cluster operator.

Before You Begin

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

This command ensures that the environment for this lab is set up properly.

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

Instructions

1. Install the OpenShift file integrity operator by using the `file-integrity-operator.yaml` file located in the `/home/student/D0380/labs/operators-review` directory. This file has errors that must be fixed for a successful installation. Find these errors by using the `oc apply` command.
 - 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Change to the `~/D0380/labs/operators-review` directory. Apply the `file-integrity-operator.yaml` file and identify the errors.

```
[student@workstation ~]$ cd ~/D0380/labs/operators-review
[student@workstation operators-review]$ oc apply -f file-integrity-operator.yaml
unable to decode "file-integrity-operator.yaml": Object 'Kind' is missing in
'{"apiVersion":"v1","kind":"Namespace","metadata":{"labels":{"openshift.io/
cluster-monitoring":"true"},"name":"openshift-file-integrity"}}'
Error from server (NotFound): error when creating "file-integrity-operator.yaml":
namespaces "openshift-file-integrity" not found
Error from server (NotFound): error when creating "file-integrity-operator.yaml":
namespaces "openshift-file-integrity" not found
```

- 1.3. Edit the `file-integrity-operator.yaml` file with your favorite editor or use the `sed` command to fix the kind field name for the namespace object.

```
[student@workstation operators-review]$ sed -i 's/^kind:/\'
file-integrity-operator.yaml
```

- 1.4. Verify the changes made to the `file-integrity-operator.yaml` file.

```
[student@workstation operators-review]$ cat file-integrity-operator.yaml
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-metering
...
...output omitted...
```

```
[student@workstation operators-review]$ oc apply -f file-integrity-operator.yaml
namespace/openshift-file-integrity created
operatorgroupoperators.coreos.com/file-integrity-operator created
subscriptionoperators.coreos.com/file-integrity-operator-sub created
```

2. Update the file integrity operator channel to "**4.6**" by modifying the `channel` field in the `file-integrity-operator.yaml` file. Use the `oc apply` command for the update.

- 2.1. Edit the `file-integrity-operator.yaml` and change the value of the `channel` field from `alpha` to "`4.6`".



Note

Numeric channel values must be enclosed in double quotes.

- 2.2. Apply the change by using `oc apply`.

```
[student@workstation operators-review]$ oc apply -f file-integrity-operator.yaml
namespace/openshift-file-integrity unchanged
operatorgroupoperators.coreos.com/file-integrity-operator unchanged
subscriptionoperators.coreos.com/file-integrity-operator-sub configured
```

- 2.3. Verify the update by checking the pod status.

```
[student@workstation operators-review]$ oc get subscription \
  file-integrity-operator-sub -n openshift-file-integrity
NAME                  PACKAGE          SOURCE      CHANNEL
file-integrity-operator-sub  file-integrity-operator  redhat-operators  4.6
```

3. Delete the project `openshift-cluster-storage-operator` to confirm that the CVO restores the cluster storage operator.

- 3.1. Delete the `openshift-cluster-storage-operator` project.

```
[student@workstation operators-review]$ oc delete project \
  openshift-cluster-storage-operator
```

- 3.2. Run the `watch` command to verify that the project is restored.

```
[student@workstation operators-review]$ watch oc get project \
  openshift-cluster-storage-operator
```



Note

It can take up to five minutes to restore the project.

- 3.3. After the `openshift-cluster-storage-operator` project is restored, press `Ctrl+C` to exit the `watch` command.
- 3.4. Change to the `/home/student` directory.

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

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

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

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

Summary

In this chapter, you learned:

- About operators and controllers.
- How to install operators from the OpenShift OperatorHub.
- How to examine cluster operators.

Chapter 4

Implementing GitOps with Jenkins

Goal

Implement a GitOps workflow using containerized Jenkins to administer an OpenShift cluster.

Objectives

- Describe Jenkins concepts and Jenkins Pipeline concepts.
- Deploy Jenkins on OpenShift using standard templates and image streams.
- Configure cluster operators using resource files in a declarative fashion.
- Automate the configuration of cluster operators from resources in Git using Jenkins Pipelines.

Sections

- Introducing Jenkins Declarative Pipelines (and Quiz)
- Deploying Jenkins on OpenShift (and Guided Exercise)
- Configuring OpenShift Resources Using a Declarative GitOps Workflow (and Guided Exercise)
- Configuring OpenShift using GitOps and Jenkins (and Guided Exercise)

Lab

Implementing GitOps with Jenkins

Introducing Jenkins Declarative Pipelines

Objectives

After completing this section, you should be able to describe Jenkins concepts and Jenkins Pipeline concepts.

Introducing Continuous Integration (CI) and Continuous Deployment (CD)

Jenkins is currently the most popular tool for automating software development processes. Created as a tool for managing software build processes, Jenkins evolved to deliver the Continuous Integration (CI) process, and is also flexible enough to support Continuous Deployment (CD) and GitOps processes.

The following figure shows a sample CI/CD workflow, illustrating how CI and CD processes are usually integrated into a larger process that allows a developer to make changes to an application source code and have quick access to a running instance of that application.

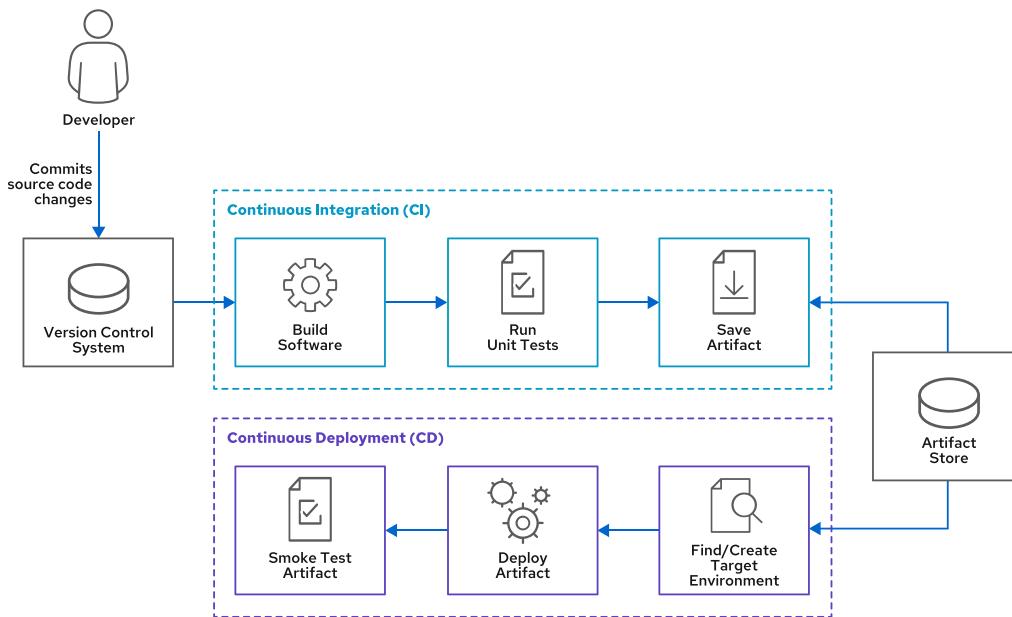


Figure 4.1: Sample CI/CD Workflow

Continuous Integration (CI) Workflows

Continuous Integration (CI) is the process of automatically building a software piece from its source code and any required dependencies, such as programming libraries, whenever the source code changes.

To be successful and reliable, CI workflows require automated testing, such as *unit testing*. A more sophisticated CI workflow might include many testing activities, such as security scanning and code coverage reports.

The final software artifact generated by a CI process can be anything that makes sense as a distributable piece of software, such as a native executable file, an RPM package, a Java library archive (JAR file), or a container image.

Continuous Delivery (CD) Workflows

Continuous Deployment (CD) is the process of automatically deploying a software piece to a target environment, such as development or production, whenever the source code changes.

To be successful and reliable, CI workflows also require automated testing, such as *integration testing*. It is also common to add manual approval gates to a CI workflow because other testing activities, such as *functional testing* and *user acceptance testing*, might not be fully automated.

CD workflows can depend on accessing a preconfigured target environment and be limited by the availability of that environment. More sophisticated CD workflows might create a target environment in a cloud environment, for example, by running Ansible Playbooks.

Container runtimes and Kubernetes clusters are popular ways to quickly provide temporary target environments for running tests before deploying in a real production environment.

GitOps Workflows

Compare and contrast the following with the diagram of the CI/CD workflow presented previously.

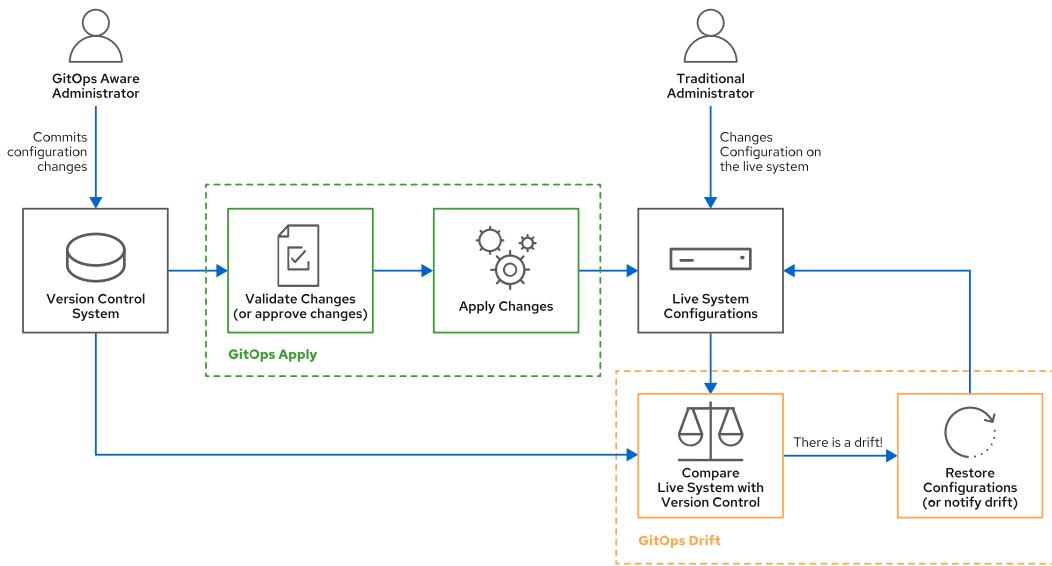


Figure 4.2: Sample GitOps Workflow

GitOps is a process that stores configurations of a target environment, such as a server or a cluster, in files managed by a version control system. It works under the assumption that system administrators do not change configurations on a live system directly. The best practice is to change the configurations under version control, and optionally have a fellow system administrator review them, before applying to a live system.

Typical GitOps workflows have two pieces: one that takes configurations from version control and applies those configurations to live systems, and another that verifies if the configurations from live systems have changed (drifted) compared to the current configuration under version control.

Comparing CI/CD and GitOps

CI/CD workflows are designed from a developer's point of view and GitOps workflows are designed from a system administrator's point of view.

Both GitOps and CI/CD assume that input comes from a version control system, and that some companion tool automatically reacts to changes committed to version control to produce some desired outcomes.

The outcomes from a CI/CD workflow is a running application and the side effect of a GitOps workflow is a live configuration.

It is a common pattern that a CI/CD process generates artifacts that serve as input to a GitOps process. For example, a CI/CD workflow generates a container image for an application and tests that application in a staging environment. A GitOps Workflow sets up the production environment for that same application.

Most tools that are capable of automating CI/CD processes are also capable of automating GitOps processes. For some organizations, tools designed for GitOps are replacing a CI/CD tools to perform the CD part of the workflow.

Introducing Tools for CI/CD

OpenShift features, such as Source-to-Image (S2I), build configurations, deployment configurations, and image streams, work together to implement simple CI/CD processes. If you deploy an application from source code using the `oc new-app` command, then OpenShift sets up these resources in a CI/CD workflow.

When you require more flexibility and power than provided by S2I, OpenShift integrates well with external CI/CD tools such as Jenkins. OpenShift includes container images and templates to deploy containerized Jenkins servers that are discussed elsewhere in this course.

Other popular CI/CD tools include OpenShift Pipelines, based on the Tekton open source project (Technical Preview as of Red Hat OpenShift Container Platform 4.6), and Jenkins X, an open source project that is creating a cloud-native variant of Jenkins.

There are also newer tools that specialize in GitOps processes, such as ArgoCD. These tools are usually not flexible enough to implement complete CI/CD workflows, but they compensate by making simpler to implement and manage GitOps workflows.

It is becoming a recommended best practice within the Kubernetes community to combine a Kubernetes-native CI/CD tool, such as OpenShift Pipelines, with a Kubernetes-native GitOps tool, such as ArgoCD, in larger integrated processes.

Introducing Jenkins

Jenkins is the leading open source automation server; it supports building, deploying, and automating any software development task.

Jenkins started as the Hudson build server, with a focus on building Java applications. In time, it evolved into a more general build automation server capable of automating generic CI and other software development workflows.

IT organizations that have already invested in Jenkins favor using it to implement GitOps workflows. Some organizations considering GitOps realize that they must also implement CI/CD to realize the benefits of GitOps and opt to automate it all with Jenkins. Other organizations consider it a major disadvantage that Jenkins is not Kubernetes-native, and look for alternatives.

Jenkins is a Java application that can be deployed to Servlet containers, such as Tomcat and Jetty, or application servers, such as JBoss EAP. Because Jenkins is coded in Java, Jenkins servers can run on any operating system, which contributed to its popularity.

Despite being coded in Java, Jenkins is known for its wide ecosystem of plug-ins that target most other popular programming language run times.

Describing Essential Jenkins Concepts

Jenkins is a very sophisticated automation server. The following are essential concepts and terms for using and managing Jenkins.

Project (or Job)

A script that describes a workflow that Jenkins should perform, such as building an application.

Pipeline

A kind of Job that follows the pipeline concept and syntax and describes a workflow as a sequence of steps that are grouped in stages.

Build

A single execution of a project, including its runtime logs and output artifacts. This term comes from Jenkins' origins as a software build server.

Node

A server or container that runs builds.

Worker

A thread in a master node that either runs a build or orchestrates available agents.

Workspace

A file system folder, dedicated to a project and sometimes also to a node, where builds store data that is either temporary, or reused between multiple builds of the same project.

Credential

A Jenkins construct that provides projects and builds with access credentials to external resources. There are different credentials to store user name and password pairs, SSH keys, and other credential types.

Plug-in

Almost all of Jenkins functionality is extensible by plug-ins written in Java. There are many community plug-ins that support different kinds of nodes, credentials, and programming languages.

Types of Jenkins Nodes

There are two kinds of Jenkins nodes:

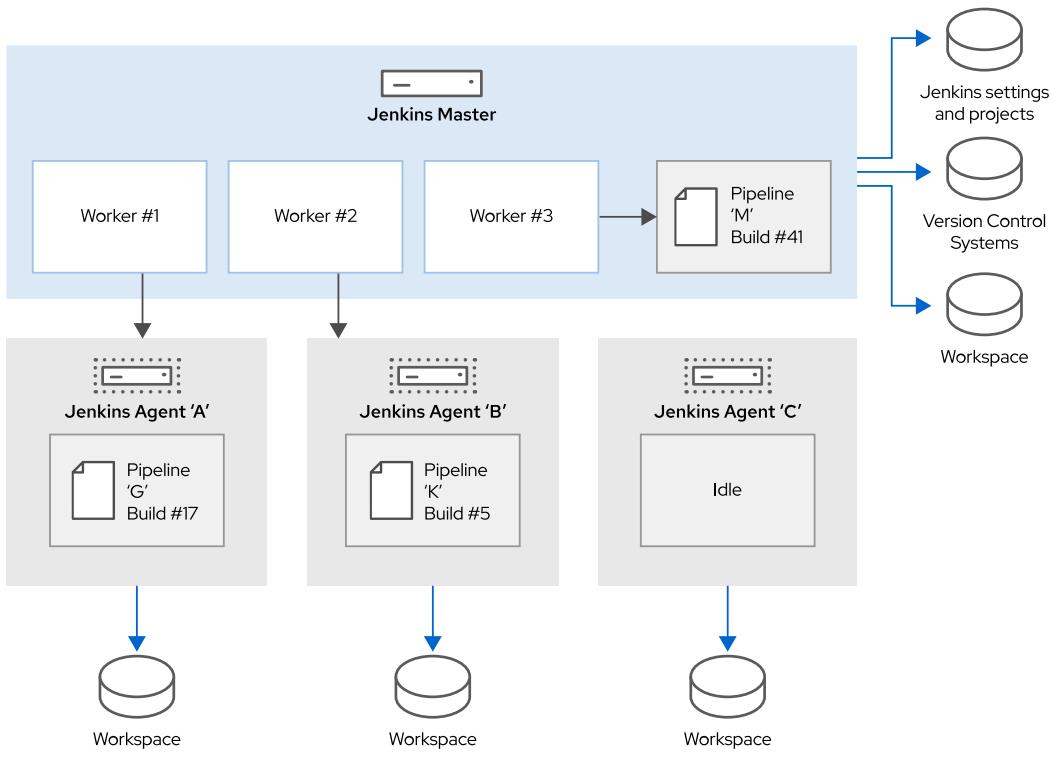
Master

Stores definitions of projects and their builds.

Agent

Run builds (or parts of a build) under the control of a master node.

A Jenkins instance contains one (and only one) master node and zero or more agent nodes. A master node is also an agent node, so a simple Jenkins deployment can have a single node. The following figure shows how the previous concepts of Jenkins projects and builds relate to the types of Jenkins nodes.

**Figure 4.3: Jenkins Architecture**

A master node provides a web UI and REST APIs to manage projects and builds.

Agents can be bare-metal servers, local containers, or pods in a Kubernetes cluster. Agent nodes also spread the load of running jobs, freeing up resources and capacity of a master node.

You can tune agent nodes according to the specific requirements of an application, such as the version of an operating system, independent of the masters. For example, a Jenkins master running Linux could use a Jenkins agent running Windows in a multiplatform application.

Managing Jenkins

The Jenkins web UI, also known as the *Jenkins Dashboard* because of its welcome page, is a web application that enables managing all aspects of a Jenkins server. The following list illustrates the main things that you can manage through the Jenkins Dashboard:

- Projects and builds for each project.
- Folders that group projects related to a business unit.
- Security settings that allow a Jenkins user to manage the server, or only those projects the user owns.
- Installed and available plug-ins.
- Online and offline agent nodes.

Jenkins also provides both a Java-based CLI and a REST API that allow limited management of a Jenkins server, including creating projects and starting builds.

A standard Jenkins setup provides few automation capabilities and is very GUI-oriented. Several plug-ins, for example the Configuration as Code (CasC) plug-in, add more extensive automation capabilities to Jenkins. These plug-ins are beyond the scope for this course.

Introducing Jenkins Declarative Pipelines

You can define a pipeline from the Jenkins Dashboard using an interactive graphical builder, and store it in a Jenkins instance. The preferred approach is writing a `Jenkinsfile` using a text editor and storing it on a version control system, such as Git.

A `Jenkinsfile` is a text file using Groovy syntax, which is very similar to Java and JavaScript. There are two possible styles for writing a `Jenkinsfile`:

Scripted pipelines

Pipelines that start with a `node` directive and define imperative scripts using the full Groovy programming language.

Declarative pipelines

Pipelines that start with a `pipeline` directive and define declarative scripts using a special-purpose domain-specific language (DSL) that is a subset of Groovy.

Many Jenkins plug-ins provide DSL statements and directives to perform tasks, such as running Java Maven builds, running NUnit integration tests, resolving NPM dependencies, and building container images. Note that these plug-ins may depend on software tools, such as the Java Development Kit (JDK) and the Node Package Manager (NPM) on agent nodes.

The declarative style is preferred because it is simpler. If you find a plug-in whose DSL statements require the scripted style, then you can add `script` blocks to a declarative pipeline and use these statements.

Describing the Structure of a `Jenkinsfile`

The following example illustrates a minimal `Jenkinsfile`, using the declarative style, that includes comments and variable substitution.

```
pipeline { ①
    agent any ②
    stages {
        stage('Example') { ③
            // use interactive input judiciously ④
            input { ⑤
                message 'Should we continue?'
                ok 'Yes, we should.'
                parameters {
                    string(name: 'PERSON', defaultValue: 'Mr Jenkins', ⑥
                        description: 'Who should I say hello to?')
                }
            }
            steps {
                echo "Hello, ${PERSON}, nice to meet you." ⑦
            }
        }
    }
}
```

① Most directives require brackets.

- ❷ This entire pipeline runs in any agent available, including the master node.
- ❸ The Jenkins Dashboard displays the name of each stage and the overall status for each build.
- ❹ Example of a single-line comment.
- ❺ Sample input directive that displays a text entry box and assigns the value to the PERSON variable.
- ❻ Line breaks and indentation are allowed inside a statement or directive.
- ❼ Pipelines (and Groovy) support string quoting, escaping, and variable substitution rules similar to Bash commands.

Most directives inside a declarative pipeline are optional.

- The `pipeline` directive requires one `agent` and one `stages` directive.
- The `stages` directive requires one or more `stage` directives.
- A `stage` directive requires one or more `steps` directives.

The following directives are allowed inside `pipelines` directive:

- `triggers`: defines conditions that fire automatic execution of builds from that pipeline.
- `options`: defines general configuration settings for the pipeline, overriding most properties from the web UI, for example: timeouts for running builds and retention of build logs.
- `parameters`: defines parameters that a user, or an upstream pipeline, can provide for running a build.
- `environment`: defines environment variables available inside a pipeline or a stage.
- `agent`: defines which agent nodes should execute, either all stages or a single stage of the pipeline.

Some directives, for example `agent`, can occur at different levels inside a pipeline, for example, at either `pipeline` or `stage`.

Most directives can be defined in any order inside a pipeline or stage.

The `stage` directive defines a logical piece of a workflow, for example, building, testing, or deploying an application.

Besides its mandatory `steps` directive, and the optional directives that were already introduced, a `stage` can also include:

- `when`: defines conditions under which the stage is executed.
- `input`: allows a stage to display interactive prompts and wait for an answer.

Inside a `steps` directive you include statements such as `echo`, `input`, and `sh` that are provided by Jenkins and its plug-ins.

An optional `post` directive can exist inside either the `pipeline` or `stages` directives. It defines steps that execute after a stage (or all stages) are completed to allow tasks, such as clean up and recovery from an error.



References

CI/CD from the Wikipedia at

<https://en.wikipedia.org/wiki/CI/CD>

Jenkins Glossary at

<https://www.jenkins.io/doc/book/glossary/>

Jenkins Pipeline Syntax at

<https://www.jenkins.io/doc/book/pipeline/syntax/>

► Quiz

Introducing Jenkins Declarative Pipelines Quiz

Choose the correct answers to the following questions:

- 1. **Which two of the following statements about Jenkins, CI/CD, and GitOps are correct? (Choose two.)**
- a. Jenkins is a general-purpose process automation server that is popular among software developers.
 - b. Jenkins is a container-native automation service that orchestrates containers as part of a larger process.
 - c. Jenkins is an automation server created to manage CI processes.
 - d. Jenkins is an automation server created to manage GitOps processes.
- 2. **Choose the correct statement about Jenkins nodes:**
- a. Master nodes can only manage projects and cannot run builds.
 - b. Both master and agent nodes can manage projects and run builds.
 - c. Agent nodes cannot manage projects nor run builds.
 - d. Both master and agent nodes can run builds.
- 3. **Choose the correct statement about managing Jenkins:**
- a. A standard Jenkins deployment requires using a web UI to perform most management tasks.
 - b. The Jenkins REST API provides easy access to all settings of a Jenkins instance and its agent nodes.
 - c. The Jenkins CLI can automatically generate pipelines for common scenarios.
 - d. The `properties` directive of a declarative pipeline offers a limited subset of the project settings available through the Jenkins web UI.
- 4. **Choose the correct statement about Jenkins pipeline projects:**
- a. Scripted pipelines are designed for CI/CD processes and declarative pipelines are designed for GitOps projects.
 - b. Writing scripted pipelines requires basic knowledge of Groovy programming.
 - c. Only declarative pipelines support running builds on containerized agents.
 - d. Only scripted pipelines can make use of most Jenkins plug-ins.

► 5. **Given the structure of Jenkinsfiles and declarative pipelines, which two statements are correct? (Choose two.)**

- a. The `agent` directive allows selecting an agent that is preconfigured with requirements of the pipeline.
- b. The `stage` directive provides shortcuts for performing common development tasks without any coding.
- c. The `steps` directive contains statements that perform the actual tasks of building, testing, and deploying pieces of software.
- d. All directives except `steps` are optional in a declarative pipeline.
- e. All statements inside a `steps` directive come from specialized plug-ins and there is no way of running standard shell commands.
- f. The `post` directive includes statements that are executed only after all stages are successful.

► Solution

Introducing Jenkins Declarative Pipelines Quiz

Choose the correct answers to the following questions:

- 1. **Which two of the following statements about Jenkins, CI/CD, and GitOps are correct? (Choose two.)**
- a. Jenkins is a general-purpose process automation server that is popular among software developers.
 - b. Jenkins is a container-native automation service that orchestrates containers as part of a larger process.
 - c. Jenkins is an automation server created to manage CI processes.
 - d. Jenkins is an automation server created to manage GitOps processes.
- 2. **Choose the correct statement about Jenkins nodes:**
- a. Master nodes can only manage projects and cannot run builds.
 - b. Both master and agent nodes can manage projects and run builds.
 - c. Agent nodes cannot manage projects nor run builds.
 - d. Both master and agent nodes can run builds.
- 3. **Choose the correct statement about managing Jenkins:**
- a. A standard Jenkins deployment requires using a web UI to perform most management tasks.
 - b. The Jenkins REST API provides easy access to all settings of a Jenkins instance and its agent nodes.
 - c. The Jenkins CLI can automatically generate pipelines for common scenarios.
 - d. The properties directive of a declarative pipeline offers a limited subset of the project settings available through the Jenkins web UI.
- 4. **Choose the correct statement about Jenkins pipeline projects:**
- a. Scripted pipelines are designed for CI/CD processes and declarative pipelines are designed for GitOps projects.
 - b. Writing scripted pipelines requires basic knowledge of Groovy programming.
 - c. Only declarative pipelines support running builds on containerized agents.
 - d. Only scripted pipelines can make use of most Jenkins plug-ins.

► 5. **Given the structure of Jenkinsfiles and declarative pipelines, which two statements are correct? (Choose two.)**

- a. The `agent` directive allows selecting an agent that is preconfigured with requirements of the pipeline.
- b. The `stage` directive provides shortcuts for performing common development tasks without any coding.
- c. The `steps` directive contains statements that perform the actual tasks of building, testing, and deploying pieces of software.
- d. All directives except `steps` are optional in a declarative pipeline.
- e. All statements inside a `steps` directive come from specialized plug-ins and there is no way of running standard shell commands.
- f. The `post` directive includes statements that are executed only after all stages are successful.

Deploying Jenkins on OpenShift

Objectives

After completing this section, you should be able to:

- Deploy Jenkins on OpenShift using standard templates and image streams.
- Deploy a CI/CD development pipeline on Jenkins.
- Manage Jenkins projects and builds using the Jenkins web UI.
- Create API tokens for scripts that automate Jenkins tasks.

Introducing Red Hat Jenkins Images and Templates for OpenShift

Red Hat provides several supported Jenkins images as part of Red Hat OpenShift Container Platform, including:

`registry.redhat.io/openshift4/ose-jenkins`

A Jenkins master image that is preconfigured with a number of Jenkins plug-ins that integrate with OpenShift.

`registry.redhat.io/openshift4/ose-jenkins-agent-maven`

A Jenkins agent image that provides Java development tools.

`registry.redhat.io/openshift4/ose-jenkins-agent-nodejs`

A Jenkins agent image that provides Node.js development tools.

`registry.redhat.io/openshift4/ose-jenkins-agent-base`

A base image for building custom agent images. Use this base image to create a new Jenkins agent image if the previous agent images do not provide the tools that you need. All of the Red Hat Jenkins agent images also include the OpenShift CLI because it is required by the OpenShift Pipeline DSL plug-in.

Issues related to these images, such as security vulnerabilities, fall under the Service-Level Agreement (SLA) of the Red Hat OpenShift Container Platform if you deploy these images on OpenShift clusters. Other ways of deploying Jenkins, including using these same images under a local container run time, are supported on a best-effort basis.

OpenShift also provides predefined image streams for the master, Java agent, and Node.js agent in the `openshift` namespace.

The recommended way to deploy Jenkins on OpenShift is using the predefined templates in the `openshift` namespace, among them:

- `jenkins-ephemeral` deploys a Jenkins server using ephemeral storage for quick testing.
- `jenkins-persistent` deploys a Jenkins server using persistent storage for development and production environments.

There are also versions of these templates that deploy Jenkins integrated with OpenShift cluster Alerting and Metering.

Describing Jenkins Plug-ins for OpenShift

The Jenkins master image from Red Hat includes several plug-ins that allow Jenkins to work when integrated with an OpenShift cluster:

Kubernetes plug-in

Creates agents on-demand as pods.

Kubernetes Credentials plug-in

Authenticates to OpenShift using credentials from a service account.

OpenShift Sync plug-in

Creates Jenkins credentials, accounts, and agents from OpenShift image streams, configuration maps, service accounts, and secrets.

OpenShift Jenkins Pipeline DSL

Also known as *OpenShift Client plug-in*, provides imperative DSL statements that deploy applications and manage OpenShift resources.

These plug-ins are Open Source software, so nothing prevents you from adding these plug-ins to an external Jenkins instance, running outside of OpenShift. The Jenkins instance could manage both projects that target OpenShift, and also projects that target other deployment platforms.

However, remember that the Red Hat OpenShift Platform Service-Level Agreement (SLA) only covers scenarios where Jenkins is deployed on OpenShift using Red Hat container images for Jenkins. Support for Jenkins servers deployed outside of OpenShift, and for the OpenShift Jenkins plug-ins deployed outside of Red Hat Jenkins images, is best-effort only.

If you require additional Jenkins plug-ins that do not come preinstalled with Red Hat Jenkins container images, the product documentation describes how to use Source-to-Image (S2I) to build child Jenkins images that are fully supported, excepting issues related to the plug-ins that you add, which are given best-effort support.

Deploying Jenkins on OpenShift Using Standard Templates

All Jenkins templates provided with OpenShift create a deployment configuration to manage an Jenkins master pod. These templates do not deploy any agent pod. As required by Jenkins builds, these are created on demand and the Jenkins master takes the role of a controller for its agent pods.

If you choose a persistent template to deploy Jenkins, then it also creates a persistent volume claim for the Jenkins configuration folder. If an external artifact server, such as a Nexus server or a container image registry, is unavailable, then you can use that same volume to store build artifacts.

If you run builds on the master node, then those builds reuse the volume to store workspace folders. If you run builds on agent pods, then those builds use ephemeral storage for workspace folders so that every build starts from a clean state. If you must save artifacts from builds running on agent pods, then your pipeline must send artifacts for storage on an external server or the master node.

Do not scale the Jenkins deployment configuration to multiple replica pods because Jenkins does not support multiple instances writing to the same configuration folder.

That is, High Availability (HA) for a Jenkins instance on OpenShift comes from Kubernetes ability to restart a pod managed in a deployment configuration, and from the storage provider's ability to retain data in persistent volumes.

The standard templates define parameters and default values for persistent volume size and memory requirements, among others. Small Jenkins instances require no changes to these default parameter values and can be deployed by creating a project and then running:

```
[user@host ~]$ oc new-app --template jenkins-persistent
```

The standard templates also create a service account named `jenkins` for all interactions with the OpenShift cluster. The Jenkins service account is assigned the `edit` role on its OpenShift project, so it is ready to deploy applications to the same project that runs the Jenkins instance.

If your Jenkins instance requires additional rights, then you must add more project or cluster roles to that service account, for example, to allow Jenkins to create resources in other projects.

Deployment Topologies for Jenkins on OpenShift

In theory, a single, large Jenkins instance could manage all projects in all OpenShift clusters for an organization. Some organizations prefer this kind of central CI/CD server that is strictly managed by a corporate IT operations team. This large Jenkins instance could hold credentials to authenticate to multiple OpenShift clusters.

Because such a large Jenkins instance may be hard to manage, many organizations run multiple Jenkins instances in a single cluster or across multiple OpenShift clusters.

It is also easier to manage permissions for these Jenkins instances if they are in different projects.

A common pattern is: each development group inside an organization manages its own Jenkins instance, and that instance does not manage projects from other groups.

Thus, the IT operations team is free from the burden of managing Jenkins instances for development teams. OpenShift role-based access permissions, combined with OpenShift resource limits and resource quotas, are sufficient to ensure that these development teams cannot break their clusters with their pipelines. Also, developers can grant access to Jenkins to projects that they own.

Organizations that use Jenkins to implement GitOps workflows can create one or more dedicated Jenkins instances with either cluster administrator privileges or custom roles that allow limited access to a few cluster operators. Organizations restrict access to these instances to cluster administrators.

Mapping OpenShift Roles to Jenkins User Permissions

Jenkins provides authentication and authorization mechanisms that can be mostly ignored when using images from Red Hat. These images automatically translate standard OpenShift project roles into Jenkins permissions as follows:

admin

Manage the Jenkins instance.

edit

Create and edit projects in the Jenkins instance.

view

View status, logs, and artifacts of builds from the Jenkins instance.

When an OpenShift user logs in to Jenkins, the OpenShift Sync plug-in automatically defines a Jenkins user by concatenating the OpenShift user name and its roles. The same Jenkins plug-in manages translation of OpenShift role bindings to Jenkins permissions.

Creating a Pipeline Project Using the Jenkins Web UI

Older Red Hat OpenShift Container Platform releases supported embedding Jenkins pipelines inside a Build Configuration resource, using the pipeline strategy. That feature, and the integration of these pipeline build configurations with the OpenShift web console, allowed OpenShift users to avoid the Jenkins web UI for basic tasks such as creating a pipeline project and running builds.

Red Hat OpenShift Container Platform 4.3 deprecated the pipeline build strategy because build configurations are being replaced by a new build system and operator, integrated with the OpenShift Pipelines feature. Note that the Jenkins container images, templates, and plug-ins are not deprecated. Only the integration of Jenkins and OpenShift build configurations is deprecated.

Jenkins users are advised to either use the Jenkins web UI to manage pipeline projects and builds or to add plug-ins, such as the Configuration as Code (CasC) plug-in.

To access the Jenkins web interface, use the host name of the single route created by the Jenkins template. Open the host name in a web browser to display the OpenShift log in page. After successfully logging in, the Jenkins Dashboard displays.

The **New Item** link in the Jenkins Dashboard page allows you to create many project types and other Jenkins configuration items, such as folders to group projects and projects of different types.

The two most common project types are:

Pipeline

Runs a pipeline taking as input a single branch from a version control system repository.

Multibranch pipeline

Automatically creates new projects when new branches are detected in a version control system repository. All these projects share the same pipeline definition that must be flexible enough to avoid conflicts between builds in different branches.

Managing Pipeline Builds Using the Jenkins Web UI

The Jenkins web UI allows creating, changing, and deleting projects. It also allows starting, stopping, deleting, and inspecting builds.

A Jenkins instance also provides an alternative Ajax-based web UI called *Blue Ocean* for monitoring pipelines and managing builds. Although Blue Ocean is considered more developer-friendly, it does not have capabilities to manage Jenkins servers. The standard Jenkins web UI is a traditional web application and is required to administer a Jenkins server.

The main elements of the Jenkins web UI are:

- The initial page, which is called the **Jenkins Dashboard**. From there you can navigate to items (projects and folders).
- A bread crumb trail that shows your context within the UI. The **Jenkins** link always returns you to the **Jenkins Dashboard**.
- A side bar menu with links. These links change according to the context within the UI: **Dashboard**, **folder**, **project**, or **build**.

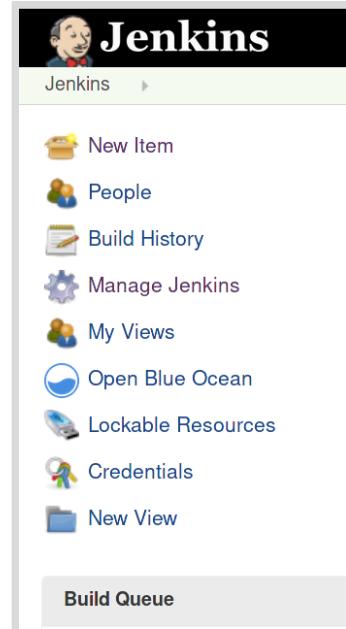


Figure 4.4: Jenkins web UI bread crumb and side bar menu

The body of the Jenkins Dashboard page shows current folders and projects. If you click a folder, then you see its nested projects and folders. If you click a project, then you see a list of builds for that project.

If you click a build of a pipeline project, then you see the **Stage View** of that build; each stage is colored green or red depending on its success (green) or failed (red) status. You can click each stage to see its logs, or click **Console output** to see the complete logs of the entire build.

Multibranch pipeline projects are actually folders with autogenerated pipeline projects for each branch, but project settings are available only from the Multibranch pipeline folder.

Generating Jenkins API Tokens

To use either the Jenkins API or the Jenkins CLI, you require an API token. That token replaces the password in an HTTP BASIC authentication header, for example:

```
[user@host ~] curl --user 'jenkins-user:token' \
  https://jenkins-host/resource-path
```

Jenkins users must keep the token value safe because it never expires. It is not possible to retrieve the token value after it is generated. Jenkins associates each token with a name so that if you suspect that a token is compromised, then you can delete it.

To generate the token, enter the Jenkins user configuration page by clicking the user name close to the **log out** link, and then clicking **Configure**. From this page you can also list and delete tokens previously generated for your Jenkins user.



References

For more information about the Jenkins master container image, refer to the *Configuring Jenkins images* section in the *Using images* chapter in the Red Hat OpenShift Container Platform 4.6 *Images* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#images-other-jenkins

For more information about the Jenkins agent container images, refer to the *Jenkins agent* section in the *Using images* chapter in the Red Hat OpenShift Container Platform 4.6 *Images* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index#images-other-jenkins-agent

Kubernetes plug-in for Jenkins

<https://github.com/jenkinsci/kubernetes-plugin>

Kubernetes Credentials plug-in for Jenkins

<https://github.com/jenkinsci/kubernetes-credentials-plugin>

OpenShift Sync plug-in for Jenkins

<https://github.com/jenkinsci/openshift-sync-plugin>

OpenShift Jenkins Pipeline (DSL) plug-in

<https://github.com/jenkinsci/openshift-client-plugin>

► Guided Exercise

Deploying Jenkins on OpenShift

In this exercise you will deploy and validate a simple Continuous Integration/Continuous Deployment (CI/CD) pipeline for a sample application.

Outcomes

You should be able to:

- Deploy a Jenkins master on OpenShift.
- Create a CI/CD pipeline project using a Jenkins file in GitHub from the Jenkins web UI.
- Start and monitor a pipeline build from the Jenkins web UI and the OpenShift CLI.
- Generate a Jenkins API token.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The Jenkins master and Node.js agent container images from Red Hat.
- A personal account at GitHub.
- The source code for the "hello, world" sample application.
- The `Jenkinsfile` to build and deploy the sample application.

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

This command ensures that the cluster API is reachable and downloads sources of the sample application.

```
[student@workstation ~]$ lab gitops-deploy start
```

Instructions

► 1. Create a GitHub project to host the sample application and also the `Jenkinsfile`.

- 1.1. Open a web browser and access <https://github.com>.

If you do not have a GitHub account, then click **Sign Up** and follow the instructions to create a personal account.

If you do have a GitHub account, then click **Sign In** and follow the instructions to log in using your personal account.

**Important**

Beginning August 13, 2021, GitHub no longer accept account passwords when authenticating operations from the command line.

You need to create a Personal Access Token (PAT) with *public_repo* permission granted as described at <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>.

- 1.2. Create a new empty repository named `gitops-deploy`.

On the **Repositories** page, click **New** to enter the **Create a new repository** page. Type `gitops-deploy` in the **Repository name** field. Leave all other fields at their default values and click **Create repository**.

On the **Quick setup** page, click the clipboard icon to copy the HTTPS URL of your repository.

Do not close your web browser, you will come back to GitHub a few times during this exercise.

- 1.3. Open a terminal on the **workstation** machine, and then clone the repository into your home folder.

```
[student@workstation ~]$ git clone \
  https://github.com/youraccount/gitops-deploy.git
Cloning into 'gitops-deploy'...
warning: You appear to have cloned an empty repository.
```

- 1.4. Copy the sample application sources and the **Jenkinsfile** from the `~/D0380/labs/gitops-deploy/hello` folder into your local clone of the git repository.

```
[student@workstation ~]$ cp ~/D0380/labs/gitops-deploy/hello/* ~/gitops-deploy
[student@workstation ~]$ cd ~/gitops-deploy
[student@workstation gitops-deploy]$ ls
app.js  Jenkinsfile  package.json
```

- 2. Review the **Jenkinsfile** for a sample CI/CD pipeline.

- 2.1. Inspect the **Jenkinsfile** and review its stages.

You do not need to understand all steps inside the **Jenkinsfile**. These stages work as-is and are ready to build and deploy the sample application.

The pipeline uses the OpenShift Client plug-in DSL to create a new OpenShift project that generates its name from the branch name and build number. Then, the pipeline builds and deploys the sample application from its source code, emulating the steps a developer would perform using the `oc new-app` command.

Observe that there are multiple echo statements inside each step of each stage. These statements can help you to relate each step to console output from the Jenkins Master.

```
[student@workstation gitops-deploy]$ less Jenkinsfile
pipeline {
    environment {
        ...output omitted...
```

```

stages {
    stage('create') {
        ...output omitted...
    }
    stage('build') {
        ...output omitted...
    }
    stage('deploy') {
        ...output omitted...
    }
    stage('test') {
        ...output omitted...
    }
}
post {
    ...output omitted...
}
}

```

2.2. Review the test stage.

It waits for an interactive prompt before proceeding, giving you time to inspect OpenShift resources in the middle of a Jenkins build, and then tests the application using the `curl` command.

```

[student@workstation gitops-deploy]$ less Jenkinsfile
...output omitted...
stage('test') {
    input {
        message 'About to test the application'
        ok 'Ok'
    }
    steps {
        echo "Check that '${env.APP}.${env.DOMAIN}' returns HTTP 200"
        sh "curl -s --fail ${env.APP}.${env.DOMAIN}"
    }
}
...output omitted...

```

2.3. Commit the sources and the Jenkinsfile, and then push to GitHub.

```

[student@workstation gitops-deploy]$ git add *
[student@workstation gitops-deploy]$ git commit -m 'sample app and pipeline'
...output omitted...
create mode 100644 Jenkinsfile
...output omitted...
[student@workstation gitops-deploy]$ git branch -M main
[student@workstation gitops-deploy]$ git push -u origin main
...output omitted...
* [new branch]      main -> main

```

2.4. Switch to your web browser and refresh the GitHub page that shows your repository. Verify that you see your Jenkinsfile and sample application files there.

- 3. As a developer, create a project in OpenShift and deploy Jenkins using the persistent template.

3.1. Switch to your terminal and log in on OpenShift as the developer user.

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

3.2. Create the `gitops-deploy` project.

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

3.3. List all standard Jenkins templates provided with OpenShift.

```
[student@workstation gitops-deploy]$ oc get template -n openshift | grep jenkins
jenkins-ephemeral           Jenkins service, without persistent storage....
jenkins-ephemeral-monitored Jenkins service, without persistent storage....
jenkins-persistent           Jenkins service, with persistent storage....
jenkins-persistent-monitored Jenkins service, with persistent storage....
```

3.4. List the parameters of the `jenkins-persistent` standard template. Note that it provides parameters that allow you to customize such things as storage size and memory requests for larger environments. Note also that all parameters have default values.

```
[student@workstation gitops-deploy]$ oc process jenkins-persistent --parameters \
-n openshift
NAME                  DESCRIPTION
VALUE
...output omitted...
MEMORY_LIMIT          Maximum amount of memory the container can use. ...
1Gi
VOLUME_CAPACITY       Volume space available for data, e.g. 512Mi, 2Gi. ...
1Gi
...output omitted...
JENKINS_IMAGE_STREAM_TAG Name of the ImageStreamTag to be used for the Jenki...
jenkins:2
...output omitted...
```

3.5. Deploy Jenkins using the `jenkins-persistent` template. Set the `JENKINS_IMAGE_STREAM_TAG` parameter to use the updated image stream `jenkins:v4.8` and accept the default values for all of the other template parameters.

```
[student@workstation gitops-deploy]$ oc new-app --template jenkins-persistent \
-p JENKINS_IMAGE_STREAM_TAG=jenkins:v4.8
...output omitted...
--> Creating resources ...
route.route.openshift.io "jenkins" created
persistentvolumeclaim "jenkins" created
deploymentconfig.apps.openshift.io "jenkins" created
serviceaccount "jenkins" created
```

```
rolebinding.authorization.openshift.io "jenkins_edit" created
service "jenkins-jnlp" created
service "jenkins" created
--> Success
...output omitted...
```

**Note**

The updated image stream is provided by the classroom environment. This image includes a fix for <https://issues.jenkins.io/projects/JENKINS/issues/JENKINS-64685>.

- 4. As a cluster administrator, grant to the Jenkins service account permission to create OpenShift projects.

- 4.1. Log in to OpenShift as the `admin` user.

```
[student@workstation gitops-deploy]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 4.2. Grant the Jenkins service account permission to create new projects.

```
[student@workstation gitops-deploy]$ oc adm policy add-cluster-role-to-user \
  self-provisioner -z jenkins -n gitops-deploy
clusterrole.rbac.authorization.k8s.io/self-provisioner added: "jenkins"
```

**Note**

The `oc adm policy` command creates a cluster role binding resource, which is not namespaced, and references a service account, which is namespaced. Thus, the `-n` option to specify the project of the service account is required.

- 5. As a developer, use the Jenkins web UI to create a Jenkins project that takes its `Jenkinsfile` from a Git repository.

- 5.1. Log in on OpenShift as the `developer` user and enter the `gitops-deploy` project in OpenShift.

```
[student@workstation gitops-deploy]$ oc login -u developer -p developer
Login successful.
...output omitted...
[student@workstation gitops-deploy]$ oc project gitops-deploy
Already on project "gitops-deploy" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 5.2. Wait until the Jenkins master pod is ready and running. It takes time for the pod to make its single container ready because Jenkins needs time to initialize.

To confirm that Jenkins is fully initialized, look for the message "Jenkins is fully up and running" in the container logs.

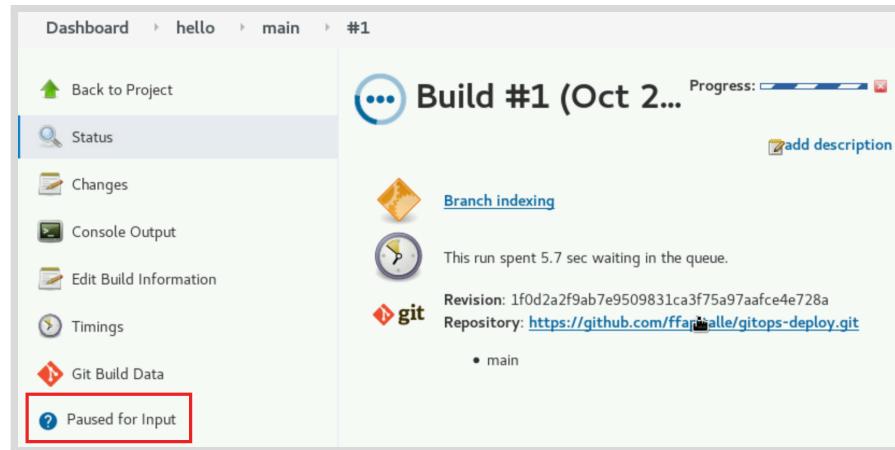
```
[student@workstation gitops-deploy]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
jenkins-1-deploy  0/1     Completed   0          2m57s
jenkins-1-xvwwr  1/1     Running    0          2m54s

[student@workstation gitops-deploy]$ oc logs jenkins-1-xvwwr | \
grep 'up and running'
2021-10-27 18:09:13 INFO      hudson.WebAppMain$3 run Jenkins is fully up and
running
```

- 5.3. Find the host name of the Jenkins master. Do not close your terminal, you will come back to it a few times during this exercise.

```
[student@workstation gitops-deploy]$ oc get routes
NAME      HOST/PORT   ...
jenkins  jenkins-gitops-deploy.apps.ocp4.example.com ...
```

- 5.4. Open a new tab in the web browser on the `workstation` machine, and then enter the Jenkins web UI using the URL obtained in the previous step.
If prompted, accept the TLS certificate for the Jenkins web UI, and then click **Log in with OpenShift**.
Also accept the TLS certificate of the OpenShift OAuth server, if prompted, and then click **htpasswd_provider** to enter the OpenShift log in credentials page.
Log in as **developer** with the **developer** password, and then click **Allow selected permissions**. OpenShift redirects your browser to the Jenkins **Dashboard** page.
- 5.5. Create a new multibranch pipeline project for the sample application.
Click **New Item** on the Jenkins Dashboard page, and then type `hello` as the item name. Select **Multibranch Pipeline** and click **Ok**.
Under the **Branch Sources** heading, select **Add Source → Git** and type the HTTPS URL of your GitHub repository, `https://github.com/youraccount/gitops-deploy.git`, in the **Project Repository** field. Because your Git repository is public you do not provide any credentials.
Leave all other fields unchanged and click **Save**.
- 5.6. Verify that Jenkins finds the **main** branch.
Jenkins automatically starts a **Scan Multibranch Pipeline** job and displays its logs. Wait until it finishes and verifies that the job logs indicate that Jenkins found a **main** branch. If a **main** branch is not found, click **Configure** and fix your Jenkins project settings, and then click **Scan Multibranch Pipeline Now**.
- 5.7. Verify that there is a build running for the main branch.
Click either **Up** or **Jenkins** to go to the Jenkins **Dashboard** page. Click `hello` to enter the new project and click `main` to enter the project page.
If you do not see any build, then click **Build Now** to start a build. If you see a build that is in progress, click on the build number to enter the build page.
After the build reaches the test stage, it is paused while waiting for user interaction. Leave the build paused at the test stage for now.



Do not close your web browser. You will come back to the Jenkins web UI a few times during this exercise.



Note

If you see errors during the build, then hover the mouse over each stage and click **Logs** to find clues about what caused the errors and review your previous steps.

To change your Jenkins project settings, click **hello** and then click **Configure**.

To change your Git repository, commit and push your changes and then click **Build Now** to start a new build.

- 6. As a developer, monitor a build from the Jenkins web UI and also from the OpenShift CLI.

- 6.1. Click **Jenkins** to enter the Jenkins Dashboard page. Notice that there is a representation of an agent running a build under the heading **Build executor status**.

From the information displayed, you can identify the name of the agent, the name of the project, the name of the branch, and the number of the build.



- 6.2. Switch to your terminal and verify that there is a pod with the same name as the agent you saw in the previous step.

```
[student@workstation gitops-gitops]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
jenkins-1-deploy  0/1     Completed   0          18m
jenkins-1-xvwwr   1/1     Running    0          18m
nodejs-56b7s      1/1     Running    0          14m
```

- 6.3. Verify that there is no controller resource for the Jenkins agent pod. It runs as an unmanaged, stand-alone pod.

```
[student@workstation gitops-gitops]$ oc status
In project gitops-deploy on server https://api.ocp4.example.com:6443

svc/jenkins-jnlp - 172.30.249.140:50000
https://jenkins-gitops-deploy.apps.ocp4.example.com (redirects) (svc/jenkins)
dc/jenkins deploys openshift/jenkins:2
deployment #1 deployed 12 minutes ago - 1 pod

pod/nodejs-56b7s runs .../jenkins-agent-nodejs:latest
...output omitted...
```

- 6.4. Verify that there is a project named `hello-main-1` where the sample application was built and deployed:



Note

Jenkins creates a new OpenShift project, named *job-branch-build*, for each pipeline execution.

```
[student@workstation gitops-gitops]$ oc status -n hello-main-1
In project hello-main-1 on server https://api.ocp4.example.com:6443

http://hello-main-1.apps.ocp4.example.com to pod port 8080-tcp (svc/nodeapp)
deployment/nodeapp deploys istag/nodeapp:latest <-
bc/nodeapp source builds https://github.com/youraccount/gitops-deploy.git#main
on openshift/nodejs:12-ubi8
deployment #2 running for 2 minutes - 1 pod
deployment #1 deployed 3 minutes ago

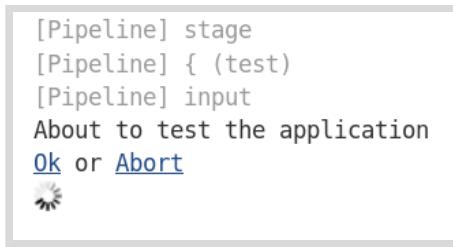
...output omitted...
```

- 6.5. Switch to your web browser and inspect the build logs in the Jenkins web UI.

From the Jenkins Dashboard page, click **hello** and then click **main** to return to your project page.

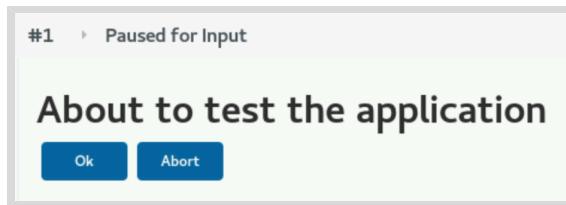
Click the number of the build, #1, to enter the **Build** page, and then click **Console Output** to view the build logs. Search for messages from the `echo` steps in your pipeline.

At the end of the page, just before a spinning disk, there is the message from the `input` directive of the `test` stage. Do not click any of the **Ok** or **Abort** links.



- 6.6. Allow the build to continue.

Click **Paused for Input** to enter to the input page, and then click **OK**.



The build finishes successfully after a few moments.

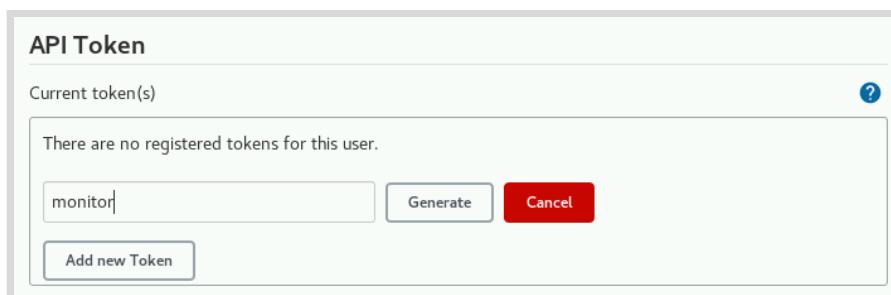
- 6.7. Switch to your terminal and verify that both the agent pod and the `hello-main-1` project are no longer displayed. Note that the project might be available for a few moments before OpenShift deletes it.

```
[student@workstation gitops-gitops]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
jenkins-1-deploy  0/1     Completed   0          26m
jenkins-1-xvwrr  1/1     Running    0          26m

[student@workstation gitops-gitops]$ oc get projects
NAME          DISPLAY NAME   STATUS
gitops-deploy                         Active
```

► 7. Monitor your Jenkins build using the Jenkins API.

- 7.1. Switch to your web browser in the Jenkins web UI and click **developer**, near the **log out** option.
- 7.2. In the **developer** Jenkins user page, click **Configure**, go to the **API Token** section and click **Add new Token**. Type a name for the token, such as `monitor`, and then click **Generate**.



- 7.3. Select and copy to the clipboard the long hexadecimal string, which is the value of the token.

The screenshot shows the Jenkins API Token configuration page. It displays a single token entry for 'monitor' with the value '117ea49d3a27e41307af7eb578ab74d840'. Below the token value is a yellow warning message: '⚠ Copy this token now, because it cannot be recovered in the future.' There are also 'Add new Token' and 'Delete' buttons.

- 7.4. Switch back to your terminal and create the `/home/student/developer-token.txt` file using a text editor. Paste the value of the token you got from the previous step.
- 7.5. Change to your home folder and inspect the `monitor-build.sh` script in the `/home/student/D0380/labs/gitops-deploy/sh` folder.
The script uses the Jenkins API token from the previous step to invoke the Jenkins API and get the result of the latest build. Do not make any change to the script.

```
[student@workstation gitops-deploy]$ cd ~
[student@workstation ~]$ cat ~/D0380/labs/gitops-deploy/sh/monitor-build.sh
#!/bin/bash

token=$( cat ~/developer-token.txt )
host=$( oc get route jenkins -o jsonpath='{.spec.host}' )
job="https://${host}/job/hello/job/main"

if json=$( curl --fail -sk "${job}/lastBuild/api/json" --user "developer-admin-edit-view:${token}" )
then
...output omitted...
```

- 7.6. Run the `monitor-build.sh` script to verify that your Jenkins API token is accepted by your Jenkins instance.

```
[student@workstation ~]$ ~/D0380/labs/gitops-deploy/sh/monitor-build.sh
Result of the last build (#1) from hello/master is: SUCCESS.
```

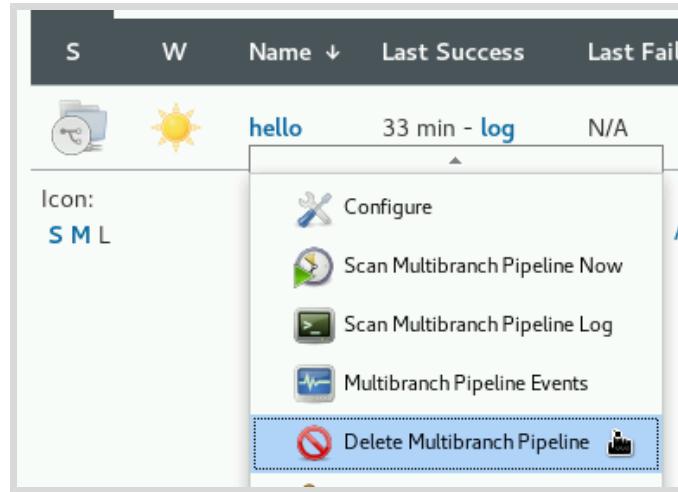
- 8. As the developer user, delete the Jenkins project and your GitHub repository.



Note

Do not delete the OpenShift project. You will reuse this project and the Jenkins server you just deployed in a subsequent exercise.

- 8.1. As the `developer` user, delete the Jenkins project.
Switch to the Jenkins web UI and click **Jenkins** to return to the Jenkins Dashboard page. Click the down arrow near **hello**, and then click **Delete Multibranch Pipeline**.



Click **Yes** to confirm the operation. It might be necessary to scroll down the page to find the **Yes** button.

- 8.2. Switch to your web browser and from the GitHub repository page, click **Settings**, and then scroll down until you find the **Danger Zone** header. Click **Delete this repository** and follow the instructions to confirm deleting the repository.
- 8.3. Remove your local clone of the sample application repository.

```
[student@workstation ~]$ rm -rf gitops-deploy
```

- 8.4. Remove your Jenkins API token.

```
[student@workstation ~]$ rm developer-token.txt
```

Finish

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

```
[student@workstation ~]$ lab gitops-deploy finish
```

Configuring OpenShift Resources Using a Declarative GitOps Workflow

Objectives

After completing this section, you should be able to:

- Describe GitOps concepts
- Configure cluster operators using resource files in a declarative fashion
- Compare live OpenShift resources with resource files
- Delete Resources with a Declarative Approach

Reviewing GitOps Concepts

GitOps processes follow infrastructure-as-code (IaC) practices: all configurations should come from files that are versioned, tested, and reviewed, such as source code in a Continuous Integration and Continuous Deployment (CI/CD) process.

Though early descriptions of GitOps workflows were based in GitHub proprietary pull requests, GitOps does not require a Git server nor a GitHub account. Any version control system and branch-based workflow could fit a GitOps process.

The main challenge of implementing GitOps, compared to CI/CD, is that "configurations" do not produce stand alone artifacts, such as binary executables, that you can store, copy, compare, and replace. Configurations change the state of a live system.

Viable GitOps processes usually require declarative configuration files, similar to Ansible Playbooks: imperative configurations (using the command module) often cause maintenance problems, whereas declarative configurations (using idempotent modules) are more reliable and also easier to apply and maintain.

The realization of GitOps processes with OpenShift is made easier if configuration files for cluster operators, custom resources, and standard resources, such as configuration maps and cron jobs, are stored as either YAML or JSON files in a version control system. This may be challenging for system administrators used to imperative OpenShift CLI commands such as `oc scale`.

Comparing Imperative and Declarative Resource Management with OpenShift

The OpenShift CLI provides a rich set of imperative commands to manage application resources, such as the `oc new-app` and `oc set` commands. Using imperative commands provides the following benefits:

- Easy to learn and use, lowering the barrier to entry for developers and system administrators.
- Requires less typing and provides clearer error messages.
- Does not require a deep knowledge of OpenShift resource types and their attributes.

Imperative `oc` commands work well for a CI/CD process but unfortunately not for a GitOps process.

The OpenShift CLI also provides a small set of declarative commands to manage application resources from either YAML or JSON files, such as the `oc apply` command. Using declarative commands provides the following advantages:

- A single syntax for creating, deleting, and updating multiple resource types because all resource-specific details are inside resource files.
- Ease of reusing the same resource files for multiple automation tools, such as Ansible Playbooks and GitOps tools.
- Ease of comparing multiple releases of source resource files and interpreting version control log outputs, compared to scripts using imperative commands.

To use declarative commands, you must write resource definition files using YAML or JSON syntaxes. To create these YAML files, a deeper knowledge of OpenShift resource types is required.

Using declarative commands also has some disadvantages; often resource files look correct but do not produce the expected results. Sometimes these are called "silent errors." Examples of silent errors include:

- Kubernetes silently accepts resource files that define unknown attributes, for example, because of a typing mistake in the attribute name.
- In YAML files, it is easy to type an incorrect number of indentation spaces and add an attribute to an incorrect nested structure.

These disadvantages encourage beginner OpenShift administrators and developers to prefer imperative commands. Fortunately, the benefits of automation surpass the disadvantages of declarative commands.

Some OpenShift CLI commands, such as `oc apply`, mitigate these disadvantages by supporting the `--validate` option. This option validates resource definitions against the same resource type metadata that provides attribute descriptions to the `oc explain` command.

Creating Resource Files from Scratch

To create a resource file from scratch, you have to find samples from Red Hat OpenShift Container Platform product documentation, upstream Kubernetes documentation, or from operators that define new custom resources types.

OpenShift provides online help for all predefined resource types, including OpenShift API extensions and custom resources from operators, using the following commands:

- The `oc api-resources` command lists all resource types currently enabled in the cluster, including custom resources.
- The `oc api-versions` command shows available API versions, including APIs defined by operators and Kubernetes extensions. There can be multiple API versions of the same resource type for backwards compatibility with previous versions of OpenShift and Kubernetes.
- The `oc explain` command lists attributes of a resource type and their descriptions, including custom resources defined by operators that comply with the Operator Life-cycle Manager (OLM). Use dot notation to look at structured resources, for example: `oc explain route.spec.host`

Red Hat generally recommends that you use these commands and documentation sources to write your resource files from scratch.

It is a good idea to create and configure a resource using imperative commands, and then export the resource to a file to use as a starting point. Unfortunately, this approach might be difficult because the exported file contains:

- Attributes with values that are generated at run time, such as `createTimeStamp` and `status`.
- Attributes with default values that are unnecessary and sometimes change between Kubernetes and OpenShift releases.
- Labels and annotations added at run time by resource controllers, admission plug-ins, and operators.

It is a best practice remove all of the above to create a minimum resource file that defines only the set of attributes that you require because the longer the resource file, the harder to write, review, and maintain it.

Another reason to write resource files from scratch, instead of exporting live resources, is that resource attributes are exported in alphabetical order. You might prefer a more logical ordering of attributes, for example, listing `metadata.name` at the beginning of your resource file, near `kind`.

OpenShift standard templates are a useful starting point to find sample resource definitions, but some of them might need cleaning and reordering of attributes for easier reading.

Finding Changes in OpenShift Resources

Live OpenShift resources have a number of attributes, labels, and annotations that only matter at run time, complicating efforts to compare resource files on disk or under version control with live resources in a cluster. Exporting an OpenShift resource and comparing it with a text file using standard `diff` tools requires significant pre or postprocessing to remove all the noise.

The complexity of pre and postprocessing varies with each resource type, and operators can add new custom resource types that your pre and postprocessing scripts do not know about.

Fortunately, OpenShift provides a safe and reliable way to compare of resource files with live resources using the `oc diff` command:

- The `oc diff` command sends resource definitions to the OpenShift API and lets all running controllers make changes, but does not apply them, similar to using the `--dry-run` option.
- Then, OpenShift API returns the change set that would have been applied by `oc apply` if it were not a `--dry-run`.

In this way, run time attributes, default values, and ordering of attributes are ignored because they would not create actual changes with `oc apply`.

Another benefit of using the `oc diff` command is that it ignores YAML file comments.

Preprocessing Resource Files with Kustomize

More sophisticated GitOps processes usually require some form of text preprocessing of resource files to handle requirements such as:

- Attribute values that change between development and production environments.
- Inserting text files inside configuration maps.
- Encoding binary files inside secrets.

The OpenShift CLI embeds the *Kustomize* utility to perform these and other preprocessing tasks. Many OpenShift commands, such as `oc apply` and `oc diff`, accept the `-k`

`kustomize_folder` option to preprocess resource files before submitting them to the OpenShift API.

Kustomize requires that the `kustomize_folder` folder contains a `kustomization.yaml` file, as in the following example that generates two resources, a deployment and a TLS secret:

```
resources:
- deployment.yaml ①
secretGenerator:
- name: mycert ②
  files:
  - tls.crt=priv-cert.crt ③
  - tls.key=priv-cert.key
  type: kubernetes.io/tls ④
generatorOptions:
  disableNameSuffixHash: true ⑤
```

- ① Kustomize takes the `deployment.yaml` file as is. It must exist inside the Kustomize folder.
- ② Kustomize generates one secret named `mycert` in the current project.
- ③ Kustomize encodes `priv-cert.crt` and `priv-cert.keys` file contents to Base64 and places the results and their respective keys inside the generated secret.
- ④ Kustomize copies any other attributes verbatim to the generated secret.
- ⑤ Kustomize generators accept a few options. This one disables adding a suffix to the names of the generated secrets.

To test that your Kustomize configuration file contains no syntax errors, use the `oc kustomize kustomize_folder` command. This command outputs the results of Kustomize preprocessing without making any changes to files inside `kustomize_folder` folder nor to live resources in your cluster.

Deleting Resources with a Declarative Approach

The `oc apply` command can create and change OpenShift resources but it cannot delete resources. Kustomize has this same limitation.

When using a declarative approach, you might create a resource that you will later want to remove. Removing the resource from your resource files does not delete the resource, so you must run imperative `oc delete` commands to delete them.

To delete a resource that is preexisting, for example, a resource that was created by installing OpenShift or an operator, you must use an imperative `oc delete` command.

To delete resources in a manner that is consistent with a declarative approach, do not list the names of the resources to delete as arguments to the `oc delete` command. Instead, provide a folder with stub resource files that includes only the `apiVersion`, `kind`, `metadata.name`, and optionally the `metadata.namespace` attributes, and then use the `oc delete -f folder` command.

Thus, a complete declarative approach to manage OpenShift resources requires two folders:

- One folder, containing resource files for the `oc apply` command, defines the expected state of a set of resources.

- Another folder, containing resource files for the `oc delete` command, defines list of resources that should not exist.

If you pass complete resource files to the `oc delete` command, it ignores any fields not required to identify the resources to delete. This behavior is useful when you want to delete all resources previously created from either a set of resource files or the output of Kustomize.



References

GitOps: A Path to More Self-service IT at
<https://queue.acm.org/detail.cfm?id=3237207/>

Declarative Management of Kubernetes Objects Using Configuration Files at
<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/>

Declarative Management of Kubernetes Objects Using Kustomize at
<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/>

How to Delete Objects at
<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/#how-to-delete-objects>

► Guided Exercise

Configuring OpenShift Resources Using a Declarative GitOps Workflow

In this exercise you will configure HTPasswd authentication from resource files in a declarative way.

Outcomes

You should be able to:

- Configure an HTPasswd Identity Provider (IdP) from YAML files.
- Configure an HTPasswd secret using Kustomize files.
- Compare HTPasswd resources from an OpenShift cluster with local YAML files ignoring status and default attributes.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- YAML files with incomplete HTPasswd Identity Provider (IdP) resources.

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

This command ensures that the cluster API is reachable and downloads all sample files required by this exercise.

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

If you are locked out of the cluster in the middle of this exercise because you made mistakes changing the authentication settings, then running the `lab gitops-resources start` command again restores the initial authentication settings of your classroom.

Instructions

- 1. Complete the Kustomize folder for configuring an HTPasswd IdP.
- 1.1. Enter the `~/D0380/labs/gitops-resources` folder and verify that it contains two folders.
Make all your changes to the `config` folder. The `orig` folder provides a pristine copy of the initial classroom settings for your reference.

```
[student@workstation ~]$ cd ~/D0380/labs/gitops-resources  
[student@workstation gitops-resources]$ ls  
config orig
```

- 1.2. Inspect the resource files inside the ~/D0380/labs/gitops-resources/config folder. The following files are ready to use and require no changes:
 - **oauth.yaml**: configures the OpenShift Authentication Operator with an HTPasswd IdP.
 - **htpasswd-secret-data**: defines HTPasswd users and passwords. Besides the initial admin and developer users, this file adds the kustom user with password redhat123.
- 1.3. Create a Kustomize configuration file named **kustomization.yaml** inside the ~/D0380/labs/gitops-resources/config folder.

**Note**

If you are unsure about the indentation spaces for each line when you make edits to YAML files, then refer to the solution files in the ~/D0380/labs/gitops-resources/orig folder.

Add the **oauth.yaml** file in the list of resources. Add a generator for an HTPasswd secret named **htpasswd-secret**, as referenced by the **oauth.yaml** file. Add the **namespace** attribute so that the secret lands in the **openshift-config** namespace as required by the HTPasswd IdP. Set the **disableNameSuffixHash** generator option to **true**.

```
resources:  
- oauth.yaml  
secretGenerator:  
- name: htpasswd-secret  
  namespace: openshift-config  
  files:  
  - htpasswd=htpasswd-secret-data  
generatorOptions:  
  disableNameSuffixHash: true
```

- 1.4. Validate that your Kustomize configuration inside the **config** folder generates an OAuth resource and also a secret resource.

```
[student@workstation gitops-resources]$ oc kustomize config  
apiVersion: v1  
...output omitted...  
kind: Secret  
metadata:  
  name: htpasswd-secret  
  namespace: openshift-config  
type: Opaque  
---  
apiVersion: config.openshift.io/v1  
kind: OAuth  
...output omitted...  
spec:  
  identityProviders:  
    - htpasswd:
```

```
fileData:
  name: htpasswd-secret
...output omitted...
```

- 2. As a cluster administrator, apply your configuration files to the cluster and validate that you can log in as the `kustom` user.

- 2.1. Log in on OpenShift as the `admin` user.

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

- 2.2. Apply all resources from the `~/D0380/labs/gitops-resources/config` folder using Kustomize. If you get warnings similar to the ones in the following sample outputs, then you can safely ignore them.

```
[student@workstation gitops-resources]$ oc apply -k config
Warning: oc apply should be used on resource created by either oc create --save-config or oc apply
secret/htpasswd-secret configured
Warning: oc apply should be used on resource created by either oc create --save-config or oc apply
oauth.config.openshift.io/cluster configured
```

- 2.3. Wait until a new set of OAuth pods are ready and running. Also wait until all terminating pods from the previous deployment are gone.

```
[student@workstation gitops-resources]$ oc get pods -n openshift-authentication
NAME                      READY   STATUS    RESTARTS   AGE
oauth-openshift-55fb849579-9mz99   1/1     Running   0          4m54s
oauth-openshift-55fb849579-r2mhn   1/1     Running   0          4m46s
```

- 2.4. Log in to OpenShift as the `kustom` user to confirm that your new authentication settings are applied.

```
[student@workstation gitops-resources]$ oc login -u kustom -p redhat123
Login successful.
...output omitted...
```

- 3. As a cluster administrator, delete the `kustom` user without changing your Kustomize folder.

- 3.1. Log in to OpenShift as the `admin` user.

```
[student@workstation gitops-resources]$ oc login -u admin -p redhat
Login successful.
...output omitted...
```

- 3.2. Extract the HTPasswd secret data to a temporary file.

```
[student@workstation gitops-resources]$ oc extract secret/htpasswd-secret \
-n openshift-config --confirm --to /tmp/
/tmp/htpasswd
```

- 3.3. Remove the kustom user from the temporary file.

```
[student@workstation gitops-resources]$ htpasswd -D /tmp/htpasswd kustom
Deleting password for user kustom
```

- 3.4. Replace the HTPasswd secret data with the contents of the temporary file.

```
[student@workstation gitops-resources]$ oc set data secret/htpasswd-secret \
-n openshift-config --from-file htpasswd=/tmp/htpasswd
secret/htpasswd-secret data updated
```

- 3.5. Wait until a new set of OAuth pods are ready and running. Also wait until all terminating pods from the previous deployment are gone.

```
[student@workstation gitops-resources]$ oc get pods -n openshift-authentication
NAME                  READY   STATUS    RESTARTS   AGE
oauth-openshift-856784b8c8-2566b   1/1     Running   0          82s
oauth-openshift-856784b8c8-6bzsb   1/1     Running   0          90s
```

- 3.6. Log in to OpenShift as kustom to confirm that the HTPasswd secret was updated.

```
[student@workstation gitops-resources]$ oc login -u kustom -p redhat123
Login failed (401 Unauthorized)
Verify that you have provided correct credentials.
```

- ▶ 4. Compare HTPasswd IdP and secret resources with YAML files on disk using the diff and the oc diff commands.

- 4.1. Export the OAuth configuration resource and the HTPasswd secret to temporary files.

```
[student@workstation gitops-resources]$ oc extract secret/htpasswd-secret \
-n openshift-config --confirm --to /tmp/
/tmp/htpasswd
```

```
[student@workstation gitops-resources]$ oc get oauth cluster \
-o yaml > /tmp/oauth-config.yaml
```

- 4.2. Compare the temporary files with the files on your ~/D0380/labs/gitops-resources/config folder using the diff command.

Verify that in addition to changes to the HTPasswd data, it also reports a number of other differences, even though you made no changes to the OAuth configuration.

```
[student@workstation gitops-resources]$ diff /tmp/htpasswd \
  config/htpasswd-secret-data
2a3
...output omitted...
[student@workstation gitops-resources]$ diff /tmp/oauth-config.yaml \
  config/oauth.yaml
5,6d4
...output omitted...
```

- 4.3. Compare the live OAuth configuration resource and the HTPasswd secret with the files on your Kustomize folder using the `oc diff` command.

Verify that only differences related to the HTPasswd secret data are displayed.

```
[student@workstation gitops-resources]$ oc diff -k config
...output omitted...
@@ -1,6 +1,6 @@
apiVersion: v1
data:
- htpasswd: YWRt..Lwo=
+ htpasswd: YWRt..MAo=
kind: Secret
metadata:
  annotations:
...output omitted...
```

- ▶ 5. Restore the default authorization settings for your classroom using the files from the `~/DO380/labs/gitops-resources/orig` folder.

```
[student@workstation gitops-resources]$ oc apply -k orig
secret/htpasswd-secret configured
oauth.config.openshift.io/cluster unchanged
```

Change to your home folder.

```
[student@workstation gitops-resources]$ cd ~
```

Finish

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

```
[student@workstation ~]$ lab gitops-resources finish
```

Configuring OpenShift using GitOps and Jenkins

Objectives

After completing this section, you should be able to:

- Automate the configuration of cluster operators from resources in Git using Jenkins Pipelines.
- Describe pipeline settings using `Jenkinsfile` directives instead of the Jenkins web UI.

Designing Jenkins Pipelines for GitOps

The following figure illustrates a sample GitOps workflow without detailing tools and other implementation details:

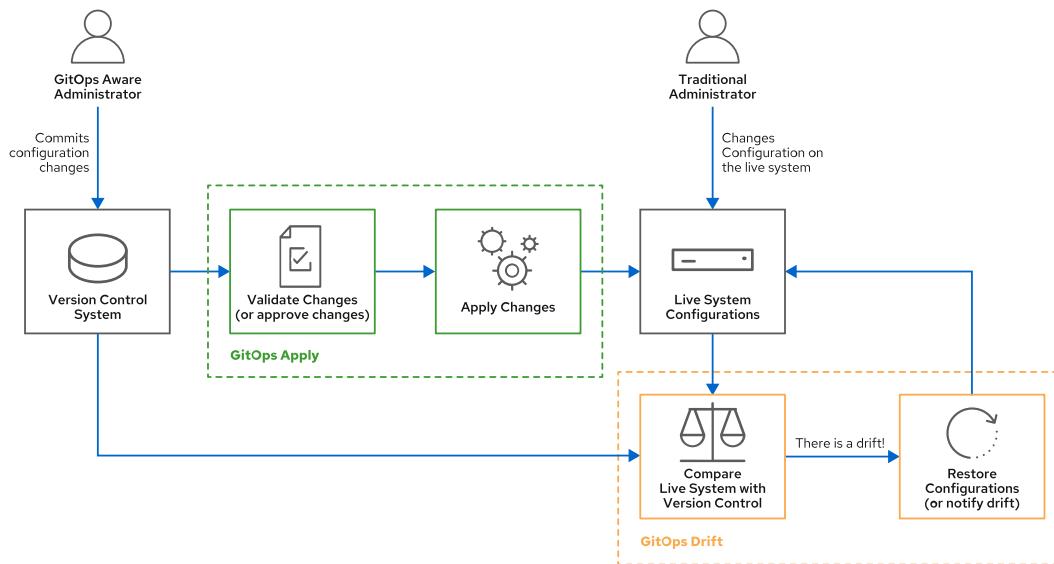


Figure 4.12: Sample GitOps Workflow

One possible design approach translates this workflow into Jenkins using two pipelines:

- An *Apply* pipeline that applies configuration changes from a version control system (VCS) to a live cluster.
- A *Drift* detection pipeline that detects when live cluster configurations changed outside of the expected GitOps process.

These pipelines primarily use the `oc apply` and `oc diff` commands, introduced elsewhere.

The following sections present a variation of this design and walk through a sample implementation.

First, you learn about some common pipeline settings and how to use them. Then, you explore using these settings to implement Apply and Drift Pipelines to realize a GitOps workflow.

A real-world implementation would be more sophisticated.

Configuring a Pipeline from a Jenkinsfile

Traditional Jenkins usage is very GUI-oriented. To follow configuration-as-code principles and be consistent with GitOps, avoid setting pipeline and multibranch pipeline project types from the Jenkins web UI.

Fortunately, most settings from the web UI are also available from the `triggers`, `options`, and `agent` directives of a pipeline.

Triggering Pipeline Builds

The `trigger` directive defines when Jenkins starts a build of a pipeline. Pipelines without a `trigger` directive must be started manually through the Jenkins web UI.

The main DSL statements allowed inside a `trigger` directive are:

- `pollSCM` takes a crontab expression to check a VCS repository periodically for changes. When it finds changes (new commits or merges) it starts a new build.
- `cron` starts new builds according to a crontab expression. It starts a new build periodically and unconditionally.
- `upstream` starts a pipeline build after another pipeline completes a build. It is a way of concatenating multiple pipelines into a larger workflow.

Both the `pollSCM` and `cron` directives accept `H/number` as an alternative to `*/number` for fixed time intervals. Using `H` takes a random number inside the interval and `*` starts from zero and leads to a more even distribution of builds over time.

A `triggers` directive can take multiple statements, for example:

```
triggers {
    pollSCM ('H/20 * * * *')
    upstream (upstreamProjects: 'library/main', threshold:
        hudson.model.Result.SUCCESS)
}
```

Jenkins also supports starting a build from an automated process using a web hook. Web hooks are Jenkins URLs that embed a random secret for authentication and the name of a VCS repository. When Jenkins receives a web hook, it starts builds for all pipelines that would poll that VCS repository.

Using web hooks requires that your VCS can connect to your Jenkins instance, which might be prevented by firewall rules, especially when your VCS is an externally hosted service such as GitHub.

Selecting an Agent to Run a Build

The `agent` directive defines the agent for a pipeline or a stage. Its `node` directive selects an agent by its `label`.

Non-containerized Jenkins instances that use preconfigured servers as agent nodes require you to configure your agent servers with labels so a `node` directive can select them.

The OpenShift Sync plug-in automatically labels the Red Hat Jenkins Agent container images with the name of their image streams: `nodejs` and `maven`. The OpenShift Sync plug-in also creates multiple instances (pods) for each agent if there are concurrent builds.

Red Hat recommends that all pipelines on OpenShift run as disposable agent pods instead of running on the Jenkins master. This configuration makes your Jenkins instance more reliable, requires less memory from the Jenkins master, and distributes the load of multiple builds among your OpenShift cluster nodes.

The following example configures a pipeline to run using the standard Node.js container image from Red Hat:

```
agent {  
    node {  
        label 'nodejs'  
    }  
}
```

Configuring Options of a Pipeline

The `options` directive supports different settings for a pipeline or a stage, for example:

- `disableConcurrentBuilds` prevents multiple builds of the same pipeline from building in parallel. For example, it avoids concurrent and conflicting updates to the same cluster operator in a GitOps pipeline. With a CI/CD workflow, it could prevent concurrent updates to the same test database.
- `buildDiscarder` removes old builds, their logs, and their artifacts from the Jenkins instance. If you do not discard these, then your master node could fill up all its configuration folder volume.
- `timeout` puts a time limit for the duration of a build. If a build takes longer than the designated time limit, then it is aborted and reported as failed.

Among the arguments for a `buildDiscarder`, the `logRotator` directive supports keeping some of the most recent builds, as in the following example:

```
options {  
    buildDiscarder (logRotator (numToKeepStr: '30', artifactNumToKeepStr: '30'))  
    disableConcurrentBuilds ()  
    timeout (time: 1, unit: 'HOURS')  
}
```

Designing Stages of a Pipeline

When you design and implement a pipeline, it is important to consider which kind of pipeline project to use. Do you require support for running builds from multiple branches?

- Pipeline projects ignore all branches except the one specified at project creation time, which is usually your master or main branch. You must merge any other branch to the main branch for the pipeline to act on your changes.
- Multibranch pipeline projects can run parallel builds for different branches and require that you write your pipeline to work reliably under these conditions.

If you use a branch-based workflow for approving changes, you probably require a multibranch pipeline. One way of writing a pipeline for a multibranch pipeline project is to either perform or skip stages based on the branch running the build.

Skipping Stages in a Pipeline

The `when` directive allows a stage to be executed or skipped under some conditions, for example:

- `branch` specifies the name of a branch that runs the stage in a multibranch pipeline.
- `changeset` specifies that the stage runs for changes affecting a set of files from VCS. File changes are specified using Maven file set syntax.
- `expression` allows using a Groovy expression to refer to values of parameters and environment variables.

Using multiple conditions inside the same `when` directive requires using the `allOf` and `anyOf` directives, for example:

```
when {
  allOf {
    branch 'master'
    changeset '/*.js'
  }
}
```

Using the DSL of the OpenShift Client Plug-in or Raw OpenShift CLI Commands

The OpenShift Client plug-in offers a Domain-Specific Language (DSL) designed for imperative CI/CD processes. Sophisticated CI/CD pipelines that perform complex manipulations of OpenShift resources usually benefit from using the DSL.

As powerful as the OpenShift Client DSL is, it may not be the best choice for a GitOps workflow. A system administrator without Java or JavaScript programming knowledge might find understanding the Groovy syntax of a Jenkins DSL challenging.

Among the disadvantages of using the OpenShift Client DSL to implement a GitOps workflow:

- The DSL always assumes a current project reads back all resources it creates or changes. It forces you to break commands that manipulate resources from different namespaces into multiple steps.

That behavior works for CI/CD pipelines because they usually build and deploy a single application that lives inside a single OpenShift project.

GitOps pipelines frequently mix namespaced and non-namespaced resources, or resources from multiple namespaces (of multiple operators), that a single `oc apply` command can manage.

- The DSL increases verbosity for simple declarative GitOps workflows because it is designed for scripted pipelines and requires adding `script` directives to declarative pipelines.
- Even if your pipeline never changes OpenShift credentials (it runs as the Jenkins service account) and never changes the project, the DSL requires `withCluster` and `withProject` Groovy blocks, further increasing verbosity.

The following example shows a pipeline stage that applies OpenShift resource files using the DSL:

```
stage ('Apply resources') {
    steps {
        script {
            openshift.withCluster () {
                openshift.withProject ('openshift-config') {
                    openshift.apply ('-k config')
                }
            }
        }
    }
}
```

The following shows the same example stage, but coded using the `sh` step to invoke the OpenShift CLI:

```
stage ('Apply resources') {
    steps {
        sh 'oc apply -k config'
    }
}
```

Given the simplicity of most GitOps declarative pipelines, using the OpenShift Client DSL is usually overkill. Invoking `oc` commands is easier and fits a large number of scenarios.

Designing and Implementing the Apply Pipelines

Joining the concepts of OpenShift declarative resource management, Jenkins declarative pipelines, and `Jenkinsfile` configuration directives enables you to implement a GitOps workflow. You start with the Apply pipeline, and then you review an implementation of the Drift pipeline.

The Apply Pipeline supports multiple branches, as required by a typical GitOps process, because you must make changes to a branch and allow someone to review your changes before applying them to a live cluster. You also require at least minimal validation of your resource file to prevent incorrect resource files from being merged to the main branch.

A sample Apply Pipeline has the following stages:

- *Validate*: Checks the syntax of the input configuration files to catch syntax errors early on. This stage runs on all of your branches.
- *Apply*: Applies the configuration changes to a cluster. This stage runs on your main branch only.
- *Test*: Tests that the configurations are live on a cluster by validating their expected side effects. That stage also runs only on your main branch.

The following sections discuss the main pieces of a `Jenkinsfile` for the Apply Pipeline.

Configuring the Apply Pipeline

The Apply Pipeline polls the VCS periodically for changes and disallows parallel builds because overlapping builds can intermittently fail as they apply and test changes to the same live cluster and the same cluster operators.

Any standard agent can work, but the Node.js agent is the lightest one.

Validating Configuration Files

The Validate stage is the only stage that runs for all branches.

Be warned that it is impossible to thoroughly validate OpenShift resource files before applying them to a live cluster. Make your best effort to validate using the `--dry-run` and `--validate` options. However, sometimes a resource works for `oc apply` command with the `--dry-run` option and fails without that option.

You cannot perform reliable validation of your resource files because your cluster might have multiple resource controllers and admission plug-ins, which depend on each other; when you use the `--dry-run` option, changes are not visible across controllers and plug-ins.

However, for most common scenarios your best efforts will be sufficient.

It is possible to write a pipeline that provisions a temporary OpenShift cluster on-demand in a cloud provider, but it may not be a quick nor cheap operation to perform for every configuration change.

The following is a complete implementation of a Validate stage of a GitOps Apply pipeline. This example assumes that you have a single live cluster:

```
stage ('Validate resources') {
    steps {
        sh 'oc apply --dry-run -k config'
    }
}
```

Applying Changes to Configuration Files

The Apply stage only applies changes to the cluster when running builds from the master branch because these are approved changes.

The Apply stage waits until each of the affected cluster operators act on the configuration changes. If the Apply state does not wait, then the subsequent Test stage could fail intermittently.

The details vary for each cluster operator but the following is a common scenario:

- First, wait for each operator to start progressing.
- Then, wait for all pods managed by the operator to redeploy.

Some cluster operators might also require that you make sure that there are no old pods remaining from a previous deployment.

The following listing shows a generic example of waiting for a cluster operator and its redeployment:

```
stage ('Apply resources') {
    when {
        branch 'master'
    }
    steps {
        sh 'oc apply -k config'
```

```
// May require multiple of the following:  
sh 'oc wait co/operatorname --for condition=Progressing \  
--timeout 15s || true'  
sh 'oc rollout status deployment/operatormanagedapp \  
-n operatornamespace -w --timeout 360s'  
}  
}
```

Testing Configuration Changes

The Test stage only tests side-effects of configuration changes when running builds from the master branch, similar to the Apply stage.

The logic on the test stage can vary, from a simple and incomplete smoke test to verify that your cluster is still up after your configuration changes, to a very specific check of the intended side-effects of your configuration files.

Sometimes, your test steps also have side-effects that may require a post directive to undo and clean up.

The following is a skeleton of a Test stage in a GitOps Apply pipeline.

```
stage ('Test configuration changes') {  
when {  
    branch 'master'  
}  
steps {  
    // oc commands to test side-effects of previous changes  
    ...output omitted...  
}  
post {  
    always {  
        // oc commands to clean up side-effects of the tests  
        ...output omitted...  
    }  
}
```

Designing and Implementing the Drift Pipeline

The Drift pipeline is simpler than the Apply pipeline because it has no side-effects to be tested and undone. It is also simpler because it is not part of reviewing and approving changes to configuration files and does not have to be a multibranch pipeline.

A sample Drift pipeline has the following stages:

- *Check*: verifies if the configuration of the cluster matches the configuration files in version control. If it does not match, then there is a configuration drift.
- *Remediate*: if a configuration drift is found, then take action to fix the issue and restore the cluster to the latest approved configurations.

An alternative action for the Remediate stage is to notify IT Operations about the issue, for example, by sending an email message or opening a support ticket.

The example implementation makes the Remediate state a `post` directive instead of an actual stage. It executes only when the Check stage fails because it finds a drift.

The following sections describe the main pieces of a `Jenkinsfile` for the Drift pipeline.

Configuring the Drift Pipeline

Run the Drift pipeline periodically because unexpected configuration changes could occur at any time. Any of the standard agents will work, but the Node.js agent is the lightest one.

Because the Drift pipeline has no side-effects of its own, parallel builds are not an issue.

Checking for Configuration Drift

The Drift pipeline saves a configuration drift report for auditing purposes. This report is the output of `oc diff`.

The Check stage fails to build if a drift is found, so these builds are displayed in red in the Jenkins web UI. If there is no drift, then the build succeeds.

The following is one way to implement the Check stage of a GitOps Drift pipeline:

```
stage ('Check resource drift') {
    steps {
        sh 'oc diff -k config | tee drift-report.txt'
        sh '! test -s drift-report.txt'
    }
}
```

Firing the Apply Pipeline

If the Apply pipeline remediates cluster configurations, it does not have to duplicate the logic required to validate, apply, and test configuration changes. That logic is already part of the Apply pipeline.

Jenkins supports running a pipeline as a step in another pipeline using the `build` statement.

The `post` directive either saves the drift report generated by the Check stage, or creates a no-drift report for auditing purposes in the Jenkins instance using the `archiveArtifacts` statement. Both reports are delivered as text files.

A Drift pipeline could use an HTML beautifier of `diff` command outputs and the Jenkins HTML Publisher plug-in to generate better-looking drift and no-drift reports.

The following is an example of the `post` directive that saves these reports and fires the Apply pipeline to restore the cluster configurations to the state in version control.

```
post {
    failure {
        archiveArtifacts artifacts: '*.txt'
        build job: 'apply/master'
    }
    success {
        sh 'rm drift-report.txt'
```

```
sh 'echo \'There is no configuration drift\' > no-drift.txt'
archiveArtifacts artifacts: '*.txt'
}
```

Improving the Design of Your GitOps Pipelines

Red Hat recommends that you create a shell script with a test suite that is invoked from a `Jenkinsfile`, instead of coding tests directly in a `Jenkinsfile`.

A related best practice for implementing complex logic outside of a pipeline is creating Groovy libraries. This may not be an exciting prospect to a non-developer system administrator, but it is an attractive alternative to developers implementing CI/CD workflows.

As your Test and Apply stages become more complex, you can create multiple pairs of Apply and Drift pipelines, for example, one for the settings of each cluster operator. Each of these pipeline pairs would take input from a different VCS repository. This approach makes your Test stages simpler, and allows you to control who can change configurations for each cluster operator using VCS security rules.

If your OpenShift cluster allows inbound access from your VCS, you can use a web hook instead of periodic VCS checks. This approach saves a few cluster compute resources otherwise wasted by polling VCS.

Eventually, you will need to roll back changes from a failed build of the Apply pipeline. Minimally, add some notification capabilities so that the IT team takes action, such as manually reverting to the latest known good configuration in version control.

A GitOps tool native to Kubernetes, such as ArgoCD, can observe resources and perform drift checking only when resources change, instead of performing drift checking periodically. This approach saves a few cluster compute resources wasted comparing live resources with version control when there is no change to the live resources.

Consider using different Git repositories to store your pipelines and your configuration files. This approach makes the whole process more reliable because it decreases the chances of human error.



References

Jenkins Pipeline Syntax at
<https://www.jenkins.io/doc/book/pipeline/syntax/>

► Guided Exercise

Configuring OpenShift using GitOps and Jenkins

In this exercise you will automate the configuration of HTPasswd authentication and group membership from resource files in Git using Jenkins.

Outcomes

You should be able to:

- Create a Jenkins pipeline that applies HTPasswd Identity Provider (IdP) settings and RBAC permissions from YAML files in a Git repository.
- Create another Jenkins pipeline that compares configuration in an OpenShift cluster with YAML files and, if there are differences, remediates the configuration drift.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The Jenkins master and Node.js agent container images from Red Hat.
- A running Jenkins master in the `gitops-deploy` OpenShift project, as completed in *Guided Exercise: Deploying Jenkins on OpenShift*.
- A personal account at GitHub.
- Resource files for configuring an HTPasswd IdP and RBAC permissions.

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

This command ensures that the cluster API is reachable and downloads sample resource files. It also verifies that Jenkins is running in the `gitops-deploy` project and deploys Jenkins if not.

```
[student@workstation ~]$ lab gitops-gitops start
```

Instructions

- 1. Create a GitHub project to host your sample configuration files.
- 1.1. Open a web browser and access <https://github.com>.
If you do not have a GitHub account, then click **Sign Up** and follow the instructions to create a personal account.
If you have a GitHub account, then click **Sign In** and follow the instructions to log in using your personal account.

**Important**

Beginning August 13, 2021, GitHub no longer accept account passwords when authenticating operations from the command line.

You need to create a Personal Access Token (PAT) with *public_repo* permission granted as described at <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>.

- 1.2. Create a new empty repository named `gitops-gitops`.

On the **Repositories** page, click **New** to enter the **Create a new repository** page. Type `gitops-gitops` in the **Repository name** field. Leave all other fields at their default values and click **Create repository**.

On the **Quick setup** page, click on the clipboard icon to copy the HTTPS URL of your repository.

- 1.3. Open a terminal on the **workstation** machine and clone the new, empty repository into your home folder. Paste the URL obtained in the previous step into your terminal.

```
[student@workstation ~]$ git clone \
https://github.com/youraccount/gitops-gitops.git
Cloning into 'gitops-gitops'...
warning: You appear to have cloned an empty repository.
```

- 1.4. Copy the contents of the `~/D0380/labs/gitops-gitops/` folder into your local clone of the git repository, and then enter your local clone.

```
[student@workstation ~]$ cp -r ~/D0380/labs/gitops-gitops/* ~/gitops-gitops
[student@workstation ~]$ cd ~/gitops-gitops
[student@workstation gitops-gitops]$ ls
config Jenkinsfile-apply Jenkinsfile-drift orig wait_oauth.sh
```

- 1.5. Inspect the sample configuration files in the `~/gitops-gitops/config` folder. These files are ready for this exercise and you should not change any of them until specifically instructed to do so.

- `oauth.yaml`: configures the OpenShift Authentication Operator with an HTPasswd IdP.
- `htpasswd-secret-data`: defines HTPasswd users and passwords. Besides the initial `admin` and `developer` users, it defines the `testuser` user for the Test stage of the Apply Pipeline.
- `project-leaders.yaml`: defines the `project-leaders` group and its members.
- `self-provisioners.yaml`: defines the groups allowed to create OpenShift projects.
- `kustomization.yaml`: defines the Kustomize configuration that generates the HTPasswd secret.

- 1.6. Commit the configuration files and push to GitHub.

```
[student@workstation gitops-gitops]$ git add config
[student@workstation gitops-gitops]$ git commit -m 'initial auth + rbac settings'
...output omitted...
create mode 100644 config/self-provisioners.yaml
...output omitted...
[student@workstation gitops-gitops]$ git branch -M main
[student@workstation gitops-gitops]$ git push -u origin main
...output omitted...
* [new branch]      main -> main
```

- ▶ 2. Verify that your Jenkins master is ready and running and assign cluster administration rights to the Jenkins service account.
- 2.1. Log in to OpenShift as the `admin` user and enter the `gitops-deploy` project in OpenShift.

```
[student@workstation gitops-gitops]$ oc login -u admin -p redhat \
https://api.ocp4.example.com:6443
Login successful.
...output omitted...
[student@workstation gitops-gitops]$ oc project gitops-deploy
Now using project "gitops-deploy" on server "https://api.ocp4.example.com:6443".
```

- 2.2. Verify that the Jenkins pod is ready and running, and that it is fully initialized. Look for the message "Jenkins is fully up and running". You might get this message more than once.

```
[student@workstation gitops-gitops]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
jenkins-1-bft7r   1/1     Running   0          2m54s
jenkins-1-deploy   0/1     Completed  0          2m57s
[student@workstation gitops-gitops]$ oc logs jenkins-1-bft7r | \
grep 'up and running'
2021-10-28 17:57:37 INFO    hudson.WebAppMain$3 run Jenkins is fully up and
running
```

- 2.3. Remove the `developer` user from the `gitops-deploy` project. You do not want to allow developers to create and build pipelines that configure the cluster.
Also, remove the ability of the Jenkins service account to create new projects. This permission becomes unnecessary because of the actions performed in the next step.

```
[student@workstation gitops-gitops]$ oc policy remove-role-from-user admin \
developer
clusterrole.rbac.authorization.k8s.io/admin removed: "developer"

[student@workstation gitops-gitops]$ oc adm policy remove-cluster-role-from-user \
self-provisioner -z jenkins
clusterrole.rbac.authorization.k8s.io/self-provisioner removed: "jenkins"
```

- 2.4. Assign cluster administrator privileges to the Jenkins service account.

Remember that cluster roles are not namespaced and the service account is in the current project, so you do not require the `-n` option.

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

► 3. Complete the Jenkins file for your Apply Pipeline.

Configure your pipeline to pool its Git repository for changes every 3 minutes, retain only the latest 5 builds, and use a Node.js agent. Also, add steps to the Validate, Apply, and Test stages to your pipeline.

- 3.1. Open the file named `Jenkinsfile-apply` in the `~/gitops-gitops` folder using a text editor. To save you time, this file provides a skeleton for the Apply pipeline; you will complete configuration settings and steps for each stage as you perform the following steps.

```
pipeline {
  triggers {
  }
  options {
  }
  agent {
  }
  stages {
    stage ('Validate configuration resources') {
      steps {
      }
    }
    stage ('Apply resources') {
      steps {
      }
    }
    stage ('Verify test user') {
      steps {
      }
    }
  }
}
```

- 3.2. Complete the `triggers` and `options` directives.

Pool your Git repository at three-minute intervals and disable concurrent builds.

```
pipeline {
  triggers {
    pollSCM ('H/3 * * * *')
  }
  options {
    disableConcurrentBuilds()
  }
  stages {
    ...output omitted...
  }
}
```

3.3. Complete the agent directive.

Add a node directive that runs all your stages using the agent image with the label `nodejs`.

```
pipeline {
    ...output omitted...
    agent {
        node {
            label 'nodejs'
        }
    }
    stages {
        ...output omitted...
```

3.4. Complete the Validate stage.

Add steps that validate configuration files by performing a dry run with resource validation of the `oc apply` command using Kustomize.

```
pipeline {
    ...output omitted...
    stages {
        stage ('Validate configuration resources') {
            steps {
                sh 'oc apply --dry-run --validate -k config'
            }
        }
    }
    ...output omitted...
```

3.5. Complete the Apply stage.

Add steps using the `oc apply` command with Kustomize, and a condition that performs the stage only for the `main` branch. Also, call the `wait_oauth.sh` script that waits until the OAuth pods redeploy to ensure that the new authentication settings are in effect.

```
pipeline {
    ...output omitted...
    stages {
        ...output omitted...
        stage ('Apply resources') {
            when {
                branch 'main'
            }
            steps {
                sh 'oc apply -k config'
                sh './wait_oauth.sh'
            }
        }
    }
    ...output omitted...
```

3.6. Complete the Test stage.

Add steps that log in to OpenShift as `testuser` and attempt to create a project; expect project creation to fail. This test proves that regular users (not members of the `project-leaders` group) cannot create projects.

Also, add a condition that performs this stage only in the `main` branch.

```
pipeline {
    ...output omitted...
    stage ('Verify test user') {
        when {
            branch 'main'
        }
        steps {
            sh 'oc login -u testuser -p redhat123 --insecure-skip-tls-verify \
https://kubernetes.default.svc:443'
            sh 'oc new-project test-testuser || true'
        }
    }
}
```

- 3.7. Save your changes, commit, and then push to GitHub. Also, commit and push the `wait_oauth.sh` script.

Before you commit and push, you can compare your new `Jenkinsfile-apply` file with the contents of `Jenkinsfile-apply` in the `~/D0380/solutions/gitops-gitops` folder.

```
[student@workstation gitops-gitops]$ diff Jenkinsfile-apply \
~/D0380/solutions/gitops-gitops/Jenkinsfile-apply
[student@workstation gitops-gitops]$ git add Jenkinsfile-apply wait_oauth.sh
[student@workstation gitops-gitops]$ git commit -m 'Apply Pipeline definition'
...output omitted...
[student@workstation gitops-gitops]$ git push
...output omitted...
```

- 3.8. Inspect the `wait_oauth.sh` script that you invoke from the `Apply` stage. It is ready to use and contains commands that wait until:

- The Authentication Cluster Operator recognizes the changes.
- A new set of OAuth pods is available.
- The previous set of OAuth pods are removed.

```
#!/bin/bash

oc wait co/authentication --for condition=Progressing \
--timeout 90s
oc rollout status -n openshift-authentication deployment/oauth-openshift \
--timeout 90s
while [ -n "$(oc get pod -n openshift-authentication -o \
jsonpath='{.items[?(@.metadata.deletionTimestamp != "")].metadata.name}')" ]
```

```
do
    sleep 3
done
```

► 4. Create and test the Apply Pipeline in Jenkins.

- 4.1. Get the host name of the Jenkins master. Do not close your terminal, you will come back to it a few times during this exercise.

```
[student@workstation gitops-gitops]$ oc get routes
NAME      HOST/PORT
jenkins   jenkins-gitops-deploy.apps.ocp4.example.com ...
```

- 4.2. Open the Jenkins web UI using a web browser and the URL from the previous step. If necessary, accept the TLS certificate when prompted, and then click **Log in with OpenShift**. Also, accept the TLS certificate of the OpenShift OAuth server, if necessary, and then click **htpasswd_provider** to enter the OpenShift log in page. Log in as the **admin** user with the **redhat** password, and then click **Allow selected permissions**. OpenShift redirects your browser to the Jenkins **Dashboard** page.
- 4.3. Create a new multibranch pipeline project for the Apply Pipeline. Click **New Item** on the Jenkins **Dashboard** page, and then type **apply** as the item name. Select **Multibranch Pipeline**, and then click **Ok**. Under the **Branch Sources** heading, select **Add Source → Git** and type the HTTPS URL of your GitHub repository, `https://github.com/youraccount/gitops-gitops.git`, in the **Project Repository** field. Because your Git repository is public, you do not provide any credentials. Under the **Build Configuration** heading, type `Jenkinsfile-apply` in the **Script Path** field. Do not set any other fields and click **Save**.
- 4.4. Verify that Jenkins finds the **main** branch. Jenkins automatically starts a **Scan Multibranch Pipeline** job and displays its logs. Wait until it finishes and verifies that the logs of the job indicate that Jenkins found a **main** branch. If a **main** branch is not found, click **Configure**, fix your Jenkins project settings, and then click **Scan Multibranch Pipeline Now**.
- 4.5. Verify that the **main** branch has a successful build. Click either **Up** or **Jenkins** to go to the Jenkins **Dashboard** page. Click **apply** to enter the Apply project, then click **main** to list builds of the **main** branch of your Git repository. If you do not see any build, then click **Build Now** to start a build. If you see a build that is in progress, then wait until it finishes with a successful status. If you see errors during the build, click on the build number to enter the build page and click **Console Output** to find clues about what caused these errors. Fix your **Jenkinsfile**, commit, and push your changes. Then, click **Build Now** to start a new build. Do not close your web browser. You will come back to the Jenkins web UI a few times during this exercise.

- 5. Add a new HTPasswd user to the HTPasswd data file on Git, and then verify that the Apply pipeline builds and adds the new user to the cluster.
- 5.1. Add the `newdev` user with password `redhat123` to the `htpasswd-secret-data` file in the `~/gitops-gitops/config` folder.

```
[student@workstation gitops-gitops]$ htpasswd -b config/htpasswd-secret-data \
newdev redhat123
Adding password for user newdev
```

- 5.2. Add the `newdev` user as a member of the `project-leaders` group.
Use a text editor to open the `project-leaders.yaml` file in the `~/gitops-gitops/config` folder, and then add a `newdev` entry to the `users` array.

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  name: project-leaders
users:
  - developer
  - newdev
```

- 5.3. Save your changes, commit, and push to GitHub.

```
[student@workstation gitops-gitops]$ git add config/htpasswd-secret-data
[student@workstation gitops-gitops]$ git add config/project-leaders.yaml
[student@workstation gitops-gitops]$ git commit -m 'Add new project leader'
...output omitted...
[student@workstation gitops-gitops]$ git push
...output omitted...
```

- 5.4. Switch to the Jenkins web UI and enter the main branch of the Apply project, if you are not already there. Wait until a new build starts and completes successfully. Click on the build number to enter the build page.
Verify that there is a reference to the latest commit made to the branch in the Changes section. Click **Console Output** and review the logs to verify that the `oc apply` command changed the HTPasswd secret, the `oauth-openshift` deployment successfully rolled out, and the `test-user` user can log in to OpenShift but cannot create a new project.
- 5.5. Switch to a terminal, log in to OpenShift as the `newdev` user, and verify you can create new projects. This proves again that the pipeline applied to the cluster all settings that you made to the configuration files in your Git repository.

```
[student@workstation gitops-gitops]$ oc login -u newdev -p redhat123
Login successful.
...output omitted...
[student@workstation gitops-gitops]$ oc new-project test-newdev
Now using project "test-newdev" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

- 5.6. Log in to OpenShift as a cluster administrator, verify that the newdev user is a member of the project-leaders group, and delete the test-newdev project before proceeding to the next step.

```
[student@workstation gitops-gitops]$ oc login -u admin -p redhat
Login successful.
...output omitted...
[student@workstation gitops-gitops]$ oc get groups
NAME          USERS
project-leaders developer, newdev
[student@workstation gitops-gitops]$ oc delete project test-newdev
project.project.openshift.io "test-newdev" deleted
```

▶ 6. Complete a Jenkinsfile for the Drift Pipeline.

Configure the pipeline to build every 5 minutes using a Node.js agent. Also, add steps to the Check stage and to the post actions that generate either a no-drift.txt or a drift-report.txt file and fire the Apply pipeline, depending on the success status of the Check stage.

- 6.1. Use a text editor to open the file named Jenkinsfile-drift in the ~/gitops-gitops folder and complete the triggers directive.

```
pipeline {
    triggers {
        cron ('H/5 * * * *')
    }
    agent {
        node {
            label 'nodejs'
        }
    }
    stages {
        ...output omitted...
```

- 6.2. Complete the Check stage using the oc diff command with Kustomize, and save the output to a text file. Make sure that the stage fails if the text file is not empty.

```
pipeline {
    ...output omitted...
    stages {
        stage ('Check resource drift') {
            steps {
                sh 'oc diff -k config | tee drift-report.txt'
                sh '! test -s drift-report.txt'
            }
        }
    }
    ...output omitted...
```

- 6.3. Complete the post directive to either save the drift report text file and fire the Apply Pipeline, or to create and save a no-drift report text file.

```

pipeline {
    ...output omitted...
    post {
        failure {
            archiveArtifacts artifacts: '*.txt'
            build job: 'apply/main'
        }
        success {
            sh 'rm drift-report.txt'
            sh 'echo \'There is no configuration drift\' > no-drift.txt'
            archiveArtifacts artifacts: '*.txt'
        }
    }
}

```

6.4. Save your changes, commit, and push to GitHub.

Before you commit and push, you can compare your new `Jenkinsfile-drift` file with the contents of `Jenkinsfile-drift` in the `~/D0380/solutions/gitops-gitops` folder.

```

[student@workstation gitops-gitops]$ diff Jenkinsfile-drift \
~/D0380/solutions/gitops-gitops/Jenkinsfile-drift
[student@workstation gitops-gitops]$ git add Jenkinsfile-drift
[student@workstation gitops-gitops]$ git commit -m 'Drift Pipeline definition'
...output omitted...
[student@workstation gitops-gitops]$ git push
...output omitted...

```

► 7. Create and test a project in Jenkins for the Drift pipeline.

7.1. Switch to the Jenkins web UI and create a new pipeline project for the Drift pipeline.

Click **Jenkins** to enter the Jenkins **Dashboard** page. Click **New Item** and type **drift** as the item name. Select **Pipeline**, and then click **Ok**.

Under the **Pipeline** heading, select **Pipeline script from SCM**. Select **Git** in the **SCM** field and type the HTTPS URL of your GitHub repository, `https://github.com/youraccount/gitops-gitops.git`, in the **Repository URL** field. Set **Branch Specifier** to `'*/main'`. Then type `Jenkinsfile-drift` in the **Script Path** field.

Do not set any other fields and click **Save**.

7.2. Verify that Jenkins can successfully build the Drift pipeline.

Jenkins should be on the **Builds** page for the Drift pipeline. Click **Build Now** to start a new build and wait until it finishes successfully.

If you see errors during the build, click on the build number to enter the build page and click **Console Output** to find clues about what caused these errors.

After you have a successful build, click on the build number to enter the build page to verify it produced an artifact named `no-drift.txt`.

► 8. Remove the `newdev` user from the `project-leaders` group in your cluster and verify that the Drift pipeline restores group membership.

- 8.1. Change the live group in your cluster using the `oc adm groups remove-users` command to remove the `newdev` member.

```
[student@workstation gitops-gitops]$ oc adm groups remove-users \
  project-leaders newdev
group.user.openshift.io./project-leaders removed: "newdev"
```

- 8.2. Verify that the `project-leaders` group does not contain the `newdev` user.

The Drift pipeline might fire and restore group membership between this and the previous step. If that happens, then ignore the difference in outputs and proceed to the next step.

```
[student@workstation gitops-gitops]$ oc get group project-leaders
NAME      USERS
project-leaders developer
```

- 8.3. Wait until Jenkins starts a new build of the Drift pipeline, and the Drift pipeline starts a new build of the Apply pipeline to restore cluster settings.

Switch to the Jenkins web UI and enter the `drift` project if you are not already there.

Wait until a new build starts and terminates with a failure. When you switch to your web browser, Jenkins might already be finished with the build.

Click on the build number to enter the build page and verify that the build produced an artifact named `drift-report.txt`. Click the **Open Blue Ocean** and verify also that the **Check resource drift** stage failed, its logs show the same information as the `drift-report.txt` file, and the `newdev` user was removed from the `project-leaders` group.

In the logs of the **Declarative: post actions** stage, verify that it scheduled a build of the `apply/main` project. Make a note of the build number so you can verify that build of the Apply pipeline.

By the time you finish verifying stages and logs, you will likely see a new build of the Drift pipeline that finds no drift.

- 8.4. Verify that a build of the Apply pipeline was started by the Drift pipeline.

Click **Jenkins**, then **apply**, then **main**, and then click the number of the build you got from the previous step.

A page displays with the message "Started by upstream project drift" and the build number of the project.

- 8.5. Switch to the terminal and verify that the `newdev` user is again a member of the `project-leaders` group

```
[student@workstation gitops-gitops]$ oc get group project-leaders
NAME      USERS
project-leaders developer, newdev
```

- 9. Clean up your Jenkins instance and remove it from your OpenShift cluster. Also remove your repository from GitHub.

- 9.1. Switch to the Jenkins web UI and remove the Drift pipeline first, followed by the Apply pipeline.

Click **Jenkins**, click the down arrow close to **drift**, click **Delete Pipeline**, and then click **Yes** to confirm the operation.

Click **Jenkins**, click the down arrow close to **apply**, click **Delete Multibranch Pipeline** and then click **Yes** to confirm the operation.

- 9.2. Switch to the GitHub repository page, and click **Settings** and scroll down until you find the **Danger Zone** header. Click **Delete this repository** and follow the instructions to confirm deleting the repository.
- 9.3. Switch to the terminal, enter your home folder, and revoke cluster administration privileges from the Jenkins service account.

The role binding remains even after the project and the service account are removed, so any developer could create new projects reusing these names to get cluster administrator privileges.

```
[student@workstation gitops-gitops]$ cd ~  
[student@workstation ~]$ oc adm policy remove-cluster-role-from-user \  
  cluster-admin -z jenkins -n gitops-deploy  
...output omitted...  
clusterrole.rbac.authorization.k8s.io/cluster-admin removed: "jenkins"
```

- 9.4. Delete the **gitops-deploy** project.

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

- 9.5. Delete the **gitops-gitops** local folder.

```
[student@workstation ~]$ rm -rf ~/gitops-gitops
```

- 9.6. Restore your cluster initial authentication settings:

```
[student@workstation ~]$ oc apply -k ~/D0380/labs/gitops-gitops/orig  
clusterrolebinding.rbac.authorization.k8s.io/self-provisioners configured  
secret/htpasswd-secret configured  
oauth.config.openshift.io/cluster unchanged
```

Finish

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

```
[student@workstation ~]$ lab gitops-gitops finish
```

► Lab

Implementing GitOps with Jenkins

In this lab, you will implement a GitOps workflow to automate the Samples cluster operator configuration and remove some of the standard templates from your cluster.

Outcomes

You should be able to:

- Deploy a Jenkins instance with cluster administration rights on OpenShift using an Ansible Playbook.
- Complete and deploy a Jenkins pipeline that applies settings for the `Samples` cluster operator from YAML files in a Git repository.
- Use YAML resource files to exclude standard templates for Ruby on Rails applications from your cluster and configure the `Samples cluster operator` so these templates are not recreated.
- Generate a Jenkins API token.

The `Samples` cluster operator manages standard templates and image streams in a cluster. It usually acts only during installation and upgrades of OpenShift, and it also watches the `openshift` namespace to prevent accidental deletion of these standard resources.



Note

To make this review lab shorter, you will not deploy a Drift pipeline; therefore you will implement an incomplete GitOps workflow.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The Jenkins and Node.js agent container images from Red Hat.
- Initial files for the `Samples` operator configuration resources, the `Apply` pipeline, and the Ansible Playbook.

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

This command ensures that the cluster API is reachable and downloads sources of the sample application.

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

Instructions

1. Complete an Ansible Playbook that deploys Jenkins on the `gitops-review` project.
An incomplete playbook is available in the `deploy-jenkins.yaml` file in the `/home/student/D0380/labs/gitops-review` folder. That playbook uses YAML resource files in the `/home/student/D0380/labs/gitops-review/resources` folder.
Complete the playbook and its resource files to use the correct OpenShift template for a Jenkins instance with persistent storage, and also to grant cluster administration rights to the Jenkins service account.
If you are unsure about the indentation in your YAML files, you can review the solution files in the `/home/student/D0380/solutions/gitops-review` folder.



Note

The playbook uses the `oc` command to process a template because that functionality is not available from the `k8s` Ansible module.

2. Run the playbook and verify that Jenkins is ready and running, and that its service account has cluster administrator privileges.
Also, log in to the Jenkins web UI as the `admin` user to verify that your Jenkins instance is fully operational.
3. Create a public GitHub repository named `gitops-review` for your GitOps workflow, and then clone that repository to your home folder.
From the `/home/student/D0380/labs/gitops-review` folder, copy the `Jenkinsfile-apply` file and the `samples` and `delete` folders to your repository, and then commit and push these files to GitHub.



Note

The stub resource files must be present in the `delete` folder because the `oc apply` command and Kustomize cannot delete Kubernetes resources. The incomplete `Jenkinsfile` already contains an `oc delete` command to handle these.

4. Create a Kustomize configuration file in the `samples` folder of your Git repository that generates configuration resources for the `Samples` cluster operator.

You already copied the `samples.yaml` file with the configuration for the `Samples` cluster operator to that folder. Do not make any change to that file and use it as input to Kustomize.



Note

If you want to understand the contents of the `samples.yaml` file, use the `oc explain config.spec` command, for example:

```
[student@workstation ~]$ oc explain config.spec \
--api-version samples.operator.openshift.io/v1
```



Note

Using Kustomize is not required by the scenario of this review lab. It uses Kustomize as a recommended practice for generic GitOps workflows.

5. Use your YAML files to change the configuration of the `Samples` cluster operator and also delete the standard templates for `Ruby on Rails` applications.

After you verify that your YAML files work as expected, commit and push them to GitHub.

Then, restore the default configuration for the `Samples` operator using the Kustomize folder at `~/D0380/labs/gitops-review/default`. Do not make any change to this folder.

Also, verify that the `Samples` cluster operator restored the templates you deleted.

6. Complete the `Jenkinsfile` for the Apply pipeline.

You already copied an incomplete `Jenkinsfile` named `Jenkinsfile-apply` to your Git repository.

Complete the `Jenkinsfile` to apply YAML resource files from the `samples` folder of your Git repository. Also, make sure that your pipeline only applies and tests changes for builds running from the `master` branch.

The pipeline also deletes resources using stub files in the `delete` folder of your Git repository, and verifies that no template for `Ruby on Rails` applications exists in the `openshift` namespace. Do not make changes to the steps that perform these actions.

If you are unsure about your changes to the `Jenkinsfile`, you can review the solution files in the `/home/student/D0380/solutions/gitops-review` folder.

7. Create a Jenkins multibranch pipeline project named `apply` for your Apply pipeline.

Configure your Jenkins project to take the `Jenkinsfile` from your GitHub repository.

After Jenkins successfully runs a build from your project, verify that the `openshift` namespace contains no template related to `Ruby on Rails` applications.

8. Generate a Jenkins API token.

Use the Jenkins web UI to access the configuration for the `admin` user and save the token to the `/home/student/jenkins-api-token.txt` file.

The `grade` command requires that token to invoke the Jenkins API and start a build of your Apply pipeline.

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 successfully.

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

Finish

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

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

The **lab** command deletes the OpenShift project you created and restores default cluster settings. It also deletes the clone of your **gitops-review** Git repository from your home folder and the Jenkins API token file.

If you wish, you can also delete the **gitops-review** repository from your GitHub account.

► Solution

Implementing GitOps with Jenkins

In this lab, you will implement a GitOps workflow to automate the Samples cluster operator configuration and remove some of the standard templates from your cluster.

Outcomes

You should be able to:

- Deploy a Jenkins instance with cluster administration rights on OpenShift using an Ansible Playbook.
- Complete and deploy a Jenkins pipeline that applies settings for the `Samples` cluster operator from YAML files in a Git repository.
- Use YAML resource files to exclude standard templates for Ruby on Rails applications from your cluster and configure the `Samples cluster operator` so these templates are not recreated.
- Generate a Jenkins API token.

The `Samples` cluster operator manages standard templates and image streams in a cluster. It usually acts only during installation and upgrades of OpenShift, and it also watches the `openshift` namespace to prevent accidental deletion of these standard resources.



Note

To make this review lab shorter, you will not deploy a Drift pipeline; therefore you will implement an incomplete GitOps workflow.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The Jenkins and Node.js agent container images from Red Hat.
- Initial files for the `Samples` operator configuration resources, the `Apply` pipeline, and the Ansible Playbook.

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

This command ensures that the cluster API is reachable and downloads sources of the sample application.

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

Instructions

1. Complete an Ansible Playbook that deploys Jenkins on the `gitops-review` project.
An incomplete playbook is available in the `deploy-jenkins.yaml` file in the `/home/student/D0380/labs/gitops-review` folder. That playbook uses YAML resource files in the `/home/student/D0380/labs/gitops-review/resources` folder.
Complete the playbook and its resource files to use the correct OpenShift template for a Jenkins instance with persistent storage, and also to grant cluster administration rights to the Jenkins service account.
If you are unsure about the indentation in your YAML files, you can review the solution files in the `/home/student/D0380/solutions/gitops-review` folder.



Note

The playbook uses the `oc` command to process a template because that functionality is not available from the `k8s` Ansible module.

- 1.1. Inspect the incomplete playbook.

Open the `deploy-jenkins.yaml` file in the `~/D0380/labs/gitops-review` folder with a text editor and verify that it contains five Ansible tasks.

You will change only the first, second, and fourth tasks, which are highlighted in the following listing.

```
- name: Deploy Jenkins for GitOps
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Create project for Jenkins
      ...output omitted...
    - name: Process the Jenkins standard template
      ...output omitted...
    - name: Deploy Jenkins using resources from the standard template
      ...output omitted...
    - name: Grant cluster-administrator privileges to the Jenkins service account
      ...output omitted...
    - name: Wait until Jenkins is available
      ...output omitted...
```

Leave the playbook file open in the text editor for the following steps.

- 1.2. Complete the resource file for the first task, `Create project for Jenkins`, to create the `gitops-review` project.

This task uses the `project.yaml` file to define a project request resource. Make no changes to the task.

```
...output omitted...
- name: Create project for Jenkins
  k8s:
    state: present
    src: "{{ playbook_dir + '/resources/project.yaml' }}"
...output omitted...
```

Open a second terminal, and then open the project .yaml file in the ~/D0380/labs/gitops-review/resources folder with a text editor to specify gitops-review as the name of the project in the project request resource.

```
apiVersion: project.openshift.io/v1
kind: ProjectRequest
metadata:
  name: gitops-review
description: Project to host a Jenkins instance with cluster admin rights
```

Save your changes and leave the second terminal open for the following steps.

- 1.3. Complete the second task, Process the Jenkins standard template, to use the jenkins-persistent standard template.

Switch to your first terminal and change the oc process command inside the task as follows:

```
...output omitted...
- name: Process the Jenkins standard template
  shell: "oc process jenkins-persistent -p JENKINS_IMAGE_STREAM_TAG=jenkins:v4.8
-n openshift -o yaml"
  register: template_resources
...output omitted...
```

- 1.4. Complete the fourth task, Grant cluster administrator privileges to Jenkins, to assign the cluster-admin cluster role to the jenkins service account inside the gitops-review project.

That task uses the role-binding.yaml file to define a project request resource. Make no changes to the task.

```
...output omitted...
- name: Grant cluster administrator privileges to the Jenkins service account
  k8s:
    state: present
    src: "{{ playbook_dir + '/resources/role-binding.yaml' }}"
...output omitted...
```

Switch to your second terminal and open the role-binding.yaml file in the ~/D0380/labs/gitops-review/resources folder with a text editor. Specify the name of the cluster role and of the service account in the cluster role binding resource as follows:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: jenkins-gitops-review
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
```

```
- kind: ServiceAccount
  name: jenkins
  namespace: gitops-review
```

- 1.5. Switch to your first terminal, save changes to your playbook, and close your text editor.

You can compare your edits with the solution files in the /home/student/D0380/solutions/gitops-review folder.

2. Run the playbook and verify that Jenkins is ready and running, and that its service account has cluster administrator privileges.

Also, log in to the Jenkins web UI as the `admin` user to verify that your Jenkins instance is fully operational.

- 2.1. Log in to OpenShift as a cluster administrator and navigate to the /home/student/D0380/labs/gitops-review folder.

```
[student@workstation ~]$ oc login -u admin -p redhat \
  https://api.ocp4.example.com:6443
Login successful.
...output omitted...
[student@workstation ~]$ cd ~/D0380/labs/gitops-review
```

- 2.2. Run the `deploy-jenkins.yaml` Ansible Playbook.

```
[student@workstation gitops-review]$ ansible-playbook deploy-jenkins.yaml
PLAY [Deploy Jenkins for GitOps]
...output omitted...
PLAY RECAP ****
localhost                  : ok=5    changed=4    unreachable=0    failed=0
skipped=0      rescued=0   ignored=0
```



Note

The task [Wait until Jenkins is available] will take 1-2 minutes to complete.



Important

The lab script deploys a global Ansible configuration file to /etc/ansible/ansible.cfg. This global Ansible configuration file sets `become: true`. The `become: true` option makes playbook tasks execute as the root user. The lab script runs as the root user and logs in to OpenShift as a user with administrator privileges.

The previous `ansible-playbook` command interacts with OpenShift as a user with administrator privileges.

- 2.3. Enter the `gitops-review` project and verify that the Jenkins master pod is ready and running.

Also confirm that Jenkins is fully initialized by looking for the message "Jenkins is fully up and running" in the container logs.

```
[student@workstation gitops-review]$ oc project gitops-review
Now using project "gitops-review" on server "https://api.ocp4.example.com:6443".
[student@workstation gitops-review]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
jenkins-1-5f46k   1/1     Running   0          3m17s
jenkins-1-deploy 0/1     Completed  0          3m20s
[student@workstation gitops-review]$ oc logs jenkins-1-5f46k | \
  grep 'up and running'
2021-10-29 13:20:09 INFO    hudson.WebAppMain$3 run Jenkins is fully up and
running
```

- 2.4. Verify that the Jenkins service account has cluster administrator privileges.

One way to check these privileges is to verify if the service account is among the subjects allowed to list nodes.

```
[student@workstation gitops-review]$ oc policy who-can get node | grep jenkins
system:serviceaccount:gitops-review:jenkins
```

- 2.5. Find the host name of your Jenkins instance. Do not close your terminal, you will come back to it a few times during this exercise.

```
[student@workstation gitops-review]$ oc get routes
NAME      HOST/PORT
jenkins   jenkins-gitops-review.apps.ocp4.example.com ...
```

- 2.6. Open a web browser on the `workstation` machine, and then enter the Jenkins web UI using the URL obtained in the previous step.

If prompted, accept the TLS certificate for the Jenkins web UI, and then click **Log in with OpenShift**.

Also accept the TLS certificate of the OpenShift OAuth server, if prompted, and then click `htpasswd_provider` to enter the OpenShift log in credentials page.

Log in as the `admin` user with the `redhat` password, and then click **Allow selected permissions**. OpenShift redirects your browser to the Jenkins **Dashboard** page.

Do not close your web browser, you will come back to the Jenkins web UI later in this exercise.

3. Create a public GitHub repository named `gitops-review` for your GitOps workflow, and then clone that repository to your home folder.

From the `/home/student/D0380/labs/gitops-review` folder, copy the `Jenkinsfile-apply` file and the `samples` and `delete` folders to your repository, and then commit and push these files to GitHub.



Note

The stub resource files must be present in the `delete` folder because the `oc apply` command and Kustomize cannot delete Kubernetes resources. The incomplete `Jenkinsfile` already contains an `oc delete` command to handle these.

- 3.1. Open a web browser and access <https://github.com>.

If you do not have a GitHub account, then click **Sign Up** and follow the instructions to create a personal account.

If you do have a GitHub account, then click **Sign In** and follow the instructions to log in using your personal account.

3.2. Create a new empty repository named gitops-review.

On the **Repositories** page, click **New** to enter the **Create a new repository** page. Type **gitops-review** in the **Repository name** field. Leave all other fields at their default values and click **Create repository**.

On the **Quick setup** page, click the clipboard icon to copy the HTTPS URL of your repository.

3.3. Switch to your terminal and clone the repository into your home folder.

```
[student@workstation gitops-review]$ cd ~  
[student@workstation ~]$ git clone \  
  https://github.com/youraccount/gitops-review.git  
Cloning into 'gitops-review'...  
warning: You appear to have cloned an empty repository.
```

3.4. Copy the incomplete **Jenkinsfile**, the configuration folder of the Samples cluster operator, and the folder with stub files from the **~/D0380/labs/gitops-review** folder into your local clone of the git repository.

```
[student@workstation ~]$ cp ~/D0380/labs/gitops-review/Jenkinsfile-apply \  
  ~/gitops-review  
[student@workstation ~]$ cp -r ~/D0380/labs/gitops-review/samples ~/gitops-review  
[student@workstation ~]$ cp -r ~/D0380/labs/gitops-review/delete ~/gitops-review  
[student@workstation ~]$ ls ~/gitops-review  
delete    Jenkinsfile-apply      samples
```

4. Create a Kustomize configuration file in the **samples** folder of your Git repository that generates configuration resources for the Samples cluster operator.

You already copied the **samples.yaml** file with the configuration for the Samples cluster operator to that folder. Do not make any change to that file and use it as input to Kustomize.



Note

If you want to understand the contents of the **samples.yaml** file, use the **oc explain** command, for example:

```
[student@workstation ~]$ oc explain config.spec \  
  --api-version samples.operator.openshift.io/v1
```



Note

Using Kustomize is not required by the scenario of this review lab. It uses Kustomize as a recommended practice for generic GitOps workflows.

4.1. Enter your Git repository clone folder and inspect its **samples** folder.

It contains a single resource file named `samples.yaml`.

```
[student@workstation ~]$ cd ~/gitops-review
[student@workstation gitops-review]$ ls samples
samples.yaml
```

4.2. Inspect the `samples/samples.yaml` resource file.

The `skippedTemplates` attribute lists all standard templates related to Ruby on Rails applications and makes the Samples cluster operator delete them from the openshift namespace. Do not make any change to this file.

```
[student@workstation gitops-review]$ cat samples/samples.yaml
apiVersion: samples.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  architectures:
  - x86_64
  managementState: Managed
  skippedTemplates:
  - rails-pgsql-persistent
  - rails-postgresql-example
```

4.3. Create the Kustomize configuration file that lists all resource files in the `samples` folder.

Use a text editor to create the `kustomization.yaml` file in the `samples` folder and add the following lines.

```
resources:
- samples.yaml
```

4.4. Validate your Kustomize configurations.

```
[student@workstation gitops-review]$ oc kustomize samples
apiVersion: samples.operator.openshift.io/v1
kind: Config
...output omitted...
  skippedTemplates:
  - rails-pgsql-persistent
  - rails-postgresql-example
```

5. Use your YAML files to change the configuration of the Samples cluster operator and also delete the standard templates for Ruby on Rails applications.

After you verify that your YAML files work as expected, commit and push them to GitHub.

Then, restore the default configuration for the Samples operator using the Kustomize folder at `~/DO380/labs/gitops-review/default`. Do not make any change to this folder. Also, verify that the Samples cluster operator restored the templates you deleted.

5.1. List all templates in your cluster related to Ruby on Rails.

```
[student@workstation gitops-review]$ oc get template -n openshift | grep rails
rails-pgsql-persistent      An example Rails application ...
rails-postgresql-example    An example Rails application ...
```

- 5.2. Apply the Samples cluster operator configuration from the samples folder.

```
[student@workstation gitops-review]$ oc apply -k samples
config.samples.operator.openshift.io/cluster configured
```

- 5.3. Delete the templates for Ruby on Rails applications using the stub files from the delete folder.

```
[student@workstation gitops-review]$ oc delete -f delete
template.template.openshift.io "rails-pgsql-persistent" deleted
template.template.openshift.io "rails-postgresql-example" deleted
```

- 5.4. Once again, list all templates in your cluster related to Ruby on Rails. There should be none.

```
[student@workstation gitops-review]$ oc get template -n openshift | grep rails
```

- 5.5. Commit your samples and delete folders and then push to GitHub.

```
[student@workstation gitops-review]$ git add samples
[student@workstation gitops-review]$ git add delete
[student@workstation gitops-review]$ git commit -m 'Exclude templates for Ruby'
...output omitted...
create mode 100644 delete/templates.yaml
create mode 100644 samples/kustomization.yaml
create mode 100644 samples/samples.yaml
[student@workstation gitops-review]$ git branch -M main
[student@workstation gitops-review]$ git push -u origin main
...output omitted...
* [new branch]      main -> main
```

- 5.6. Restore the default configuration for the Samples cluster operator.

```
[student@workstation gitops-review]$ oc apply -k \
~/DO380/labs/gitops-review/default
config.samples.operator.openshift.io/cluster configured
```

- 5.7. Make sure that the templates related to Ruby on Rails are back to your cluster before moving to the next step.

```
[student@workstation gitops-review]$ oc get template -n openshift | grep rails
rails-pgsql-persistent      An example Rails application ...
rails-postgresql-example    An example Rails application ...
```

6. Complete the Jenkinsfile for the Apply pipeline.

You already copied an incomplete `Jenkinsfile` named `Jenkinsfile-apply` to your Git repository.

Complete the `Jenkinsfile` to apply YAML resource files from the `samples` folder of your Git repository. Also, make sure that your pipeline only applies and tests changes for builds running from the `master` branch.

The pipeline also deletes resources using stub files in the `delete` folder of your Git repository, and verifies that no template for Ruby on Rails applications exists in the `openshift` namespace. Do not make changes to the steps that perform these actions.

If you are unsure about your changes to the `Jenkinsfile`, you can review the solution files in the `/home/student/D0380/solutions/gitops-review` folder.

6.1. Inspect the incomplete `Jenkinsfile`.

Open the `Jenkinsfile-apply` file in the `/home/student/gitops-review` folder with a text editor and verify that its pipeline contains three stages. You will change only the `Apply` and `Verify` stages, which are highlighted in the following listing.

The pipeline also contains a few directives. You will only change the `triggers` directive, as highlighted in the following listing. The following steps describe each change in detail.

```
pipeline {
    triggers {
        ...output omitted...
    options {
        ...output omitted...
    agent {
        ...output omitted...
    stages {
        stage ('Validate configuration resources') {
            ...output omitted...
        stage ('Apply resources') {
            ...output omitted...
        stage ('Verify deleted templates') {
            ...output omitted...
        }
    }
}
```

6.2. Configure your pipeline to poll its Git repository every five minutes.

Change your `triggers` directive as follows.

```
pipeline {
    triggers {
        pollSCM ('H/5 * * * *')
    }
    options {
        disableConcurrentBuilds()
    }
    agent {
        node {
            label 'nodejs'
        }
    }
}
```

```
stages {
    stage ('Validate configuration resources') {
        ...output omitted...
```

- 6.3. Complete your Apply stage to run from the `main` branch only, and to apply the configurations from the `samples` folder using Kustomize.

```
...output omitted...
stage ('Validate configuration resources') {
    ...output omitted...
stage ('Apply resources') {
    when {
        branch 'main'
    }
    steps {
        sh 'oc apply -k samples'
        // ignore delete errors caused by templates deleted by previous runs
        sh 'oc delete -f delete --wait || true'
    }
}
stage ('Verify deleted templates') {
    ...output omitted...
```

- 6.4. Complete your Verify stage to only run from the `main` branch.

```
...output omitted...
stage ('Apply resources') {
    ...output omitted...
stage ('Verify deleted templates') {
    when {
        branch 'main'
    }
    steps {
        sh '! oc get templates -n openshift | grep rails'
    }
}
}
```

- 6.5. Commit your `Jenkinsfile` and then push to GitHub.

```
[student@workstation gitops-review]$ git add Jenkinsfile-apply
[student@workstation gitops-review]$ git commit -m 'Completed Apply pipeline'
...output omitted...
create mode 100644 samples/Jenkinsfile-apply
...output omitted...
[student@workstation gitops-review]$ git push
...output omitted...
cc54e2b..2284328 main -> main
```

7. Create a Jenkins multibranch pipeline project named `apply` for your Apply pipeline. Configure your Jenkins project to take the `Jenkinsfile` from your GitHub repository.

After Jenkins successfully runs a build from your project, verify that the `openshift` namespace contains no template related to Ruby on Rails applications.

- 7.1. Create a new multibranch pipeline project for the Apply pipeline.

Click **New Item** on the Jenkins Dashboard page, and then type `apply` as the item name. Select **Multibranch Pipeline**, and then click **Ok**.

Under the **Branch Sources** heading, select **Add Source → Git** and type the HTTPS URL of your GitHub repository, `https://github.com/youraccount/gitops-review.git`, in the **Project Repository** field. Because your Git repository is public, you do not provide any credentials.

Under the **Build Configuration** heading, type `Jenkinsfile-apply` in the **Script Path** field.

Do not set any other fields and click **Save**.

- 7.2. Verify that Jenkins finds the main branch.

Jenkins automatically starts a **Scan Multibranch Pipeline** job and displays its logs. Wait until it finishes and verifies that the logs of the job indicate that Jenkins found a main branch.

- 7.3. Wait until Jenkins builds your main branch successfully.

Click either **Up** or **Jenkins** to go to the Jenkins Dashboard page. Click **apply** to enter the Apply project, then click **main** to list builds of the **main** branch of your Git repository.

It might take a few moments until a build starts and its stages are displayed. After the Jenkins build starts, wait until it finishes successfully.

- 7.4. Verify that your cluster now contains no standard templates related to Ruby on Rails.

```
[student@workstation gitops-review]$ oc get template -n openshift | grep rails
```

8. Generate a Jenkins API token.

Use the Jenkins web UI to access the configuration for the **admin** user and save the token to the `/home/student/jenkins-api-token.txt` file.

The `grade` command requires that token to invoke the Jenkins API and start a build of your Apply pipeline.

- 8.1. Switch to the open web browser displaying the Jenkins web UI and click **admin**, found near the **logout** button.
- 8.2. On the **admin** Jenkins user page, click **Configure** and then click **Add new Token**. Type a name for the token, such as '`grade`', and click **Generate**.

API Token	
Current token(s)	There are no registered tokens for this user.

Generate **Cancel**

Add new Token

- 8.3. Select and copy the long hexadecimal string, which is the value of the token.

The screenshot shows the Jenkins API Token configuration page. It displays a single token named "grade" with the value "11af45125bfbe5523865687b12ab3ca4f7". Below the token list, there is a yellow warning message: "⚠️ Copy this token now, because it cannot be recovered in the future." A "Copy" button is located next to the token value. There is also a "Delete" button and a help icon (question mark) in the top right corner.

- 8.4. Switch back to your terminal and create the /home/student/jenkins-api-token.txt file using a text editor. Paste the value of the token you got from the previous step into this file.
- 8.5. Change to the /home/student directory.

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

Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work.

Correct any reported failures and rerun the command until successfully.

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

Finish

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

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

The `lab` command deletes the OpenShift project you created and restores default cluster settings. It also deletes the clone of your `gitops-review` Git repository from your home folder and the Jenkins API token file.

If you wish, you can also delete the `gitops-review` repository from your GitHub account.

Summary

In this chapter, you learned:

- Jenkins is a popular software development automation server.
- Red Hat provides a set of Jenkins container images, which includes plug-ins for interacting with OpenShift and Kubernetes.
- OpenShift comes with a few standard templates that makes it easy to deploy Jenkins using images from Red Hat.
- CI/CD workflows are usually developer-centric, follow an imperative style, and make heavy use of the DSL of the OpenShift Client plug-in.
- GitOps workflows are usually system administrator-centric, follow an declarative style, and make use of YAML files and Kustomize.
- A GitOps workflow does two things:
 - Apply configuration changes from files under version control.
 - Detect when configuration drifts from the state under version control, then reconcile or notify.
- Declarative GitOps workflows for Kubernetes are based on the `oc apply` and `oc diff` commands.
- CI/CD and GitOps workflows are implemented with Jenkins as pipelines
- Jenkins pipelines are composed of stages and each stage performs a number of steps.
- A step can perform any task required by its workflow, for example, building a container image, setting RBAC permissions, and applying cluster operator configurations.

Chapter 5

Configuring Enterprise Authentication

Goal

Configure OpenShift integration with enterprise identity providers

Objectives

- Configure OpenShift to authenticate using an IdM user and credentials.
- Automate group synchronization with an LDAP directory.

Sections

- Configuring the LDAP Identity Provider (and Guided Exercise)
- Synchronizing OpenShift Groups with LDAP (and Guided Exercise)

Lab

Configuring OpenShift Enterprise Authentication

Configuring the LDAP Identity Provider

Objectives

After completing this section, you should be able to:

- Configure OpenShift to authenticate using an IdM user and credentials.
- Troubleshoot issues with IdM user login integration.

Enterprise Authentication

OpenShift clusters have two categories of users: normal users and service accounts. Normal users represent human operators and are assumed to be managed by an independent system, such as HTPasswd, LDAP or OpenStack Keystone. Service Accounts are managed by OpenShift and are usually associated to projects to perform specific tasks with different levels of authorization. When the OpenShift API receives a request with no authentication or an invalid authentication, this request is assumed to be made by the **anonymous** user.

Users can be associated to groups, which can also be managed by an external service.

Authentication Methods

OpenShift API requests use one of the following methods to authenticate:

- OAuth Access Tokens: the API reads a token from the HTTP headers to verify that the request is authenticated.
- X.509 Client Certificates: the API checks the certificate validity to verify the authenticity of the request.

OpenShift OAuth Server

The OpenShift OAuth server uses an identity provider to validate the identity of users and issue OAuth tokens so users can interact with the API.

Identity Providers

OpenShift supports the following Identity Providers:

- HTPasswd
- Keystone
- LDAP
- Basic authentication
- Request header
- GitHub or GitHub Enterprise
- GitLab
- Google
- OpenID Connect

Configuring the LDAP Identity Provider

To configure an LDAP server as an identity provider:

Create a secret with the IdM admin user password. This secret will be used for the LDAP identity provider:

```
[user@host ~]$ oc create secret generic ldap-secret \
--from-literal=bindPassword=${LDAP_ADMIN_PASSWORD} \
-n openshift-config
```

TLS communication needs certificate authority validation, in this case, the CA is obtained from the IdM, which is configured with the classroom.

Create a configuration map containing the IdM Certificate Authority's root certificate to make OpenShift trust the IdM certificates:

```
[user@host ~]$ oc create configmap ca-config-map \
--from-file=\
ca.crt=<(curl http://idm.ocp-$[GUID].example.com/ipa/config/ca.crt) \
-n openshift-config
```

Create a file to modify the OpenShift OAuth configuration by adding the LDAP identity provider and the other elements required for proper configuration.

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - name: ldapidp
      mappingMethod: claim
      type: LDAP
      ldap:
        attributes:
          id:
            - dn
          email:
            - mail
          name:
            - cn
        preferredUsername:
          - uid
      bindDN: "uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com"
      bindPassword:
        name: ldap-secret
      ca:
        name: ca-config-map
      insecure: false
      url: "ldaps://idm.ocp4.example.com/
cn=users,cn=accounts,dc=ocp4,dc=example,dc=com?uid"
```

**Note**

- The `url: "ldaps://...?uid"` fragment must be typed as a single line.

Apply the custom resource and wait until the pods in the `openshift-authentication` namespace are restarted so that the identity provider is active:

```
[user@host ~]$ oc apply -f tmp/ldap-cr.yml
```

Verify that you are able to log in with the IdM users. If further users are added to the IdM, then they will be immediately available and the OpenShift OAuth pods will not be restarted, unlike in the HTPasswd use case.

```
[user@host ~]$ oc login -u admin -p ${LDAP_ADMIN_PASSWORD}  
[user@host ~]$ oc whoami
```

To verify if there are any issues with the IdM integration, check the pod status and logs of the `deployment.apps/oauth-openshift` deployment in the `openshift-authentication` namespace.

```
[user@host ~]$ oc get pods -n openshift-authentication  
[user@host ~]$ oc logs deployment.apps/oauth-openshift
```

When users log in using the LDAP server as identity provider, OpenShift creates new user and identity entries.

```
[user@host ~]$ oc get user  
[user@host ~]$ oc get identity
```

**Note**

To perform a clean up, remove both the user and the identity. If the user is removed but not the identity, then you will not be able to log in with that user and will get 500 errors.

LDAP Users and Role-based Access Control

Role-based Access Control (RBAC) is a technique to assign privileges to users and groups in a project or cluster. The OpenShift RBAC implementation uses roles as descriptors of permissions, and role bindings to define the relationship between roles and users and groups.

Cluster administrators can use RBAC to delegate tasks or to give developers permission to perform administrative operations in their projects.

Add the `cluster-admin` role to users of the IdM that need cluster administration privileges.

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

Troubleshooting

Authentication troubleshooting can be a complex task; the things to check include:

- Authentication Operator Logs.
- Oauth Pods status.

- LDAP server logs.

The steps to troubleshoot are nearly identical to the steps taken while performing initial configuration as detailed earlier in this chapter.

The identity manager logs, status, and network communication with the OpenShift Cluster should be checked while troubleshooting.



References

- For more information, refer to the *Configuring an LDAP identity provider* section in the *Configuring identity providers* chapter, in the Red Hat OpenShift Container Platform 4.6 *Authentication and authorization* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/authentication_and_authorization/index#configuring-ldap-identity-provider

► Guided Exercise

Configuring the LDAP Identity Provider

In this exercise you will configure an LDAP Identity provider for OpenShift.

Outcomes

You should be able to:

- Add an LDAP Identity Provider (IdP) to the OAuth operator.
- Show logs of failed and successful log in attempts.

Before You Begin

To perform this exercise, ensure you have:

- A running OpenShift cluster.
- Administrative access to the OpenShift cluster.

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

This command ensures that the environment for this lab is properly setup.

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

Instructions

Allow users stored in an Identity Management (IdM) system to log in to the OpenShift Cluster.

► 1. Create the support resources for the identity provider configuration.

- 1.1. Log in to your OpenShift cluster as the **admin** user.

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

...output omitted...
```

- 1.2. Create a **Secret** with the LDAP bind password, which is required for identity provider configuration.

```
[student@workstation ~]$ oc create secret generic ldap-secret \
  --from-literal=bindPassword='Redhat123@!' \
  -n openshift-config
secret/ldap-secret created
```

- 1.3. Create a ConfigMap containing the IdM certificate to trust.

```
[student@workstation ~]$ curl -O http://idm.ocp4.example.com/ipa/config/ca.crt
...output omitted...
[student@workstation ~]$ oc create configmap -n openshift-config \
ca-config-map --from-file=ca.crt
configmap/ca-config-map created
```

- 2. Modify the OpenShift OAuth configuration.

- 2.1. Change to the ~/D0380/labs/auth-ldap/ directory.

```
[student@workstation ~]$ cd ~/D0380/labs/auth-ldap/
```

- 2.2. Edit the ldap-cr.yaml file and change the dn, mail, cn, uid, ldap-secret and ca-config-map values as shown.

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - htpasswd:
        fileData:
          name: htpasswd-secret
        mappingMethod: claim
        name: htpasswd_provider
        type: HTPasswd
    - name: ldapidp
        mappingMethod: claim
        type: LDAP
    - ldap:
        attributes:
          id:
            - dn
          email:
            - mail
          name:
            - cn
          preferredUsername:
            - uid
        bindDN: "uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com"
        bindPassword:
          name: ldap-secret
        ca:
          name: ca-config-map
        insecure: false
        url: "ldaps://idm.ocp4.example.com/
cn=users,cn=accounts,dc=ocp4,dc=example,dc=com?uid"
```

**Note**

- The url: "ldaps://...?uid" fragment must be typed as a single line.
- The `htpasswd_provider` identity provider defines the `admin` user that you use in exercises and labs. The `ldap-cr.yaml` file includes the `htpasswd_provider` identity provider so that the `admin` user is always available.

2.3. Apply the custom resource. You can safely ignore the warning that displays.

```
[student@workstation auth-ldap]$ oc apply -f ldap-cr.yaml
Warning: oc apply should be used on resource created by either oc create --save-config or oc apply
oauth.config.openshift.io/cluster configured
```

2.4. Wait until the `oauth-openshift` pods in the `openshift-authentication` namespace finish redeploying. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation auth-ldap]$ watch oc get pods -n openshift-authentication
NAME                  READY   STATUS    RESTARTS   AGE
oauth-openshift-6d57946ff7-757k8   1/1     Running   0          45s
oauth-openshift-6d57946ff7-fxcbg   1/1     Running   0          39s
```

▶ 3. Change to the `/home/student/` directory.

```
[student@workstation auth-ldap]$ cd
```

▶ 4. To verify the configuration, log in as the LDAP user `openshift-user` and the password `openshift-user`.

```
[student@workstation ~]$ oc login -u openshift-user -p openshift-user
```

Finish

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

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

Synchronizing OpenShift Groups with LDAP

Objectives

After completing this section, you should be able to:

- Automate group synchronization with an LDAP directory.
- Troubleshoot group synchronization issues.

Synchronizing LDAP Groups

Groups are used to manage user permissions in bulk.

OpenShift can synchronize LDAP groups by querying the LDAP server and creating them internally. This process should be run periodically to ensure groups are kept up to date. OpenShift administrators can grant permissions based on LDAP groups, but these permissions must be assigned using the OpenShift RBAC system and are not inherited from the IdM.

The LDAPSsyncConfig resource contains the settings that the OpenShift cluster needs for group synchronization, which can be performed manually or using a CronJob.

Configuring LDAP Group Synchronization

To create an LDAPSsyncConfig resource and test it, complete the following two steps.

First, create a file containing an LDAPSsyncConfig resource definition. The CronJob resource uses a Secret and a ConfigMap resources instead of embedding the password and a local path.

```
kind: LDAPSsyncConfig
apiVersion: v1
url: ldaps://idm.ocp4.example.com
bindDN: uid=ldap_user_for_sync,cn=users,cn=accounts,dc=example,dc=com
bindPassword: ldap_user_for_sync_password
insecure: false
ca: /path/to/ca.crt
rfc2307:
  groupsQuery:
    baseDN: "cn=groups,cn=accounts,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
    filter: (objectClass=posixgroup)
  groupUIDAttribute: dn
  groupNameAttributes: [ cn ]
  groupMembershipAttributes: [ member ]
  usersQuery:
    baseDN: "cn=accounts,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
```

```
userUIDAttribute: dn
userNameAttributes: [ cn ]
tolerateMemberNotFoundErrors: false
tolerateMemberOutOfScopeErrors: true
```

Execute a dry run to verify the synchronization process.

```
[user@host ~]$ oc adm groups sync --sync-config tmp/ldap-sync.yml
apiVersion: v1
items:
- metadata:
  annotations:
    openshift.io/ldap.sync-time: "2020-04-14T09:26:29Z"
    openshift.io/ldap.uid: cn=admins,cn=groups,cn=accounts,dc=example,dc=com
    openshift.io/ldap.url: idm.ocp4.example.com:636
  creationTimestamp: "2020-03-30T14:15:49Z"
  labels:
    openshift.io/ldap.host: idm.ocp4.example.com
  name: admins
  resourceVersion: "2752973"
  selfLink: /apis/user.openshift.io/v1/groups/admins
  uid: f4bd4e3a-7290-11ea-b166-0a580a80001b
users:
- Administrator
...
...
```

In case of failure, the synchronization process displays errors that are similar to the following:

```
[user@host ~]$ oc adm groups sync --sync-config tmp/ldap-sync.yml
...
error: could not bind to the LDAP server: LDAP Result Code 49 "Invalid
Credentials":
```

Scheduling LDAP Group Synchronization

The following steps detail how to create a CronJob to perform a periodic LDAP group synchronization.

- Store the LDAP bind password in an OpenShift Secret so the CronJob can access the password in a secure way.
- Store the LDAPSConfig and the IdM certificate in a ConfigMap so the CronJob can use them.

Create a new LDAPSConfig.

```
kind: LDAPSConfig
apiVersion: v1
url: ldaps://idm.ocp4.example.com
bindDN: uid=admin,cn=users,cn=accounts,dc=example,dc=com
bindPassword:
  file: /etc/secrets/bindPassword
  insecure: false
ca: /etc/config/ca.crt
```

```
rfc2307:  
  groupsQuery:  
    baseDN: "cn=groups,cn=accounts,dc=example,dc=com"  
    scope: sub  
    derefAliases: never  
    pageSize: 0  
    filter: (objectClass=ipausergroup)  
  groupUIDAttribute: dn  
  groupNameAttributes: [ cn ]  
  groupMembershipAttributes: [ member ]  
  usersQuery:  
    baseDN: "cn=users,cn=accounts,dc=example,dc=com"  
    scope: sub  
    derefAliases: never  
    pageSize: 0  
  userUIDAttribute: dn  
  userNameAttributes: [ uid ]
```

Create a Secret.

```
[user@host ~]$ oc create secret generic ldap-secret \  
--from-literal bindPassword=r3dh4t
```

Create a ConfigMap.

```
[user@host ~]$ oc create configmap ldap-config --from-file \  
ldap-group-sync.yaml=tmp/ldap-sync-config-cronjob.yml,ca.crt=tmp/ca.crt
```

Create a CronJob.

```
apiVersion: batch/v1beta1  
kind: CronJob  
metadata:  
  name: group-sync  
spec:  
  schedule: "*/1 * * * *" ①  
  jobTemplate:  
    spec:  
      template:  
        spec:  
          restartPolicy: Never  
          containers:  
            - name: group-sync  
              image: registry.redhat.io/openshift4/ose-cli:v4.5 ②  
              command:  
                - /bin/sh  
                - -c  
                - oc adm groups sync --sync-config /etc/config/ldap-group-sync.yaml --  
confirm ③  
          volumeMounts: ④  
            - mountPath: "/etc/config"  
              name: "ldap-sync-volume"  
            - mountPath: "/etc/secrets"
```

```

        name: "ldap-bind-password"
volumes:
  - name: "ldap-sync-volume"
    configMap:
      name: ldap-config
  - name: "ldap-bind-password"
    secret:
      secretName: ldap-secret
serviceAccountName: ldap-group-syncer-sa ⑤
serviceAccount: ldap-group-syncer-sa

```

- ① Schedule the job to run every minute to have shorter test cycles.
- ② Provide an OCI image to run the CronJob. This image must contain the oc command.
- ③ Execute the sync command with the --confirm option in the CronJob.
- ④ Provide the Secret and ConfigMap to the CronJob.
- ⑤ Provide a ServiceAccount that can execute get, list, create, update verbs on the groups resource.

**Note**

The `- oc adm groups sync ... --confirm` fragment must be typed as a single line.

Verify LDAP Group Synchronization

Use the `watch` command to inspect the CronJob execution.

```
[user@host ~]$ watch oc get cronjobs,jobs,pods
NAME          SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
cronjob.batch/group-sync  */1 * * * *   False     0        33s           2m5s

NAME          COMPLETIONS   DURATION   AGE
job.batch/group-sync-1597933740  1/1        4s         84s
job.batch/group-sync-1597933800  1/1        4s         24s

NAME          READY   STATUS      RESTARTS   AGE
pod/group-sync-1597933740-4d5t8  0/1    Completed   0          84s
pod/group-sync-1597933800-jlwgl  0/1    Completed   0          24s
```

Check the pod logs.

```
[user@host ~]$ oc logs pod/group-sync-1586857680-6969r
group/admins
...
```

Review LDAP Users Permissions and Group Membership

Verify a group definition and membership.

```
[user@host ~]$ oc get group admins -o yaml
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: "2020-04-14T09:50:45Z"
    openshift.io/ldap.uid: cn=admins,cn=groups,cn=accounts,dc=ocp-example,dc=do380,dc=dev,dc=nextcle,dc=com
    openshift.io/ldap.url: idm.ocp4.example.com:636
  creationTimestamp: "2020-03-30T14:15:49Z"
  labels:
    openshift.io/ldap.host: idm.ocp4.example.com
  name: admins
  resourceVersion: "2802446"
  selfLink: /apis/user.openshift.io/v1/groups/admins
  uid: f4bd4e3a-7290-11ea-b166-0a580a80001b
users:
- admin
```

Granting Permissions to IdM User Groups

OpenShift does not inherit permissions from the LDAP server; permissions must be added to each group.

Roles can be added to LDAP groups in the same way as regular OpenShift groups after the group synchronization is performed.

```
[user@host ~]$ oc adm policy add-cluster-role-to-group cluster-admin admins
```



References

- For more information, refer to the *Syncing LDAP groups* chapter in the Red Hat OpenShift Container Platform 4.6 *Authentication* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/authentication_and_authorization/ldap-syncing
- For information about Active Directory nested groups, refer to the *LDAP nested membership sync example* section in the *Syncing LDAP groups* chapter, which is in the Red Hat OpenShift Container Platform 4.6 *Authentication* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/authentication_and_authorization/ldap-syncing#ldap-syncing-active-dir_ldap-syncing-groups

► Guided Exercise

Synchronizing OpenShift Groups with LDAP

In this exercise you will configure OpenShift for LDAP group synchronization.

Outcomes

You should be able to:

- Create a CronJob to perform the synchronization.
- Use the `oc policy who-can` command to list users and groups who can perform specific actions on specific resources according to a scenario.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that the environment for this lab is properly setup.

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

Instructions

- 1. Perform a dry run of the group synchronization process.

- 1.1. Log into your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Change to the `~/D0380/labs/auth-ldapsync/` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/auth-ldapsync/
```

- 1.3. Fetch the IdM certificate to trust in the synchronization process.

```
[student@workstation auth-ldapsync]$ curl -O \
http://idm.ocp4.example.com/ipa/config/ca.crt
...output omitted...
2020-09-09 09:51:47 (300 MB/s) - 'ca.crt' saved [1655/1655]
```

14. Edit the `ldap-sync.yml` file and change the `bindPassword` and `ca` values as shown.

```
kind: LDAPSConfig
apiVersion: v1
url: ldaps://idm.ocp4.example.com
bindDN: uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com
bindPassword: Redhat123@!
insecure: false
ca: ca.crt
rfc2307:
...output omitted...
```

15. Run the synchronization without performing any action.

```
[student@workstation auth-ldapsync]$ oc adm groups sync \
--sync-config ldap-sync.yml
apiVersion: v1
items:
- metadata:
  annotations:
    openshift.io/ldap.sync-time: 2020-07-07T15:15:34-0400
    openshift.io/ldap.uid:
  cn=admins,cn=groups,cn=accounts,dc=ocp4,dc=example,dc=com
    openshift.io/ldap.url: idm.ocp4.example.com:636
  creationTimestamp: null
  labels:
    openshift.io/ldap.host: idm.ocp4.example.com
  name: admins
  users:
  - admin
...output omitted...
```

- 2. Create the `auth-ldapsync` project.

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

- 3. Create a service account with suitable minimal permissions to manage groups.

- 3.1. Review the file named `rbac.yml`, which contains an `ldap-group-syncer` service account, a cluster role with write access to groups, and a cluster role binding.
- 3.2. Deploy the OpenShift resources.

```
[student@workstation auth-ldapsync]$ oc create -f rbac.yml
serviceaccount/ldap-group-syncer created
clusterrole.rbac.authorization.k8s.io/ldap-group-syncer created
clusterrolebinding.rbac.authorization.k8s.io/ldap-group-syncer created
```

► 4. Create the configuration for the CronJob.

4.1. Modify the `cronjob.yml` file to match the following.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: group-sync
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
            - name: hello
              image: registry.redhat.io/openshift4/ose-cli:v4.5
              command:
                - /bin/sh
                - -c
                - - oc adm groups sync --sync-config /etc/config/cron-ldap-sync.yml --
confirm
          volumeMounts:
            - mountPath: "/etc/config"
              name: "ldap-sync-volume"
            - mountPath: "/etc/secrets"
              name: "ldap-bind-password"
          volumes:
            - name: "ldap-sync-volume"
              configMap:
                name: ldap-config
            - name: "ldap-bind-password"
              secret:
                secretName: ldap-secret
  serviceAccountName: ldap-group-syncer
  serviceAccount: ldap-group-syncer
```



Note

- The `- oc adm groups sync ... --confirm` fragment must be typed as a single line.

4.2. Copy `ldap-sync.yml` to a new file named `cron-ldap-sync.yml`. Edit the `cron-ldap-sync.yml` file and change the `bindPassword` and `ca` values as shown.

```
kind: LDAPSNCConfig
apiVersion: v1
url: ldaps://idm.ocp4.example.com
bindDN: uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com
bindPassword:
  file: /etc/secrets/bindPassword
insecure: false
ca: /etc/config/ca.crt
rfc2307:
  ...output omitted...
```

- 4.3. Create a ConfigMap containing the LDAPSNCConfig and the trusted certificate.

```
[student@workstation auth-ldapsync]$ oc create configmap ldap-config \
--from-file cron-ldap-sync.yml=cron-ldap-sync.yml,ca.crt=ca.crt
configmap/ldap-config created
```

- 4.4. Create a Secret containing the LDAP bind password.

```
[student@workstation auth-ldapsync]$ oc create secret generic ldap-secret \
--from-literal bindPassword='Redhat123@!'
secret/ldap-secret created
```

► 5. Create a CronJob to automatically run the audit script on a schedule.

- 5.1. Create the CronJob.

```
[student@workstation auth-ldapsync]$ oc create -f cronjob.yml
cronjob.batch/group-sync created
```

- 5.2. Watch the CronJob, Job, and Pod resources until the group synchronization completes. Press **Ctrl+C** to exit the `watch` command.

```
[student@workstation auth-ldapsync]$ watch oc get cronjobs,jobs,pods
NAME          SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
cronjob.batch/group-sync  */1 * * * *   False     0        35s           50s

NAME          COMPLETIONS   DURATION   AGE
job.batch/group-sync-1595239320  1/1       16s        28s

NAME          READY   STATUS    RESTARTS   AGE
pod/group-sync-1595239320-zl7h6  0/1    Completed   0        28s
```

- 5.3. Review that the user `openshift-admin` is in the `openshift-admins` group.

```
[student@workstation auth-ldapsync]$ oc get group openshift-admins
NAME          USERS
openshift-admins  openshift-admin
```

- 5.4. Change to the `/home/student/` directory.

```
[student@workstation auth-ldapsync]$ cd
```

- 6. Temporarily, give the LDAP group `openshift-admins` cluster administrators privileges.

- 6.1. Note that the `openshift-admin` user cannot view the OAuth cluster resource.

```
[student@workstation ~]$ oc policy who-can get oauth
resourceaccessreviewresponse.authorization.openshift.io/<unknown>

Namespace: auth-ldapsync
Verb:       get
Resource:   oauths.config.openshift.io

Users:     admin
          system:admin
...output omitted...
Groups:   system:cluster-admins
          system:cluster-readers
          system:masters
```

- 6.2. Grant the permissions.

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

- 6.3. Note that `openshift-admin` can view the OAuth cluster resource.

```
[student@workstation ~]$ oc policy who-can get oauth
...output omitted...
Groups: openshift-admins
...output omitted...
```

- 6.4. Revoke the permissions.

```
[student@workstation ~]$ oc adm policy remove-cluster-role-from-group \
  cluster-admin openshift-admins
clusterrole.rbac.authorization.k8s.io/cluster-admin removed: "openshift-admins"
```

- 7. Delete the `auth-ldapsync` project.

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

Finish

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

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

► Lab

Configuring OpenShift Enterprise Authentication

In this lab, you will configure LDAP authentication for the OpenShift Container Platform.

Outcomes

You should be able to:

- Configure authentication using LDAP.
- Synchronize groups using LDAP.
- Grant a cluster role to a group synchronized from LDAP.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the `workstation` machine, use the `lab` command to prepare your system for this exercise.

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

Instructions

- Create an `ldap-secret` secret in the `openshift-config` namespace that contains the LDAP bind password `Redhat123@!` in the `bindPassword` key.
- Create a `ca-config-map` configuration map in the `openshift-config` namespace that contains the IdM certificate in the `ca.crt` key. The IdM certificate is at `http://idm.ocp4.example.com/ipa/config/ca.crt`.
- Add an LDAP identity provider to the OpenShift OAuth configuration. You can use the `~/D0380/labs/auth-review/ldap-cr.yaml` file as a template.
Configure the LDAP identity provider to:
 - Bind to the LDAP identity provider as
`"uid=admin, cn=users, cn=accounts, dc=ocp4, dc=example, dc=com"`.
 - Use `"ldaps://idm.ocp4.example.com/cn=users, cn=accounts, dc=ocp4, dc=example, dc=com?uid"` as the LDAP URL.
 - Use the `uid` LDAP attribute as the preferred user name.
 - Use the `dn` LDAP attribute as the id.
- Verify that you can log in as the `openshift-user` user with the `openshift-user` password.

5. Create an `ldap-group-syncer` cluster role with permissions to get, list, create, and update groups. Create a new `auth-review` project. Create a `ldap-group-syncer` service account. Bind the `ldap-group-syncer` cluster role to the `ldap-group-syncer` service account.
6. Create a cron job that runs the `oc adm groups sync` command every minute as the `ldap-group-syncer` service account. The cron job uses a configuration map that contains an LDAP group synchronization configuration file and the IdM certificate. The LDAP group synchronization configuration uses a secret containing the LDAP bind password.
You can use the `~/D0380/labs/auth-review/cronjob.yml` file as a template for the cron job. You can use the `~/D0380/labs/auth-review/cron-ldap-sync.yml` file as a template for the LDAP group synchronization configuration file.
 - Use `ldaps://idm.ocp4.example.com/` as the IdM LDAP URL.
 - Bind to the LDAP identity provider as "`uid=admin, cn=users, cn=accounts, dc=ocp4, dc=example, dc=com`" using the `Redhat123@!` password.
 - Use the RFC 2307 schema for synchronization.
 - Use `cn=groups, cn=accounts, dc=ocp4, dc=example, dc=com` as the base DN for the groups query.
 - Use `(objectClass=ipausergroup)` for the groups query filter.
 - Use the `dn` LDAP attribute as the group UID attribute.
 - Use the `cn` LDAP attribute as the group name attribute.
 - Use the `member` LDAP attribute as the group membership attribute.
 - Use `cn=users, cn=accounts, dc=ocp4, dc=example, dc=com` as the base DN for the users query.
 - Use the `dn` LDAP attribute as the user UID attribute.
 - Use the `uid` LDAP attribute as the user name attribute.

7. Verify that the `openshift-admins` group exists, and that the `openshift-admins` group contains the `openshift-admin` user.

8. Grant the `cluster-admin` cluster role to the `openshift-admins` group.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

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

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

► Solution

Configuring OpenShift Enterprise Authentication

In this lab, you will configure LDAP authentication for the OpenShift Container Platform.

Outcomes

You should be able to:

- Configure authentication using LDAP.
- Synchronize groups using LDAP.
- Grant a cluster role to a group synchronized from LDAP.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the workstation machine, use the `lab` command to prepare your system for this exercise.

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

Instructions

1. Create an `ldap-secret` secret in the `openshift-config` namespace that contains the LDAP bind password `Redhat123@!` in the `bindPassword` key.
 - 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Create a secret with the LDAP bind password, which is required for the identity provider configuration.

```
[student@workstation ~]$ oc create secret generic ldap-secret \
--from-literal=bindPassword='Redhat123@!' \
-n openshift-config
secret/ldap-secret created
```

2. Create a `ca-config-map` configuration map in the `openshift-config` namespace that contains the IdM certificate in the `ca.crt` key. The IdM certificate is at `http://idm.ocp4.example.com/ipa/config/ca.crt`.

```
[student@workstation ~]$ curl -O http://idm.ocp4.example.com/ipa/config/ca.crt  
...output omitted...  
[student@workstation ~]$ oc create configmap -n openshift-config \  
ca-config-map --from-file=ca.crt  
configmap/ca-config-map created
```

3. Add an LDAP identity provider to the OpenShift OAuth configuration. You can use the `~/D0380/labs/auth-review/ldap-cr.yaml` file as a template.

Configure the LDAP identity provider to:

- Bind to the LDAP identity provider as `"uid=admin, cn=users, cn=accounts, dc=ocp4, dc=example, dc=com"`.
- Use `"ldaps://idm.ocp4.example.com/`
`cn=users, cn=accounts, dc=ocp4, dc=example, dc=com?uid"` as the LDAP URL.
- Use the `uid` LDAP attribute as the preferred user name.
- Use the `dn` LDAP attribute as the id.

3.1. Change to the `~/D0380/labs/auth-review` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/auth-review
```

3.2. Edit the `ldap-cr.yaml` file. Ensure the file matches the following:

```
apiVersion: config.openshift.io/v1  
kind: OAuth  
metadata:  
  name: cluster  
spec:  
  identityProviders:  
    - htpasswd:  
        fileData:  
          name: htpasswd-secret  
        mappingMethod: claim  
        name: htpasswd_provider  
        type: HTPasswd  
    - name: ldapidp  
        mappingMethod: claim  
        type: LDAP  
        ldap:  
          attributes:  
            id:  
              - dn  
            preferredUsername:  
              - uid  
          bindDN: "uid=admin, cn=users, cn=accounts, dc=ocp4, dc=example, dc=com"  
          bindPassword:  
            name: ldap-secret
```

```
ca:  
  name: ca-config-map  
  insecure: false  
  url: "ldaps://idm.ocp4.example.com/  
cn=users,cn=accounts,dc=ocp4,dc=example,dc=com?uid"
```

**Note**

- The `url: "ldaps://...?uid"` fragment must be typed as a single line.
- The `htpasswd_provider` identity provider defines the `admin` user that you use in exercises and labs. The `ldap-cr.yml` file includes the `htpasswd_provider` identity provider so that the `admin` user is always available.

3.3. Apply the custom resource. You can safely ignore the warning that displays.

```
[student@workstation auth-review]$ oc apply -f ldap-cr.yml  
Warning: oc apply should be used on resource created by either oc create --save-  
config or oc apply  
oauth.config.openshift.io/cluster configured
```

4. Verify that you can log in as the `openshift-user` user with the `openshift-user` password.

4.1. Wait until the `oauth-openshift` pods in the `openshift-authentication` namespace finish redeploying. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation auth-review]$ watch oc get pod -n openshift-authentication
```

4.2. Log in as the `openshift-user` user with the `openshift-user` password.

```
[student@workstation auth-review]$ oc login -u openshift-user -p openshift-user  
Login successful.  
...output omitted...
```

4.3. Log in as the `admin` user with the `redhat` password.

```
[student@workstation auth-review]$ oc login -u admin -p redhat  
Login successful.  
...output omitted...
```

5. Create an `ldap-group-syncer` cluster role with permissions to get, list, create, and update groups. Create a new `auth-review` project. Create a `ldap-group-syncer` service account. Bind the `ldap-group-syncer` cluster role to the `ldap-group-syncer` service account.

5.1. Create an `ldap-group-syncer` cluster role with permissions to get, list, create, and update groups.

```
[student@workstation auth-review]$ oc create clusterrole ldap-group-syncer \  
--resource=groups --verb=get,list,create,update  
clusterrole.rbac.authorization.k8s.io/ldap-group-syncer created
```

5.2. Create a new auth-review project.

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

5.3. Create the ldap-group-syncer service account and grant the permissions.

```
[student@workstation auth-review]$ oc create serviceaccount ldap-group-syncer
serviceaccount/ldap-group-syncer created

[student@workstation auth-review]$ oc create clusterrolebinding \
    ldap-group-syncer --clusterrole=ldap-group-syncer \
    --serviceaccount=auth-review:ldap-group-syncer
clusterrolebinding.rbac.authorization.k8s.io/ldap-group-syncer created
```

6. Create a cron job that runs the `oc adm groups sync` command every minute as the `ldap-group-syncer` service account. The cron job uses a configuration map that contains an LDAP group synchronization configuration file and the IdM certificate. The LDAP group synchronization configuration uses a secret containing the LDAP bind password.

You can use the `~/D0380/labs/auth-review/cronjob.yml` file as a template for the cron job. You can use the `~/D0380/labs/auth-review/cron-ldap-sync.yml` file as a template for the LDAP group synchronization configuration file.

- Use `ldaps://idm.ocp4.example.com/` as the IdM LDAP URL.
- Bind to the LDAP identity provider as "`uid=admin, cn=users, cn=accounts, dc=ocp4, dc=example, dc=com`" using the `Redhat123@!` password.
- Use the RFC 2307 schema for synchronization.
- Use `cn=groups, cn=accounts, dc=ocp4, dc=example, dc=com` as the base DN for the groups query.
- Use `(objectClass=ipausergroup)` for the groups query filter.
- Use the `dn` LDAP attribute as the group UID attribute.
- Use the `cn` LDAP attribute as the group name attribute.
- Use the `member` LDAP attribute as the group membership attribute.
- Use `cn=users, cn=accounts, dc=ocp4, dc=example, dc=com` as the base DN for the users query.
- Use the `dn` LDAP attribute as the user UID attribute.
- Use the `uid` LDAP attribute as the user name attribute.

6.1. Modify the `cronjob.yml` file to match the following.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: group-sync
```

```

namespace: auth-review
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
            - name: hello
              image: registry.redhat.io/openshift4/ose-cli:v4.5
              command:
                - /bin/sh
                - -c
                - - oc adm groups sync --sync-config /etc/config/ldap-group-sync.yaml --confirm
              volumeMounts:
                - mountPath: "/etc/config"
                  name: "ldap-sync-volume"
                - mountPath: "/etc/secrets"
                  name: "ldap-bind-password"
              volumes:
                - name: "ldap-sync-volume"
                  configMap:
                    name: ldap-config
                - name: "ldap-bind-password"
                  secret:
                    secretName: ldap-secret
              serviceAccountName: ldap-group-syncer
              serviceAccount: ldap-group-syncer

```

**Note**

The `- oc adm groups sync ... --confirm` fragment must be typed as a single line.

- 6.2. Edit the `cron-ldap-sync.yaml` file and change the `bindPassword` and `ca` values as shown.

```

kind: LDAPSsyncConfig
apiVersion: v1
url: ldaps://idm.ocp4.example.com
bindDN: uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com
bindPassword:
  file: /etc/secrets/bindPassword
insecure: false
ca: /etc/config/ca.crt
rfc2307:
  ...output omitted...

```

- 6.3. Create a configuration map containing the `LDAPSsyncConfig` and the trusted certificate.

```
[student@workstation auth-review]$ curl -O \
  http://idm.ocp4.example.com/ipa/config/ca.crt
...output omitted...

[student@workstation auth-review]$ oc create configmap \
  -n auth-review ldap-config \
  --from-file ldap-group-sync.yaml=cron-ldap-sync.yml,ca.crt=ca.crt
configmap/ldap-config created
```

- 6.4. Create a secret containing the LDAP bind password.

```
[student@workstation auth-review]$ oc create secret generic \
  -n auth-review ldap-secret \
  --from-literal bindPassword='Redhat123@!'
secret/ldap-secret created
```

- 6.5. Create a cron job that runs the `oc adm groups sync` command every minute as the `ldap-group-syncer` service account, using the configuration map from the previous step.

```
[student@workstation auth-review]$ oc create -f cronjob.yaml
cronjob.batch/group-sync created
```

7. Verify that the `openshift-admins` group exists, and that the `openshift-admins` group contains the `openshift-admin` user.

- 7.1. Wait for the cron job to complete at least one execution. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation auth-review]$ watch oc get -n auth-review cronjob,job,pod
```

- 7.2. Verify that the `openshift-admins` group exists. Verify that the `openshift-admins` group contains the `openshift-admin` user.

```
[student@workstation auth-review]$ oc get group
NAME          USERS
...output omitted...
openshift-admins    openshift-admin
...output omitted...
```

8. Grant the `cluster-admin` cluster role to the `openshift-admins` group.

- 8.1. Grant the permissions.

```
[student@workstation auth-review]$ oc adm policy add-cluster-role-to-group \
  cluster-admin openshift-admins
clusterrole.rbac.authorization.k8s.io/cluster-admin added: "openshift-admins"
```

- 8.2. Change to the `/home/student` directory.

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

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

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

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

Summary

In this chapter, you learned:

- Enterprise Identity Management Configuration for OpenShift.
- OpenShift Roles, Identities, groups and their relations to Enterprise Identity Management.
- Automatization for Identity Management.

Chapter 6

Configuring Trusted TLS Certificates

Goal

Configure trusted TLS certificates for external access to cluster services and applications.

Objectives

- Change the wildcard certificate used by the ingress controller operator.
- Configure an application to use the trusted certificate bundle.
- Describe common issues with OpenShift certificates.
- Integrating OpenShift with an Enterprise Certificate Authority (and Guided Exercise)
- Configuring Applications to Trust the Enterprise Certificate Authority (and Guided Exercise)
- Troubleshooting OpenShift Certificates (and Guided Exercise)

Sections

Configuring Trusted TLS Certificates

Lab

Integrating OpenShift with an Enterprise Certificate Authority

Objectives

After completing this section, you should be able to:

- Change the wildcard certificate used by the ingress controller operator.
- Replace the master API certificate.
- Create an edge route using a wildcard certificate.

Describing the Default Wildcard Certificate

The Red Hat OpenShift Container Platform installer creates an internal certificate authority (CA) and uses this CA to sign additional certificates.

The certificate used by the ingress controller operator is a wildcard certificate for all routes in the .apps subdomain for your cluster, such as .apps.ocp4.example.com. Routes for the web console, Grafana, Prometheus, and OAuth use this same wildcard certificate.

When using the wildcard certificate signed by the internal OpenShift CA, the web console prompts users with a warning indicating that the connection is not secure.

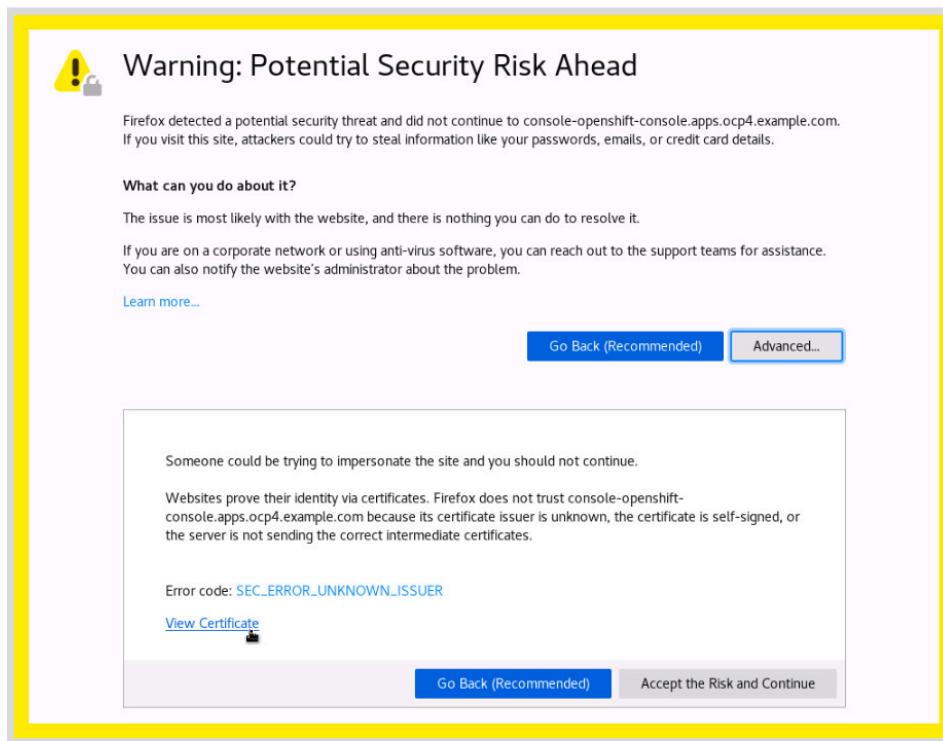


Figure 6.1: Certificate Warning

The View Certificate link provides details about the certificate.

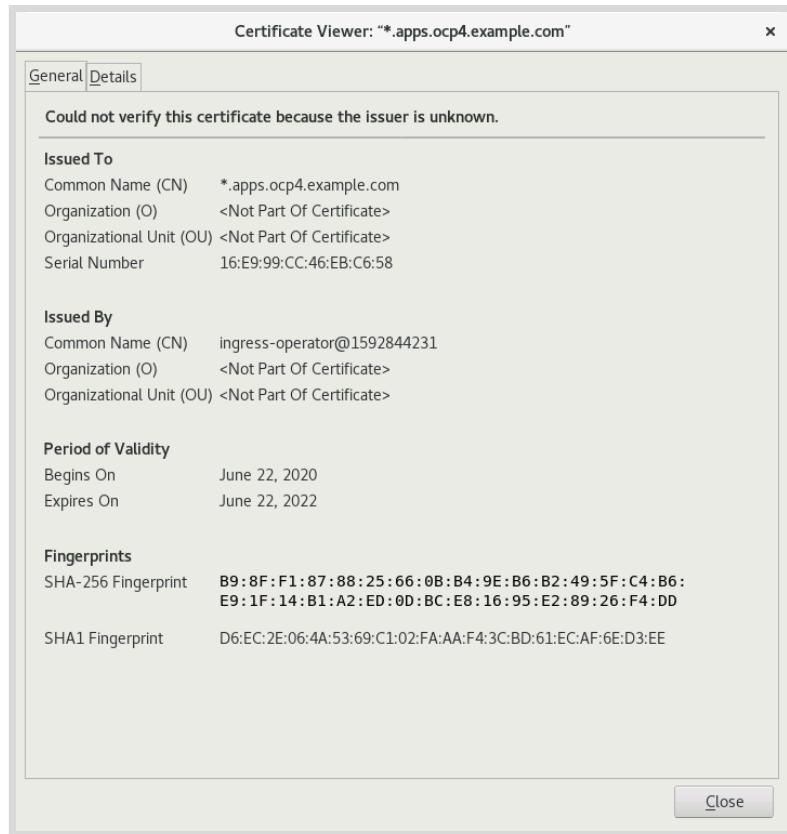


Figure 6.2: Default Ingress Certificate

If OpenShift certificates are not signed by a recognized certificate authority, then users attempting to access the cluster must add one or more exceptions to enable using the certificates.

Although using these exceptions might be acceptable for internal use, this solution is frequently insufficient for client-facing URLs.

Companies can use a certificate signed by a recognized certificate authority, such as Let's Encrypt, or use a certificate signed by their own enterprise certificate authority.



Note

The only clients that implicitly trust certificates signed by the internal OpenShift certificate authority are other components within the OpenShift cluster. Therefore, these certificates should not be replaced.

Changing the Ingress Controller Operator Certificate

The ingress operator configures the ingress controller to route traffic into the OpenShift environment. The certificate used by the ingress controller can be updated so that it uses a certificate signed by a recognized certificate authority, or by your own enterprise CA. Changing the ingress controller operator to use a different certificate and its associated key only requires a handful of steps. Before starting this process, you need:

- The new certificate and key in PEM format.

- The certificate must have a `subjectAltName` extension of `*.apps.<OPENSIGHT-DOMAIN>`, such as `*.apps.ocp4.example.com`, that enables using the certificate as a wildcard certificate for the `.apps` subdomain.



Important

Changing the certificate used by the ingress controller operator does not affect certificates signed by the internal OpenShift certificate authority.

To begin, create a new configuration map in the `openshift-config` namespace.

Prefix the file path with `ca-bundle.crt=` to name the data key in the configuration map as `ca-bundle.crt`. This configuration map can contain one or more certificates. For example, you can combine the new wildcard certificate and the certificate authority used to sign the wildcard certificate into one file. Add these certificates to the certificate bundle if additional certificates are needed to proxy out of your cluster.

Although it may not seem intuitive, some OpenShift components communicate with each other using external-facing URLs.

Adding your certificates to the cluster proxy ensures that your web console pods can trust the authentication pods and vice versa.

This step is not needed if the certificate is signed by a certificate authority that already exists in the Red Hat CoreOS (RHCOS) trust bundle.

```
[user@host ~]$ oc create configmap <CONFIGMAP-NAME> \
--from-file ca-bundle.crt=<PATH-TO-CERTIFICATE> \
-n openshift-config
```

Configure the cluster proxy to use the new configuration map. This step injects the certificate information contained in your configuration map into other configuration maps labeled with `config.openshift.io/inject-trusted-cabundle=true`.

As with the preceding step, this step is not needed if the certificate is signed by a certificate authority that already exists in the RHCOS trust bundle.

There are several ways to modify the cluster proxy, such as using `oc edit`, `oc patch`, or modifying a configuration file under version control and then using `oc apply`.

Additional changes to the cluster proxy can support a custom PKI infrastructure.

```
[user@host ~]$ oc patch proxy/cluster --type=merge \
--patch='{"spec":{"trustedCA":{"name":"<CONFIGMAP-NAME>"}}'
```



Important

If you fail to perform the preceding two steps when using a certificate signed by your enterprise CA, then you will be unable to log into the web console.

Create a new TLS secret in the `openshift-ingress` namespace using the new certificate and its corresponding key. The OpenShift ingress operator uses this secret.

```
[user@host ~]$ oc create secret tls <SECRET-NAME> \
--cert <PATH-TO-CERTIFICATE> \
--key <PATH-TO-KEY> \
-n openshift-ingress
```

Modify the default ingress controller operator in the `openshift-ingress-operator` namespace so that `defaultCertificate` uses the newly created secret.

```
[user@host ~]$ oc patch ingresscontroller.operator/default \
-n openshift-ingress-operator --type=merge \
--patch='{"spec": {"defaultCertificate": {"name": "<SECRET-NAME>"}}}'
```

If the change is successful, then new router pods in the `openshift-ingress` namespace deploy and change to a running status.

```
[user@host ~]$ watch oc get pods -n openshift-ingress
```

Replacing the Master API Certificate

The OpenShift master API uses a different certificate than the certificate used by the ingress controller. Changing the master API certificate allows users to log in securely using the `oc` command. As with the ingress controller certificate, a certificate signed by a recognized certificate authority or by your company enterprise CA can replace the master API certificate.

To change the master API certificate, you need:

- The master API certificate and key in PEM format.
- The certificate is issued to the URL used to access the master API, such as `api.ocp4.example.com`.
- The `subjectAltName` extension for the certificate contains the URL used to access the master API, such as `DNS:api.ocp4.example.com`.
- If the certificate is signed by your enterprise CA, then you can create a combined certificate by concatenating the master API certificate and the CA certificate into one PEM file. Concatenate additional certificates into the combined PEM file as necessary to establish a chain of trust.

```
[user@host ~]$ cat WILDCARD.pem CA.pem > COMBINED-CERT.pem
```

Change the master API certificate with the following steps. Create a new TLS secret in the `openshift-config` namespace using the master API certificate and key. Use the combined PEM certificate file for the `--cert` option if you created one.

```
[user@host ~]$ oc create secret tls <SECRET-NAME> \
--cert <PATH-TO-CERTIFICATE> \
--key <PATH-TO-KEY> \
-n openshift-config
```

Modify the cluster API server to use the new secret.

This can be accomplished by using `oc edit`, `oc patch`, or by modifying a file under version control and then using `oc apply`.

```
[user@host ~]$ oc patch apiserver cluster --type=merge \
-p '[{"spec": {"servingCerts": {"namedCertificates": '\
'[""], '\
'"servingCertificate": {"name": "<SECRET-NAME>"}]}]}'
```

If the API server updated successfully, then new `kube-apiserver` pods in the `openshift-kube-apiserver` namespace are created.

The `kube-apiserver` cluster operator transitions to a progressing state while the pods are created.

<code>oc get clusteroperator/kube-apiserver</code>					
NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
<code>kube-apiserver</code>	4.6.36	True	True	False	7d

The pods are ready when the `kube-apiserver` cluster operator is no longer progressing.

<code>oc get clusteroperator/kube-apiserver</code>					
NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
<code>kube-apiserver</code>	4.6.36	True	False	False	7d

<code>oc get pods -l app=openshift-kube-apiserver \ -n openshift-kube-apiserver</code>					
NAME	READY	STATUS	RESTARTS	AGE	
<code>kube-apiserver-master01</code>	5/5	Running	0	9m4s	
<code>kube-apiserver-master02</code>	5/5	Running	0	5m52s	
<code>kube-apiserver-master03</code>	5/5	Running	0	2m	



Note

It can take ten minutes or more before all of the `kube-apiserver` pods finish deploying.

Verifying the New Certificate

Your system can be configured to trust your enterprise CA using the following two steps:

- Copy your enterprise CA certificate to the `/etc/pki/ca-trust/source/anchors` directory. Change the name of the certificate if it conflicts with an existing file name in that directory.
- Run the `update-ca-trust extract` command.

Access the web console to verify the new certificate. Use the `oc whoami` command to find the web console URL.

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

After you enter the URL in your web browser, the web console displays with a lock icon in the URL address bar indicating that the connection is secure.

If your enterprise certificate authority signed the certificate, then your web browser might indicate that it does not recognize the certificate issuer.

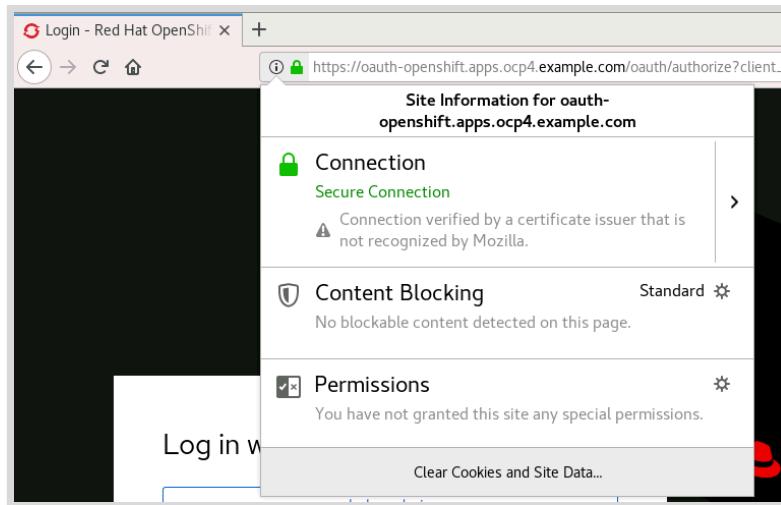


Figure 6.3: Secured Firefox Connection

If a recognized certificate authority signed the master API certificate, or if you configure your system to trust your enterprise CA, then you can log in securely using the `oc login` command.

If you had previously logged in insecurely, you can delete the `~/.kube` directory before using the `oc login` command.

Alternatively, use the `--certificate-authority` option with the `oc login` command to specify the location of the CA certificate used to sign the master API certificate.

Creating an Edge Route Using the New Wildcard Certificate

Both new and existing OpenShift applications can immediately take advantage of the new wildcard certificate by creating an edge route. While an edge route only secures traffic between the router and the user, it does not require making any modifications to the application container image.

You can still use re-encrypt routes and pass-through routes to secure traffic between the user and the application pods.

Use the `oc create route edge` to create a new edge route.

```
[user@host ~]$ oc create route edge <ROUTE-NAME> --service <SERVICE>
```



Important

When using a wildcard certificate signed by an enterprise CA, any device not configured to trust the CA will require the user to make an exception in order to allow using the certificate. This is especially true for external customers.



References

- For more information, refer to the *Configuring certificates* chapter in the Red Hat OpenShift Container Platform 4.6 *Authentication* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/security_and_compliance/index#configuring-certificates
- For more information, refer to the *Configuring the cluster wide proxy* chapter in the Red Hat OpenShift Container Platform 4.6 *Networking* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/networking/index#enable-cluster-wide-proxy
- For more information, refer to the *Adding API server certificates* section in the *Configuring certificates* chapter in the Red Hat OpenShift Container Platform 4.6 *Security and compliance* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/security_and_compliance/index#api-server-certificates

► Guided Exercise

Integrating OpenShift with an Enterprise Certificate Authority

In this exercise you will configure your OpenShift cluster so that the ingress controller and master API use a certificate signed by the classroom certificate authority.

Outcomes

You should be able to:

- Inspect a new certificate signed by the classroom certificate authority.
- Configure the ingress controller to use the new certificate.
- Configure the master API to use the new certificate.
- Validate the new certificate by accessing the web console and by running the `oc login` command.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that the cluster API is reachable, downloads files needed for the exercise, and generates a new certificate signed by the classroom certificate authority.

```
[student@workstation ~]$ lab certificates-enterprise-ca start
```

Instructions

- 1. Inspect the new certificate to view its subject, issuer, dates, and subject alternate name.

- 1.1. Change to the `~/D0380/labs/certificates-enterprise-ca` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/certificates-enterprise-ca
```

- 1.2. The `/usr/local/lib/ansible/certs/wildcard-api.yml` Ansible Playbook created the `wildcard-api.pem` certificate and the `wildcard-api-key.pem` key using the classroom CA file located at `/etc/pki/tls/certs/classroom-ca.pem` to sign the certificate.

Run the `openssl` command to view details about the certificate. The certificate is a wildcard for any URL ending in `.apps.ocp4.example.com`. It also covers the master API URL `api.ocp4.example.com`.

```
[student@workstation certificates-enterprise-ca]$ openssl x509 -in \
  wildcard-api.pem -noout -subject -issuer -ext 'subjectAltName' -dates
subject=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN =
*.apps.ocp4.example.com
issuer=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN = GLS
Training Classroom Certificate Authority
X509v3 Subject Alternative Name:
DNS:*.apps.ocp4.example.com, DNS:api.ocp4.example.com
notBefore=Sep 2 14:11:33 2021 GMT
notAfter=Aug 31 14:11:33 2031 GMT
```

- ▶ 2. Create an additional certificate combining the new certificate and the classroom CA certificate.

```
[student@workstation certificates-enterprise-ca]$ cat wildcard-api.pem \
/etc/pki/tls/certs/classroom-ca.pem > combined-cert.pem
```

- ▶ 3. Create a new TLS secret using the combined certificate.

- 3.1. Log in to your OpenShift cluster as the `admin` user.

```
[student@workstation certificates-enterprise-ca]$ oc login -u admin -p redhat \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 3.2. Create a new TLS secret in the `openshift-config` namespace using `combined-cert.pem` and `wildcard-api-key.pem`.

```
[student@workstation certificates-enterprise-ca]$ oc create secret tls \
custom-tls --cert combined-cert.pem --key wildcard-api-key.pem \
-n openshift-config
secret/custom-tls created
```

- ▶ 4. Configure the API server to use the `custom-tls` secret.

- 4.1. Modify the `apiserver-cluster.yaml` file to match the bold lines below. Specify the URL of the master API and the name of the TLS secret that you created.

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  servingCerts:
    namedCertificates:
      - names:
```

```
- api.ocp4.example.com
servingCertificate:
  name: custom-tls
```

- 4.2. Apply the changes using the `oc apply` command.

```
[student@workstation certificates-enterprise-ca]$ oc apply \
-f apiserver-cluster.yaml
Warning: oc apply should be used on resource created by either oc create --save-
config or oc apply
apiserver.config.openshift.io/cluster configured
```

- 5. If the change is successful, then the `kube-apiserver` pods in the `openshift-kube-apiserver` namespace redeploy. It can take ten minutes or more before the pods are ready. Wait until the `PROGRESSING` column for the `kube-apiserver` cluster operator has a value of `True`. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation certificates-enterprise-ca]$ watch oc get \
clusteroperator/kube-apiserver
Every 2.0s: oc get clusteroperator/kube-apiserver
workstation.lab.example.com: ...

NAME          VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
kube-apiserver  4.6.36    True        True         False      28d
```



Note

The `kube-apiserver` pods go into `CrashLoopBackOff` and `Error` states before ultimately reaching a `Running` status with all four containers running in each pod. Each pod might restart multiple times. It can take ten minutes or more before all of the `kube-apiserver` pods finish deploying.

- 6. Create a new configuration map in the `openshift-config` namespace using the combined certificate. Ensure that the configuration map uses a data key of `ca-bundle.crt`.

```
[student@workstation certificates-enterprise-ca]$ oc create configmap \
combined-certs --from-file ca-bundle.crt=combined-cert.pem \
-n openshift-config
configmap/combined-certs created
```

- 7. Modify the cluster proxy so that it adds the new configuration map to the trusted certificate bundle.
- 7.1. Modify the `proxy-cluster.yaml` file to match the bold line in the following example. Specify the name of the configuration map that you just created.

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: combined-certs
```

7.2. Apply the changes using the `oc apply` command.

```
[student@workstation certificates-enterprise-ca]$ oc apply -f proxy-cluster.yaml
Warning: oc apply should be used on resource created by either oc create --save-config or oc apply
proxy.config.openshift.io/cluster configured
```

- ▶ 8. Create a TLS secret named `custom-tls-bundle` in the `openshift-ingress` namespace. Use `custom-combined-certs` as the certificate and `wildcard-api-key.pem` as the key.

```
[student@workstation certificates-enterprise-ca]$ oc create secret tls \
  custom-tls-bundle --cert combined-cert.pem --key wildcard-api-key.pem \
  -n openshift-ingress
secret/custom-tls-bundle created
```

- ▶ 9. Modify the ingress controller operator to use the `custom-tls-bundle` secret. A successful change redeploys the router pods in the `openshift-ingress` namespace.

9.1. Modify the `ingresscontrollers.yaml` file to match the bold line below. Specify the name of the TLS secret that you just created.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  finalizers:
  - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
  name: default
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: custom-tls-bundle
  replicas: 2
```

9.2. Apply the changes using the `oc apply` command.

```
[student@workstation certificates-enterprise-ca]$ oc apply \
  -f ingresscontrollers.yaml
Warning: oc apply should be used on resource created by either oc create --save-config or oc apply
ingresscontroller.operator.openshift.io/default configured
```

9.3. Change to the `/home/student` directory.

```
[student@workstation certificates-enterprise-ca]$ cd
```

- ▶ 10. Wait until the new `router-default` pods in the `openshift-ingress` namespace finish redeploying and the previous router pods disappear. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation ~]$ watch oc get pods -n openshift-ingress
```



Important

If the previous router pods have not fully terminated, then accessing the OpenShift web console URL in the next step might still produce a certificate warning.

- ▶ 11. Identify the route to the OpenShift web console, and then access the URL in Firefox. Your system trusts any certificate signed by the `/etc/pki/tls/certs/classroom-ca.pem` CA file.

- 11.1. Identify the OpenShift web console URL.

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

- 11.2. Open the web console URL. Firefox displays a secure lock icon in the address bar.

- `https://console-openshift-console.apps.ocp4.example.com`.

- ▶ 12. Previously in this exercise, the `kube-apiserver` pods in the `openshift-kube-apiserver` namespace were redeployed. Confirm that the redeployment finished, and then test the certificate used by the OpenShift master API.

- 12.1. Wait until the `PROGRESSING` column for the `kube-apiserver` cluster operator has a value of `False`. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation certificates-enterprise-ca]$ watch \  
"oc get clusteroperator/kube-apiserver ; \  
oc get pods -l app=openshift-kube-apiserver -n openshift-kube-apiserver"
```

```
Every 2.0s: oc get clusteroperator/kube-apiserver  
workstation.lab.example.com: ...  
NAME          VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE  
kube-apiserver  4.6.36   True       False        False      27d  
NAME          READY     STATUS      RESTARTS  AGE  
kube-apiserver-master01  5/5     Running     0          9m4s  
kube-apiserver-master02  5/5     Running     0          5m52s  
kube-apiserver-master03  5/5     Running     0          2m
```

- 12.2. Log out of OpenShift.

```
[student@workstation ~]$ oc logout  
Logged "admin" out on "https://api.ocp4.example.com:6443"
```

12.3. Remove the existing /home/student/.kube directory.

```
[student@workstation ~]$ rm -rf ~/.kube
```

12.4. Log in to your OpenShift cluster as the `admin` user. Because your system trusts the master API certificate, you are not prompted to accept an insecure connection.

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

...output omitted...
```

Finish

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

```
[student@workstation ~]$ lab certificates-enterprise-ca finish
```

Configuring Applications to Trust the Enterprise Certificate Authority

Objectives

After completing this section, you should be able to:

- Include an enterprise CA certificate in the trusted certificate bundle.
- Configure an application to use the trusted certificate bundle.

Including Additional Trusted Certificate Authorities

When using your own enterprise CA, OpenShift can include the enterprise CA certificate in a trusted certificate authority bundle. This approach is useful if an application running in OpenShift must communicate with secure URLs that have been signed by your enterprise CA.

By default, applications do not trust the enterprise CA.

If your OpenShift cluster uses certificates signed by your enterprise CA, then check to see if your enterprise CA certificate is already included. Identify the configuration map used by the cluster proxy.

```
[user@host ~]$ oc get proxy/cluster \
-o jsonpath='{.spec.trustedCA.name}{"\n"}'
<CONFIGMAP-NAME>
```

Extract and then view the contents of the identified configuration map. Your own certificates, such as a wildcard certificate and enterprise CA certificate, are listed at the top.

Comments might exist in the configuration map providing information about your certificates.

```
[user@host ~]$ oc extract configmap <CONFIGMAP-NAME> \
-n openshift-config --confirm
ca-bundle.crt

[user@host ~]$ less ca-bundle.crt
...output omitted...
```

If the configuration map does not contain the enterprise CA certificate, then modify the configuration map to append the certificate. Combine the wildcard certificate and the enterprise CA certificate in a new PEM file.

Adding comments, such as `# Wildcard Cert` above the wildcard certificate and `# Enterprise CA`, above the enterprise CA certificate makes it easier to identify the certificates when viewing the configuration map at a later time.

Replace the previously identified configuration map with the new certificate.

```
[user@host ~]$ oc set data configmap <CONFIGMAP-NAME> \
--from-file ca-bundle.crt=<PATH-TO-NEW-CERTIFICATE> -n openshift-config
```

Injecting the Trusted CA Bundle

The Cluster Network Operator watches for changes to the trusted CA configuration map identified in the cluster proxy. The pod injects additional certificates into all configuration maps with the label of `config.openshift.io/inject-trusted-cabundle=true`. Within the configuration map, additional certificates appear above the existing trusted certificate authority bundle.

To use the trusted CA bundle in a pod, create an empty configuration map and then use the `oc label` command to add the `config.openshift.io/inject-trusted-cabundle=true` label.

```
[user@host ~]$ oc create configmap <CONFIGMAP-NAME>
[user@host ~]$ oc label configmap <CONFIGMAP-NAME> \
  config.openshift.io/inject-trusted-cabundle=true
```

Mounting a Trusted CA Bundle

Mount a configuration map with the label of `config.openshift.io/inject-trusted-cabundle=true` into a pod so that the pod trusts the certificate authorities identified in the configuration map.

If the configuration map contains your enterprise CA certificate, then your application trusts certificates signed by your enterprise CA. There are various ways to mount the configuration map.

The `oc set volume` command can modify part of the configuration, but you must follow it with the `oc edit` command to finish the configuration.

Alternatively, if the file is under version control, edit the configuration file and apply the changes with the `oc apply` command.

```
[user@host ~]$ oc set volume dc/<DC-NAME> -t configmap \
  --name trusted-ca --add --read-only=true \
  --mount-path /etc/pki/ca-trust/extracted/pem \
  --configmap-name <CONFIGMAP-NAME>
```

The `oc set volume` command makes the following changes in bold.

```
...output omitted...
spec:
  containers:
    - image: quay.io/redhattraining/hello-world-nginx@sha256:...
      imagePullPolicy: IfNotPresent
      name: hello
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
        - mountPath: /etc/pki/ca-trust/extracted/pem
          name: trusted-ca
```

```
readOnly: true
dnsPolicy: ClusterFirst
restartPolicy: Always
schedulerName: default-scheduler
securityContext: {}
terminationGracePeriodSeconds: 30
volumes:
- configMap:
  defaultMode: 420
  name: <CONFIGMAP-NAME>
  name: trusted-ca
...output omitted...
```

If you made some of the changes with the `oc set volume` command, then use the `oc edit` command to finish modifying the deployment or deployment configuration.

Add the lines in bold so that the pod mounts the certificate bundle at `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem`.

```
[user@host ~]$ oc edit dc/<DC-NAME>
...output omitted...
volumes:
- configMap:
  defaultMode: 420
  items:
  - key: ca-bundle.crt
    path: tls-ca-bundle.pem
  name: <CONFIGMAP-NAME>
  name: trusted-ca
...output omitted...
```

Verifying the Mounted Bundle

If you successfully changed the application deployment or deployment configuration, then application pods redeploy. Access an application pod to verify that it trusts certificates signed by your enterprise CA.

```
[user@host ~]$ oc rsh hello-3-65qs7
```

The `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem` file contains the certificate for your enterprise CA. Depending on the container image, you might be able to inspect the certificate file using `less`, `head`, `cat`, `grep`, or a similar command. The following example assumes that your enterprise CA signed the certificate for `hello.apps.ocp4.example.com`.

```
sh-4.4$ curl https://hello.apps.ocp4.example.com
<html>
<body>
<h1>Hello, world from nginx!</h1>
</body>
</html>
```



References

For more information, refer to the *Configuring a custom PKI* chapter in the Red Hat OpenShift Container Platform 4.6 Networking documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/networking/index#configuring-a-custom-pki

► Guided Exercise

Configuring Applications to Trust the Enterprise Certificate Authority

In this exercise you will modify the deployment of an application so that it trusts certificates signed by your enterprise certificate authority.

Outcomes

You should be able to:

- Identify the configuration map used by the cluster proxy.
- Verify the certificates in the cluster proxy configuration map.
- Create a new configuration map that is injected with the trusted certificate bundle.
- Mount your configuration map inside a pod so that it trusts certificates signed by your enterprise CA.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that the cluster API is reachable and ensures that your cluster uses a certificate signed by the classroom enterprise CA.

```
[student@workstation ~]$ lab certificates-app-trust start
```

Instructions

- 1. Determine if the cluster proxy trusts additional certificates. The listed name is a configuration map in the `openshift-config` namespace.
- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Display the cluster proxy resource definition in YAML format. The `name` attribute under `trustedCA` is the name of the configuration map in the `openshift-config` namespace.

```
[student@workstation ~]$ oc get proxy/cluster -o yaml
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  creationTimestamp: "2021-09-28T17:18:19Z"
  generation: 2
  name: cluster
  resourceVersion: "39884"
  selfLink: /apis/config.openshift.io/v1/proxies/cluster
  uid: c87ee674-4ddc-3efe-a74e-dfe25da5d7b3
spec:
  trustedCA:
    name: combined-certs
status: {}
```

- ▶ 2. Search for Classroom in the content of the data key of the identified configuration map. The wildcard certificate and the certificate for your enterprise CA must be contained in the configuration map.

```
[student@workstation ~]$ oc get configmap combined-certs -n openshift-config \
-o jsonpath='{.data.*}' | grep Classroom
# Classroom Wildcard Certificate
# GLS Training Classroom Certificate Authority
```



Note

If the comment for the wildcard certificate displays # Classroom Wildcard & Master API Certificate instead of # Classroom Wildcard Certificate, then the API server also uses the wildcard certificate. Either wildcard certificate is valid, and the same comment appears a few times in this exercise.

- ▶ 3. Create the certificates-app-trust project with two applications named hello1 and hello2. Use the cluster wildcard certificate to create edge routes for each application.

- 3.1. Create the certificates-app-trust project.

```
[student@workstation ~]$ oc new-project certificates-app-trust
Now using project "certificates-app-trust" on server
"https://api.ocp4.example.com:6443".
...output omitted...
```

- 3.2. Use the container image located at quay.io/redhattraining/hello-world-nginx:v1.0 to create the hello1 application.

```
[student@workstation ~]$ oc new-app --name hello1 \
--docker-image quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Success
...output omitted...
```

- 3.3. Use the cluster wildcard certificate to create an edge route for the hello1 application.

```
[student@workstation ~]$ oc create route edge --service hello1 \
--hostname hello1-trust.apps.ocp4.example.com
route.route.openshift.io/hello1 created
```

- 3.4. Use the container image located at quay.io/redhattraining/hello-world-nginx:v1.0 to create the hello2 application.

```
[student@workstation ~]$ oc new-app --name hello2 \
--docker-image quay.io/redhattraining/hello-world-nginx:v1.0
...output omitted...
--> Success
...output omitted...
```

- 3.5. Use the cluster wildcard certificate to create an edge route for the hello2 application.

```
[student@workstation ~]$ oc create route edge --service hello2 \
--hostname hello2-trust.apps.ocp4.example.com
route.route.openshift.io/hello2 created
```

- ▶ 4. Use the curl command to access both application routes from the workstation machine. Both routes are secure because the workstation machine trusts any certificate signed by the classroom enterprise CA. The edge routes use the wildcard certificate for the cluster.

- 4.1. Identify the routes.

```
[student@workstation ~]$ oc get routes
NAME      HOST/PORT          ...      PORT      TERMINATION ...
hello1    hello1-trust.apps.ocp4.example.com  ...  8080-tcp  edge   ...
hello2    hello2-trust.apps.ocp4.example.com  ...  8080-tcp  edge   ...
```

- 4.2. Use the curl command to access hello1-trust.apps.ocp4.example.com.

```
[student@workstation ~]$ curl https://hello1-trust.apps.ocp4.example.com
<html>
<body>
<h1>Hello, world from nginx!</h1>
</body>
</html>
```

- 4.3. Use the curl command to access hello2-trust.apps.ocp4.example.com.

```
[student@workstation ~]$ curl https://hello2-trust.apps.ocp4.example.com
<html>
<body>
<h1>Hello, world from nginx!</h1>
</body>
</html>
```

- ▶ 5. Connect to the `hello1` application pod and attempt to access the route for the `hello2` application. This attempt fails because the `hello1` pods do not trust certificates signed by the enterprise CA.

- 5.1. Identify the deployments for the example applications.

```
[student@workstation ~]$ oc get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
hello1    1/1     1           1           61s
hello2    1/1     1           1           48s
```

- 5.2. Connect to the `hello1` application pod.

```
[student@workstation ~]$ oc exec -it deployment/hello1 -- /bin/bash
```

- 5.3. Attempt to access the route for the `hello2` application. This attempt fails because the `hello1` pod does not trust the enterprise CA.

```
bash-4.4$ curl https://hello2-trust.apps.ocp4.example.com
curl: (60) SSL certificate problem: self signed certificate in certificate chain
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

- 5.4. Exit the pod.

```
bash-4.4$ exit
```

- ▶ 6. Inject the trusted certificate bundle into a new `ca-certs` configuration map.

- 6.1. Create the `ca-certs` configuration map.

```
[student@workstation ~]$ oc create configmap ca-certs
configmap/ca-certs created
```

- 6.2. Label the configuration map with `config.openshift.io/inject-trusted-cabundle=true`.

```
[student@workstation ~]$ oc label configmap ca-certs \
  config.openshift.io/inject-trusted-cabundle=true
configmap/ca-certs labeled
```

- 6.3. Display the contents of the configuration map. Although it was originally empty, the cluster network operator injected certificates into it. There are many certificates in the configuration map; the certificates from combined-certs were injected at the top.

```
[student@workstation ~]$ oc get configmap ca-certs -o yaml | head -n 6
apiVersion: v1
data:
  ca-bundle.crt: |
    # Classroom Wildcard Certificate
    -----BEGIN CERTIFICATE-----
    AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8gISIjJCUMJygpKissLS4v
```

- ▶ 7. Mount the ca-certs configuration map in the hello1 application pod, ensuring that the hello1 application trusts certificates signed by the classroom enterprise CA.
- 7.1. Use the `oc set volume` command to mount the ca-certs configuration map into the hello1 deployment.

```
[student@workstation ~]$ oc set volume deployment/hello1 -t configmap \
--name trusted-ca --add --read-only=true \
--mount-path /etc/pki/ca-trust/extracted/pem \
--configmap-name ca-certs
deployment.apps/hello1 volume updated
```

- 7.2. Modify the hello1 deployment to add additional information about the configuration map. Add the lines in bold and ensure the correct indentation (using spaces rather than tabs) before saving.

```
[student@workstation ~]$ oc edit deployment/hello1
...output omitted...
  volumes:
  - configMap:
      defaultMode: 420
      items:
      - key: ca-bundle.crt
        path: tls-ca-bundle.pem
      name: ca-certs
      name: trusted-ca
  ...output omitted...
```

This modification creates the `tls-ca-bundle.pem` file at the volume mount point (`/etc/pki/ca-trust/extracted/pem/`) using the data from the `ca-bundle.crt` key contained in the `ca-certs` configuration map.

- ▶ 8. Connect to the new hello1 application pod and attempt to access the route for the hello2 application.
- 8.1. Wait until the pods in the hello1 application finish redeploying. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation ~]$ watch oc get pods -l deployment=hello1
```

Chapter 6 | Configuring Trusted TLS Certificates

8.2. List the pods for the `hello1` application.

```
[student@workstation ~]$ oc get pods -l deployment=hello1
NAME           READY   STATUS    RESTARTS   AGE
hello1-566ccb7f5-f5r94   1/1     Running   0          41s
```

8.3. List the routes for the `hello2` application.

```
[student@workstation ~]$ oc get routes -l app=hello2
NAME      HOST/PORT
hello2   hello2-trust.apps.ocp4.example.com ...
```

8.4. Connect to the new `hello1` application pod.

```
[student@workstation ~]$ oc exec -it deployment/hello1 -- /bin/bash
```

8.5. Check for classroom certificates in `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem`.

```
bash-4.4$ grep Classroom /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem
# Classroom Wildcard Certificate
# GLS Training Classroom Certificate Authority
```

8.6. Access the route for the `hello2` application. The `curl` command works because the pod now trusts certificates signed by the classroom enterprise CA.

```
bash-4.4$ curl https://hello2-trust.apps.ocp4.example.com
<html>
<body>
  <h1>Hello, world from nginx!</h1>
</body>
</html>
```

8.7. Exit from the pod.

```
bash-4.4$ exit
```

► 9. Clean up your environment. Delete the `certificates-app-trust` project.

```
[student@workstation ~]$ oc delete project certificates-app-trust
project.project.openshift.io "certificates-app-trust" deleted
```

Finish

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

```
[student@workstation ~]$ lab certificates-app-trust finish
```

Troubleshooting OpenShift Certificates

Objectives

After completing this section, you should be able to:

- Describe common issues with OpenShift certificates.
- Renew expired OpenShift certificates.

Describing Common Issues with OpenShift Certificates

OpenShift needs several TLS certificates to run. The required certificates include certificates for *internal* communication between nodes and services, and certificates for *external* communication that depends on OpenShift routes and ingresses.

OpenShift relies on several resources to be able to use your organization certificates, including:

- Configuration maps
- Secrets
- Custom resources, such as `apiserver`

When you configure OpenShift to use your organization certificates for the ingress controller operator or the API server, there are many ways to troubleshoot these certificates, including reviewing the resource via the web console, using the command-line interface, or using tools such as `openssl`.

One critical administrative task is the monitoring of custom certificate expiry dates, and the renewal of those certificates before production is affected.

OpenShift makes it possible to update certificates without disrupting the applications. Operators monitor resources such as configuration maps or secrets and automatically redeploy the services responsible for serving the certificates.

Troubleshooting Certificates in OpenShift

Multiple resources are responsible for ensuring that your organization certificates are properly used, including secrets, configuration maps, and custom resources.

If the API server certificate expires, you can still log in to the web console or use the CLI, however, the following message will appear:

```
The server is using an invalid certificate: x509: certificate has expired or is  
not yet valid
```

If you add a certificate for the API server (when it previously used the default ingress controller certificate), or you remove a specific API server certificate, then the `kube-apiserver` pods in the `openshift-kube-apiserver` namespace redeploy.

Watch the progressing status of the `kube-apiserver` cluster operator to monitor the deployment of the `kube-apiserver` pods. During the redeployment, the progressing status is True. The progressing status changes to False after the redeployment completes.

```
[user@host ~]$ watch oc get clusteroperator/kube-apiserver
```

**Important**

You can experience inconsistent behavior with `oc` commands during the redeployment of the `kube-apiserver` pods. As the pods deploy, you might connect to a pod that has a different certificate than the one used when you initially logged in.

Running an `oc` command might return the message, "Unable to connect to the server: x509: certificate signed by unknown authority". Eventually all `oc` commands might return that message.

Run the `oc logout` command followed by the `oc login` command after all of the `kube-apiserver` pods use the new certificate.

Renewing the API Server Certificate

To renew the API server certificate, you must identify the name of the secret containing the certificate used by the API server.

```
[user@host ~]$ oc get apiserver/cluster -o yaml  
...output omitted...  
spec:  
  servingCerts:  
    namedCertificates:  
    - names:  
      - <API-URL> ①  
    servingCertificate:  
      name: <SECRET-NAME> ②
```

- ① The API server URL.
- ② The name of the secret that contains the certificate.

Extract the secret, and then use the `openssl` command to inspect the certificate.

```
[user@host ~]$ oc extract secret/<SECRET-NAME> -n openshift-config --confirm  
tls.crt  
tls.key  
  
[user@host ~]$ openssl x509 -in tls.crt -noout -dates  
notBefore=Apr 15 21:12:55 2016 GMT  
notAfter=Apr 15 21:13:55 2021 GMT
```

If the certificate has expired or will expire soon, then follow your company procedures to request a new certificate. As with creating the initial certificate for the API server, the certificate signing request for the certificate renewal must contain the `subjectAltName` extension for the URL used to access the API server, such as `DNS:api.ocp4.example.com`.

Chapter 6 | Configuring Trusted TLS Certificates

After obtaining a new certificate from your organization administrator, you can renew the certificate in place by running the following command.

```
[user@host ~]$ oc set data secret <SECRET-NAME> \
--from-file tls.crt=<PATH-TO-NEW-CERTIFICATE> \
--from-file tls.key=<PATH-TO-KEY> \
-n openshift-config
secret/<SECRET-NAME> data updated
```

Renewing the Ingress Controller Certificate

The OpenShift ingress controller manages routes for internal services, such as OAuth, the web console, and Prometheus. The ingress controller might use the cluster proxy, which also relies on certificates.

To renew the ingress controller certificate, you must identify the name of the secret containing the certificate used by the ingress controller.

```
[user@host ~]$ oc get ingresscontroller/default -n openshift-ingress-operator \
-o jsonpath='{.spec.defaultCertificate.name}{"\n"}'
<SECRET-NAME>
```

Extract the secret, and then use the `openssl` command to inspect the certificate.

```
[user@host ~]$ oc extract secret/<SECRET-NAME> -n openshift-ingress --confirm
tls.crt
tls.key

[user@host ~]$ openssl x509 -in tls.crt -noout -dates
notBefore=May 15 11:12:23 2019 GMT
notAfter=May 15 11:13:23 2021 GMT
```

If the certificate has expired or will expire soon, follow your company procedures to request a new certificate. As with creating the initial wildcard certificate, the certificate signing request for the certificate renewal must contain the `subjectAltName` extension of `*.apps.<OPENSHIFT-DOMAIN>`, such as `*.apps.ocp4.example.com`. After obtaining a new certificate, the secret can be updated in place using the `oc set data secret` command.

```
[user@host ~]$ oc set data secret <SECRET-NAME> \
--from-file tls.crt=<PATH-TO-NEW-CERTIFICATE> \
--from-file tls.key=<PATH-TO-KEY> \
-n openshift-config
secret/<SECRET-NAME> data updated
```

Updating the certificate instructs the router pods to redeploy in the `openshift-ingress` namespace.

After the `router-default` pods finish redeploying, you can use the following `curl` command to confirm the renewal of the certificate. Although this example checks the certificate for the OAuth URL, the same certificate is used for the web console, Prometheus, Grafana, and more.

```
[user@host ~]$ curl -v -k \
https://oauth-openshift.apps.ocp4.example.com 2>&1 | grep -w date
* start date: Jul 21 18:15:22 2021 GMT
* expire date: Jul 21 18:15:22 2022 GMT
```

If the cluster proxy uses the same certificate, then the configuration map identified for the cluster proxy must be updated as well. Identify the name of the configuration map used by the cluster proxy.

```
[user@host ~]$ oc get proxy/cluster -o jsonpath='{.spec.trustedCA.name}{"\n"}' \
<CONFIGMAP-NAME> ①
```

- ① The configuration map exists in the openshift-config namespace.

The configuration map can be updated in place using the `oc set data configmap` command.

```
[user@host ~]$ oc set data configmap <CONFIGMAP-NAME> \
--from-file ca-bundle.crt=<PATH-TO-NEW-CERTIFICATE> -n openshift-config
configmap/<CONFIGMAP-NAME> data updated
```

Troubleshooting Certificates Renewal

Should the certificate not update in the cluster, check the following:

- Use the `openssl` command to ensure that the new certificate is valid.
- Verify that the `notBefore` date is in the past and the `notAfter` date is in the future.

```
[user@host ~]$ openssl x509 -in <PATH-TO-CERTIFICATE> -noout -dates
notBefore=Jul 21 19:07:50 2021 GMT
notAfter=Jul 19 19:07:50 2026 GMT
```

After updating a certificate, you can compare the certificate serial number of the secret with the certificate file to make sure that they match. A mismatch indicates that the secret did not update properly.

```
[user@host ~]$ oc get secret <SECRET-NAME> -n openshift-config \
-o jsonpath='{.data.}' | base64 -di | openssl x509 -noout -serial*
serial=DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

[user@host ~]$ openssl x509 -in <PATH-TO-CERTIFICATE> -noout -serial
serial=DA39A3EE5E6B4B0D3255BFEF95601890AFD80709
```

Troubleshooting API Server Certificates

The `kube-apiserver` pods do not redeploy for an in place certificate update. Run the `oc get events` command in the `openshift-kube-apiserver` namespace to verify that the `kube-apiserver` pods use the updated certificate.

```
[user@host ~]$ oc get events --sort-by='.lastTimestamp' \
-n openshift-kube-apiserver
...output omitted...
30s ... CertificateUpdated ... pod/kube-apiserver-master01 ... Wrote updated
secret: openshift-kube-apiserver/user-serving-cert-000
30s ... CertificateUpdated ... pod/kube-apiserver-master02 ... Wrote updated
secret: openshift-kube-apiserver/user-serving-cert-000
30s ... CertificateUpdated ... pod/kube-apiserver-master03 ... Wrote updated
secret: openshift-kube-apiserver/user-serving-cert-000
```

Troubleshooting Ingress Controller Certificates

For the ingress controller, ensure that new `router-default` pods in the `openshift-ingress` namespace finished redeploying. Expect that each router pod has a status of `Running` and that each pod displays `1/1` in the `READY` column. Also expect that router pods associated with a previous replica set are not displayed.

```
[user@host ~]$ oc get pods -n openshift-ingress
NAME                  READY   STATUS    RESTARTS   AGE
router-default-55df6c587-96dhv   1/1     Running   0          9m
router-default-55df6c587-bf4qp   1/1     Running   0          9m
```



References

- For more information, refer to the *Configuring Certificates* chapter in the Red Hat OpenShift Container Platform 4.6 Security and compliance documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/security_and_compliance/index#replacing-default-ingress
- **How do I configure a CA and sign certificates using OpenSSL in Red Hat Enterprise Linux?**
<https://access.redhat.com/solutions/15497>

► Guided Exercise

Troubleshooting OpenShift Certificates

In this exercise, you will renew expired certificates and apply the renewed certificates to the default ingress controller operator and master API.

Outcomes

You should be able to:

- Identify expired OpenShift certificates.
- Configure the OpenShift cluster to use renewed certificates.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the **workstation** machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and configures the cluster to use certificates that are set to expire one minute in the future.

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

Instructions

► 1. Review the TLS certificates for the ingress controller.

- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```



Note

You might receive a warning regarding using insecure connections. Answer `y` to proceed.

- 1.2. Change to the `~/D0380/labs/certificates-troubleshoot` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/certificates-troubleshoot
```

- 1.3. Use the `curl` command to display information about the TLS certificate that secures traffic to the OpenShift web console.

```
[student@workstation certificates-troubleshoot]$ curl -v -k \
https://console.openshift-console.apps.ocp4.example.com 2>&1 | \
grep -E 'date|expired'
* start date: Jun 9 12:49:23 2021 GMT
* expire date: Jun 9 12:50:23 2021 GMT
* SSL certificate verify result: certificate has expired (10), continuing anyway.
```

The certificate expired at the date and time listed on the `expire date` line.



Note

The router pods in the `openshift-ingress` namespace must finish redeploying with the new (expired) certificate. If you do not see a `certificate has expired` message, then wait for a minute and try again.

- 1.4. If your enterprise CA signed the wildcard certificate for your OpenShift cluster, then the cluster-wide proxy stores certificate information in a configuration map. Run the `oc get` command to retrieve the name of the configuration map used by the cluster proxy.

```
[student@workstation certificates-troubleshoot]$ oc get proxy/cluster \
-o jsonpath='{.spec.trustedCA.name}{"\n"}'
wildcard-bundle
```



Note

If you get an error regarding the X509 certificate being expired or signed by unknown authority, then try to log in to the cluster using the `--insecure-skip-tls-verify` flag, and execute the command again.

```
[student@workstation certificates-troubleshoot]$ oc login -u admin -p redhat \
--insecure-skip-tls-verify https://api.ocp4.example.com:6443
```

- 1.5. Extract the `wildcard-bundle` configuration map to the current directory. The `--confirm` option overwrites the file if it already exist in the destination.

```
[student@workstation certificates-troubleshoot]$ oc extract \
configmap/wildcard-bundle -n openshift-config --to ./ --confirm
ca-bundle.crt
```

- 1.6. Use the `openssl` command to read the certificates. The `-dates` option displays the dates and times of certificate validity; the `-serial` option displays the certificate serial number.

```
[student@workstation certificates-troubleshoot]$ openssl x509 \
-in ca-bundle.crt -noout -subject -issuer -dates -serial
subject=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN =
*.apps.ocp4.example.com
issuer=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN = GLS
Training Classroom Certificate Authority
notBefore=Oct 29 00:14:45 2021 GMT
notAfter=Oct 29 00:15:45 2021 GMT
serial=7ECC993EB3CF0E8942A92B687D9F5152CB730727
```

**Note**

Although your certificate for CN = *.apps.ocp4.example.com displays different information, expect that the notAfter date (in GMT) is in the past.

The ca-bundle.crt file has two certificates and openssl only reads the first one.

```
[student@workstation certificates-troubleshoot]$ grep '^#' ca-bundle.crt
# Classroom Wildcard Certificate
# GLS Training Classroom Certificate Authority
```

- Run the `oc get` command to retrieve the ingress controller secret name. The secret contains the TLS certificate and the associated private key.

```
[student@workstation certificates-troubleshoot]$ oc get \
-ingresscontroller/default -n openshift-ingress-operator \
-o jsonpath='{.spec.defaultCertificate.name}{"\n"}'
wildcard-tls
```

- Extract the wildcard-tls secret and save the output on the current directory. The `--confirm` option overwrites the files if they already exist in the destination.

```
[student@workstation certificates-troubleshoot]$ oc extract secret/wildcard-tls \
-n openshift-ingress --to ./ --confirm
tls.crt
tls.key
```

- Use the `openssl` command to read the certificate. The `-dates` option displays the dates and times of certificate validity and the `-serial` option displays the certificate serial number.

```
[student@workstation certificates-troubleshoot]$ openssl x509 -in tls.crt \
-noout -subject -issuer -dates -serial
subject=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN =
*.apps.ocp4.example.com
issuer=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN = GLS
Training Classroom Certificate Authority
notBefore=Oct 29 00:14:45 2021 GMT
notAfter=Oct 29 00:15:45 2021 GMT
serial=7ECC993EB3CF0E8942A92B687D9F5152CB730727
```

The dates and `serial` line match the output from the `ca-bundle.crt` certificate. The `notAfter` line indicates that the certificate expired just after starting this exercise.



Note

The `tls.crt` file has two certificates and `openssl` only reads the first one.

```
[student@workstation certificates-troubleshoot]$ grep '^#' tls.crt
# Classroom Wildcard Certificate
# GLS Training Classroom Certificate Authority
```

- 2. Renew the ingress controller wildcard certificate. Ensure the certificate is valid for ten years.

- 2.1. Run the `renew_wildcard.sh` script. The script uses an Ansible Playbook to create a new wildcard certificate with an expiration date set to 3650 days from now (about ten years). The script stores files in the current directory.

```
[student@workstation certificates-troubleshoot]$ ./renew_wildcard.sh
...output omitted...
TASK [Create a combined certificate] ****
ok: [localhost]

TASK [Create a hard-link to the combined certificate] ****
ok: [localhost]

PLAY RECAP ****
localhost: ok=18    changed=6      unreachable=0      failed=0      skipped=3
rescued=0      ignored=0
```

- 2.2. Use the `openssl` command to verify that the `wildcard-combined.pem` certificate expires in about ten years.

```
[student@workstation certificates-troubleshoot]$ openssl x509 \
-in wildcard-combined.pem -noout -subject -issuer -dates -serial
subject=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN =
*.apps.ocp4.example.com
issuer=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN = GLS
Training Classroom Certificate Authority
notBefore=Oct 29 00:22:34 2021 GMT
notAfter=Oct 27 00:22:34 2031 GMT
serial=4ABEFC4A1663CBA79C7042FD26DDB4122AD87346
```

**Note**

The `wildcard-combined.pem` file has two certificates and `openssl` only reads the first one.

```
[student@workstation certificates-troubleshoot]$ grep '^#' wildcard-combined.pem
# Classroom Wildcard Certificate
# GLS Training Classroom Certificate Authority
```

► 3. Update the OpenShift cluster proxy and ingress controller operator.

- 3.1. You must also update the TLS secret that the ingress controller uses. The secret resides in the `openshift-ingress` namespace.

Update the existing `wildcard-tls` secret in the `openshift-ingress` namespace using the renewed `wildcard-combined.pem` certificate and the existing `wildcard-key.pem` private key.

```
[student@workstation certificates-troubleshoot]$ oc set data \
secret/wildcard-tls -n openshift-ingress \
--from-file tls.crt=wildcard-combined.pem \
--from-file tls.key=wildcard-key.pem
secret/wildcard-tls data updated
```

► 4. Confirm the renewal of the ingress controller wildcard certificate. From the `workstation` machine, run the following `curl` command. The output indicates an expiration date of about ten years in the future.

```
[student@workstation certificates-troubleshoot]$ curl -v -k \
https://console-openshift-console.apps.ocp4.example.com 2>&1 | grep -w date
* start date: Oct 29 00:22:34 2021 GMT
* expire date: Oct 27 00:22:34 2031 GMT
```

**Note**

It can take a minute before the router pods use the updated certificate. You can get different outputs from the `curl` command above depending on which router pod processes the request.

► 5. Review the TLS certificates of the API server.

- 5.1. Delete the `~/.kube` directory and run the `oc login` command. The message indicates that the API certificate has expired or is not yet valid. Type `y` to bypass the certificate check.

```
[student@workstation certificates-troubleshoot]$ rm -rf ~/.kube
[student@workstation certificates-troubleshoot]$ oc login -u admin -p redhat \
https://api.ocp4.example.com:6443
```

```
The server is using an invalid certificate: x509: certificate has expired or is
not yet valid
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.
Use insecure connections? (y/n): y

Login successful.

...output omitted...
```



Note

The start script redeploys the API server pods in the `openshift-kube-apiserver` namespace using an expiring certificate. If the pods have not finished redeploying, then the `invalid certificate` warning does not display. It is safe to continue the exercise even if you do not see the warning.

- 5.2. The `cluster` resource is a custom resource of type `apiserver` and defines the certificate that the API server uses. Use the `oc get` command to review the resource configuration.

```
[student@workstation certificates-troubleshoot]$ oc get apiserver/cluster \
-o yaml | tail
...output omitted...
spec:
  servingCerts:
    namedCertificates:
      - names:
          - api.ocp4.example.com
    servingCertificate:
      name: api-tls ①
```

① The secret that contains the certificates.

- 5.3. Extract the `api-tls` secret and save the output to the current directory. The `--confirm` option overwrites the files if they already exist in the destination.

```
[student@workstation certificates-troubleshoot]$ oc extract secret/api-tls \
-n openshift-config --to ./ --confirm
tls.crt
tls.key
```

- 5.4. Use the `openssl` command to read the certificate. The `-issuer` and `-dates` options display the certificate authority that issued the certificate and the dates and times of certificate validity.

```
[student@workstation certificates-troubleshoot]$ openssl x509 -in tls.crt \
  -noout -subject -issuer -dates -serial
subject=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN =
api.ocp4.example.com
issuer=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN = GLS
Training Classroom Certificate Authority
notBefore=Oct 29 00:14:53 2021 GMT
notAfter=Oct 29 00:15:53 2021 GMT
serial=714A26872F9262B69BC3A52EA55B464E1920EC10
```

The GLS Training Classroom Certificate Authority CA issued the certificate. Although your dates are different, expect that the certificate expired because it was only valid for one minute.



Note

The `tls.crt` file has two certificates and `openssl` only reads the first one.

```
[student@workstation certificates-troubleshoot]$ grep '^#' tls.crt
# Master API Certificate
# GLS Training Classroom Certificate Authority
```

▶ 6. Renew the API certificate.

- 6.1. Run the `renew_api.sh` script. The script uses an Ansible Playbook to create a new API certificate with an expiration date set to 3650 days from now (about ten years). The script stores files in the current directory.

```
[student@workstation certificates-troubleshoot]$ ./renew_api.sh
...output omitted...
TASK [Create a combined certificate] *****
ok: [localhost]

TASK [Create a hard-link to the combined certificate] *****
ok: [localhost]

PLAY RECAP *****
localhost : ok=18    changed=6      unreachable=0      failed=0
  skipped=3    rescued=0     ignored=0
```

- 6.2. List the certificates in the `~/D0380/labs/certificates-troubleshoot` directory.

```
[student@workstation certificates-troubleshoot]$ ls *.pem | grep -v key
api-combined.pem
api.pem
wildcard-combined.pem
wildcard.pem
```

- 6.3. Use the `openssl` command to display information about the `api-combined.pem` certificate.

```
[student@workstation certificates-troubleshoot]$ openssl x509 \
-in api-combined.pem -noout -subject -issuer -dates -serial
subject=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN =
api.ocp4.example.com
issuer=C = US, ST = NC, L = Raleigh, O = "Red Hat, Inc.", OU = Training, CN = GLS
Training Classroom Certificate Authority
notBefore=Oct 29 00:30:20 2021 GMT
notAfter=Oct 27 00:30:20 2031 GMT
serial=30904D7078C79BB97736D50EC5FC6A21E8EE492F
```

The output indicates an expiration date of about ten years from now.



Note

The `api-combined.pem` file has two certificates and `openssl` only reads the first one.

```
[student@workstation certificates-troubleshoot]$ grep '^#' api-combined.pem
# Master API Certificate
# GLS Training Classroom Certificate Authority
```

▶ 7. Update the secret for the API certificate.

- 7.1. Update the existing `api-tls` secret in the `openshift-config` namespace so that it uses the renewed `api-combined.pem` certificate and the existing `api-key.pem` private key.

```
[student@workstation certificates-troubleshoot]$ oc set data secret/api-tls \
-n openshift-config \
--from-file tls.crt=api-combined.pem \
--from-file tls.key=api-key.pem
secret/api-tls data updated
```

- 7.2. If the `oc set data` command succeeds, then the certificate used by each API server pod is updated. Run the following command to view the latest events in the `openshift-kube-apiserver` namespace.

```
[student@workstation certificates-troubleshoot]$ oc get events \
-n openshift-kube-apiserver --sort-by='lastTimestamp' | tail
...output omitted...
60s ... CertificateUpdated ... pod/kube-apiserver-master01 ... Wrote updated
secret: openshift-kube-apiserver/user-serving-cert-000
60s ... CertificateUpdated ... pod/kube-apiserver-master02 ... Wrote updated
secret: openshift-kube-apiserver/user-serving-cert-000
60s ... CertificateUpdated ... pod/kube-apiserver-master03 ... Wrote updated
secret: openshift-kube-apiserver/user-serving-cert-000
```

- 7.3. Change to the `/home/student` directory.

```
[student@workstation certificates-troubleshoot]$ cd
```

- 7.4. Run the following `curl` command to ensure that the certificate is renewed.

```
[student@workstation ~]$ curl -v -k https://api.ocp4.example.com:6443 2>&1 | \
  grep -w date
* start date: Oct 29 00:30:20 2021 GMT
* expire date: Oct 27 00:30:20 2031 GMT
< date: Fri, 29 Oct 2021 00:34:29 GMT
```

**Note**

You might need to run the `curl` command more than once before you see the updated certificate. Although your dates are different, expect that the certificate expires in about ten years.

You can get different outputs from the `curl` command above depending on which API server pod processes the request.

- 7.5. Delete the `~/.kube` directory and run the `oc login` command. Ensure that the warning about an expired certificate no longer displays. If it does, cancel the login operation with `Ctrl+C` and try again after a few seconds.

```
[student@workstation ~]$ rm -rf ~/.kube

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

...output omitted...
```

**Note**

You can get different error messages while trying to log into the cluster. This happens because the API server pods are being restarted. It can take ten minutes or more before all of the `kube-apiserver` finish deploying.

Finish

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

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

▶ Lab

Configuring Trusted TLS Certificates

In this lab, you will configure OpenShift to use certificates signed by an enterprise certificate authority.

Outcomes

You should be able to:

- Update the ingress controller operator and the master API to use certificates signed by the classroom certificate authority.
- Confirm that `https://console-openshift-console.apps.ocp4.example.com` and `https://api.ocp4.example.com:6443` use the updated certificates.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the workstation machine, use the `lab` command to prepare your system for this exercise.

This command:

- Configures the OpenShift cluster to use default certificates.
- Creates a classroom CA certificate at `/etc/pki/tls/certs/classroom-ca.pem`.
- Downloads sample YAML files to `/home/student/D0380/labs/certificates-review/`.
- Creates a combined wildcard and master API certificate signed by `/etc/pki/tls/certs/classroom-ca.pem`.

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



Important

If the `lab` command returns the message, "Cannot login to OpenShift as the 'kubeadmin' user with the password of '<PASSWORD>'", then the `kube-apiserver` pods in the `openshift-kube-apiserver` namespace have not finished redeploying.

Monitor the progress of the `kube-apiserver` cluster operator with the `watch oc get clusteroperator/kube-apiserver` command until the only message displayed is "Unable to connect to the server: x509: certificate signed by unknown authority".

Then, press `Ctrl+C` to exit the `watch` command, and then run the `lab` command again.

**Note**

- Consider using the *Configuring certificates* chapter of the Red Hat OpenShift Container Platform Authentication guide as a reference.

Instructions

- As the OpenShift `admin` user, update the cluster-wide proxy to use the `/home/student/D0380/labs/certificates-review/review-combined.pem` certificate.
- Update the ingress controller operator to use the `~/D0380/labs/certificates-review/review-combined.pem` certificate and `~/D0380/labs/certificates-review/review-key.pem` key.
- Update the API server to use the `~/D0380/labs/certificates-review/review-combined.pem` certificate and `~/D0380/labs/certificates-review/review-key.pem` key.
- Verify that the web console URL uses the updated certificate `https://console-openshift-console.apps.ocp4.example.com`.
- Verify that the API server URL uses the updated certificate `https://api.ocp4.example.com:6443`.

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 certificates-review grade
```

Finish

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

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

► Solution

Configuring Trusted TLS Certificates

In this lab, you will configure OpenShift to use certificates signed by an enterprise certificate authority.

Outcomes

You should be able to:

- Update the ingress controller operator and the master API to use certificates signed by the classroom certificate authority.
- Confirm that `https://console-openshift-console.apps.ocp4.example.com` and `https://api.ocp4.example.com:6443` use the updated certificates.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the workstation machine, use the `lab` command to prepare your system for this exercise.

This command:

- Configures the OpenShift cluster to use default certificates.
- Creates a classroom CA certificate at `/etc/pki/tls/certs/classroom-ca.pem`.
- Downloads sample YAML files to `/home/student/D0380/labs/certificates-review/`.
- Creates a combined wildcard and master API certificate signed by `/etc/pki/tls/certs/classroom-ca.pem`.

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



Important

If the `lab` command returns the message, "Cannot login to OpenShift as the 'kubeadmin' user with the password of '<PASSWORD>'", then the `kube-apiserver` pods in the `openshift-kube-apiserver` namespace have not finished redeploying.

Monitor the progress of the `kube-apiserver` cluster operator with the `watch oc get clusteroperator/kube-apiserver` command until the only message displayed is "Unable to connect to the server: x509: certificate signed by unknown authority".

Then, press `Ctrl+C` to exit the `watch` command, and then run the `lab` command again.

**Note**

- Consider using the *Configuring certificates* chapter of the Red Hat OpenShift Container Platform Authentication guide as a reference.

Instructions

- As the OpenShift admin user, update the cluster-wide proxy to use the /home/student/D0380/labs/certificates-review/review-combined.pem certificate.

- Log in to your OpenShift cluster as the admin user.

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://api.ocp4.example.com:6443
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.
Use insecure connections? (y/n): y

Login successful.

...output omitted...
```

**Note**

You might get an error x509: certificate signed by unknown authority while attempting to log in. This happens because the API server pods are being restarted. Please wait a few seconds and attempt to log in again. It can take ten minutes or more before all of the kube-apiserver pods finish deploying.

- Change to the ~/D0380/labs/certificates-review directory.

```
[student@workstation ~]$ cd ~/D0380/labs/certificates-review
```

- Create the review-bundle configuration map in the openshift-config namespace using the review-combined.pem certificate.

```
[student@workstation certificates-review]$ oc create configmap review-bundle \
-n openshift-config --from-file ca-bundle.crt=review-combined.pem
configmap/review-bundle created
```

- Modify the proxy-cluster.yml file to replace <CONFIGMAP-NAME> with the name of the configuration map that you created (review-bundle).

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: review-bundle
```

15. Update the proxy/cluster resource to use the review-bundle configuration map.

```
[student@workstation certificates-review]$ oc apply -f proxy-cluster.yml
proxy.config.openshift.io/cluster configured
```



Note

The following `oc patch` command is an alternative to editing `proxy-cluster.yml` and using the `oc apply` command.

```
$ oc patch proxy/cluster --type=merge \
-p '{"spec": {"trustedCA": {"name": "review-bundle"}}}'
```

2. Update the ingress controller operator to use the `~/D0380/labs/certificates-review/review-combined.pem` certificate and `~/D0380/labs/certificates-review/review-key.pem` key.
 - 2.1. Create the `review-tls` secret in the `openshift-ingress` namespace using the `review-combined.pem` certificate and `review-key.pem` key.

```
[student@workstation certificates-review]$ oc create secret tls review-tls \
-n openshift-ingress --cert review-combined.pem --key review-key.pem
secret/review-tls created
```

- 2.2. Modify the `ingresscontrollers.yaml` file to replace <SECRET-NAME> with the name of the secret that you created (`review-tls`).

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  finalizers:
  - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
  name: default
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: review-tls
  replicas: 2
```

- 2.3. Update the `ingresscontrollers/default` resource to use the `review-tls` secret.

```
[student@workstation certificates-review]$ oc apply -f ingresscontrollers.yml
ingresscontroller.operator.openshift.io/default configured
```

**Note**

The following `oc patch` command is an alternative to editing `ingresscontrollers.yml` and using the `oc apply` command.

```
$ oc patch ingresscontroller.operator default \
-n openshift-ingress-operator --type=merge \
-p '{"spec": {"defaultCertificate": {"name": "review-tls"}}}'
```

3. Update the API server to use the `~/D0380/labs/certificates-review/review-combined.pem` certificate and `~/D0380/labs/certificates-review/review-key.pem` key.

- 3.1. Create the `review-tls` secret in the `openshift-config` namespace using the `review-combined.pem` certificate and `review-key.pem` key.

```
[student@workstation certificates-review]$ oc create secret tls review-tls \
-n openshift-config --cert review-combined.pem --key review-key.pem
secret/review-tls created
```

- 3.2. Modify the `apiserver-cluster.yml` file.

- Replace <MASTER-API> with `api.ocp4.example.com`.
- Replace <SECRET-NAME> with the name of the secret that you created (`review-tls`).

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  servingCerts:
    namedCertificates:
    - names:
      - api.ocp4.example.com
    servingCertificate:
      name: review-tls
```

- 3.3. Update the `apiserver/cluster` resource to use the `review-tls` secret.

```
[student@workstation certificates-review]$ oc apply -f apiserver-cluster.yml
apiserver.config.openshift.io/cluster configured
```

**Note**

The following `oc patch` command is an alternative to editing `apiserver-cluster.yaml` and using the `oc apply` command.

```
$ oc patch apiserver cluster --type=merge \
-p '{"spec":{"servingCerts":{"namedCertificates":'`\
'[{"names":["api.ocp4.example.com"],'`\
'"servingCertificate":{"name":"review-tls"}]}]}}'
```

4. Verify that the web console URL uses the updated certificate `https://console-openshift-console.apps.ocp4.example.com`.
 - 4.1. A successful update to the `ingresscontrollers/default` resource redeloys the router pods in the `openshift-ingress` namespace. Verify the redeployment of the router pods. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation certificates-review]$ watch oc get pods \
-n openshift-ingress
NAME                  READY   STATUS    RESTARTS   AGE
router-default-567c74bcdd-6pvzz  1/1     Running   0          4m3s
router-default-567c74bcdd-prv7m  1/1     Running   0          3m45s
```

- 4.2. Access the web console URL using either the `curl` command or the Firefox web browser.
 - `https://console-openshift-console.apps.ocp4.example.com`

```
[student@workstation certificates-review]$ curl -v $(oc whoami --show-console) \
2>&1 | grep 'SSL certificate verify ok.'
*   SSL certificate verify ok.
```

If you access the URL using Firefox, then the URL field displays a green lock icon.

5. Verify that the API server URL uses the updated certificate `https://api.ocp4.example.com:6443`.
 - 5.1. A successful update to the `apiserver/cluster` resource redeloys the `apiserver` pods in the `openshift-kube-apiserver` namespace. Verify the redeployment of the API server pods.

The `kube-apiserver` cluster operator should no longer be progressing. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation certificates-review]$ watch \
"oc get clusteroperator/kube-apiserver ; \
oc get pods -l app=openshift-kube-apiserver -n openshift-kube-apiserver"

Every 2.0s: oc get clusteroperator/kube-apiserver
workstation.lab.example.com: ...
NAME              VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
kube-apiserver  4.6.36   True       False        False     27d
NAME              READY   STATUS    RESTARTS   AGE
```

kube-apiserver-master01	5/5	Running	0	9m4s
kube-apiserver-master02	5/5	Running	0	5m52s
kube-apiserver-master03	5/5	Running	0	2m



Note

It can take ten minutes or more before all of the kube-apiserver pods finish deploying.

5.2. Change to the /home/student directory.

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

5.3. Remove the ~/ .kube directory.

```
[student@workstation ~]$ rm -rf ~/.kube
```

5.4. Log in to your OpenShift cluster as the admin user.

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

...output omitted...
```

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 certificates-review grade
```

Finish

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

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

Summary

In this chapter, you learned:

- The OpenShift installer creates an internal certificate authority (CA) and uses this CA to sign additional certificates.
- Because the default OpenShift certificates are not signed by a recognized certificate authority, users that access the cluster must add one or more exceptions to accept OpenShift certificates.
- The ingress operator configures the ingress controller to route traffic into the OpenShift environment. You can update the certificate that the ingress controller uses with a certificate signed by a recognized certificate authority, or by your own enterprise CA.
- Changing the ingress controller operator to use a different certificate and its associated key only requires a handful of steps. These steps include acquiring the new certificate and key in PEM format, and the certificate must define a wildcard domain in the `subjectAltName` field. This allows OpenShift to use that certificate as a wildcard certificate for the `.apps` subdomain.
- OpenShift makes it possible to update certificates in place without disrupting the applications because operators monitor resources, such as configuration or secrets, and automatically redeploy the services responsible for serving the certificates.

Chapter 7

Configuring Dedicated Node Pools

Goal

Add nodes to an OpenShift cluster with custom configurations.

Objectives

- Add nodes to an OpenShift cluster on user-provisioned infrastructure.
- Create custom machine configuration pools and edit existing machine configurations.

Sections

- Adding Compute Nodes (and Guided Exercise)
- Creating Custom Machine Config Pools (and Guided Exercise)

Adding Compute Nodes

Objectives

After completing this section, you should be able to add nodes to an OpenShift cluster on user-provisioned infrastructure.

Discussing Node Scaling

Add more instances to your OpenShift cluster to increase the capacity of the cluster.

The steps for performing instance scaling operations differ for installer-provisioned and user-provisioned OpenShift clusters.

The Machine API automatically performs scaling operations for supported cloud providers. Modify the number of replicas specified in a Machine Set, and OpenShift communicates with the cloud provider to provision or deprovision instances.

In user-provisioned infrastructure, you must manually provision new instances. The exact strategy for provisioning is specific to your data center and your IT processes, however the base steps remain the same:

1. Update the compute node Ignition file with an updated TLS certificate.
2. Install Red Hat Enterprise Linux CoreOS (RHCOS) from an ISO image or using a Preboot eXecution Environment (PXE) boot.
3. Add the new instance to the ingress load balancer.
4. Approve Certificate Signing Requests (CSRs) to allow the compute node to join the cluster.

Discussing Ignition

RHCOS is a RHEL-based operating system managed by the OpenShift Container Platform.

Red Hat discourages directly manipulating a RHCOS configuration. Instead, provide initial instance configuration in the form of Ignition files.

Ignition is a provisioning tool designed to configure RHCOS on first boot. Supply configuration in a JSON-formatted `.ign` file to declare changes to the disk, such as partitions, systemd units, certificates, and custom files.

The OpenShift installer generates the following Ignition files:

- `bootstrap.ign`
- `master.ign`
- `worker.ign`

Specify the `worker.ign` Ignition file as a kernel parameter when adding a new compute node.

After the instance is provisioned, changes to RHCOS are managed by the Machine Config Operator. These changes are also specified in the Ignition format, but are limited in scope.

Updating the Worker TLS Certificate

The OpenShift installer creates the Ignition files with an initial bootstrapping TLS certificate. In most production installations, this certificate expires 24 hours after installation.

- Fetch the latest TLS certificate from the Machine Config Server listening on the control plane port 22623.
- Run the `openssl s_client -connect api-int.ocp4.example.com:22623 -showcerts` command to retrieve the certificate.
- Extract the text wrapped between the BEGIN CERTIFICATE and END CERTIFICATE markers.
- Replace the base64-encoded certificate in the `worker.ign` as follows.
- Use the `base64` command to encode the certificate text.

```
{  
  "ignition": {  
    "config": {  
      "append": [  
        {  
          "source": "https://api-int.ocp4.example.com:22623/config/worker",  
          "verification": {}  
        }  
      ]  
    },  
    "security": {  
      "tls": {  
        "certificateAuthorities": [  
          {  
            "source": "data:text/plain;charset=utf-8;base64,LS0...==", ❶  
            "verification": {}  
          }  
        ]  
      }  
    },  
    "timeouts": {},  
    "version": "2.2.0"  
  },  
  "networkd": {},  
  "passwd": {},  
  "storage": {},  
  "systemd": {}  
}
```

- ❶ Specify the TLS certificate as a base64-encoded value in the TLS source field of the `certificateAuthorities` list.



Note

Use the `openssl x509 -text -noout` command to view information about the certificate such as the issuer, subject, and expiration date.

Installing with PXE boot

Install RHCOS from an ISO image or by using PXE boot. PXE booting performs a network-based installation and is a common choice in an enterprise data center.

PXE relies on a set of very basic technologies:

- Dynamic Host Configuration Protocol (DHCP) for locating instances.
- Trivial File Transfer Protocol (TFTP) for serving the PXE files.
- HTTP for the ISO images and configuration files.

The RHCOS PXE configuration specifies the location of a kernel file, OS images, and an Ignition configuration file.

Discussing Ingress Load Balancing

When adding compute nodes to a cluster, add the instance to external load balancer back-end pools. For example, OpenShift schedules router pods on compute nodes. The ingress load balancer must direct traffic to the new compute node instance.

OpenShift supports a variety of load balancers. The choice of specific load balancer technology is outside the scope of this course.

The following is an example of the HAProxy configuration file `/etc/haproxy/haproxy.cfg`. HAProxy listens on port 443 and routes traffic to either of the two servers.

Example HAProxy configuration file

```
frontend ingress_secure
    bind *:443
    mode tcp
    default_backend ingress_secure_backend
backend ingress_secure_backend
    balance roundrobin
    mode tcp
    server worker01 192.168.50.13:443 check
    server worker02 192.168.50.14:443 check
```

Approving Certificate Signing Requests

When a new node attempts to join the cluster, OpenShift creates two Certificate Signing Requests (CSRs). First, the Machine Config Operator client requests a CSR:

`system:serviceaccount:openshift-machine-config-operator:node-bootstrapper`. Then, after you approve the node bootstrapper CSR, OpenShift creates a CSR with the new node as the requester, such as `system:node:worker04`.

Approve the certificate using `oc adm certificate approve CSR` command as follows.

```
[user@host ~]$ oc get csr -A
NAME          AGE      REQUESTOR
             CONDITION
csr-87kbl    75m      system:node:master01
              Approved,Issued
csr-knchlw   106m     system:node:worker03
              Approved,Issued
csr-lqcbd    4m19s    system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper Pending

[user@host ~]$ oc adm certificate approve csr-lqcbd
certificatesigningrequest.certificates.k8s.io/csr-lqcbd approved
```

Alternatively, approve all CSRs using the xargs command.

```
[user@host ~]$ oc get csr -A -o name | xargs oc adm certificate approve
```

Adding RHEL Compute Nodes

RHEL instances can join an OpenShift cluster as compute machines. RHEL instances are not automatically managed and updated by OpenShift operators. A system administrator must be responsible for update management and operations.



References

For more information on adding workers, refer to the *Red Hat Enterprise Linux CoreOS (RHCOS)* chapter in the Red Hat OpenShift Container Platform 4.6 *Architecture* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/architecture/architecture-rhcos

For more information on RHEL compute nodes, refer to the *Adding RHEL compute machines to an OpenShift Container Platform cluster* chapter in the Red Hat OpenShift Container Platform 4.6 *Machine management* documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/machine_management/index#adding-rhel-compute

► Guided Exercise

Adding Compute Nodes

In this exercise you will add compute nodes to an OpenShift cluster.

Outcomes

You should be able to:

- Examine PXE and Ignition files required for adding new nodes to a cluster.
- Update the compute node Ignition file with a new certificate.
- Update the HAProxy configuration to add the new nodes to the load balancer.

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 pools-adding-workers start
```

Instructions

- 1. From workstation, use ssh to connect to the utility machine as the `lab` user.

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

- 2. Examine the PXE files in the `/var/lib/tftpboot/pxelinux.cfg` directory.

- 2.1. List the files, and notice that three of the file names end with `-NOOP`. Worker machines attempt to load a file that matches `01-` followed by the MAC address of the machine. The `-NOOP` suffix prevents the new workers from loading the file.

```
[lab@utility ~]$ ls /var/lib/tftpboot/pxelinux.cfg/
01-52-54-00-00-32-09 01-52-54-00-00-32-0b 01-52-54-00-00-32-0d
01-52-54-00-00-32-0f 01-52-54-00-00-32-0a 01-52-54-00-00-32-0c
01-52-54-00-00-32-0e 01-52-54-00-00-32-10-NOOP 01-52-54-00-00-32-11-NOOP
01-52-54-00-00-32-12-NOOP
```

- 2.2. View the contents of one of the PXE files.

```
[lab@utility ~]$ cat /var/lib/tftpboot/pxelinux.cfg/01-52-54-00-00-32-10-NOOP
default menu.c32
prompt 0
timeout 50
menu title **** OpenShift 4 Worker PXE Boot Menu ****
```

```
label Install CoreOS 4.6.8 Worker Node
kernel /openshift4/4.6.36/rhcos-4.6.8-x86_64-live-kernel-x86_64
append ip=dhcp rd_neednet=1 coreos.inst.install_dev=vda console=tty0
console=ttyS0 coreos.inst=yes coreos.live.rootfs_url=http://192.168.50.254:8080/
openshift4/images/rhcos-4.6.8-x86_64-live-rootfs.x86_64.img
coreos.inst.ignition_url=http://192.168.50.254:8080/openshift4/ignitions/ocp4/
worker.ign initrd=/openshift4/4.6.36/rhcos-4.6.8-x86_64-live-initramfs.x86_64.img
```

The PXE file references an Ignition file that is also hosted on the utility machine.

► 3. View the worker Ignition file and examine the embedded certificate.

- 3.1. List the contents of the `/var/www/html/openshift4/ignitions/ocp4/worker.ign` Ignition file. Pipe the output to `jq` for readability.

```
[lab@utility ~]$ sudo ls /var/www/html/openshift4/ignitions/ocp4/
bootstrap.ign master.ign worker.ign
[lab@utility ~]$ sudo cat /var/www/html/openshift4/ignitions/ocp4/worker.ign \
| jq
{
  "ignition": {
    ...output omitted...
  },
  "security": {
    "tls": {
      "certificateAuthorities": [
        {
          "source": "data:text/plain;charset=utf-8;base64,LS0tL...cg=="
        }
      ],
      ...output omitted...
    }
  }
}
```

- 3.2. Copy the base64-encoded value of the `source` data to your clipboard by selecting it and pressing `Ctrl+Shift+C`. Do not include the `data:text/plain;charset=utf-8;base64,` prefix.
- 3.3. Use the `echo` command to print the copied base64-encoded certificate and pipe the output to the `base64 -d` command.

```
[lab@utility ~]$ echo "LS0tL...g==" | base64 -d
-----BEGIN CERTIFICATE-----
MIIDEDCCAfigAwIBAgIIZeDuWKYIU6YwDQYJKoZIhvcNAQELBQAwJjESMBAGA1UE
CxMJb3BlbnNoaWZ0MRAWDgYDVQQDEwdyb290LWNhMB4XDTIwMDUyODE5MTAwNFo
...output omitted...
01PU1obMUx/pTzerQb0067kcMPo=
-----END CERTIFICATE-----
```

- 3.4. Use the `openssl` command to examine the x509 certificate.

```
[lab@utility ~]$ echo "LS0tL...g==" | base64 -d | \
openssl x509 -noout -text
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 7341129457124031398 (0x65e0ee58a608bba6)
Signature Algorithm: sha256WithRSAEncryption
```

```
Issuer: OU = openshift, CN = root-ca
Validity
    Not Before: May 28 19:10:04 2020 GMT
    Not After : May 26 19:10:04 2030 GMT
Subject: OU = openshift, CN = root-ca
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
...output omitted...
```

In a production environment, the certificate embedded in the worker Ignition file might expire 24 hours after initial installation.

- ▶ 4. Exit the SSH session on **utility** to return to **workstation**.
- ▶ 5. Inspect and then run the **prep-utility.yml** Ansible Playbook.
 - 5.1. Change to the `~/D0380/labs/pools-adding-workers/` directory and open the **prep-utility.yml** file.

```
[student@workstation ~]$ cd ~/D0380/labs/pools-adding-workers/
[student@workstation pools-adding-workers]$ vim prep-utility.yml
```

- 5.2. Review the tasks defined in the playbook. Ansible updates the worker Ignition file with a new certificate from `api-int.ocp4.example.com`, copies the PXE files to paths without the "-NOOP" suffix, and restarts the TFTP socket. The final playbook tasks update the HAProxy configuration to include the new workers.
- 5.3. Run the Ansible Playbook.

```
[student@workstation pools-adding-workers]$ ansible-playbook prep-utility.yml

PLAY [Add new workers] ****
TASK [Find OpenShift version directory] ****
ok: [utility]

...output omitted...

PLAY RECAP ****
utility      : ok=9    changed=5    unreachable=0   failed=0   skipped=0 ...
```

- ▶ 6. Reboot workers 04-06 using the **ROL Lab Environment** controls. The unconfigured machines automatically reboot every five minutes. Manually reboot the machines from ROL to avoid waiting.
 - 6.1. Navigate to the **ROL Lab Environment** tab to control the classroom virtual machines.
 - 6.2. Click **Open Console** next to **worker04**, and then press any key or press **Ctrl+Alt+Del** to reboot the machine. Return to the **ROL Lab Environment** tab and repeat the process for **worker05** and **worker06**.



Note

The workers will reboot a couple times during the PXE boot process.

- 7. Review the HAProxy configuration on the **utility** machine to view the new worker nodes. The classroom runs HAProxy to round-robin load balance ingress traffic to the worker nodes. In a previous step, the Ansible Playbook added the workers to the ingress back end.
- 7.1. From **workstation**, use **ssh** to run the **cat** command on the **utility** machine as the **lab** user.

```
[student@workstation pools-adding-workers]$ ssh lab@utility \
  cat /etc/haproxy/haproxy.cfg
  ...output omitted...
```

- 7.2. Find and review the servers listed for the **ingress_insecure_backend** and **ingress_secure_backend**, located near the end of the file.

► 8. Inspect and then run the **approve-csrs.sh** Bash script to approve the worker CSRs.

 - 8.1. Log in to your OpenShift cluster as the **admin** user.

```
[student@workstation pools-adding-workers]$ oc login -u admin -p redhat \
  https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

- 8.2. Review the **approve-csrs.sh** script. The script approves pending CSRs in a **while** loop until it counts six worker nodes.
- 8.3. Execute the script.

**Note**

It will take several minutes for the new worker machines to join the cluster.

```
[student@workstation pools-adding-workers]$ ./approve-csrs.sh
Worker count is 3.
certificatesigningrequest.certificates.k8s.io/csr-8mstv approved
Worker count is 4.
certificatesigningrequest.certificates.k8s.io/csr-g6sjn approved
certificatesigningrequest.certificates.k8s.io/csr-4j84s approved
certificatesigningrequest.certificates.k8s.io/csr-n4hhq approved
Worker count is 5.
certificatesigningrequest.certificates.k8s.io/csr-wh64h approved
Worker count is 6.
certificatesigningrequest.certificates.k8s.io/csr-kvhtg approved
```

- 9. Run **oc get nodes** to list the three new nodes.

**Note**

It will take a minute or two for all workers to reach a **Ready** state.

```
[student@workstation pools-adding-workers]$ oc get nodes
NAME      STATUS    ROLES   AGE     VERSION
master01  Ready     master   23d    v1.19.0+b00ba52
master02  Ready     master   23d    v1.19.0+b00ba52
master03  Ready     master   23d    v1.19.0+b00ba52
worker01  Ready     worker   23d    v1.19.0+b00ba52
worker02  Ready     worker   23d    v1.19.0+b00ba52
worker03  Ready     worker   23d    v1.19.0+b00ba52
worker04  Ready     worker   6m49s  v1.19.0+b00ba52
worker05  Ready     worker   6m47s  v1.19.0+b00ba52
worker06  Ready     worker   6m48s  v1.19.0+b00ba52
```

- 10. Change to the /home/student/ directory.

```
[student@workstation pools-adding-workers]$ cd
```

Finish

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

The compute nodes added in this exercise will remain available for the rest of the course. They are used in subsequent chapters.

```
[student@workstation ~]$ lab pools-adding-workers finish
```

Creating Custom Machine Config Pools

Objectives

After completing this section, you should be able to create custom machine configuration pools and edit existing machine configurations.

Discussing the Machine Config Operator

The Machine Config Operator (MCO) manages instance configuration changes and operating system upgrades. OpenShift administrators can set custom node configurations by declaring `MachineConfig` and `MachineConfigPool` resources.

The MCO defines two custom resources.

`MachineConfig (MC)`

Machine Configs declare instance customizations using the Ignition config format. Machine Configs are labeled with a role such as `worker` or `infra`.

`MachineConfigPool (MCP)`

Machine Config Pools use labels to match one or more Machine Configs to one or more nodes. This creates a pool of nodes with the same configuration. The Machine Config Operator uses the Machine Config Pool to track status as it applies Machine Configs to the nodes.

Creating Machine Configs

Machine Configs use the Ignition configuration format. Ignition was originally designed as an immutable configuration tool to execute only during initial instance provisioning. The Machine Config Daemon supports a limited set of configuration changes including the following:

- Configuring core user SSH authorized keys.
- Declaring Systemd units.
- Writing custom files.

The Machine Config Operator reads Machine Configs alphanumerically, from `00-*` to `99-*`. Changes to the same file are overwritten by Machine Configs named with a higher number. The resulting compilation of Machine Configs is stored in a "rendered" Machine Config resource.

Machine Configs are specified with a `machineconfiguration.openshift.io/role` label. List Machine Configs for a specific role using the `--selector` argument.

```
[user@host ~]$ oc get machineconfig \
  --selector=machineconfiguration.openshift.io/role=worker
  NAME          GENERATEDBYCONTROLLER   IGNITIONVERSION AGE
  00-worker      910f22cb1550a...2b1566ce5f  2.2.0        20d
  01-worker-container-runtime 910f22cb1550a...2b1566ce5f  2.2.0        20d
  01-worker-kubelet    910f22cb1550a...2b1566ce5f  2.2.0        20d
  99-worker-registries 910f22cb1550a...2b1566ce5f  2.2.0        20d
  99-worker-ssh       910f22cb1550a...2b1566ce5f  2.2.0        20d
```

Writing Custom Files

Specify custom file contents, such as the Systemd configuration or TLS certificates, in an escaped format following the data URL scheme standard. File contents are typically Base64 encoded in ignition files. Other content, like Systemd units or ssh keys, are not Base64 encoded. In a Terminal, use the `base64` and `base64 -d` commands to encode files or standard input.

Journald MachineConfig example

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker ①
  name: 60-journald ②
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-8;base64,VGVzdGluYXk= ③
            filesystem: root
            mode: 0644
            path: /etc/systemd/journald.conf
```

- ① Label Machine Config resources by role.
- ② Prefix the name with a two-digit number that specifies when to apply the configuration, relative to Machine Configs belonging to the same Machine Config Pool.
- ③ Use the data URL format to embed escaped file content. Base64 encoding is common for Ignition files.

Creating Machine Config Pools

Machine Config Pools specify a `machineConfigSelector` and a `nodeSelector`. By default, OpenShift includes only `master` and `worker` Machine Config Pools, but custom pools can be added.

Custom pools are a composition of `worker` and custom Machine Configs. The `machineConfigSelector` match expression selects both `worker` and the custom role label. Nodes assigned to the custom pool use Machine Configs from the `worker` with additions supplied by the custom role. This is critical so that `worker` operating system updates managed by OpenShift are applied to the machines in the pool.

The following `MachineConfigPool` specification demonstrates creating a separate pool for machine learning (ml) nodes.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: ml
spec:
```

```
machineConfigSelector:  
  matchExpressions:  
    - key: machineconfiguration.openshift.io/role  
      operator: In  
      values: [worker, ml] ❶  
nodeSelector:  
  matchLabels:  
  node-role.kubernetes.io/ml: "" ❷
```

- ❶ Include both worker and ml Machine Configs.
- ❷ Apply to nodes with the node-role.kubernetes.io/ml label.

Labeling Nodes

Add a custom role to a node by labeling the node with a node-role.kubernetes.io/ROLE key. Nodes can have multiple roles.

The example below adds an `infra` node commonly used for scheduling infrastructure workloads, such as cluster logging.

```
[user@host ~]$ oc label node/worker03 node-role.kubernetes.io/infra=  
node/worker03 labeled  
  
[user@host ~]$ oc get nodes  
NAME STATUS ROLES AGE VERSION  
master01 Ready master 15d v1.18.3+012b3ec  
master02 Ready master 15d v1.18.3+012b3ec  
master03 Ready master 15d v1.18.3+012b3ec  
worker01 Ready worker 15d v1.18.3+012b3ec  
worker02 Ready worker 15d v1.18.3+012b3ec  
worker03 Ready infra,worker 15d v1.18.3+012b3ec
```

Notice that nodes can be assigned multiple roles, and Machine Config Pools can select multiple Machine Configs. Because the `infra` Machine Config Pool includes both `worker` and `infra` Machine Configs, you can remove the `worker` role from the node.

```
[user@host ~]$ oc label node/worker03 node-role.kubernetes.io/worker=  
node/worker03 labeled  
  
[user@host ~]$ oc get nodes  
NAME STATUS ROLES AGE VERSION  
NAME STATUS ROLES AGE VERSION  
master01 Ready master 15d v1.18.3+012b3ec  
master02 Ready master 15d v1.18.3+012b3ec  
master03 Ready master 15d v1.18.3+012b3ec  
worker01 Ready worker 15d v1.18.3+012b3ec  
worker02 Ready worker 15d v1.18.3+012b3ec  
worker03 Ready infra 15d v1.18.3+012b3ec
```

Alternatively, add custom roles to the existing worker roles.

```
[user@host ~]$ oc get nodes
NAME      STATUS   ROLES      AGE      VERSION
master01  Ready    master     15d     v1.18.3+012b3ec
master02  Ready    master     15d     v1.18.3+012b3ec
master03  Ready    master     15d     v1.18.3+012b3ec
worker01  Ready    worker,app 15d     v1.18.3+012b3ec
worker02  Ready    worker,app 15d     v1.18.3+012b3ec
worker03  Ready    worker,infra 15d     v1.18.3+012b3ec
```

Configuring Pod Scheduling

Specify a nodeSelector to assign workloads to nodes that match the label.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example
spec:
  replicas: 4
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      containers:
        - image: example:v1.0
          name: example
          ports:
            - containerPort: 8080
              protocol: TCP
```

To define a default node selector, edit the `defaultNodeSelector` using the `oc edit scheduler cluster` command. After editing the scheduler resource, wait several minutes for all API server replicas to restart and the change to take effect.

If a `defaultNodeSelector` is defined, you must specify a node selector to override the default. The `oc debug node` command cannot schedule on nodes other than the default. To work around this issue, create a new project with the `--node-selector=""` flag and run the debug in that namespace.

```
[user@host ~]$ oc debug node/master01
Starting pod/master01-debug ...
To use host binaries, run chroot /host

Removing debug pod ...
Error from server (BadRequest): container "container-00" in pod "master01-debug"
is not available
```

```
[user@host ~]$ oc adm new-project debug --node-selector=""
Created project debug

[user@host ~]$ oc debug node/master01 -n debug
Starting pod/master01-debug ...
To use host binaries, run chroot /host
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.2#
```

Node taints prevent workloads from being scheduled on nodes without the proper tolerations. However, several OpenShift DaemonSets declare specific tolerances that will cause unexpected behavior if these tolerances conflict with custom taints.

For example, the Machine Config Daemon tolerates `master` and `etcd` taints. If a custom `NoSchedule` taint is added, the Machine Config Daemon will not be able to apply operating system updates to the machine.

Observing Machine Config Pool Updates

The Machine Config Daemon manages the node update, using several node annotations that are useful for understanding the state of the update.

- `machine-config-daemon.v1.openshift.com/currentConfig`
- `machine-config-daemon.v1.openshift.com/desiredConfig`
- `machine-config-daemon.v1.openshift.com/state`

Remember that you can use the `oc describe` command to view annotations.

```
[user@host ~]$ oc describe node worker01
Name:           worker01
Roles:          worker
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/arch=amd64
                kubernetes.io/hostname=worker01
                kubernetes.io/os=linux
                node-role.kubernetes.io/worker=
                node.openshift.io/os_id=rhcos
Annotations:   machinecon...openshift.io/currentConfig: rendered-worker-370...bfd
                machinecon...openshift.io/desiredConfig: rendered-worker-370...bfd
                machineconfiguration.openshift.io/reason:
                machineconfiguration.openshift.io/state: Done
                volumes.kubernetes.io/controller-managed-attach-detach: true
...output omitted...
```

When the desired config does not match the current config, the Machine Config Daemon will:

1. Apply the rendered Machine Config.
2. Drain pods from the node.
3. Reboot the machine.

List the overall status of pool updates using the `oc get machineconfigpool` command. During the first update the `CONFIG` field remains blank.

```
[user@host ~]$ oc get machineconfigpool
NAME      CONFIG          UPDATED    UPDATING   DEGRADED
infra     False
master    rendered-master-716...267  True       False      False
worker    rendered-worker-573...3db  True       False      False
```



References

For more information on pod scheduling, refer to the *Placing pods on specific nodes using node selectors* section in the *Working with pods* chapter in the Red Hat OpenShift Container Platform 4.6 *Nodes* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#nodes-pods-node-selectors

For more information on managing nodes, refer to the *Working with nodes* chapter in the Red Hat OpenShift Container Platform 4.6 *Nodes* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#working-with-nodes

OpenShift Container Platform 4: How does Machine Config Pool work?

<https://www.redhat.com/en/blog/openshift-container-platform-4-how-does-machine-config-pool-work>

The "data" URL scheme

<https://tools.ietf.org/html/rfc2397>

► Guided Exercise

Creating Custom Machine Config Pools

In this exercise you will create an `infra` Machine Config Pool and add a custom Machine Config.

Outcomes

You should be able to:

- Create custom Machine Config Pools and Machine Configs.
- Observe the status of Machine Config updates.

Before You Begin

To perform this exercise, ensure you have completed *Guided Exercise: Adding Compute Nodes*. Nodes `worker04`, `worker05`, and `worker06` must be added to the cluster and in a `Ready` state.

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

```
[student@workstation ~]$ lab pools-creating start
```

Instructions

- 1. Label nodes `worker04`, `worker05`, and `worker06` with the `infra` role, and then remove the `worker` role.
- 1.1. Use the `oc label node` command to add the `node-role.kubernetes.io/infra=label`.

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

```
[student@workstation ~]$ oc get nodes
NAME      STATUS   ROLES     AGE      VERSION
master01  Ready    master    23d     v1.19.0+b00ba52
master02  Ready    master    23d     v1.19.0+b00ba52
master03  Ready    master    23d     v1.19.0+b00ba52
worker01  Ready    worker   23d     v1.19.0+b00ba52
worker02  Ready    worker   23d     v1.19.0+b00ba52
worker03  Ready    worker   23d     v1.19.0+b00ba52
worker04  Ready    worker   91m    v1.19.0+b00ba52
worker05  Ready    worker   91m    v1.19.0+b00ba52
worker06  Ready    worker   91m    v1.19.0+b00ba52
```

```
[student@workstation ~]$ oc label node/worker04 node-role.kubernetes.io/infra=node/worker04 labeled  
  
[student@workstation ~]$ oc label node/worker05 node-role.kubernetes.io/infra=node/worker05 labeled  
  
[student@workstation ~]$ oc label node/worker06 node-role.kubernetes.io/infra=node/worker06 labeled
```

- 1.2. Use the `oc label node` command to remove the `node-role.kubernetes.io/worker` label.

```
[student@workstation ~]$ oc label node/worker04 node-role.kubernetes.io/worker-node/worker04 labeled  
  
[student@workstation ~]$ oc label node/worker05 node-role.kubernetes.io/worker-node/worker05 labeled  
  
[student@workstation ~]$ oc label node/worker06 node-role.kubernetes.io/worker-node/worker06 labeled
```

- 1.3. Use the `oc get nodes` command to review the updated roles.

```
[student@workstation ~]$ oc get nodes  
NAME      STATUS   ROLES     AGE      VERSION  
master01   Ready    master    23d     v1.19.0+b00ba52  
master02   Ready    master    23d     v1.19.0+b00ba52  
master03   Ready    master    23d     v1.19.0+b00ba52  
worker01   Ready    worker   23d     v1.19.0+b00ba52  
worker02   Ready    worker   23d     v1.19.0+b00ba52  
worker03   Ready    worker   23d     v1.19.0+b00ba52  
worker04   Ready    infra    93m    v1.19.0+b00ba52  
worker05   Ready    infra    93m    v1.19.0+b00ba52  
worker06   Ready    infra    93m    v1.19.0+b00ba52
```

- 2. Change to the `~/D0380/labs/pools-creating/` directory, and then update the Machine Config defined in `motd-mc.yml` to add a custom `/etc/motd` file.

- 2.1. Encode the contents of `motd.txt` to Base64. Copy the encoded value to the clipboard.

```
[student@workstation ~]$ cd ~/D0380/labs/pools-creating  
  
[student@workstation pools-creating]$ base64 motd.txt  
VGhpcyBpcyBhIGN1c3RvbSBtZXNzYWdlIG9mIHRoZSBkYXkgZmlsZS4K
```

- 2.2. Replace the text in `motd-mc.yml` marked `CHANGE_ME`. Update the metadata label `machineconfiguration.openshift.io/role` to `infra` and the name to `50-motd`. Update the data portion of the `source` value to the base64-encoded value of `motd.txt` stored in your clipboard.

- 2.3. Verify that the completed file reads as follows.

```
[student@workstation pools-creating]$ cat motd-mc.yml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: infra
  name: 50-motd
spec:
  config:
    ignition:
      version: 3.1.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-8;base64,VGhpcpcyBh...9mIHRZS4K
            filesystem: root
            mode: 0644
            path: /etc/motd
```

- 3. Update the Machine Config Pool defined in `infra-mcp.yml` to apply `worker` and `infra` Machine Configs to nodes matching the `node-role.kubernetes.io/infra` label.
- 3.1. Replace `CHANGE_ME` in the `machineConfigSelector` with the text `[worker,infra]`.
 - 3.2. Replace `CHANGE_ME` in the `nodeSelector` with the text `node-role.kubernetes.io/infra: ""`.
 - 3.3. Verify the completed file reads as follows.

```
[student@workstation pools-creating]$ cat infra-mcp.yml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values: [worker,infra]
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/infra: ""
```

- 4. Create both resources in the cluster.

```
[student@workstation pools-creating]$ oc create -f .
machineconfigpool.machineconfiguration.openshift.io/infra created
machineconfig.machineconfiguration.openshift.io/50-motd created
```

- 5. List the Machine Configs and verify that the `50-motd` Machine Config is included.

```
[student@workstation pools-creating]$ oc get mc
NAME                GENERATEDBYCONTROLLER   IGNITIONVERSION AGE
00-master           093319...578d          3.1.0          23d
00-worker           093319...578d          3.1.0          23d
01-master-container-runtime 093319...578d 3.1.0          23d
01-master-kubelet   093319...578d          3.1.0          23d
01-worker-container-runtime 093319...578d 3.1.0          23d
01-worker-kubelet   093319...578d          3.1.0          23d
50-motd           3.1.0          16s
99-master-generated-registries 093319...578d 3.1.0          23d
99-master-ssh       3.1.0          23d
99-worker-generated-registries 093319...578d 3.1.0          23d
99-worker-ssh       3.1.0          23d
rendered-infra-1768c...ced17a9 093319...578d 3.1.0          11s
rendered-master-9512...5f5168a 093319...578d 3.1.0          23d
rendered-master-bdb2...14061d7 093319...578d 3.1.0          23d
rendered-master-e91d...1f788d9 093319...578d 3.1.0          23d
rendered-worker-9b0c...053ff2e 093319...578d 3.1.0          23d
rendered-worker-ab45...58a9caf 093319...578d 3.1.0          23d
rendered-worker-d989...606a709 093319...578d 3.1.0          23d
```

► 6. List the Machine Config Pools, and then describe the `infra` Machine Config Pool.

- 6.1. Use the `oc get mcp` command to list the `MachineConfigPool` resources. Notice that the Machine Config Pool is currently in an `UPDATING` state.

```
[student@workstation pools-creating]$ oc get mcp
NAME    CONFIG      UPDATED  UPDATING  DEGRADED ...
infra   rendered-infra-f25... False    True      False    ...
master  rendered-master-71... True     False     False    ...
worker  rendered-worker-57... True     False     False    ...
```

- 6.2. Use the `oc describe mcp/infra` command to view details about the `infra` Machine Config Pool. Review the source Machine Configs added to the Spec.

```
[student@workstation pools-creating]$ oc describe mcp/infra
Name:         infra
Namespace:
Labels:       <none>
Annotations: <none>
API Version: machineconfiguration.openshift.io/v1
Kind:         MachineConfigPool
Metadata:
  Creation Timestamp: 2020-06-14T23:55:46Z
  Generation:        2
  ...output omitted...
  Resource Version:  211637
  Self Link:         /apis/machine...openshift.io/v1/machineconfigpools/infra
  UID:              d7c2b26a-1bb9-4f8a-84fe-57ba1cb1596e
Spec:
  Configuration:
    Name:  rendered-infra-f25cb509a35fdca08b5fb488a8a68ba2
```

```
Source:
  API Version: machineconfiguration.openshift.io/v1
  Kind: MachineConfig
  Name: 00-worker
  API Version: machineconfiguration.openshift.io/v1
  Kind: MachineConfig
  Name: 01-worker-container-runtime
  API Version: machineconfiguration.openshift.io/v1
  Kind: MachineConfig
  Name: 01-worker-kubelet
  API Version: machineconfiguration.openshift.io/v1
  Kind: MachineConfig
  Name: 50-motd
...output omitted...
```

▶ 7. Check the state of the updating nodes.

- 7.1. Use the `oc get nodes` command to list the nodes. Notice that the `infra` node reports a `SchedulingDisabled` status. The node is updating with the new Machine Config.

NAME	STATUS	ROLES	AGE	VERSION
master01	Ready	master	23d	v1.19.0+b00ba52
master02	Ready	master	23d	v1.19.0+b00ba52
master03	Ready	master	23d	v1.19.0+b00ba52
worker01	Ready	worker	23d	v1.19.0+b00ba52
worker02	Ready	worker	23d	v1.19.0+b00ba52
worker03	Ready	worker	23d	v1.19.0+b00ba52
worker04	Ready, SchedulingDisabled	infra	4m33s	v1.19.0+b00ba52
worker05	Ready	infra	4m31s	v1.19.0+b00ba52
worker06	Ready	infra	4m29s	v1.19.0+b00ba52

- 7.2. Describe the node to review the `machineconfiguration.openshift.io` annotations.

```
[student@workstation pools-creating]$ oc describe node/worker05
Name: worker05
Roles: infra,worker
Labels:
  beta.kubernetes.io/arch=amd64
  beta.kubernetes.io/os=linux
  kubernetes.io/arch=amd64
  kubernetes.io/hostname=worker05
  kubernetes.io/os=linux
  node-role.kubernetes.io/infra=
  node-role.kubernetes.io/worker=
  node.openshift.io/os_id=rhcos
Annotations:
  ma...openshift.io/currentConfig: rendered-infra-f25cb...ba2
  ma...openshift.io/desiredConfig: rendered-infra-f25cb...ba2
  machineconfiguration.openshift.io/reason:
  machineconfiguration.openshift.io/state: Done
...output omitted...
```

- 8. After the update is complete, use `oc debug node/worker05` to verify that the file updated. Retry in a few minutes if the first attempt fails. When finished, use `Ctrl+D` twice to exit.

```
[student@workstation pools-creating]$ oc debug node/worker05
Starting pod/worker05-debug ...
To use host binaries, run chroot /host
Pod IP: 192.168.50.17
If you don't see a command prompt, try pressing enter.
sh-4.2# chroot /host
sh-4.4# cat /etc/motd
This is a custom message of the day file.
sh-4.4# exit
sh-4.4# exit

Removing debug pod ...
```

- 9. Change to the `/home/student/` directory.

```
[student@workstation pools-creating]$ cd
```

Finish

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

```
[student@workstation ~]$ lab pools-creating finish
```



Note

The machine config and machine config pool created in this exercise are used in following exercises. Thus, the `lab pools-creating finish` command does not revert those changes.



Note

In production environments, administrators can use dedicated `infra` nodes to host only infrastructure components such as control plane services, default routers, the logging stack, the metrics stack, or the OpenShift internal registry.

For this course, you do not add a default node selector to prevent regular applications from using the dedicated `infra` nodes because there are only six nodes in the cluster.

Summary

In this chapter, you learned:

- How to use Ignition as a provisioning tool for Red Hat Enterprise Linux CoreOS.
- How to add nodes to an existing user-provisioned OpenShift cluster by updating the worker Ignition file and PXE booting new nodes.
- How to use Machine Config Operator, MachineConfigs, and MachineConfigPools, and how they are related.
- How to create custom MachineConfigs and MachineConfig pools and how they get applied to nodes in the cluster.

Chapter 8

Configuring Persistent Storage

Goal

Configure storage providers and storage classes to ensure cluster user access to persistent volume resources.

Objectives

- Describe the components of the OpenShift storage architecture.
- Configure an application with shared file storage.
- Configure a database application with block storage.
- Use a custom resource to create persistent local volumes.

Sections

- Describing the OpenShift Storage Architecture (and Quiz)
- Provisioning Shared Storage for Applications (and Guided Exercise)
- Provisioning Block Storage for Databases (and Guided Exercise)
- Provisioning Local Block Storage (and Guided Exercise)

Lab

Configuring Persistent Storage

Describing the OpenShift Storage Architecture

Objectives

After completing this section, you should be able to describe the components of the OpenShift storage architecture.

Persistent Storage Overview

Many applications require long-term data storage, and each application has unique storage needs. Many storage technologies exist to address application storage requirements, each with different benefits, drawbacks, and costs.

With the OpenShift storage architecture, cluster administrators can provide various storage resources to OpenShift cluster users. The architecture enables administrators to limit expensive or scarce storage resources to specific projects that require specialized storage resources. Cluster administrators can also specify a default storage resource type that is scalable and affordable.

The OpenShift storage architecture has three primary components:

- Storage Classes
- Persistent Volumes
- Persistent Volume Claims

Storage Classes

OpenShift uses storage class resources to describe different types of persistent volumes in a cluster. A storage class is a cluster resource and does not contain sensitive information. Non-privileged users can view the storage classes that are available in a cluster.

A storage class describes high-level storage characteristics, such as quality of service, throughput, or storage technology. A storage class resource does not contain low-level storage implementation details. A storage class is a public resource describing a type of storage volume in the cluster that does not expose sensitive configurations or information.

A storage class must define a `provisioner` attribute. OpenShift uses a provisioner to create persistent volumes on demand. OpenShift does not contain a built-in provisioner for all storage technologies.

Persistent Volumes

A **persistent volume (PV)** is an OpenShift cluster resource that describes an allocatable storage volume. A persistent volume defines metadata and configuration details that OpenShift uses to mount the volume on the node. After OpenShift mounts a volume on a node, it can deploy pods that use the storage volume.

Each persistent volume identifies a **volume plug-in**, which is automation that OpenShift uses to mount a storage volume. The persistent volume resource defines parameters that the volume plug-in requires to mount storage. For example, an NFS persistent volume defines an `nfs` section that contains parameters to mount an NFS share. To mount the persistent volume on a node, OpenShift executes the NFS volume plug-in with parameters defined in the NFS persistent volume.

By default, only privileged users can view persistent volume resources. Persistent volumes might contain sensitive information, such as remote storage endpoints or authentication details.

Persistent Volume Claims

Non-privileged users create a **persistent volume claim (PVC)**, to request a persistent volume resource. A persistent volume claim specifies high-level storage criteria for a volume, such as:

- Storage size
- Storage class
- Access mode

OpenShift attempts to match each persistent volume claim with an available persistent volume in the cluster. If OpenShift identifies a persistent volume match for a persistent volume claim, then OpenShift binds the persistent volume to the persistent volume claim.

Describing Persistent Volumes

An example persistent volume resource definition follows:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003 1
spec:
  capacity:
    storage: 5Gi 2
    volumeMode: Filesystem 3
    accessModes: 4
      - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle 5
  storageClassName: slow 6
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs: 7
    path: /tmp
    server: 172.17.0.2
```

- ➊ The name of the persistent volume.
- ➋ The size of the underlying storage volume.
- ➌ Supported **Block** and **Filesystem** volume modes. A **Filesystem** volume is mounted as a directory, while a **Block** volume is mounted as a raw block device.
- ➍ The access mode list describes how to access the volume from one or more cluster nodes. A **ReadWriteOnce** volume can only be mounted as read-write on a single cluster node. Access modes are discussed elsewhere in this course.
- ➎ The reclaim policy is one of: **Retain**, **Recycle**, or **Delete**. The reclaim policy determines the actions to perform on a persistent volume when that persistent volume is released.
- ➏ The storage class name for the persistent volume. A persistent volume claim must request the **slow** storage class to bind to this persistent volume.

- 7 This persistent volume uses the NFS volume plug-in. The `nfs` section defines parameters that the NFS volume plug-in requires to mount the volume on a node. This section includes sensitive NFS configuration information.

Provisioning and Binding Persistent Volumes

A persistent volume claim uses a storage class name to request a particular type of storage. OpenShift examines the pool of existing persistent volumes. If the storage class name for an existing persistent volume matches the request, then the persistent volume becomes a match candidate for the request.



Note

A persistent volume claim can request storage without specifying a storage class. If a PVC does not specify a storage class, then OpenShift uses a default storage class. The cluster administrator designates one of the storage classes as the default storage class.

OpenShift further filters the match candidate list based on other request criteria. If any persistent volumes satisfy all of the request criteria, then OpenShift binds one of the persistent volumes to the persistent volume claim. A persistent volume claim is exclusive; a persistent volume cannot bind to more than one persistent volume claim.

If no existing persistent volumes satisfy the request criteria, then OpenShift attempts to provision a volume on demand. OpenShift provides the storage class provisioner with parameters defined in the storage class, ensuring that the correct type of storage is created. **Dynamic provisioning** refers to the creation of persistent volumes on-demand.



Note

If the `provisioner` attribute of a storage class is set to `kubernetes.io/no-provisioner`, then a provisioner is not available to create storage volumes.

For technologies that do not have a provisioner, you must find other methods to create storage volumes. Recommended methods include:

- **Install a storage operator**

There are many third-party and open-source operators that manage the provisioning of persistent volumes and related storage resources.

In a later exercise, you install an operator to manage local storage persistent volumes.

- **Write and use Ansible Playbooks**

Playbooks enable configuration management at scale. Use a playbook to create and manage storage volumes, persistent volumes, and storage classes.

In a later exercise, you use a playbook to create iSCSI targets, iSCSI persistent volumes, and an iSCSI storage class.

Releasing a Persistent Volume

When you delete a persistent volume claim, you indicate that you no longer need the persistent volume bound to that claim. If no pods are using the persistent volume, then OpenShift deletes

the persistent volume claim immediately. OpenShift delays deletion until no running pods are using the persistent volume.

After deleting a persistent volume claim resource, OpenShift releases the associated persistent volume.

Reclaiming a Persistent Volume

OpenShift initiates a reclaim policy for a persistent volume after OpenShift releases it.

OpenShift defines three reclaim policies:

Delete

The persistent volume is not put back into the pool of available persistent volumes. The persistent volume resource is deleted, and the underlying storage volume is also deleted. All dynamically-provisioned persistent volumes use a `Delete` reclaim policy.

Retain

OpenShift takes no immediate action to reclaim the volume. A cluster administrator must ensure proper handling of this volume, including data backup and volume deletion.

Recycle

OpenShift deletes all data on the volume, which enables volume reuse. After deletion, the volume is available to use again.



References

For more information about the OpenShift storage architecture, refer to the *Understanding Persistent Storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/index#understanding-persistent-storage

For more information about storage classes, refer to the *Defining a StorageClass* section in the *Dynamic Provisioning* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/index#defining-storage-classes_dynamic-provisioning

► Quiz

Describing the OpenShift Storage Architecture

Match the items below to their counterparts in the table.

A cluster resource that defines characteristics for a particular type of storage that users can request.

A cluster resource that defines the information that OpenShift requires to mount a volume to a node.

A project resource that defines a request for storage with specific storage characteristics.

Code that OpenShift uses to create a storage resource.

Code that OpenShift uses to mount a storage resource on a node.

OpenShift Storage Component	Description
Provisioner	
Persistent Volume	
Storage Class	
Persistent Volume Claim	
Volume Plug-in	

► Solution

Describing the OpenShift Storage Architecture

Match the items below to their counterparts in the table.

OpenShift Storage Component	Description
Provisioner	Code that OpenShift uses to create a storage resource.
Persistent Volume	A cluster resource that defines the information that OpenShift requires to mount a volume to a node.
Storage Class	A cluster resource that defines characteristics for a particular type of storage that users can request.
Persistent Volume Claim	A project resource that defines a request for storage with specific storage characteristics.
Volume Plug-in	Code that OpenShift uses to mount a storage resource on a node.

Provisioning Shared Storage for Applications

Objectives

After completing this section, you should be able to configure an application with shared file storage.

Describing Shared Storage

Many modern applications use shared storage. Shared storage enables application scaling and parallel processing of data.

Examples of applications using shared storage:

Continuous Integration/Continuous Delivery pipelines

An application clones source code to a shared location. Agents execute tests concurrently on the shared files.

Artificial Intelligence Workloads

Model training algorithms process large data sets in parallel. Each parallel process has access to the same shared data set.

Streaming applications

An application collects unstructured data and writes that data to a common location. This data is processed at a later time.

Modern storage technologies fall into three broad categories:

File Storage

File storage technologies store data as files in a file hierarchy. Network-attached Storage (NAS) is an example of a file storage technology. Common NAS technologies include:

- Network File System (NFS)
- Server Message Block (SMB)
- Common Internet File System (CIFS)

Block Storage

Block storage technologies store data in blocks on a block storage device. Examples of block storage include:

- Storage Area Networks (SAN)
- Amazon Elastic Block Store (EBS)
- Google Cloud Persistent Disk

Object Storage

Object storage technologies store data as an object with arbitrary metadata. Objects are not stored in a file system hierarchy. Examples of object storage include:

- Amazon Simple Storage Service (S3)
- Red Hat Ceph Storage
- OpenStack Object Storage (Swift)

Use file or object storage technologies for shared storage.

Because block storage technologies allow only single-client access to a storage volume, block storage is not shareable.

Accessing Persistent Shared Storage

Before deploying a pod that requires persistent storage, Kubernetes first mounts the storage volume on the scheduled node. A persistent volume contains information that Kubernetes requires to mount the storage volume on a node. This information includes a list of access modes supported by the underlying storage volume.

An access mode indicates the ability (or inability) to mount a storage volume on many different nodes based on the capabilities of the storage provider. Kubernetes defines three access modes:

Persistent Volume Access Modes

Access Mode	CLI Abbreviation	Description
ReadWriteMany	RWX	Kubernetes can mount the volume as read-write on many nodes.
ReadOnlyMany	ROX	Kubernetes can mount the volume as read-only on many nodes.
ReadWriteOnce	RwO	Kubernetes can mount the volume as read-write on only a single node.



Note

An application that requires shared storage must request a persistent volume with a `ReadOnlyMany` or `ReadWriteMany` access mode.

Kubernetes uses a `volume plug-in` to mount a storage volume on a node. Each different storage technology corresponds to a separate volume plug-in. Every persistent volume specifies a volume plug-in. A persistent volume must also define metadata that the volume plug-in requires to mount the storage volume on a node.

The following table lists the built-in OpenShift volume plug-ins, along with the supported access modes for the plug-in:

OpenShift Volume Plug-in support for Access Modes

Volume Plug-in	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWS EBS	Yes	No	No
Azure File	Yes	Yes	Yes
Azure Disk	Yes	No	No
Cinder	Yes	No	No
Fibre Channel	Yes	Yes	No
GCE Persistent Disk	Yes	No	No
HostPath	Yes	No	No

Volume Plug-in	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
iSCSI	Yes	Yes	No
LocalVolume	Yes	No	No
NFS	Yes	Yes	Yes
VMware vSphere	Yes	No	No

**Note**

OpenShift can only mount a persistent volume with one access mode at a time.

For example, OpenShift can mount an iSCSI volume as either `ReadWriteOnce` or `ReadOnlyMany`, but OpenShift cannot simultaneously mount a given iSCSI volume as `ReadWriteOnce` on one node, and `ReadOnlyMany` on other nodes.

Managing Shared Storage Resources

OpenShift uses storage classes to manage different types of storage, including shared storage. Each storage class defines a `provisioner` attribute. A provisioner automates many persistent volume life cycle tasks, namely creating a persistent volume on demand.

OpenShift contains several internal provisioners. However, OpenShift does not provide an internal provisioner for each volume plug-in.

Internal Provisioner support

Volume Plug-in	Internal Provisioner	Storage Class Provisioner Value
AWS EBS	Yes	<code>kubernetes.io/aws-ebs</code>
Azure File	Yes	<code>kubernetes.io/azure-file</code>
Azure Disk	Yes	<code>kubernetes.io/azure-disk</code>
Cinder	Yes	<code>kubernetes.io/cinder</code>
Fibre Channel	No	
GCE Persistent Disk	Yes	<code>kubernetes.io/gce-pd</code>
HostPath	No	
iSCSI	No	
LocalVolume	No	
NFS	No	
VMware vSphere	Yes	<code>kubernetes.io/vsphere-volume</code>

**Note**

If you create a storage class for a volume plug-in that does not have a corresponding provisioner, use a storage class provisioner value of `kubernetes.io/no-provisioner`.

OpenShift contains two built-in volume plug-ins for shared storage: NFS and Azure File. OpenShift has built-in support to provision Azure File persistent volumes dynamically, but not NFS persistent volumes.

Additional shared storage options are available in OpenShift. To use these options, you must install additional components in the cluster.

Managing Azure File Persistent Volumes

OpenShift has a built-in provisioner for Azure File persistent volumes. To use Azure File volumes in OpenShift, first, create a secret that contains credentials for your Azure Storage account. Second, create a storage class for each Azure File tier you require for the OpenShift cluster.

Each storage class definition contains a reference to the Azure storage credentials secret. OpenShift uses the Azure File provisioner to manage all Azure File persistent volumes in the cluster. An example Azure File storage class definition follows:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: azurefile-lrs 1
provisioner: kubernetes.io/azure-file 2
parameters:
  skuName: Standard_LRS 3
  storageAccount: azure_storage_account_name 4
  secretNamespace: my-secret-namespace 5
  secretName: azure-storage-credentials 6
```

- 1** The name of the storage class. To request this type of storage, a persistent volume claim uses a `StorageClassName` value of `azurefile-lrs`.
- 2** A provisioner value of `kubernetes.io/azure-file` instructs OpenShift to use the Azure Files provisioner to create persistent volumes. The Azure File provisioner uses values in the `parameters` section to create persistent volumes with specific characteristics.
- 3** The `skuName` parameter specifies an Azure File storage tier. The `Standard_LRS` value corresponds to standard locally-redundant storage (LRS).
- 4** The provisioner requires an account name to authenticate to the Azure Files API.
- 5** **6** The provisioner requires sensitive credentials to authenticate to the Azure Files API. The provisioner retrieves sensitive credentials from a secret resource and then uses them to authenticate to the Azure Files API.

Managing NFS Persistent Volumes

NFS is another option for shared file storage in OpenShift. OpenShift does not, however, have a built-in NFS provisioner.

You have two options to manage NFS persistent volumes:

- Create and use playbooks (or other automation) to provision a pool of static NFS persistent volumes**

Write playbooks to create a set of NFS storage classes and persistent volumes. Ensure each persistent volume defines a storage class and configures NFS mount options according to that storage class.

If the cluster requires additional NFS persistent volumes, then rerun the playbook with updated parameters.

- Add external components, such as an operator or custom provisioner, to automate the management of NFS persistent volumes**

Many third-party and open source solutions exist to help with the management of NFS shares.

Red Hat Container Storage uses operators to manage the dynamic provisioning of NFS persistent volumes.



Note

The classroom environment uses an external, open-source NFS provisioner. The provisioner dynamically creates NFS persistent volumes from an existing NFS server. See the *References* for more information.

Red Hat does not recommend using this provisioner in production environments.

Providing Other Shared Storage Resources

Red Hat works with partners and the open-source community to provide other shared storage solutions. Use the OpenShift marketplace to install other storage operators that can manage shared persistent volumes resources.

Setting a Default Storage Class

To simplify storage requests, OpenShift allows users to create a PVC that does not specify a storage class. With this feature, users do not need to know the various storage tiers that are available in a particular OpenShift cluster.

If a PVC does not specify a storage class, then OpenShift tries to match the PVC to a persistent volume from the default storage class. You must designate an appropriate default storage class for the OpenShift cluster. Use a default storage class that is scalable, cost-effective, and easy to manage.

The default storage class resource for a cluster contains a specific metadata annotation:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" ①
...output omitted...
```

- ❶ The default storage class must define a `storageclass.kubernetes.io/is-default-class` annotation, which has a value of "true".

Use the `oc annotate` command to create or update an annotation. For example, the following command sets the standard storage class as the default storage class:

```
[user@host ~]$ oc annotate storageclass standard \
--overwrite "storageclass.kubernetes.io/is-default-class=true"
```

You can also store the annotation YAML snippet in a file and use the content of the file to patch the storage class resource:

```
[user@host ~]$ cat set_default.yml
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"

[user@host ~]$ oc patch storageclass standard -p "$(cat set_default.yml)"
```

OpenShift labels the default storage class in the output of the `oc get storageclass` command:

```
[user@host ~]$ oc get storageclasses
NAME          PROVISIONER      RECLAIMPOLICY   ...
standard (default) nfs-storage   Delete         ...
gold           azure-file-storage Delete         ...
```

You must define only one default storage class. If more than one default exists, then OpenShift does not provision a persistent volume for a PVC that does not specify a storage class.

Restricting Access to Storage Resources

Because all storage resources are not the same, you might need to implement restrictions on storage resources in the cluster.

Use a resource quota to restrict resource consumption per project. Quotas are often used to restrict CPU and memory usage in a project, but you can also use quotas to restrict storage resource usage.

For example, consider a hypothetical cluster that provides both a standard and a fast storage class. The standard storage class provides inexpensive yet scalable storage, with average I/O performance. The fast storage class provides low latency and high throughput storage, but it is expensive. To encourage users to consume standard storage resources, you must use a resource quota to restrict fast storage resources.

Storage Resources Managed by Resource Quotas

Resource Name	Description
<code>requests.storage</code>	The sum of storage space requests across all persistent volume claims can not exceed this value.

Resource Name	Description
<code>persistentvolumeclaims</code>	The total number of persistent volume claim resources that can exist in a project.
<code><storage-class-name>.storageclass.storage.k8s.io/requests.storage</code>	The sum of storage space requests for the across all persistent volume claims, but only for the <code><storage-class-name></code> storage class.
<code><storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims</code>	The total number of persistent volume claim resources for the <code><storage-class-name></code> storage class that can exist in a project.

An example resource quota follows:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage ①
  namespace: test ②
spec:
  hard:
    persistentvolumeclaims: 4 ③
    requests.storage: 100G ④
```

① ② A 'storage' resource quota that restricts resources for the `test` namespace (project).

- ③ The quota restricts the project to four persistent volume claims.
- ④ The quota restricts the project to 100 GB of total persistent storage.

You can also use the `oc create quota` command to create a resource quota from the command line.



References

For more information about Persistent Volume access modes, refer to the *Access Modes* section in the *Understanding Persistent Storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/index#pv-access-modesUnderstanding-persistent-storage

For more information about Azure File persistent storage, refer to the *Persistent storage using Azure File* section in the *Configuring persistent storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/configuring-persistent-storage#persistent-storage-using-azure-file

For more information about NFS persistent storage, refer to the *Persistent storage using NFS* section in the *Configuring persistent storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/configuring-persistent-storage#persistent-storage-using-nfs

For more information about resource quotas, refer to the *Resource Quotas per Project* section in the *Quotas* chapter in the Red Hat OpenShift Container Platform 4.6 Applications documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/applications/index#quotas-setting-per-project

Kubernetes NFS Subdir External Provisioner

<https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner>

► Guided Exercise

Provisioning Shared Storage for Applications

In this exercise you will deploy a back-end batch processing job to access shared file storage.

Outcomes

You should be able to:

- Identify the default storage class for an OpenShift cluster.
- Set the default storage class for an OpenShift cluster.
- Create a storage quota for a project.
- Create a Persistent Volume claim for shared storage.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the **workstation** machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable and that no default storage class exists.

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

Instructions

- 1. As the `admin` user, create a `storage-file` project with a restriction of three persistent volume claims and 5 GB of storage. Add the `developer` user as a project administrator.
- 1.1. Log in to the cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Create the `storage-file` project.

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

- 1.3. Create a storage quota for the project. Restrict the project to 3 persistent volume claims and 5 GB of storage.

```
[student@workstation ~]$ oc create quota storage \
--hard=requests.storage=5G,persistentvolumeclaims=3
resourcequota/storage created
```

- 1.4. Add the developer user as a project administrator.

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



Note

If you have not previously authenticated as the developer user, then you will see a warning message that indicates the developer user is not found.

OpenShift creates a developer user resource only after the developer user authenticates for the first time.

- 2. The developer user manages an application that requires shared storage. The application consists of three components: a HTTPD front end, a back-end image processing application, and a Redis service to manage a work queue for the back-end application. The ~/D0380/labs/storage-file directory contains resource definitions to deploy these components.

The front-end application defines a persistent volume claim for shared storage in the resources/httpd/pvc.yml file. Create a photo-share persistent volume claim from the resources/httpd/pvc.yml file. Expect the photo-share persistent volume claim to have a Pending status.

- 2.1. Change to the ~/D0380/labs/storage-file directory.

```
[student@workstation ~]$ cd ~/D0380/labs/storage-file
```

- 2.2. Log in to the cluster as the developer user.

```
[student@workstation storage-file]$ oc login -u developer -p developer
Login successful.
...output omitted...
```

- 2.3. Switch to the storage-file project.

```
[student@workstation storage-file]$ oc project storage-file
Already on project "storage-file" on server "https://api.ocp4.example.com:6443".
```

- 2.4. Display and review the contents of the resources/httpd/pvc.yml file.

```
[student@workstation storage-file]$ cat resources/httpd/pvc.yml
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  name: photo-share
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi
```

- 2.5. Use the `resources/httpd/pvc.yml` file to create the `photo-share` persistent volume claim.

```
[student@workstation storage-file]$ oc apply -f resources/httpd/pvc.yml
persistentvolumeclaim/photo-share created
```

- 2.6. Display the status of all persistent volume claims in the project.

```
[student@workstation storage-file]$ oc get persistentvolumeclaims
NAME      STATUS      VOLUME      CAPACITY      ACCESS MODES      STORAGECLASS      ...
photo-share  Pending
```

- 2.7. Use the `oc describe` command to display events for the `photo-share` persistent volume claim.

```
[student@workstation storage-file]$ oc describe persistentvolumeclaim photo-share
...output omitted...
Events:
  Type    Reason          ...    Message
  ----  -----          ...    -----
  Normal  FailedBinding  ...    No persistent volumes available for this claim and
  no storage class is set
```

The `resources/httpd/pvc.yml` file does not declare a storage class, and a default storage class is not defined for the cluster.

- 3. Edit the `resources/httpd/pvc.yml` file to specify shared file storage from the cluster NFS storage provider. After you edit the file, reprovise the `photo-share` persistent volume claim.

- 3.1. Display the list of available storage classes.

```
[student@workstation storage-file]$ oc get storageclasses
NAME      PROVISIONER          RECLAIMPOLICY      ...
nfs-storage  k8s-sigs.io/nfs-subdir-external-provisioner  Delete      ...
```

- 3.2. Use your preferred editor to modify the `resources/httpd/pvc.yml` file. Specify the `nfs-storage` storage class for the `photo-share` persistent volume:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: photo-share
```

```
spec:  
  accessModes:  
    - ReadWriteMany  
storageClassName: nfs-storage  
  resources:  
    requests:  
      storage: 3Gi
```

Save the changes.

- 3.3. Delete the existing photo-share persistent volume claim. Next, use the modified resources/httpd/pvc.yml file to re-create the photo-share persistent volume claim.

```
[student@workstation storage-file]$ oc delete persistentvolumeclaim photo-share  
persistentvolumeclaim "photo-share" deleted  
  
[student@workstation storage-file]$ oc apply -f resources/httpd/pvc.yml  
persistentvolumeclaim/photo-share created
```

- 3.4. Use the oc get persistentvolumeclaims command to verify that an NFS persistent volume binds to the photo-share persistent volume claim.

```
[student@workstation storage-file]$ oc get persistentvolumeclaims  
NAME      STATUS  VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE  
photo-share  Bound   pvc-...  3Gi       RWX           nfs-storage  10s
```

- 4. Deploy the front-end HTTPD application and Redis service. Execute the simulate_activity.sh script to simulate users uploading photos from the front-end application. After the simulation, one hundred photos are present on the NFS persistent volume.

- 4.1. The resources/httpd directory contains resource definitions for the front-end application. Use the oc apply -k command to deploy the front-end application.

```
[student@workstation storage-file]$ oc apply -k resources/httpd/  
service/httpd created  
deployment.apps/httpd created  
route.route.openshift.io/httpd created  
persistentvolumeclaim/photo-share configured
```

The front-end application exposes a route. If you navigate to the URL, then you see that no images exist in the data directory.

- `http://httpd-storage-file.apps.ocp4.example.com/data`

- 4.2. The resources/redis directory contains resource definitions for the Redis service, which hosts a work queue for the back-end application. Use the oc apply -k command to deploy the Redis service.

```
[student@workstation storage-file]$ oc apply -k resources/redis/  
service/redis created  
deployment.apps/redis created
```

- 4.3. Wait for the Redis pod to become ready.

```
[student@workstation storage-file]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
httpd-5c57dc69f-vnl7p   1/1     Running   0          2m
redis-59d8d5c4bb-xcsgn  1/1     Running   0          115s
```

**Note**

If you execute the `show_queue.sh` script after the Redis service is ready, then the output indicates that the processing queue is an empty array.

- 4.4. Execute the `./simulate_activity.sh` script to simulate users uploading photos from the front-end application. The script uploads 100 photos to the `uploads` subdirectory on the shared volume.

```
[student@workstation storage-file]$ ./simulate_activity.sh
HTTP pod: httpd-5c57dc69f-vnl7p
Redis pod: redis-59d8d5c4bb-xcsgn
df2212f6e1ad2862.jpg
df8a8403e4545ab5.jpg
...output omitted...
97
98
99
100
```

If you navigate to the URL in a browser, then a `uploads` subdirectory is present with 100 images.

- `http://httpd-storage-file.apps.ocp4.example.com/data`

If you execute the `show_queue.sh` script, then the same set of image file names is present in the processing queue.

- 5. Deploy the back-end image processing job to detect objects in uploaded photos. Verify that all three back-end worker pods are able to process files from the shared persistent volume.

- 5.1. Use the `oc apply -k` command to deploy the resources in the `resources/backend` subdirectory. Wait until the worker pods have a `Running` status. The pods may be in a `ContainerCreating` status for a couple minutes while the application container image downloads to each node from the registry.

```
[student@workstation storage-file]$ oc apply -k resources/backend/
job.batch/worker created
```

```
[student@workstation storage-file]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
httpd-5c57dc69f-dh6ms   1/1     Running   0          39m
redis-59d8d5c4bb-zrtcz  1/1     Running   0          38m
redis-client        1/1     Running   0          10m
```

worker-4kv28	1/1	Running	0	59s
worker-h7z65	1/1	Running	0	58s
worker-v6n8h	1/1	Running	0	59s

- 5.2. The job completes approximately two minutes after you started it. Wait for the job to finish.

```
[student@workstation storage-file]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
httpd-5c57dc69f-dh6ms  1/1     Running   0          41m
redis-59d8d5c4bb-zrtcz 1/1     Running   0          40m
redis-client      1/1     Running   0          12m
worker-4kv28       0/1     Completed  0          3m5s
worker-h7z65       0/1     Completed  0          3m4s
worker-v6n8h       0/1     Completed  0          3m5s
```

- 5.3. Review the logs of each worker pod to verify that each worker pod processes files from the shared storage volume.

```
[student@workstation storage-file]$ oc logs worker-4kv28 | tail
...output omitted...
Working on /data/uploads/e0d3452ed12d6c68.jpg
Working on /data/uploads/e0581d87281bb10a.jpg
Working on /data/uploads/df8a8403e4545ab5.jpg
Waiting for work
Queue empty, exiting

[student@workstation storage-file]$ oc logs worker-4kv28 | \
grep -i "Working on" | wc -l
34

[student@workstation storage-file]$ oc logs worker-h7z65 | \
grep -i "Working on" | wc -l
33

[student@workstation storage-file]$ oc logs worker-v6n8h | \
grep -i "Working on" | wc -l
33
```

The three worker pods are able to access shared file storage to process all one hundred image files.

- 6. As the `admin` user, set the `nfs-storage` storage class as the default storage class. When you set a default storage class, cluster users do not need to specify a storage class to create a persistent volume claim.

- 6.1. Log in as the `admin` user.

```
[student@workstation storage-file]$ oc login -u admin -p redhat
Login successful.

...output omitted...
```

Chapter 8 | Configuring Persistent Storage

- 6.2. Review the nfs-storage storage class resource definition. Verify that the nfs-storage storage class is not the default storage class.

```
[student@workstation storage-file]$ oc get storageclass nfs-storage -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
  creationTimestamp: "2020-05-28T19:33:31Z"
...output omitted...
```

The storageclass.kubernetes.io/is-default-class annotation is set to a value of false.

- 6.3. Display and review the contents of the set-default-storageclass.yaml patch file.

```
[student@workstation storage-file]$ cat set-default-storageclass.yaml
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
```

- 6.4. Use the contents of the set-default-storageclass.yaml patch file to set the nfs-storage storage class as the default storage class.

```
[student@workstation storage-file]$ oc patch storageclass nfs-storage \
  -p "$(cat set-default-storageclass.yaml)"
storageclass.storage.k8s.io/nfs-storage patched
```

- 6.5. Verify that the nfs-storage storage class is the default storage class for the cluster. Also verify that only one storage class is labeled as the default storage class.

```
[student@workstation storage-file]$ oc get storageclasses
NAME          PROVISIONER          RECLAIMPOLICY  ...
nfs-storage  (default)  nfs-storage-provisioner  Delete        ...
```

The nfs-storage storage class is now the default storage class for the cluster.

- 7. Change to the /home/student directory.

```
[student@workstation storage-file]$ cd
```

Finish

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

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

Provisioning Block Storage for Databases

Objectives

After completing this section, you should be able to configure a database application with block storage.

Overview of Block Storage

Block storage refers to any storage technology that stores data in fixed-size blocks. Common examples include hard disk drives (HDDs) and solid-state drives (SSDs).

Block storage is not restricted to physically attached hardware. A Storage Area Network (SAN) presents storage to networked systems as locally attached block devices. Network protocols, such as iSCSI and Fibre Channel, provide block-level access to storage devices.

Most cloud providers also have scalable, block storage solutions. Examples include Amazon Elastic Block Store (EBS), Google Persistent Disk, and Azure Disks.

Describing Block Storage Use Cases

Block storage technologies are designed for speed and efficiency. The storage system writes a data record to multiple blocks. When a data record is needed, the storage system retrieves multiple blocks. The blocks are reassembled and presented to the requesting user as a complete data record.

Databases and transactional systems are examples of applications that require high throughput and I/O performance. Many of these systems use block storage technologies because of the increased I/O performance requirements.

There are a few drawbacks to block storage:

- Block storage is not a shared read-write storage technology. Only one host at a time can write data.
- Block storage can be expensive.
- Block storage does not provide robust metadata about stored data. Adding a file system to a block device provides additional metadata for each record, such as a file name or permissions.

Accessing Block Storage

In many application scenarios, a file system is installed on a block storage device. Applications use the file system to read, write, and organize data on the block device. Without a file system, the application is responsible for organizing and writing data to the block device.

In select scenarios, applications require direct access to the block device. With direct access, applications bypass file system overhead to achieve improved I/O performance. These applications must implement custom I/O capabilities, which are tuned for increased performance. As an example, some database technologies configure raw block devices rather than specifying a database data directory.

Kubernetes and OpenShift support both block and file system persistent volumes for block storage technologies. Use the `volumeMode` attribute in a persistent volume specification to indicate the type of volume. Allowable values for the volume mode are: `Filesystem` and `Block`.

Block Volumes

As an example, the following defines a fibre channel block volume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fc-block-pv-01 ①
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  storageClassName: fc-block ②
  volumeMode: Block ③
  fc: ④
    targetWWNs: ["500...fd1"]
    lun: 0
  readOnly: false
```

- ① The name of the persistent volume.
- ② The storage class name is `fc-block`, which indicates a fibre channel (`fc`) block volume.
- ③④ OpenShift uses the fibre channel volume plug-in to mount the volume on a node.

To request a block volume, an application developer must create a PVC that declares a `Block` volume mode. The following example defines a request for a block volume.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block ①
  resources:
    requests:
      storage: 10Gi
  storageClassName: fc-block ②
```

- ① The volume mode must be set to a value of `Block`. If the PVC definition does not include a volume mode, the default value is `Filesystem`.
- ② The persistent volume must come from the `fc-block` storage class. If you do not specify a storage class, then OpenShift uses the default storage class, which might not support block volumes.

To bind to a block volume, both the PVC and PV must explicitly define a `Block` volume mode. The table that follows outlines all possible binding scenarios for PV and PVC volume mode values.

Binding scenarios by Volume Mode

PVC Volume Mode	PV Volume Mode	Result
Unspecified	Block	Does not bind.
Filesystem	Block	Does not bind.
Block	Block	The PV binds to the PVC.
Unspecified	Unspecified	The PV binds to the PVC, but the PV is not a block volume. Unspecified values default to Filesystem.

If an application requires access to a raw block device, rather than a file system, you must modify the pod specification. The following pod specification attaches a raw block device at the `/dev/xvda` device path.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
    ...output omitted...
    volumeDevices: ①
      - name: data ②
        devicePath: /dev/xvda ③
  volumes:
    - name: data ④
      persistentVolumeClaim:
        claimName: block-pvc ⑤
```

- ① Use `volumeDevices` to attach block volumes as a raw block device.
- ②③ Attach the `data` volume as a block device at the `/dev/xvda` device path.
- ④⑤ The `data` volume corresponds to persistent volume bound to the `block-pvc` persistent volume claim.

File System Volumes for Block Storage

Many applications require access to a file system, not a raw block device. For this reason, Kubernetes and OpenShift use a default volume mode of `Filesystem` for PVs and PVCs.

Both shared file storage and block storage provide file system volumes. A file system volume, which is based on block storage, often provides better I/O performance than volumes based on shared file storage. To request block storage, an application developer must specify a storage class that corresponds to block storage. If a storage class is omitted, then OpenShift uses the default storage class, which might not correspond to a block storage technology.

**Note**

In this course, all example applications request file system storage.

Sharing Block Storage

Unlike shared file storage, block storage technologies only attach to a single host at a time. Some block storage technologies can attach to multiple hosts with read-only access. However, block storage technologies restrict write access to a single host to ensure data integrity.

All block storage in Kubernetes and OpenShift has a `ReadWriteOnce` access mode. Some technologies, such as iSCSI, also support `ReadOnlyMany` access mode.

If an application, such as a database, requires write access to block storage, then the persistent volume claim must request an access mode of `ReadWriteOnce`. Because the volume only attaches to one node at a time, you can safely use a volume with a single pod.

To share a block storage volume with a different application, you must stop the current application that is using the volume. For example, to restore a database from a backup, you must first scale down the database to zero pods. A restoration application can mount the database volume after the database is scaled to zero.

Before the database application is restarted, the restoration application must finish and release the database volume.

Restricting Allocation of Block Storage Resources

Block storage resources are often expensive, and as a result, a limited cluster resource. You might want to restrict the use of block storage resources on a project-by-project basis. In this way, you ensure the availability of limited storage resources to those projects with unique storage requirements.

Use resource quotas for each project to restrict the use of particular storage resources. The following resource quota definition restricts the database project to a single block storage persistent volume.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage ①
  namespace: test ②
spec:
  hard:
    fc-block.storageclass.storage.k8s.io/persistentvolumeclaims: 1 ③
    fc-block.storageclass.storage.k8s.io/requests.storage: 100G ④
```

① ② The storage resource quota restricts storage resources for the `test` project.

③ The `test` project is restricted to one persistent volume claim for the `fc-block` storage class.

④ The `test` project is restricted to 100 GB of storage from the `fc-block` storage class.

You can also restrict access to storage class resources entirely.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  namespace: dev ①
spec:
  hard:
    fc-block.storageclass.storage.k8s.io/persistentvolumeclaims: 0 ②
```

- ① ②** The dev project cannot create a persistent volume claim that requests storage for the `fc-block` storage class.



References

For more information about block volumes, refer to the *Block Volume Support* section in the *Understanding Persistent Storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/index#block-volume-supportUnderstanding-persistent-storage

For more information about persistent iSCSI storage, refer to the *Persistent Storage Using iSCSI* section in the *Configuring persistent storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/configuring-persistent-storage#persistent-storage-using-iscsi

► Guided Exercise

Provisioning Block Storage for Databases

In this exercise you will create block storage resources for the cluster and deploy a database that uses block storage.

Outcomes

You should be able to:

- Create a storage class to make block storage resources visible to non-privileged cluster users.
- Use a storage class in a persistent volume claim to request block storage.
- Create a storage quota that restricts storage resources by storage class.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

On the **workstation** machine, use the `lab` command to prepare your system for this exercise. The command ensures that the cluster API is reachable.

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

Instructions

- 1. The `~/D0380/labs/storage-block/iscsi` directory contains playbooks to configure iSCSI block storage resources for the cluster. Review and execute the `site.yml` playbook.
- 1.1. Change to the `~/D0380/labs/storage-block/iscsi` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/storage-block/iscsi
```

- 1.2. Display the `site.yml` file.

```
[student@workstation iscsi]$ cat site.yml
- name: Install iSCSI packages
  import_playbook: install-iscsi.yml

- name: Configure a logical volume for each PV
  import_playbook: configure-lvm.yml

- name: Configure each logical volume as an iSCSI LUN
  import_playbook: configure-iscsi.yml
```

```
- name: Create a PV resource file for each LUN
  import_playbook: render-templates.yml
```

The `site.yml` file executes four other playbooks:

- The `install-iscsi.yml` playbook ensures that iSCSI packages are installed on the iSCSI server.
- The `configure-lvm.yml` playbook configures Logical Volume Management on the iSCSI server.

Each persistent volume resource corresponds to one logical volume on the iSCSI server. The playbook creates the logical volumes in the `iSCSI_vg` volume group.

- The `configure-iscsi.yml` playbook configures each logical volume in the `iSCSI_vg` volume group as an iSCSI LUN.
- The `render-templates.yml` playbook creates a persistent volume resource file in the `~/D0380/labs/storage-block/PVs` directory for each iSCSI LUN. The playbook also creates a `storageclass.yml` file and a `kustomization.yml` file in the same directory.

1.3. Execute the `site.yml` playbook.

The first time you execute the playbook, the task "Check if iSCSI target already seems configured" fails. You can safely ignore this error.

```
[student@workstation iscsi]$ ansible-playbook site.yml
...output omitted...

TASK [Render PV YAML files] ****
changed: [workstation] => (item=1G)
changed: [workstation] => (item=2G)
...output omitted...

TASK [Render kustomization.yml] ****
changed: [workstation] => (item=kustomization.yml)
changed: [workstation] => (item=storageclass.yml)

PLAY RECAP ****
utility.lab.example.com  : ok=14  changed=11  ...  rescued=1    ignored=0
workstation              : ok=3    changed=3  ...  rescued=0    ignored=0
```

- ▶ **2.** The `render-templates.yml` playbook creates a resource file for each persistent volume in the `~/D0380/labs/storage-block/PVs` directory. Review one of the persistent volume resource files. Use the `oc apply -k` command to create persistent volumes from the resource files, and then verify that iSCSI persistent volumes exist in the cluster.
- 2.1. Change to the `~/D0380/labs/storage-block/PVs` directory. List the files in the directory.

```
[student@workstation iscsi]$ cd ~/D0380/labs/storage-block/PVs

[student@workstation PVs]$ ls
iscsi_pv_00.yml  iscsi_pv_03.yml  iscsi_pv_06.yml  kustomization.yml
iscsi_pv_01.yml  iscsi_pv_04.yml  iscsi_pv_07.yml  storageclass.yml
iscsi_pv_02.yml  iscsi_pv_05.yml  iscsi_pv_08.yml
```

2.2. Display the content of the `iscsi_pv_00.yml` file.

```
[student@workstation PVs]$ cat iscsi_pv_00.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv-00 ①
spec:
  capacity:
    storage: 1Gi ②
  volumeMode: Filesystem
  storageClassName: iscsi-blk ③
  accessModes:
    - ReadWriteOnce
  iscsi: ④
    targetPortal: 172.25.250.253:3260
    iqn: iqn.2020-06.com.example:utility.lab.example.com
    lun: 0
    initiatorName: iqn.2020-06.com.example:openshift
    fsType: 'ext4'
    readOnly: false
```

- ① The persistent volume name is `iscsi-pv-00`.
- ② The size of persistent volume is set to equal the size of the corresponding logical volume on the iSCSI server.
- ③ The persistent volume has a storage class label of `iscsi-blk`.
- ④ The persistent volume uses the iSCSI volume plug-in. This section defines parameters that OpenShift requires to mount an iSCSI volume. Each persistent volume uses a different iSCSI LUN.

2.3. Log in as the `admin` user.

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

...output omitted...
```

2.4. Use the `oc apply -k` command to create persistent volumes from the resource definitions in the `~/D0380/labs/storage-block/PVs` directory.

```
[student@workstation PVs]$ oc apply -k ~/DO380/labs/storage-block/PVs
persistentvolume/iscsi-pv-00 created
persistentvolume/iscsi-pv-01 created
persistentvolume/iscsi-pv-02 created
persistentvolume/iscsi-pv-03 created
persistentvolume/iscsi-pv-04 created
persistentvolume/iscsi-pv-05 created
persistentvolume/iscsi-pv-06 created
persistentvolume/iscsi-pv-07 created
persistentvolume/iscsi-pv-08 created
```

- 2.5. Verify that nine persistent volumes exist with an `iscsi-blk` storage class attribute value.

```
[student@workstation PVs]$ oc get persistentvolumes \
-o=custom-columns='NAME:metadata.name,STORAGECLASS:spec.storageClassName'
NAME                      STORAGECLASS
iscsi-pv-00                iscsi-blk
iscsi-pv-01                iscsi-blk
iscsi-pv-02                iscsi-blk
iscsi-pv-03                iscsi-blk
iscsi-pv-04                iscsi-blk
iscsi-pv-05                iscsi-blk
iscsi-pv-06                iscsi-blk
iscsi-pv-07                iscsi-blk
iscsi-pv-08                iscsi-blk
...output omitted...
```

- 2.6. Display the list of storage classes.

```
[student@workstation PVs]$ oc get storageclasses
NAME          PROVISIONER          RECLAIMPOLICY
...
nfs-storage  (default)  k8s-sigs.io/nfs-subdir-external-provisioner  Delete
...
```

An `iscsi-blk` storage class resource does not exist, yet nine persistent volumes exist with an `iscsi-blk` storage class label.



Note

If a persistent volume claim contains a `storageClassName` value of `iscsi-blk`, then OpenShift binds one of the nine `iscsi-blk` persistent volumes to the claim.

Non-privileged users do not know that iSCSI persistent volumes exist in the cluster because they can not inspect persistent volumes.

You did not create an `iscsi-blk` storage class resource in this step to simulate the preceding scenario. In a later step, you request iSCSI storage as a non-privileged user when no iSCSI storage class exists.

**Important**

You must create an iSCSI storage class resource to ensure that non-privileged users know that iSCSI storage is available in the cluster. You create the `iscsi-blk` storage class resource in a later step.

- 3. As the `admin` user, create a `storage-block` project with a restriction of two persistent volume claims and 6 GB of storage. Further restrict the project to a single persistent volume claim for the `iscsi-blk` storage class. Add the `developer` user as a project administrator.

- 3.1. Create the `storage-block` project.

```
[student@workstation PVs]$ oc whoami
admin

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

- 3.2. Create a `storage quota` resource for the project. Restrict the project to: 2 persistent volume claims, 6 GB of requested storage, and 1 persistent volume claim for the `iscsi-blk` storage class.

```
[student@workstation PVs]$ limit1="requests.storage=6G"
[student@workstation PVs]$ limit2="persistentvolumeclaims=2"
[student@workstation PVs]$ sclass="iscsi-blk.storageclass.storage.k8s.io"
[student@workstation PVs]$ limit3="${sclass}/persistentvolumeclaims=1"
[student@workstation PVs]$ oc create quota storage \
  --hard=${limit1},${limit2},${limit3}
resourcequota/storage created
```

- 3.3. Use the `oc describe` command to review the `storage quota` resource.

```
[student@workstation PVs]$ oc describe quota storage
Name:                                     storage
Namespace:                                storage-block
Resource          Used   Hard
-----
iscsi-blk.storageclass.storage.k8s.io/persistentvolumeclaims  0     1   ①
persistentvolumeclaims                      0     2   ②
requests.storage                            0     6G  ③
```

- ① The project is restricted to one persistent volume claim for the `iscsi-blk` storage class.
- ② The project is restricted to two persistent volume claims, regardless of storage class.
- ③ The project is restricted to 6 GB of requested storage, across all persistent volume claims.

3.4. Add the developer user as a storage-block project administrator.

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

**Note**

If you have not previously authenticated as the developer user, then you see a warning message that indicates the developer user is not found.

You can safely ignore this message.

- ▶ 4. The developer user manages a set of applications to operate a database. The database requires block storage, although the backup and restore applications can use shared storage.

As the developer user, switch to the storage-block project. Review storage resources that are available to the storage-block project.

4.1. Login as the developer user and ensure you are using the storage-block project.

```
[student@workstation PVs]$ oc login -u developer -p developer
Login successful.
```

```
You have one project on this server: "storage-block"
```

```
Using project "storage-block".
```

4.2. Display a list of persistent volumes.

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

**Note**

Persistent volume resources can contain sensitive information, such as connection credentials to remote storage. For this reason, non-privileged users do not have access to persistent volume resources.

4.3. Display the list of storage class resources.

```
[student@workstation PVs]$ oc get storageclasses
NAME          PROVISIONER          RECLAIMPOLICY
nfs-storage   (default)   k8s-sigs.io/nfs-subdir-external-provisioner Delete
```

Only the nfs-storage storage class resource exists. The developer user is not aware that iSCSI persistent volumes exist in the cluster.

**Note**

Each of the iSCSI persistent volumes declare a storage class value of `iscsi-blk`.

OpenShift does not create a storage class resource from persistent volume resource definitions. As an administrator, you must ensure a storage class resource exists for any persistent volumes that you manually provision.

- 4.4. As the developer user, use the `oc describe` command to review the storage quota resource.

```
[student@workstation PVs]$ oc describe quota storage
Name:                                     storage
Namespace:                                storage-block
Resource          Used   Hard
-----
iscsi-blk.storageclass.storage.k8s.io/persistentvolumeclaims  0     1  ⓘ
persistentvolumeclaims                      0     2
requests.storage                           0     6G
```

- ① For the developer user, the storage quota provides the only indication that an iSCSI storage class exists.
- ▶ 5. The `~/D0380/labs/storage-block/test-db` directory contains resource definitions to deploy database management applications.

The `postgres` subdirectory contains resource definition files to deploy a Postgresql database. Ensure that the database uses iSCSI block storage, which is defined in the `postgres/statefulset.yml` file. Deploy the database.

Inspect the storage quota object, which updates after you deploy the database.

- 5.1. Change to the `~/D0380/labs/storage-block/test-db` subdirectory.

```
[student@workstation PVs]$ cd ~/D0380/labs/storage-block/test-db
```

- 5.2. Edit the `volumeClaimTemplates` section of the `postgres/statefulset.yml` file to match the following:

```
volumeClaimTemplates:
- metadata:
  name: data
spec:
  storageClassName: iscsi-blk
  accessModes: [ "ReadWriteOnce" ]
  resources:
    requests:
      storage: 2Gi
```

Save the file.

**Note**

A solution file is available at: ~/D0380/solutions/storage-block/test-db/postgres/statefulset.yml.

5.3. Deploy the database.

```
[student@workstation test-db]$ oc apply -k postgres/
secret/postgresql created
service/postgresql created
statefulset.apps/postgresql created
```

5.4. Verify that the persistent volume claim binds to one of the iSCSI persistent volumes.

```
[student@workstation test-db]$ oc get persistentvolumeclaims
NAME          STATUS  VOLUME      CAPACITY  ACCESS MODES  STORAGECLASS ...
data-postgresql-0  Bound   iscsi-pv-01  2Gi       RWO          iscsi-blk     ...
```

5.5. Inspect the storage quota resource after deploying the database.

```
[student@workstation test-db]$ oc describe quota storage
Name:                                     storage
Namespace:                                storage-block
Resource          Used   Hard
-----
iscsi-blk.storageclass.storage.k8s.io/persistentvolumeclaims  1    1  ①
persistentvolumeclaims                         1    2
requests.storage                               2Gi  6G
```

- ① The database persistent volume claim increments the persistent volume claim count for the `iscsi-blk` storage class.

► 6. As the `admin` user, create an `iscsi-blk` storage class resource.

The `~/D0380/labs/storage-block/PVs/storageclass.yml` file contains a resource definition for the `iscsi-blk` storage class.

6.1. Change to the `~/D0380/labs/storage-block/PVs` directory. Display the contents of the `storageclass.yml` file.

```
[student@workstation test-db]$ cd ~/D0380/labs/storage-block/PVs
[student@workstation PVs]$ cat storageclass.yml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: iscsi-blk ①
provisioner: kubernetes.io/no-provisioner ②
allowVolumeExpansion: false
```

- ① The storage class name is `iscsi-blk`.
② The storage class does not provision volumes on demand.

6.2. Log in as the `admin` user.

```
[student@workstation PVs]$ oc login -u admin
Logged into "https://api.ocp4.example.com:6443" as "admin" ...
...output omitted...
```

6.3. Use the `storageclass.yml` file to create the `iscsi-blk` storage class.

```
[student@workstation PVs]$ oc create -f storageclass.yml
storageclass.storage.k8s.io/iscsi-blk created
```

- 7. As the `developer` user, verify that the `iscsi-blk` storage class displays in the storage class list.

7.1. Log in as the `developer` user, and then switch to the `storage-block` project.

```
[student@workstation PVs]$ oc login -u developer
Logged into "https://api.ocp4.example.com:6443" as "developer" ...
...output omitted...
[student@workstation PVs]$ oc project storage-block
...output omitted...
```

7.2. Display the list of storage classes.

```
[student@workstation PVs]$ oc get storageclasses
NAME          PROVISIONER           RECLAIMPOLICY  ...
iscsi-blk     kubernetes.io/no-provisioner Delete   ...
nfs-storage   nfs-storage-provisioner Delete   ...
```

- 8. [OPTIONAL] Simulate database operations.

The `~/DO380/labs/storage-block/test-db` directory contains subdirectories, each with resource definitions to deploy a job that implements a specific database operation. These jobs demonstrate how to share different types of persistent volumes between pods.

In the `storage-block` project, use the `oc apply -k` command to:

- Deploy a job to initialize the database. Resources for the initialization job are in the `init_job` subdirectory. This job does not mount any persistent volumes.
- Deploy a database backup job. Resources for the backup job are in the `backup_job` subdirectory. This job mounts a shared storage persistent volume, different than the database block storage persistent volume.
- Deploy a database restoration job. Resources for the database restoration job are in the `restore_job` subdirectory. This job creates a pod that mounts both the shared storage and block storage persistent volumes.

- 8.1. As the developer user, use the `oc apply -k` command to deploy a database initialization job. The initialization job resources are in the `init_job` subdirectory.

```
[student@workstation PVs]$ cd ~/DO380/labs/storage-block/test-db

[student@workstation test-db]$ oc whoami
developer

[student@workstation test-db]$ oc apply -k init_job/
configmap/db-init created
job.batch/database-init created
```

- 8.2. After the initialization job completes, execute the `get_all.sh` script to retrieve all data from the database table.

```
[student@workstation test-db]$ oc get pods
NAME             READY   STATUS    RESTARTS   AGE
database-init-b5sr7   0/1     Completed   0          20s
postgresql-0       1/1     Running   0          18m

[student@workstation test-db]$ ./get_all.sh
You are now connected to database "sampledb" as user "postgres".
id | first_name | last_name | age |      email
---+-----+-----+-----+
 1 | John        | Doe        | 99 | jdoe@example.com
 2 | Jane        | Doe        | 99 | jdoe2@example.com
(2 rows)
```

- 8.3. Use the `oc apply -k` command to deploy a database backup job. The backup job resources are in the `backup_job` subdirectory.

```
[student@workstation test-db]$ oc apply -k backup_job/
serviceaccount/backup created
role.rbac.authorization.k8s.io/pod-exec created
rolebinding.rbac.authorization.k8s.io/pod-exec created
configmap/backup-scripts created
job.batch/backup created
persistentvolumeclaim/data-postgresql-backup created
```

- 8.4. After the backup job completes, execute the `add_new.sh` script to add new data to the database.

```
[student@workstation test-db]$ oc get pods
NAME             READY   STATUS    RESTARTS   AGE
backup-g2d2v      0/1     Completed   0          8s
database-init-b5sr7   0/1     Completed   0          19m
postgresql-0       1/1     Running   0          28m

[student@workstation test-db]$ ./add_new.sh
You are now connected to database "sampledb" as user "postgres".
INSERT 0 1
```

Because the new data is not present in the backup, the database does not contain this data when you restore from the backup.

8.5. Execute the `get_all.sh` script to retrieve all data from the database table.

```
[student@workstation test-db]$ ./get_all.sh
You are now connected to database "sampledb" as user "postgres".
 id | first_name | last_name | age |      email
---+-----+-----+-----+
 1 | John       | Doe        | 99 | jdoe@example.com
 2 | Jane       | Doe        | 99 | jdoe2@example.com
 3 | Anna       | Smith       | 98 | someuser@example.com
(3 rows)
```

8.6. Use the `oc apply -k` command to deploy a database restoration job. The restoration job resources are in the `restore_job` subdirectory.

```
[student@workstation test-db]$ oc apply -k restore_job/
serviceaccount/backup unchanged
role.rbac.authorization.k8s.io/create-jobs created
role.rbac.authorization.k8s.io/scale-statefulsets created
rolebinding.rbac.authorization.k8s.io/create-jobs created
rolebinding.rbac.authorization.k8s.io/scale-statefulsets created
configmap/restore-scripts created
job.batch/restore-controller created
```

8.7. Wait for the restoration job to complete.



Note

The restoration job deploys a `restore-controller` pod. First, the controller pod scales the database pod to zero pods, which takes the database offline. After the database is offline, another pod can mount the database volume.

Second, the controller pod deploys a `restore-db` pod, which mounts the database and backup volumes. The `restore-db` pod then copies the backup archive to the database volume.

Third, the controller pod deletes the `restore-db` pod, which allows the database pod to remount the database volume.

Finally, the controller pod scales the database pod back to one pod. The restoration job is complete when the `restore-controller` pod has a `Completed` status and the `postgresql-0` pod has a `Running` status.

```
[student@workstation test-db]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
backup-g2d2v   0/1     Completed  0          8m14s
database-init-b5sr7 0/1     Completed  0          27m
postgresql-0   1/1     Running   0          33s
restore-controller-527b4 0/1     Completed  0          52s
```

8.8. Execute the `get_all.sh` script to retrieve all data from the restored database table.

```
[student@workstation test-db]$ ./get_all.sh
You are now connected to database "sampledb" as user "postgres".
 id | first_name | last_name | age |      email
----+-----+-----+-----+
 1 | John       | Doe        | 99 | jdoe@example.com
 2 | Jane       | Doe        | 99 | jdoe2@example.com
(2 rows)
```

The database does not contain any data added after the backup job executes.

- ▶ 9. Change to the /home/student directory.

```
[student@workstation test-db]$ cd
```

Finish

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

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

Provisioning Local Block Storage

Objectives

After completing this section, you should be able to use a custom resource to create persistent local volumes.

Describing Local Volumes

Network-based block storage technologies enable remote access to storage with better I/O performance than shared file storage. In some scenarios, applications require better performance.

Locally-attached block devices can often achieve better I/O performance than remote block storage technologies. Kubernetes and OpenShift provide capability to create persistent volumes from locally-attached block devices. The persistent volume definition that follows mounts a locally-attached block device.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv-01
spec:
  capacity:
    storage: 500Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local: ①
    path: /mnt/disks/vol1 ②
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname ③
              operator: In
              values:
                - my-node ④
```

- ① The local volume plug-in defines a path attribute, which is the location of the locally-mounted storage.
- ② The /mnt/disks/vol1 path is the mount point for the locally attached block device. You must ensure the device is formatted and mounted to the node.
- ③④ A local persistent volume must define node affinity rules that restrict the persistent volume to particular node. OpenShift uses the node affinity rules to schedule pods to the same node as the local persistent volume. In this example, the persistent volume corresponds to the block storage that is mounted on the my-node cluster node.

Managing Local Volumes

Local volumes do not support dynamic provisioning. You must find an appropriate management solution to use local volumes at scale.

In a previous exercise, you used Ansible Playbooks to manage the provisioning of iSCSI block storage resources for the cluster. You can use a similar methodology to manage local volumes.

Alternatively, you can use the Local Storage Operator to manage the life cycle of local persistent volumes.

Describing the Local Storage Operator

The Local Storage Operator creates and manages local volumes attached to nodes in the cluster. The operator requires configuration defined in `LocalVolume` custom resources to create and manage local volumes.

The operator uses configuration defined in `LocalVolume` custom resources. Each resource identifies one or more block devices that the operator uses to create local volumes.

The `LocalVolume` custom resource definition that follows defines a `fast-local-storage` and a `standard-local-storage` storage class.

```
apiVersion: "local.storage.openshift.io/v1" ①
kind: "LocalVolume" ②
metadata:
  name: "local-disks" ③
spec:
  storageClassDevices:
    - storageClassName: "fast-local-storage" ④
      volumeMode: Filesystem
      devicePaths:
        - /dev/sdb ⑤
    - storageClassName: "standard-local-storage" ⑥
      volumeMode: Filesystem
      devicePaths:
        - /dev/hda ⑦
```

①② The API specification for `LocalVolume` resources.

③ The `local-disks` custom resource.

④⑤ The `fast-local-storage` storage class, which uses the `/dev/sdb` block device on each node.

⑥⑦ The `standard-local-storage` storage class, which uses the `/dev/hda` block device on each node.

Creating Local Persistent Volumes

To create local persistent volumes, you must create a `LocalVolume` custom resource in the target namespace for the Local Storage operator.

To determine the target namespace for the Local Storage operator, list the CSV resources for all namespaces:

```
[user@host ~]$ oc get clusterserviceversions --all-namespaces
NAMESPACE      NAME          ... DISPLAY     ... PHASE
...output omitted...
local-storage  local-storage-operator... Local Storage ... Succeeded
...output omitted...
```

The output of the previous command indicates that the operator is installed in the `local-storage` namespace. You must create all `LocalVolume` custom resources in the same namespace.

If the `local-volumes.yml` file contains a `LocalVolume` resource definition, then the following command creates a `LocalVolume` resource:

```
[user@host ~]$ oc create -f local-volumes.yml -n local-storage
```



References

For more information about the Local Storage Operator, refer to the *Persistent storage using local volumes* section in the *Configuring persistent storage* chapter in the Red Hat OpenShift Container Platform 4.6 Storage documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/storage/index#persistent-storage-using-local-volume

Kubernetes 1.14: Local Persistent Volumes GA

<https://kubernetes.io/blog/2019/04/04/kubernetes-1.14-local-persistent-volumes-ga/>

Volumes

<https://kubernetes.io/docs/concepts/storage/volumes/#local>

► Guided Exercise

Installing the Local Storage Operator

In this exercise you will create persistent volumes from locally-attached block storage.

Outcomes

You should be able to

- Install the Local Storage operator.
- Create a `LocalVolume` custom resource that is managed by the Local Storage Operator.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The additional worker nodes (`worker04`, `worker05`, and `worker06`), which are active and in a `Ready` state. If the additional worker nodes are not active, then ensure you have completed *Guided Exercise: Adding Compute Nodes*.

On the `workstation` machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

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

Instructions

- 1. Verify that an additional block device is attached to each of the `worker04`, `worker05`, and `worker06` nodes.

As the `admin` user, open a debug shell to one of the worker nodes. Execute the `lsblk` command to list the attached block devices for the node.

- 1.1. Log in as the `admin` user.

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

...output omitted...
```

- 1.2. Display the cluster nodes.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES   AGE     VERSION
master01   Ready     master   26d    v1.19.0+b00ba52
```

master02	Ready	master	26d	v1.19.0+b00ba52
master03	Ready	master	26d	v1.19.0+b00ba52
worker01	Ready	worker	26d	v1.19.0+b00ba52
worker02	Ready	worker	26d	v1.19.0+b00ba52
worker03	Ready	worker	26d	v1.19.0+b00ba52
worker04	Ready	infra	3d1h	v1.19.0+b00ba52
worker05	Ready	infra	3d1h	v1.19.0+b00ba52
worker06	Ready	infra	3d1h	v1.19.0+b00ba52

- 1.3. Use a debug shell to execute the `lsblk` command on the `worker06` node. Verify that the node contains a 20GB, unpartitioned `vdb` device.

```
[student@workstation ~]$ oc debug node/worker06 -- lsblk
Starting pod/worker06-debug ...
...output omitted...
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0    11:0    1   1G  0 rom
vda   252:0    0   40G  0 disk
|-vda1 252:1    384M 0 part /host/boot
|-vda2 252:2    127M 0 part /host/boot/efi
|-vda3 252:3      1M 0 part
-vda4 252:4    0 39.5G 0 part
`vdb   252:16   0   20G 0 disk

Removing debug pod ...
```

- 1.4. Verify that the `worker01` node does not contain an additional `vdb` block device. No `vdb` device is listed in the output.

```
[student@workstation ~]$ oc debug node/worker01 -- lsblk
Starting pod/worker01-debug ...
...output omitted...
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda   252:0    0   40G  0 disk
|-vda4 252:4    0 39.5G 0 part
|-vda2 252:2    127M 0 part /host/boot/efi
|-vda3 252:3      1M 0 part
`-vda1 252:1    384M 0 part /host/boot

Removing debug pod ...
```

- 2. As the `admin` user, create a `storage-local` project.

```
[student@workstation ~]$ oc whoami
admin

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

- 3. To install the Local Storage operator, first create an operator group resource in the `storage-local` project.

The `~/D0380/labs/storage-local/resources` directory contains an operator group resource definition file. Review the file and create the operator group resource.



Note

Although you can install the Local Storage Operator from the web console, this exercise uses resource definition files to install the operator from the command line.

A file-based installation method integrates with version control, GitOps, and other infrastructure-as-code techniques.

- 3.1. Change to the `~/D0380/labs/storage-local` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/storage-local
```

- 3.2. Review the `resources/operator-group.yml` file.

```
[student@workstation storage-local]$ cat resources/operator-group.yml
---
apiVersion: operators.coreos.com/v1alpha2
kind: OperatorGroup
metadata:
  name: local-operator-group
  namespace: storage-local
spec:
  targetNamespaces:
    - storage-local
```

The file defines a operator group resource that targets the `storage-local` namespace. This operator group configures RBAC access for member operators in the `storage-local` namespace.

- 3.3. Create the operator group resource.

```
[student@workstation storage-local]$ oc create \
-f resources/operator-group.yml
operatorgroup.operators.coreos.com/local-operator-group created
```

- 4. Create the subscription resource for the local storage operator.

You must add missing information to the `resources/subscription.yml` template before you create the subscription resources.

- 4.1. Review the `resources/subscription.yml` subscription template file.

```
[student@workstation storage-local]$ cat resources/subscription.yml
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator ①
  namespace: storage-local ②
spec:
```

```
channel: <CHANNEL_VALUE> ③  
installPlanApproval: Automatic  
name: <PACKAGE_MANIFEST_NAME> ④  
source: <SOURCE> ⑤  
sourceNamespace: <SOURCE_NAMESPACE> ⑥
```

- ① ② After you create the resource, the subscription name is `local-storage-operator` and it exists in the `storage-local` namespace.
- ③ Before you create the resource, you must replace `<CHANNEL_VALUE>` with the channel value for the local storage operator.
- ④ Before you create the resource, you must replace `<PACKAGE_MANIFEST_NAME>` with the name of the package manifest for the local storage operator.
- ⑤ ⑥ Before you create the resource, you must replace `<SOURCE>` and `<SOURCE_NAMESPACE>` with the correct values, which are provided in a later step.

4.2. Display all storage-related package manifests.

```
[student@workstation storage-local]$ oc get packagemanifest | grep storage  
...output omitted...  
local-storage-operator Red Hat Operators 31h  
...output omitted...
```

The package manifest name for the local storage operator is `local-storage-operator`. In a later step, you replace `<PACKAGE_MANIFEST_NAME>` in the subscription template with `local-storage-operator`.

4.3. Find the source values that you need for the subscription resource template.

Search for `Source` in the output of the `oc describe packagemanifest` command.

```
[student@workstation storage-local]$ oc describe packagemanifests \  
local-storage-operator | grep Source  
Catalog Source: redhat-operators ①  
Catalog Source Display Name: Red Hat Operators  
Catalog Source Namespace: openshift-marketplace ②  
Catalog Source Publisher: Red Hat
```

- ① In a later step, you replace `<SOURCE>` in the subscription template with `redhat-operators`.
- ② In a later step, you replace `<SOURCE_NAMESPACE>` in the subscription template with `openshift-marketplace`.

4.4. Find the channel value that you need for the subscription resource template.

Search for `Channel` in the output of the `oc describe packagemanifest` command.

```
[student@workstation storage-local]$ oc describe packagemanifests \
local-storage-operator | grep Channel
Channels:
Default Channel: 4.6 ①
```

- ① In a later step, you replace <CHANNEL> in the subscription template with 4.6.
- 4.5. Modify the `resources/subscription.yml` template file to match the correct values for the local storage operator. Then save the file.
Ensure the content of the `resources/subscription.yml` file matches the following:

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: storage-local
spec:
  channel: "4.6" ①
  installPlanApproval: Automatic
  name: local-storage-operator ②
  source: redhat-operators ③
  sourceNamespace: openshift-marketplace ④
```

- ① You must enclose the 4.6 channel value in quotes. Otherwise, the channel value is interpreted as a floating point number.
- ② The package manifest name, which is `local-storage-operator`.
- ③④ The source values for the subscription.



Note

A solution file is available at `~/DO380/solutions/storage-local/resources/subscription.yml`.

- 4.6. Use the `resources/subscription.yml` file to create a subscription resource for the local storage operator.

```
[student@workstation storage-local]$ oc apply -f resources/subscription.yml
subscription.operator.coreos.com/local-storage-operator created
```

- 5. Verify that local storage operator resources are deployed.

- 5.1. List all the pods.

```
[student@workstation storage-local]$ oc get pods
NAME                               READY   STATUS    RESTARTS   AGE
local-storage-operator-75d7454577-s475q   1/1     Running   0          32s
```

5.2. List the operator group resources.

```
[student@workstation storage-local]$ oc get operatorgroup  
NAME          AGE  
local-operator-group  10m
```

5.3. List the subscription resources.

```
[student@workstation storage-local]$ oc get subscriptions  
NAME          PACKAGE          SOURCE          CHANNEL  
local-storage-operator local-storage-operator redhat-operators  4.6
```

5.4. List the cluster service version resource for the local storage operator.

```
[student@workstation storage-local]$ oc get clusterserviceversions  
NAME          DISPLAY      ... PHASE  
local-storage-operator.4.6.0-2021... Local Storage ... Succeeded
```

**Note**

The CSV resource name in your output may differ from the preceding output.

The CSV resource name matches the pattern: `local-storage-operator-4.6.z-<DATE STRING>`.

5.5. For easy access to the CSV name in later steps, create a `CSV_NAME` variable with the value of the local storage operator CSV resource name.

```
[student@workstation storage-local]$ oc get clusterserviceversions -o name  
clusterserviceversion.operators.coreos.com/local-storage-operator.4.6.0-2021...  
  
[student@workstation storage-local]$ export CSV_NAME=$(oc get csv -o name)  
  
[student@workstation storage-local]$ echo ${CSV_NAME}  
clusterserviceversion.operators.coreos.com/local-storage-operator.4.6.0-2021...
```

▶ 6. Display the list of custom resource definition types that the operator owns. Verify that no operator-owned custom resources exist in the project.

6.1. A CSV resource specification defines a `customresourcedefinitions.owned` attribute. This attribute defines a list of custom resource types that the operator owns.

Use a jsonpath expression to display the `kind` attribute for all operator-owned resource types.

```
[student@workstation storage-local]$ oc get ${CSV_NAME} -o \  
  jsonpath='{.spec.customresourcedefinitions.owned[*].kind}{"\n"}'  
LocalVolume LocalVolumeSet LocalVolumeDiscovery LocalVolumeDiscoveryResult
```

6.2. Verify that no `LocalVolume` custom resources exist.

```
[student@workstation storage-local]$ oc get localvolumes  
No resources found in storage-local namespace.
```

▶ 7. Create a LocalVolume custom resource for the operator.

Ensure the operator creates a persistent volume for the /dev/vdb block device on each node. Ensure that each persistent volume uses a local-blk storage class name.

- 7.1. Retrieve an example LocalVolume custom resource definition from the operator CSV resource.

The alm-examples annotation contains a JSON-encoded string that defines a list of example custom resource definitions.

```
[student@workstation storage-local]$ oc get ${CSV_NAME} \  
-o jsonpath='{.metadata.annotations.alm-examples}{\"\\n\"}'  
[  
 {  
   "apiVersion": "local.storage.openshift.io/v1",  
   "kind": "LocalVolume",  
   "metadata": {  
     "name": "example"  
   },  
   "spec": {  
     "storageClassDevices": [  
       {  
         "devicePaths": [  
           "/dev/vde",  
           "/dev/vdf"  
         ],  
         "fsType": "ext4",  
         "storageClassName": "foobar",  
         "volumeMode": "Filesystem"  
       }  
     ]  
   }  
 }  
 ...output omitted...  
 ]
```



Note

The alm-examples string does not end with a new line character.

The end of the preceding jsonpath expression contains {"\n"} to ensure the prompt displays on a separate line.

- 7.2. Use the provided `get_localvolume_template.sh` script to create a LocalVolume resource definition in YAML format. The script converts the previous example from JSON to YAML and stores the example in the `resources/localvolume.yaml` file.

```
[student@workstation storage-local]$ ./get_localvolume_template.sh
Created /home/student/D0380/labs/storage-local/resources/localvolume.yml
```

7.3. Review the resources/localvolume.yml template file.

```
[student@workstation storage-local]$ cat resources/localvolume.yml
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: example ①
spec:
  storageClassDevices:
    - devicePaths:
        - /dev/vde ②
        - /dev/vdf ③
      fsType: ext4
      storageClassName: foobar ④
      volumeMode: Filesystem
```

- ① In the next step, change the name to local-storage.
- ② ③ In the next step, change the devicePaths list to only contain the /dev/vdb device.
- ④ In the next step, change the storage class name to local-blk.

7.4. Use your preferred editor to update values in the resources/localvolume.yml file. Ensure the file matches the following content, and then save the file.

```
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: local-storage
spec:
  storageClassDevices:
    - devicePaths:
        - /dev/vdb
      fsType: ext4
      storageClassName: local-blk
      volumeMode: Filesystem
```

7.5. Use the localvolume.yml file to create the LocalVolume resource.

```
[student@workstation storage-local]$ oc create -f resources/localvolume.yml
localvolume.local.storage.openshift.io/local-storage created
```

7.6. Verify that the local-storage resource exists.

```
[student@workstation storage-local]$ oc get localvolumes
NAME          AGE
local-storage  71s
```

- 8. Verify that the operator uses the custom resource to create persistent volumes.

- 8.1. After you create the LocalVolume resource, verify that the operator deploys pods to each worker node to control local storage provisioning:

```
[student@workstation storage-local]$ oc get pods
NAME                               READY   STATUS    RESTARTS   AGE
local-storage-local-diskmaker-6rqbv   1/1     Running   0          29s
...output omitted...
local-storage-local-diskmaker-zsnqg   1/1     Running   0          30s
local-storage-local-provisioner-4cqnr   1/1     Running   0          30s
...output omitted...
local-storage-local-provisioner-zkfn8   1/1     Running   0          29s
local-storage-operator-75d7454577-kzr9b   1/1     Running   0          11m
```

There are six local-diskmaker pods, and six local-provisioner pods. Each of the six nodes contains a local-diskmaker and local-provisioner pod.

- 8.2. Verify that the local storage operator creates three persistent volumes. Because the additional /dev/vdb block device exists only on the worker04, worker05, and worker06 nodes, only three persistent volumes exist.

```
[student@workstation storage-local]$ oc get persistentvolumes
NAME      CAPACITY   ...   STORAGECLASS   REASON   AGE
local-pv-21231ed8   20Gi     ...   local-blk        27s
local-pv-6013b152   20Gi     ...   local-blk        27s
local-pv-80eb4d7f   20Gi     ...   local-blk        28s
...output omitted...
```



Note

If the command output is difficult to read, try a different command to filter specific columns.

To only display the NAME, CAPACITY, and STORAGECLASS columns, use the following command:

```
[student@workstation local]$ NAME="NAME:.metadata.name"
[student@workstation local]$ CAPACITY="CAPACITY:.spec.capacity.storage"
[student@workstation local]$ CLASS="STORAGECLASS:.spec.storageClassName"
[student@workstation local]$ oc get persistentvolumes \
-o custom-columns="$NAME,$CAPACITY,$CLASS"
```

See `oc get --help` for more information about custom columns.

- 8.3. Display the list of storage classes.

```
[student@workstation storage-local]$ oc get storageclasses
NAME           PROVISIONER   ...
local-blk       kubernetes.io/no-provisioner   ...
...output omitted...
```

The operator uses information in the `LocalVolume` custom resource to create the `local-blk` storage class.

- ▶ **9.** Change to the `/home/student` directory.

```
[student@workstation storage-local]$ cd
```

Finish

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

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

▶ Lab

Configuring Persistent Storage

In this lab, you will create local block storage resources for the cluster and deploy a database that uses local block storage.

Outcomes

You should be able to:

- Use a storage class in a persistent volume claim to request block storage.
- Create a storage quota that restricts storage resources by storage class.
- Install the Local Storage Operator.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The additional worker nodes (`worker04`, `worker05`, and `worker06`) are active and in a Ready state. If the additional worker nodes are not active, then ensure that you completed *Guided Exercise: Adding Compute Nodes*.

On the `workstation` machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

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

Instructions

1. Create an operator group to start the installation of the Local Storage Operator.
As the `admin` user, create a `local-storage` operator group resource in the `storage-local` project. Ensure that the operator group targets the `storage-local` namespace.
2. To finish operator installation, create a subscription resource for the Local Storage Operator in the `storage-local` project.
Verify that a cluster service version resource exists after you create the subscription.
3. The Local Storage Operator uses configuration from `LocalVolume` resources to create and manage local persistent volumes.
Create a `LocalVolume` resource. Ensure that the operator creates a persistent volume for each `/dev/vdb` device that is attached to the cluster nodes.
Further ensure that each persistent volume:
 - Corresponds to the `local-blk` storage class.
 - Uses a `Filesystem` volume mode.

4. As the `admin` user, create a `storage-review` project. Add the `developer` user as an administrator for the project.

Create a `storage` quota that restricts the project to:

- 1 persistent volume claim for storage from the `local-blk` storage class.
- 10 GB of requested storage, regardless of storage class.

5. As the `developer` user, deploy a `postgres` database in the `storage-review` project.

The `~/DO380/labs/storage-review` directory contains a `postgres` subdirectory. This subdirectory contains resources to deploy a `postgres` database.

Modify the `postgres/statefulset.yml` file to ensure that the database uses a `local-blk` persistent volume.

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 storage-review grade
```

Finish

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

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

► Solution

Configuring Persistent Storage

In this lab, you will create local block storage resources for the cluster and deploy a database that uses local block storage.

Outcomes

You should be able to:

- Use a storage class in a persistent volume claim to request block storage.
- Create a storage quota that restricts storage resources by storage class.
- Install the Local Storage Operator.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The additional worker nodes (`worker04`, `worker05`, and `worker06`) are active and in a `Ready` state. If the additional worker nodes are not active, then ensure that you completed *Guided Exercise: Adding Compute Nodes*.

On the `workstation` machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

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

Instructions

1. Create an operator group to start the installation of the Local Storage Operator.

As the `admin` user, create a `local-storage` operator group resource in the `storage-local` project. Ensure that the operator group targets the `storage-local` namespace.

- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Create the `storage-local` project.

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

- 1.3. Create a resource file named `operatorgroup.yml` for the `local-storage` operator group that contains the following:

```
apiVersion: operators.coreos.com/v1alpha2
kind: OperatorGroup
metadata:
  name: local-storage
  namespace: storage-local
spec:
  targetNamespaces:
    - storage-local
```

- 1.4. Create the operator group resource from the resource definition file you created.

```
[student@workstation ~]$ oc apply -f operatorgroup.yml
operatorgroup.operator.coreos.com/local-storage created
```

2. To finish operator installation, create a subscription resource for the Local Storage Operator in the `storage-local` project.

Verify that a cluster service version resource exists after you create the subscription.

- 2.1. Create a subscription resource file for the local storage operator that contains the following:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: storage-local
spec:
  channel: "4.6"
  installApproval: automatic
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

- 2.2. Create the subscription resource from the resource definition file you created.

The following command assumes that the resource definition file is the `subscription.yml` file.

```
[student@workstation ~]$ oc apply -f subscription.yml
subscription.operator.coreos.com/local-storage-operator created
```

- 2.3. Verify the cluster service version resource exists for the local storage operator.

```
[student@workstation ~]$ oc get clusterserviceversions
NAME                DISPLAY      ...    PHASE
local-storage-operator.4.6.0-2021...  Local Storage  ...  Succeeded
```

3. The Local Storage Operator uses configuration from `LocalVolume` resources to create and manage local persistent volumes.

Create a `LocalVolume` resource. Ensure that the operator creates a persistent volume for each `/dev/vdb` device that is attached to the cluster nodes.

Further ensure that each persistent volume:

- Corresponds to the `local-blk` storage class.
- Uses a `Filesystem` volume mode.

- 3.1. Create a `LocalVolume` resource definition file that contains the following:

```
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: local-storage
spec:
  storageClassDevices:
    - devicePaths:
        - /dev/vdb
      fsType: ext4
  storageClassName: local-blk
  volumeMode: Filesystem
```

- 3.2. Create the `LocalVolume` resource from the resource definition file you created.

The following command assumes that the resource definition file is the `localvolume.yml` file.

```
[student@workstation ~]$ oc apply -f localvolume.yml
localvolume.local.storage.openshift.io/local-storage created
```

4. As the `admin` user, create a `storage-review` project. Add the `developer` user as an administrator for the project.

Create a `storage` quota that restricts the project to:

- 1 persistent volume claim for storage from the `local-blk` storage class.
- 10 GB of requested storage, regardless of storage class.

- 4.1. Create the `storage-review` project.

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

- 4.2. Add the `developer` user as a project administrator.

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

- 4.3. Create a storage quota for the `storage-review` project:

```
[student@workstation ~]$ sclass=local-blk.storageclass.storage.k8s.io
[student@workstation ~]$ limit1=${sclass}/persistentvolumeclaims=1
[student@workstation ~]$ limit2=requests.storage=10G
[student@workstation ~]$ oc create quota storage \
    --hard=${limit1},${limit2}
resourcequota/storage created
```

5. As the developer user, deploy a `postgres` database in the `storage-review` project.

The `~/D0380/labs/storage-review` directory contains a `postgres` subdirectory. This subdirectory contains resources to deploy a `postgres` database.

Modify the `postgres/statefulset.yml` file to ensure that the database uses a `local-blk` persistent volume.

- 5.1. Log in as the developer user.

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

- 5.2. Ensure the `storage-review` project is the active project.

```
[student@workstation ~]$ oc project storage-review
...output omitted...
```

- 5.3. Modify the `volumeClaimTemplates` section of the `~/D0380/labs/storage-review/postgres/statefulset.yml` file to match the following:

```
...output omitted...
volumeClaimTemplates:
- metadata:
    name: data
  spec:
    storageClassName: local-blk
    accessModes: [ "ReadWriteOnce" ]
    resources:
      requests:
        storage: 2Gi
```

- 5.4. Deploy the database.

```
[student@workstation ~]$ oc apply -k ~/D0380/labs/storage-review/postgres/
secret/postgresql created
service/postgresql created
statefulset.apps/postgresql created
```

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 storage-review grade
```

Finish

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

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

Summary

In this chapter, you learned:

- OpenShift provides many volume plug-ins that enable using a wide variety of storage technologies in an OpenShift cluster.
- OpenShift uses dynamic provisioning to provide on-demand storage resources for some storage technologies.
- Operators or playbooks can help manage statically-provisioned storage resources.
- Users specify a storage class in a persistent volume claim to request storage with particular characteristics. When a storage class is not specified, OpenShift uses the default storage class.

Chapter 9

Managing Cluster Monitoring and Metrics

Goal

Configure and manage the OpenShift monitoring stack.

Objectives

- List cluster monitoring features.
- Send alerts to external locations.
- Use Prometheus and Grafana to assist in diagnosing cluster problems.
- Configure Prometheus and Alertmanager to use persistent storage.

Sections

- Introducing the Cluster Monitoring Stack (and Quiz)
- Managing OpenShift Alerts (and Guided Exercise)
- Troubleshooting Using the Cluster Monitoring Stack (and Guided Exercise)
- Configuring Storage for the Cluster Monitoring Stack (and Guided Exercise)

Lab

Managing Cluster Monitoring and Metrics

Introducing the Cluster Monitoring Stack

Objectives

After completing this section, you should be able to:

- List cluster monitoring features.
- Describe alert types and identify when to silence an alert.

Listing the Cluster Monitoring Stack Components

The monitoring stack deploys the following components in the environment to provide a complete solution for monitoring the infrastructure, receiving alerts, and consulting performance graphs.

Cluster Monitoring Operator

The cluster monitoring operator (CMO) is the central component of the monitoring stack. It controls the deployed monitoring components and ensures that they are always in sync with the latest version of the CMO.

Prometheus Operator

The Prometheus operator deploys and configures both Prometheus and Alertmanager. The operator also manages the generation and configuration of configuration targets (service monitors and pod monitors).

Prometheus

Prometheus is the monitoring server.

Prometheus Adapter

The Prometheus adapter exposes cluster resources that are used for Horizontal Pod Autoscaling (HPA).

Alertmanager

Alertmanager handles alerts sent by the Prometheus server.

Kube state metrics

`kube-state-metrics` is a converter agent that exports Kubernetes objects to metrics that Prometheus can parse.

OpenShift state metrics

`openshift-state-metrics` is based on the `kube-state-metrics` and adds monitoring for OpenShift-specific resources (such as image registry metrics).

Node exporter

`node-exporter` exports low-level metrics for worker nodes.

Thanos Querier

Thanos Querier is a single, multitenant interface that enables aggregating and deduplicating cluster and user workload metrics.

Grafana

Grafana is a platform for analyzing and visualizing metrics. The Grafana dashboards provided with the monitoring stack are read-only.

Defining the Cluster Monitoring Stack Custom Resource Definitions

The cluster monitoring operator deploys and manages the life cycle of a Prometheus-based stack for monitoring the cluster. The operator creates the following five custom resource definitions (CRDs):

- **Prometheus** for deploying Prometheus instances. Display all resource instances using the following command:

```
[user@host ~]$ oc get prometheuses -A
```

- **ServiceMonitor** for managing configuration files that describe how and where to scrape the data from application services. Display all resource instances using the following command:

```
[user@host ~]$ oc get servicemonitors -A
```

- **PodMonitor** for managing configuration files for scraping data from pods. Display all resource instances using the following command:

```
[user@host ~]$ oc get podmonitors -A
```

- **Alertmanager** for deploying and managing alert manager instances. Display all resource instances using the following command:

```
[user@host ~]$ oc get alertmanagers -A
```

- **PrometheusRule** for managing rules. Rules correspond to a query set, a filter, and any other field. Display all resource instances using the following command:

```
[user@host ~]$ oc get prometheusrules -A
```

The `prometheus-k8s-rulefiles-0` configuration map in the `openshift-monitoring` namespace provides a combined view of all of the Prometheus rules.

Assigning Cluster Monitoring Roles

Assign the `cluster-monitoring-view` cluster role to a user to permit viewing cluster alerts and metrics. This role allows a user to see the Monitoring section of the OpenShift web console.

```
[user@host ~]$ oc adm policy add-cluster-role-to-user \
cluster-monitoring-view USER
```

Describing Alertmanager Features

An alert is a rule that evaluates to `true` or `false`. The rule is often based on cluster observations, such as cluster CPU utilization.

An alert is also associated with a duration. The alert rule must continue to evaluate to `true` for the defined duration for the alert to trigger.

Chapter 9 | Managing Cluster Monitoring and Metrics

You access cluster alerts from the OpenShift web console at **Monitoring → Alerting**. A brief description of the alert, the alert state, and the alert severity are displayed. View alert details by clicking the name of the alert.

Name	Severity	State	Source
AL_UpdateAvailable	Info	Firing Since Sep 10, 9:32 am	Platform
AL_Watchdog	None	Firing Since Sep 10, 9:32 am	Platform
AL_AlertmanagerReceiverNo tConfigured	Warning	Firing Since Sep 10, 9:32 am	Platform

Figure 9.1: Alert List

An alert has four states:

State	Description
Firing	The alert rule evaluates to true, and has evaluated to true for longer than the defined alert duration.
Pending	The alert rule evaluates to true, but has not evaluated to true for longer than the defined alert duration.
Silenced	The alert is Firing, but is actively being silenced. Administrators can silence an alert to temporarily deactivate it.
Not Firing	Any alert that is not Firing, Pending, or Silenced is labeled as Not Firing.

Although the **Firing**, **Silenced**, and **Pending** alert states are displayed from the web console by default, the display of each state can be toggled on or off.

Understanding Alert Rules

To stop an alert, you must fix or otherwise address any cluster conditions that cause the alert to fire.

Click the name of any alert in the OpenShift web console to display alert details. You can check alert details regardless of the state.

The screenshot shows the 'Alert Details' page for an alert named 'AlertmanagerReceiversNotConfigured'. The alert is currently in a 'Warning' state, which is labeled as 'Firing' since Sep 10, 9:32 am. The alert message states that no notifications are configured, and it links to OpenShift documentation for configuration. The alert has a severity of 'Warning' and is associated with the 'Platform' source. It is part of an alerting rule named 'AlertmanagerReceiversNotConfigured'. A 'Silence Alert' button is visible in the top right corner.

Figure 9.2: Alert Details

In this example, the `AlertmanagerReceiversNotConfigured` alert is defined in the `AlertmanagerReceiversNotConfigured` alerting rule. This alert has a severity of `Warning` and is currently in a `Firing` state.

As shown in the **Message** field, this alarm switches to `Firing` when no alerts are configured to send to a notification system. The **Message** section describes the alert.

Although hidden in this graphic, the alert detail page typically displays the evaluated metric in a graph.

The graph contains a `View in Metrics` link that redirects to the `Monitoring → Metrics` menu which displays a graph with the evaluation query of the alert.

Silencing an Alert

Silence an alert while you actively work to resolve the issue. No additional notifications about a silenced alert are sent until the silence expires.

Silence an alert using the vertical ellipsis menu at the far right of the alert row.

The screenshot shows the 'Alerts' page with three alerts listed:

- AL UpdateAvailable**: Info severity, Firing since Sep 10, 9:32 am. Description: Your upstream update recommendation service...
- AL Watchdog**: Info severity, Firing since Sep 10, 9:32 am. Description: This is an alert meant to ensure that the entire alerting pipeline...
- AL AlertmanagerReceiversNotConfigured**: Warning severity, Firing since Sep 10, 9:32 am. Description: Alerts are not configured to be sent to a notification system...

A context menu is open for the third alert, showing options: `Silence Alert` (with a speaker icon) and `View Alerting Rule`.

Figure 9.3: Silence Alert

An alert is silenced for two hours by default, but it can be silenced for a longer or shorter period.

Complete the **Creator** and **Comment** fields to provide insight into who silenced the alert and why the alert was silenced.

The **Alerting** page, accessible from the **Monitoring → Alerting** menu, displays silenced alerts by default, including when each silence will end.

Name	Severity	State	Source	
AL AlertmanagerReceiversNotConfigured	⚠ Warning	🔇 Silenced Ends Sep 10, 12:44 pm	Platform	⋮

Alerts are not configured to be sent to a notification system,...

Figure 9.4: Alert List with Silence

In this example, the `AlertManagerReceiversNotConfigured` alert has been silenced. The silenced alert is scheduled to end at 12:44pm on Sep 10. The alert will start firing again if the condition causing the alert has not been resolved by that time.



References

- For more information, refer to the *Understanding the monitoring stack* chapter in the *OpenShift Monitoring Guide* at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/monitoring/understanding-the-monitoring-stack

► Quiz

Introducing the Cluster Monitoring Stack

Choose the correct answers to the following questions:

- ▶ 1. **The cluster monitoring stack (including Prometheus and Grafana) is a separate component that must be added after installing your OpenShift cluster. (True or False)**
 - a. True
 - b. False
- ▶ 2. **The prometheusrule custom resource definition only has instances in the openshift-monitoring namespace. (True or False)**
 - a. True
 - b. False
- ▶ 3. **Which two statements about pending alerts are true? (Choose two.)**
 - a. The alert expression for the pending alert evaluates as true, but the expression has not evaluated as true for a specified duration.
 - b. The alert expression for the pending alert does not currently evaluate as true, even though it might evaluate as true at some point in the future.
 - c. Pending alerts rarely apply to critical alerts.
 - d. Pending alerts are common and should not necessarily cause concern until they move to a firing state.
- ▶ 4. **Which two statements about silenced alerts are true? (Choose two.)**
 - a. When silencing an alert, the default setting is to silence the alert indefinitely.
 - b. When silencing an alert, the default setting is to silence the alert for two hours.
 - c. Silence an alert during troubleshooting so that Alertmanager does not send additional notifications about the alert.
 - d. Silence an alert during troubleshooting to tell Prometheus to stop checking the evaluation expression for the alert.

► Solution

Introducing the Cluster Monitoring Stack

Choose the correct answers to the following questions:

- ▶ 1. **The cluster monitoring stack (including Prometheus and Grafana) is a separate component that must be added after installing your OpenShift cluster. (True or False)**
 - a. True
 - b. False
- ▶ 2. **The prometheusrule custom resource definition only has instances in the openshift-monitoring namespace. (True or False)**
 - a. True
 - b. False
- ▶ 3. **Which two statements about pending alerts are true? (Choose two.)**
 - a. The alert expression for the pending alert evaluates as true, but the expression has not evaluated as true for a specified duration.
 - b. The alert expression for the pending alert does not currently evaluate as true, even though it might evaluate as true at some point in the future.
 - c. Pending alerts rarely apply to critical alerts.
 - d. Pending alerts are common and should not necessarily cause concern until they move to a firing state.
- ▶ 4. **Which two statements about silenced alerts are true? (Choose two.)**
 - a. When silencing an alert, the default setting is to silence the alert indefinitely.
 - b. When silencing an alert, the default setting is to silence the alert for two hours.
 - c. Silence an alert during troubleshooting so that Alertmanager does not send additional notifications about the alert.
 - d. Silence an alert during troubleshooting to tell Prometheus to stop checking the evaluation expression for the alert.

Managing OpenShift Alerts

Objectives

After completing this section, you should be able to send alerts to external locations.

Describing Alertmanager Default Receivers

By default, alerts are not sent to external locations. The OpenShift web console displays alerts at **Monitoring** → **Alerting**. Also, alerts are accessible from the Alertmanager API.

```
[user@host ~]$ ALERTMANAGER="$(oc get route/alertmanager-main \
-n openshift-monitoring -o jsonpath='{.spec.host}')"

[user@host ~]$ curl -s -k -H "Authorization: Bearer $(oc sa get-token \
prometheus-k8s -n openshift-monitoring)" \
https://$ALERTMANAGER/api/v1/alerts | jq .
```

Configure Alertmanager to send alerts to external locations, such as email, PagerDuty, and HipChat, to promptly notify you about cluster problems.

Alertmanager sends alerts to the locations configured in the `alertmanager-main` secret in the `openshift-monitoring` namespace.

This is the default configuration of the `alertmanager-main` secret. The double quotes can be removed to improve readability.

```
"global":
  "resolve_timeout": "5m" ①
"receivers":
- "name": "null"
"route":
  "group_by":
    - "namespace"
  "group_interval": "5m" ②
  "group_wait": "30s" ③
  "receiver": "null" ④
  "repeat_interval": "12h" ⑤
  "routes": ⑥
    - "match":
      "alertname": "Watchdog" ⑦
      "receiver": "null"
```

- ① Value used by Alertmanager if the alert does not include a `EndsAt` field. After this time passes, it can declare the alert as resolved if it has not been updated.
- ② Time to wait before sending a notification about new alerts that are added to a group of alerts for which an initial notification has already been sent.

- ③ Time to wait before sending an initial notification for a group of alerts. Allows for waiting for an inhibiting alert to arrive or for collecting more initial alerts for the same group.
- ④ This is the default receiver for notifications. A value of `null` indicates that it does nothing. All instances of `null` are typically changed to something else, such as `default`.
- ⑤ Time to wait before sending a notification again if it has already been sent successfully for an alert.
- ⑥ A route block defines a node in a routing tree and its children. Its optional configuration parameters are inherited from its parent node if not set.
- ⑦ Define a filter of type `alertname`.

Sending Alerts to PagerDuty

If your company already uses PagerDuty, then you will likely send alerts about your OpenShift cluster to PagerDuty. You can route alerts to cellphones, pagers, and email addresses also. The Prometheus Integration Guide on the PagerDuty website describes how to configure PagerDuty for Prometheus notifications. The following configuration sends `critical` alert messages to PagerDuty.

```
"global":  
  "resolve_timeout": "5m"  
"receivers":  
  - "name": "pagerduty-notification" ①  
    "pagerduty_configs":  
      "service_key": "e679d31da9b34aa4b8bb4f412186546d" ②  
  - "name": "default"  
"route":  
  "group_by":  
    - "job"  
  "group_interval": "5m"  
  "group_wait": "30s"  
  "receiver": "default"  
  "repeat_interval": "12h"  
  "routes":  
    - "match":  
        "alertname": "Watchdog"  
        "receiver": "default"  
    - "match":  
        "severity": "critical" ③  
        "receiver": "pagerduty-notification"
```

- ① An arbitrary name that is used by an associated route.
- ② The PagerDuty integration key for Prometheus. The `service_key` property corresponds to the PagerDuty Events API v1. The `routing_key` property is used for the PagerDuty Events API v2.
- ③ The matching criteria for the alert.

Multiple PagerDuty receivers can be configured so that different types of alerts are sent to the appropriate teams. Consult the [References](#) section for additional details.

Sending Alerts to Email

Alerts can be sent by email through an SMTP server. The following example sends `critical` alerts to the `ocp-admins@example.com` email alias.

```
"global":  
  "resolve_timeout": "5m"  
  "smtp_smarthost": "utility.lab.example.com:25" ①  
  "smtp_from": "alerts@ocp4.example.com" ②  
  "smtp_auth_username": "smtp_training" ③  
  "smtp_auth_password": "Red_H4T@!" ④  
  "smtp_require_tls": false ⑤  
  
"receivers":  
  - "name": "email-notification" ⑥  
    "email_configs": ⑦  
      - "to": "ocp-admins@example.com" ⑧  
  - "name": "default"  
  
"route":  
  "group_by":  
    - "job"  
  "group_interval": "5m"  
  "group_wait": "30s"  
  "receiver": "default"  
  "repeat_interval": "12h"  
  "routes":  
    - "match":  
      "alertname": "Watchdog"  
      "receiver": "default"  
    - "match":  
      "severity": "critical"  
      "receiver": "email-notification" ⑨
```

- ① The global SMTP host. This host is used if `smarthost` is not defined in the `email_configs` for a receiver.
- ② The global email sender address. This address is used if `from` is not defined in the `email_configs` for a receiver.
- ③ The global SMTP user name for optional authentication. This user name is used if `auth_username` is not defined in the `email_configs` for a receiver.
- ④ The global SMTP password for optional authentication. This password is used if `auth_password` is not defined in the `email_configs` for a receiver.
- ⑤ A global setting specifying if TLS is required for SMTP. This setting can be overridden using `require_tls` in the `email_configs` for a receiver.
- ⑥ An arbitrary name for the receiver. A route specifies this receiver name for a match.
- ⑦ This setting indicates that the receiver will send alerts by email.
- ⑧ The `to` setting must be specified under `email_configs` and does not have an equivalent global SMTP setting.
- ⑨ The receiver to use if the `match` evaluates as `true` for the alert.

Applying a New Alertmanager Configuration

Apply a new Alertmanager configuration by updating the `alertmanager-main` secret in the `openshift-monitoring` namespace.

```
[user@host ~]$ oc set data secret/alertmanager-main \
-n openshift-monitoring --from-file=/tmp/alertmanager.yaml
secret/alertmanager-main data updated
```

A successful update generates the log message: "Completed loading of configuration file" `file=/etc/alertmanager/config/alertmanager.yaml`. Check for this message in the `Alertmanager` container within the `alertmanager-main-0` pod in the `openshift-monitoring` namespace. An incorrect configuration generates a failed log message.

```
[user@host ~]$ oc logs -f -c alertmanager alertmanager-main-0 \
-n openshift-monitoring
...output omitted...
level=info ts=2021-10-15T18:53:09.052Z caller=coordinator.go:119
component=configuration msg="Loading configuration file" file=/etc/alertmanager/
config/alertmanager.yaml
level=info ts=2021-10-15T18:53:09.052Z caller=coordinator.go:131
component=configuration msg="Completed loading of configuration file" file=/etc/
alertmanager/config/alertmanager.yaml
```



References

- **Alertmanager Configuration**
<https://prometheus.io/docs/alerting/latest/configuration/>
- **Prometheus Integration Guide**
<https://www.pagerduty.com/docs/guides/prometheus-integration-guide/>

► Guided Exercise

Managing OpenShift Alerts

In this exercise you will send alerts to the `ocp-admins@example.com` email address and resolve the `AlertmanagerReceiversNotConfigured` alert.

Outcomes

You should be able to:

- Configure Alertmanager to send email alerts.
- Review default alerts.
- Silence a firing alert.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that the `utility.lab.example.com` host is configured to receive email and configures the `ocp-admins@example.com` email alias to deliver messages to the `lab` user on `utility.lab.example.com`.

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

Instructions

- 1. Configure Alertmanager to send warning alerts to the `ocp-admins@example.com` email address.
- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Extract the existing `alertmanager-main` secret from the `openshift-monitoring` namespace to the `/tmp` directory.

```
[student@workstation ~]$ oc extract secret/alertmanager-main --to /tmp/ \
-n openshift-monitoring --confirm
/tmp/alertmanager.yaml
```

13. The default `alertmanager-main` secret contains many unnecessary quotes. Remove the quotes using the `sed` command to improve readability.

```
[student@workstation ~]$ sed -i 's://"//g' /tmp/alertmanager.yaml
```

**Important**

Removing the extraneous quote characters is not required.

In some cases, quote characters are required in a YAML file. The previous `sed` command removes all quotes, including possible required quotes.

In this exercise, none of the quote characters in the `/tmp/alertmanager.yaml` file are required.

14. Modify the `/tmp/alertmanager.yaml` file.

```
[student@workstation ~]$ vim /tmp/alertmanager.yaml
```

15. Add the lines in bold to the file. Ensure that you correctly indent each line. Replace instances of `null` with `default`. Set the `repeat_interval` to `1m` so the alert email notifications are sent more frequently.

```
global:  
  resolve_timeout: 5m  
  smtp_smarthost: 192.168.50.254:25  
  smtp_from: alerts@ocp4.example.com  
  smtp_require_tls: false  
receivers:  
  - name: default  
  - name: email-notification  
    email_configs:  
      - to: ocp-admins@example.com  
route:  
  group_by:  
    - namespace  
  group_interval: 5m  
  group_wait: 30s  
  receiver: default  
  repeat_interval: 1m  
  routes:  
    - match:  
        alertname: Watchdog  
        receiver: default  
    - match:  
        severity: warning  
        receiver: email-notification
```

**Note**

Setting a severity of `warning` sends many email messages. You change the severity to `critical` later in the exercise.

The `~/DO380/solutions/monitor-alerts/alertmanager.yaml` file contains the correct configuration.

16. Update the existing `alertmanager-main` secret in the `openshift-monitoring` namespace with the content of the `/tmp/alertmanager.yaml` file.

```
[student@workstation ~]$ oc set data secret/alertmanager-main \
-n openshift-monitoring --from-file /tmp/alertmanager.yaml
secret/alertmanager-main data updated
```

17. Follow the `alertmanager` container logs and look for the message: "Completed loading of configuration file" `file=/etc/alertmanager/config/alertmanager.yaml`

New log messages can take up to 30 seconds to display. Press `Ctrl+C` to exit the `oc logs` command.

```
[student@workstation ~]$ oc logs -f -c alertmanager alertmanager-main-0 \
-n openshift-monitoring
...output omitted...
level=info ts=2021-10-15T18:53:09.052Z caller=coordinator.go:119
component=configuration msg="Loading configuration file" file=/etc/alertmanager/
config/alertmanager.yaml
level=info ts=2021-10-15T18:53:09.052Z caller=coordinator.go:131
component=configuration msg="Completed loading of configuration file" file=/etc/
alertmanager/config/alertmanager.yaml
```

**Note**

If you see configuration errors in the logs, modify the `/tmp/alertmanager.yaml` file and reapply your changes.

- ▶ 2. The `lab` user on the `utility.lab.example.com` host receives emails sent to the `ocp-admins@example.com` email address. Verify that Alertmanager sent an email for each firing alert.

- 2.1. Use SSH to connect to the `utility.lab.example.com` host as the `lab` user.

```
[student@workstation ~]$ ssh lab@utility.lab.example.com
...output omitted...
```

- 2.2. Run the `mutt` command to access mail.

```
[lab@utility ~]$ mutt
```

Allow `mutt` to create the `/home/lab/Mail` directory.

```
/home/lab/Mail does not exist. Create it? ([yes]/no): y
```

- 2.3. The existing emails sent from `alerts@ocp4.example.com` demonstrate that Alertmanager sent email notifications for firing alerts with a severity of `warning` or greater. Press `q` to quit.

```
q:Quit d:Del u:Undel s:Save m:Mail r:Reply g:Group ?:Help
 1 N Jun 25 alerts@ocp4.exa ( 192) [FIRING:1] openshift-image-registry (I...
 2 N Aug 24 alerts@ocp4.exa ( 167) [FIRING:1] (TestAlert openshift-monit...
```

**Note**

Alertmanager sends alerts as email attachments in HTML format. Although not necessary for this exercise, you can save the attachments and transfer them to a machine with graphical capabilities.

- 2.4. Disconnect from the SSH session.

```
[lab@utility ~]$ exit
```

► 3. View alerts from the OpenShift web console.

- 3.1. Use `Firefox` to navigate to the OpenShift web console URL.

- `https://console-openshift-console.apps.ocp4.example.com`

**Note**

If Firefox displays a warning message, then add exceptions to allow using the default OpenShift certificates.

- 3.2. Log in with `htpasswd_provider` using the following credentials.

- Username: `admin`
- Password: `redhat`

- 3.3. From the OpenShift web console, navigate to `Monitoring → Alerting`.

- 3.4. Because there might be additional alerts, expect firing alerts to include the `TestAlert` and `Watchdog` alerts.

► 4. Examine the `TestAlert` alert and then silence the alert for troubleshooting.

- 4.1. Click the `TestAlert` link to get details.

This alert has a severity of `warning`, and it fires when this exercise begins. The alert message indicates that this alert is a test alert.

- 4.2. Silence the alert so that Alertmanager does not send additional notifications. Click `Silence Alert` to silence the alert.

By default, the alert is silenced for two hours. You do not need to change the `Start` or `End` fields.

In the **Creator** field, type your name.

In the **Comment** field, type **Silenced during troubleshooting**.

Click **Silence** to silence the alert.

- 4.3. Navigate to **Monitoring → Alerting** in the web console to verify that the **TestAlert** alert is no longer firing.
- ▶ 5. Although you previously configured Alertmanager to send email alerts with a severity of **warning** or greater to the `ocp-admins@example.com` email address, this configuration will likely send more messages than desired. Modify the Alertmanager configuration to send **critical** alerts to the `ocp-admins@example.com` email address.
 - 5.1. Use the `sed` command to replace **warning** with **critical** in the `/tmp/alertmanager.yaml` file.

```
[student@workstation ~]$ sed -i 's/warning/critical/' /tmp/alertmanager.yaml
```

- 5.2. Update the existing `alertmanager-main` secret in the `openshift-monitoring` namespace with the content of the `/tmp/alertmanager.yaml` file.

```
[student@workstation ~]$ oc set data secret/alertmanager-main \
-n openshift-monitoring --from-file /tmp/alertmanager.yaml
secret/alertmanager-main data updated
```

Alertmanager now sends an email for any alert with a severity of **critical** to the `ocp-admins@example.com` email address.

Finish

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

```
[student@workstation ~]$ lab monitor-alerts finish
```

Troubleshooting Using the Cluster Monitoring Stack

Objectives

After completing this section, you should be able to use Prometheus and Grafana to assist in diagnosing cluster problems.

Introducing Prometheus

Prometheus is an open source project for system monitoring and alerting.

Both Red Hat OpenShift Container Platform and Kubernetes integrate Prometheus to enable cluster metrics, monitoring, and alerting capabilities.

Prometheus gathers and stores streams of data from the cluster as time-series data. Time-series data consists of a sequence of samples, with each sample containing:

- A timestamp.
- A numeric value (such as an integer, float, or Boolean).
- A set of labels in the form of key/value pairs. The key/value pairs are used to isolate groups of related values for filtering.

For example, the `machine_cpu_cores` metric in Prometheus contains a sequence of measurement samples of the number of CPU cores for each machine.

OpenShift integrates Prometheus metrics at **Monitoring → Metrics**. A separate Prometheus user interface is accessible using the URL of the `prometheus-k8s` route in the `openshift-monitoring` namespace.

From the **Metrics** page, enter an expression, such as a metric name, and then click **Run Queries** to retrieve the most recent sample for the metric.

The following example displays the `instance:node_cpu_utilisation:rate1m` metric over time.

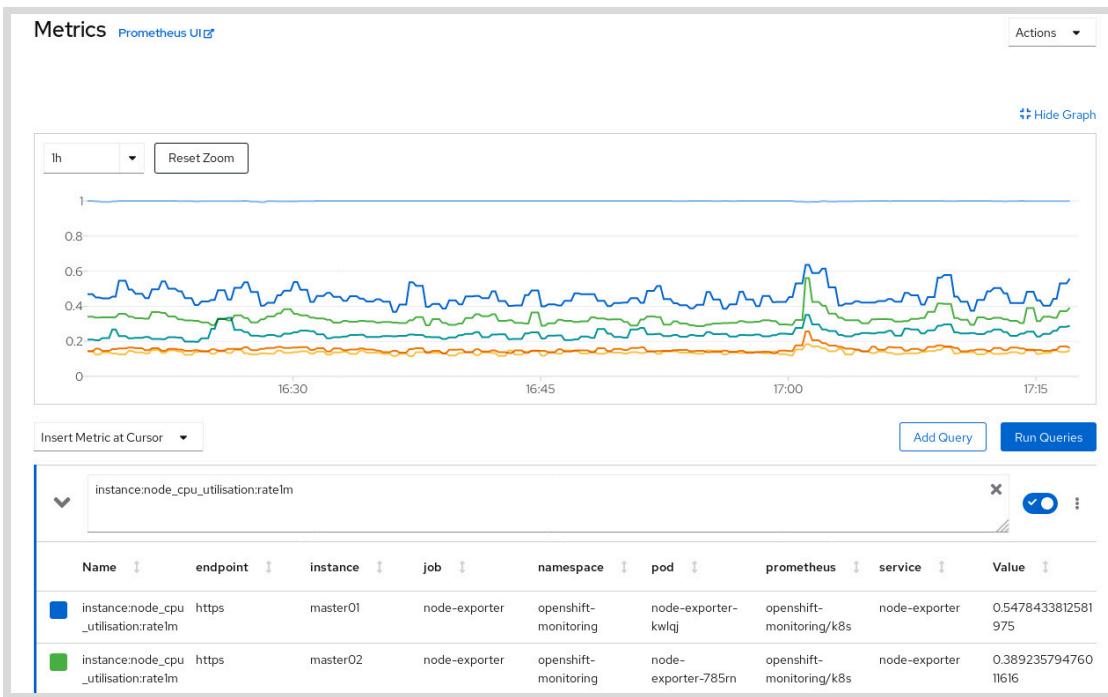


Figure 9.5: Prometheus Metrics

The metric contains data for each node instance in the cluster.

Describing Prometheus Query Language

Prometheus provides a query language, PromQL, that allows you to select and aggregate time-series data.

You can filter a metric to include only certain key/value pairs. For example, modify the previous query to show only metrics for the `worker02` node using the following expression:

```
instance:node_cpu_utilisation:rate1m{instance="worker02"}
```

Prometheus Query Language provides several operators to compute new time-series metrics. PromQL contains arithmetic operators, including addition, subtraction, multiplication, and division operators. PromQL contains comparison operators, including equality, greater than, and less than operators.

PromQL contains some built-in functions that you can include in PromQL expressions including the following:

`sum()`

Totals the value of all sample entries at a given time

`rate()`

Computes the per-second average of a time series for a given time range

`count()`

Counts the number of sample entries at a given time

Every Prometheus alert contains a Prometheus Query Language expression. Consider the `KubeCPUOvercommit` alert, available in **Monitoring** → **Alerting** → **Alerting Rules**.

The screenshot shows the configuration for the 'KubeCPUOvercommit' alert rule. It includes fields for Name (KubeCPUOvercommit), For (5m), Severity (Warning), and a detailed Expression section. The expression is a complex metric query involving the sum function, division, and comparison operators to calculate the ratio of total CPU requests to total CPU cores across namespaces and containers, then compare it to the ratio of allocatable CPU cores to the count of nodes minus one. Below the configuration is a section for Active Alerts, which is currently empty.

Figure 9.6: KubeCPUOvercommit Alert Rule

The KubeCPUOvercommit alert expression compares two ratios using the `sum` function, the division operator, and the greater-than operator:

- The left ratio is the total number of CPU requests divided by the total number of CPU cores.
- The right ratio is a count of entries minus 1 for the CPU cores metric, divided by a count of entries for the CPU metric.

Each entry in the CPU cores metric corresponds to a node. Therefore, the right side ratio corresponds to a percentage of cluster CPU capacity if a single node fails. If the expression evaluates as `true` for five minutes, then the alert starts firing with a `warning` severity.

Firing alerts can notify administrators of potential cluster problems that might require further investigation.

Introducing Grafana

Grafana is an open source analytics and visualization project intended to improve the observability of system metrics. OpenShift integrates Grafana dashboards at [Monitoring → Dashboards](#). A separate Grafana user interface is accessible using the URL of the `grafana` route in the `openshift-monitoring` namespace.

With Grafana, you can create customized dashboards to visualize key cluster metrics. Grafana dashboards frequently refresh to display current summary metrics and graphs.

Grafana graphs are interactive. With Grafana, you can further explore interesting data features and characteristics you observe in a graph.

OpenShift Container Platform provides several dashboards in Grafana. These dashboards serve as a good starting point for near real-time observability of cluster metrics and health.

After receiving an alert, an administrator might use Grafana dashboards to investigate the problem. This investigation could include checking if a specific node or project has a problem. Additionally, Grafana dashboards can help identify if a problem was temporary or if it appears to be persistent.

Grafana includes the following default dashboards:

etcd

This dashboard provides information on etcd instances running in the cluster.

Kubernetes/Compute Resources/Cluster

This dashboard provides a high-level view of cluster resources.

Kubernetes/Compute Resources/Namespace (Pods)

This dashboard displays resource usage for pods within a namespace.

Kubernetes/Compute Resources/Namespace (Workloads)

This dashboard filters resource usage first by namespace and then by workload type, such as deployment, daemonset, and statefulset. Grafana displays all workloads of the specified type within the namespace.

Kubernetes/Compute Resources/Node (Pods)

This dashboard shows pod resource usage filtered by node.

Kubernetes/Compute Resources/Pod

This dashboard displays the resource usage for individual pods. Select a namespace and a pod within the namespace.

Kubernetes/Compute Resources/Workload

This dashboard provides resources usage filtered by namespace, workload, and workload type.

Kubernetes/Networking/Cluster

This dashboard displays network usage for the cluster. Grafana sorts many items to show namespaces with the highest usage.

Prometheus

This dashboard provides detailed information about the prometheus-k8s pods running in the openshift-monitoring namespace.

USE Method/Cluster

USE is an acronym for Utilization Saturation and Errors. This dashboard displays several graphics that can identify if the cluster is over-utilized, over-saturated, or experiencing a large number of errors. Because Grafana displays all nodes in the cluster, you might be able to identify a node that is not behaving the same as the other nodes in the cluster.

The following graphic indicates that worker03 is experiencing a higher level of memory saturation compared to the other nodes in the cluster.

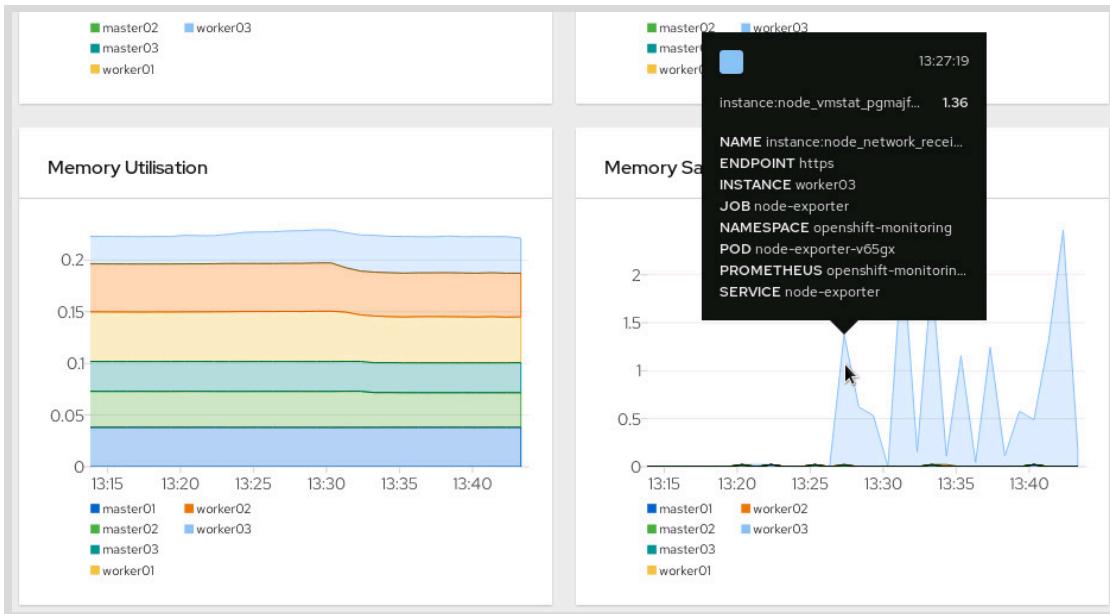


Figure 9.7: USE Method/Cluster - Memory Saturation

Additionally, worker03 is experiencing higher disk IO saturation.

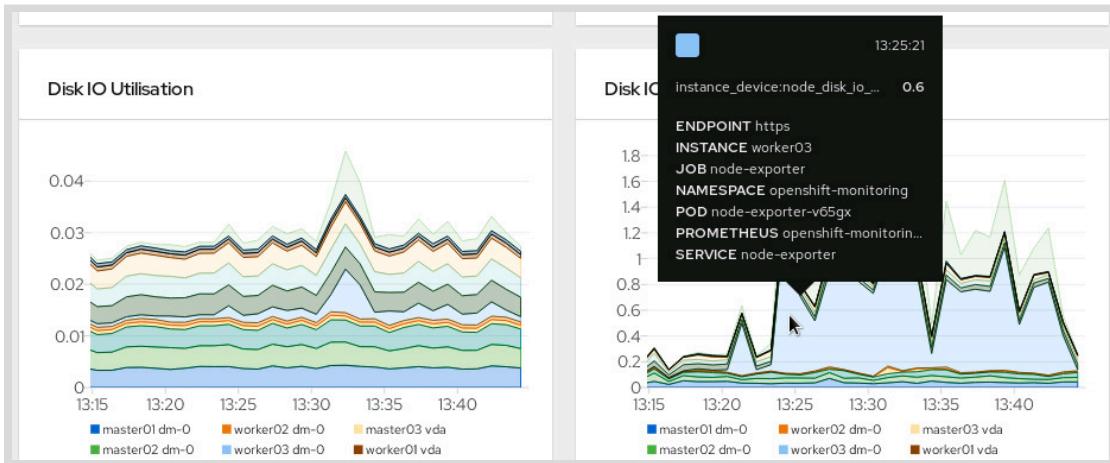


Figure 9.8: USE Method/Cluster - Disk IO Saturation

USE Method/Node

This dashboard filters by node to display potential problems using the USE (Utilization Saturation and Errors) method.



References

For more information about Prometheus, refer to the Prometheus documentation at
<https://prometheus.io/docs/introduction/overview/>

For more information about Prometheus metrics, refer to the Prometheus Data Model documentation at
https://prometheus.io/docs/concepts/data_model/

For more information on Prometheus Query Language, refer to the Prometheus documentation at
<https://prometheus.io/docs/prometheus/latest/querying/basics/>

For more information about Grafana, refer to the *Getting Started* guide at
<https://grafana.com/docs/grafana/latest/getting-started/>

For more information about the USE Method, refer to
<http://www.brendangregg.com/usemethod.html>

► Guided Exercise

Troubleshooting Using the Cluster Monitoring Stack

In this exercise you will identify a potential application problem using the cluster monitoring stack.

Outcomes

You should be able to use Grafana to identify a deployment that consumes excessive amounts of CPU and memory.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that the cluster API is reachable and deploys an application with a potential problem.

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

Instructions

- 1. Use Grafana to display cluster resource usage information within the OpenShift web console.
- 1.1. Use Firefox to navigate to the OpenShift web console URL.
 - <https://console-openshift-console.apps.ocp4.example.com>



Note

If Firefox displays a warning message, then add exceptions to allow using the default OpenShift certificates.

- 1.2. Log in with `htpasswd_provider` using the following credentials.
 - Username: admin
 - Password: redhat
- 1.3. Navigate to **Monitoring** → **Dashboards**.
- 1.4. Select the **Kubernetes/Compute Resources/Cluster** dashboard.
- 1.5. Scroll to the **CPU Quota** section, and then click the **CPU Usage** column header. You might have to click the column header more than once. Click until it turns blue with

an arrow pointing down to indicate that rows are sorted by namespaces using the highest to lowest amount of CPU.

CPU Quota						
Namespace	Pods	Workloads	CPU Usage	CPU Requests	CPU Requests %	
monitor-troubleshoot	11	2	1.46	0.5	292.81%	
openshift-kube-apiserver	-	-	0.564	5.49	10.27%	
openshift-etcd	-	-	0.382	2.19	17.45%	
openshift-monitoring	21	11	0.231	0.569	40.53%	

Figure 9.9: Namespaces with the Highest CPU Usage

Expect that the `monitor-troubleshoot` namespace is listed as one of the top five namespaces using the most amount of CPU resources. Although the `monitor-troubleshoot` namespace only requests 0.5 CPU resources, it uses considerably more. The result is that the **CPU Requests %** column reports between 100% to 500%.

If you do not see the `monitor-troubleshoot` namespace consuming an elevated amount of resources, then reload the page in your browser.

16. Scroll to the **Requests by Namespace** section and then click the **Memory Usage** column header. You might have to click the column header more than once. Click until it turns blue with an arrow pointing down to indicate that rows are sorted by namespaces using the highest to lowest amount of memory.

Requests by Namespace						
Namespace	Pods	Workloads	Memory Usage	Memory Requests	Memory Requests %	
openshift-kube-apiserver	-	-	4.76 GiB	6.23 GiB	76.35%	
monitor-troubleshoot	11	2	3.06 GiB	200 MiB	1,567.29%	
openshift-monitoring	21	11	2.58 GiB	5.08 GiB	50.81%	
openshift-etcd	-	-	1.07 GiB	3.08 GiB	34.60%	

Figure 9.10: Namespaces with the Highest Memory Usage

Expect that the `monitor-troubleshoot` namespace is listed as one of the top five namespaces using the most memory resources. As with CPU usage, the `monitor-troubleshoot` namespace uses considerably more memory than the 200 MiB requested by the pods in the namespace.

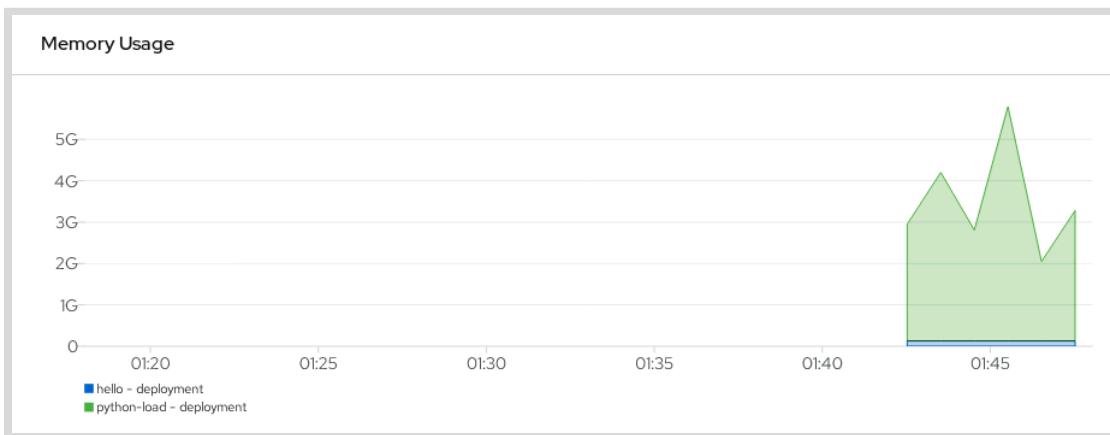
- ▶ 2. Use Grafana to identify the workload and pod that uses the most cluster resources in the `monitor-troubleshoot` namespace.
- 2.1. Select the **Kubernetes/Compute Resources/Namespace (Workloads)** dashboard. Select the `monitor-troubleshoot` namespace and leave `deployment` as the workload type.
 - 2.2. The **CPU Quota** section shows that the `hello` deployment requests a total of 0.5 CPU resources and currently uses almost no CPU resources. The `python-load` deployment does not request or limit CPU resources, but the one pod in the deployment consumes more than three CPU resources.

CPU Quota				
Workload ↑	Workload Type	Running Pods	CPU Usage	CPU Requests
hello	deployment	10	0	0.5
python-load	deployment	1	3.81	-

Figure 9.11: Deployment CPU Usage

Although the actual CPU usage in your cluster might be different, it is clear that the `python-load` deployment uses the most CPU resources.

- 2.3. The **Memory Usage** graph shows that the `python-load` deployment has dramatic increases and decreases in memory usage. However, the ten pods in the `hello` deployment consistently use a total of about 122 MiB of memory.

**Figure 9.12: Deployment Memory Usage**

- ▶ 3. You have identified a deployment that appears to be behaving erratically and might be using more resources than intended. The next step might be to identify the owner of the `python-load` deployment so that the owner can verify the application. Minimally, the owner should add either resource requests or limits for both CPU and memory to the `python-load` deployment. Although a cluster administrator can do this, setting incorrect limits might result in the pod entering a `CrashLoopBackOff` state. Similarly, a cluster administrator can implement a quota for the `monitor-troubleshoot` namespace to limit the total amount of CPU and memory resources used by the project. Because the `python-load` deployment does not currently specify requests or limits for either CPU or memory, restricting CPU and memory resources with a quota would prevent the `python-load` pod from running.

Finish

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

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

Configuring Storage for the Cluster Monitoring Stack

Objectives

After completing this section, you should be able to configure Prometheus and Alertmanager to use persistent storage.

Configuring the Cluster Monitoring Operator

After installing the cluster monitoring stack, you can configure persistent storage to prevent monitoring data loss. This configuration enables you to keep a record of the past cluster status that you can use to investigate and correlate current and past issues within the cluster.

You can configure the cluster monitoring operator by creating a configuration map named `cluster-monitoring-config` in the `openshift-monitoring` namespace. This configuration map does not exist by default. Within the `config.yaml` entry of the `data` section, specify components and their configurations.

The following cluster monitoring operator components are configurable:

- `alertmanagerMain`
- `auth`
- `ingress`
- `kubeStateMetrics`
- `nodeExporter`
- `prometheusK8s`
- `prometheusOperator`

For example, use the `prometheusK8s` component to configure persistent storage for Prometheus, and the `alertmanagerMain` component to configure persistent storage for Alertmanager.

The basic skeleton for the `cluster-monitoring-config` configuration map follows:

```
apiVersion: 1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>:
      <component-configuration-options>
```

Configuring Prometheus Persistent Storage

Add persistent storage for Prometheus if you want stored metrics to survive recreation or redeployment of the Prometheus pods.

Prometheus stores data in a custom, highly efficient format time series database (TSDB). Samples are grouped into blocks of two hours. Each two-hour block consists of a directory

containing a chunks subdirectory containing all the time series samples for that window of time, a metadata file, and an index file.

By default, Prometheus stores 15 days worth of metrics using ephemeral storage.

Prometheus Database storage requirements based on the number of nodes/pods in the cluster

Number of Nodes	Number of Pods	Prometheus storage growth per day	Prometheus storage growth per 15 days	RAM Space (per scale size)	Network (per tsdb chunk)
50	1800	6.3 GB	94 GB	6 GB	16 MB
100	3600	13 GB	195 GB	10 GB	26 MB
150	5400	19 GB	283 GB	12 GB	36 MB
200	7200	25 GB	375 GB	14 GB	46 MB

The following configuration map attaches 40 GB of persistent storage to Prometheus using the existing gp2 storage class. The configuration specifies that data is retained for 15 days.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      retention: 15d
      volumeClaimTemplate:
        metadata:
          name: prometheus-local-pv
        spec:
          storageClassName: gp2
          volumeMode: Filesystem
          resources:
            requests:
              storage: 40Gi
```

Assuming a file name of `persistent-storage-prometheus.yaml`, use the `oc apply` command to either create or update the configuration map:

```
[user@host ~]$ oc apply -f persistent-storage-prometheus.yaml
configmap/cluster-monitoring-config created
```

After applying the configuration map, OpenShift redeploys the Prometheus pods as a stateful set and reloads the configuration for Prometheus.

Run the `oc get pvc` command to retrieve the list of persistent volumes claims in the `openshift-monitoring` namespace.

```
[user@host ~]$ oc get persistentvolumeclaims
NAME                           ...   CAPACITY   ACCESS MODES   STORAGECLASS
prometheus-k8s-db-prometheus-k8s-0 ...   40Gi      RWO          gp2
prometheus-k8s-db-prometheus-k8s-1 ...   40Gi      RWO          gp2
```

Configuring Alert Manager Persistent Storage

By default, `alertmanager-main` pods running in the `openshift-monitoring` namespace use ephemeral storage. Configure Alertmanager to use persistent storage so that alerts are not lost if the `alertmanager-main` pods are recreated or redeployed.

The following example configures Alertmanager to attach a 20 GB persistent volume using the existing gp2 storage class for the cluster.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      volumeClaimTemplate:
        metadata:
          name: alertmanager-local-pv
        spec:
          storageClassName: gp2
          volumeMode: Filesystem
          resources:
            requests:
              storage: 20Gi
```

Assuming a file name of `persistent-storage-alertmanager.yaml`, apply the configuration using the following command:

```
[user@host]$ oc apply -f persistent-storage-alertmanager.yaml
configmap/cluster-monitoring-config configured
```



References

- For more information, refer to the *Scaling the Cluster Monitoring Operator* chapter in the *Scalability and Performance Guide* at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/scalability_and_performance/index#prometheus-database-storage-requirements_cluster-monitoring-operator
- **Prometheus Storage**
<https://prometheus.io/docs/prometheus/latest/storage/>

► Guided Exercise

Configuring Storage for the Cluster Monitoring Stack

In this exercise you will configure Prometheus and Alertmanager to use persistent storage.

Outcomes

You should be able to:

- Create a new configuration map for the cluster monitoring operator.
- Define persistent storage for Prometheus and Alertmanager using an existing storage class.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- An existing storage class capable of dynamically provisioning persistent volumes.

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

This command ensures that the cluster API is reachable and that Prometheus and Alertmanager use ephemeral storage.

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

Instructions

- 1. Determine if your cluster already contains a configuration map for the cluster monitoring operator.
- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Look for the `cluster-monitoring-config` configuration map in the `openshift-monitoring` namespace. This command returns an error because the `cluster-monitoring-config` configuration map does not exist by default.

```
[student@workstation ~]$ oc get configmap cluster-monitoring-config \
-n openshift-monitoring
Error from server (NotFound): configmaps "cluster-monitoring-config" not found
```

► 2. Identify storage files used by Prometheus.

- 2.1. Connect to the `prometheus` container in one of the `prometheus-k8s` pods in the `openshift-monitoring` namespace, and list files in the `/prometheus` directory. Although the Prometheus pods use ephemeral storage, the pods are stopped and restarted in the classroom environment rather than being recreated. The result is that you might see directories created on a previous day.

```
[student@workstation ~]$ oc exec -it prometheus-k8s-0 -c prometheus \
-n openshift-monitoring -- ls -l /prometheus
total 20
drwxr-sr-x. 3 nobody nobody    68 Nov  5 19:00 01FKRQHN1TSFFAS4VYDGP7A04
drwxr-sr-x. 2 nobody nobody    34 Nov  5 19:00 chunks_head
-rw-r--r--. 1 nobody nobody 20001 Nov  5 19:08 queries.active
drwxr-sr-x. 3 nobody nobody   81 Nov  5 19:00 wal
```



Note

Although your output displays the `queries.active` file and the `wal` directory, all other output will vary.

- 2.2. Connect to the `prometheus` container in one of the `prometheus-k8s` pods in the `openshift-monitoring` namespace, and check to see if the `/prometheus` directory is a mount point.

```
[student@workstation ~]$ oc exec -it prometheus-k8s-0 -c prometheus \
-n openshift-monitoring -- df -h /prometheus
Filesystem           Size  Used Avail Use% Mounted on
/dev/mapper/coreos-luks-root-nocrypt  40G  9.8G  30G  25% /prometheus
```

Although the `/prometheus` directory is a mount point, the volume definition in the `prometheus-k8s` stateful set uses `emptyDir` to indicate that it is a placeholder. The `/dev/mapper/coreos-luks-root-nocrypt` devices indicates ephemeral storage on the node.



Note

The values of your output for the `Used`, `Avail`, and `Use%` columns might differ.

- 3. Delete the Prometheus pods and notice how the new pods do not contain the files previously identified.

- 3.1. Delete the `prometheus-k8s` pods in the `openshift-monitoring` namespace.

```
[student@workstation ~]$ oc delete pods -l app=prometheus \
-n openshift-monitoring
pod "prometheus-k8s-0" deleted
pod "prometheus-k8s-1" deleted
```

- 3.2. After the new `prometheus-k8s-0` pod transitions to a running state, list the contents of the `/prometheus` directory in the `prometheus-k8s-0` pod. Aside from the `/prometheus/wal` directory, the previously listed directories no longer exist.

```
[student@workstation ~]$ oc exec -it prometheus-k8s-0 -c prometheus \
-n openshift-monitoring -- ls -l /prometheus
total 20
drwxr-sr-x. 2 nobody nobody 6 Nov 5 19:13 chunks_head
-rw-r--r--. 1 nobody nobody 20001 Nov 5 19:13 queries.active
drwxr-sr-x. 2 nobody nobody 22 Nov 5 19:13 wal
```

- ▶ 4. Configure the cluster monitoring operator so that both Prometheus and Alertmanager use persistent storage.

- 4.1. Identify the existing storage class for the cluster. You will use this storage class name for Prometheus and Alertmanager.

```
[student@workstation ~]$ oc get storageclasses
NAME          PROVISIONER
RECLAIMPOLICY ...
nfs-storage (default)   k8s-sigs.io/nfs-subdir-external-provisioner   Delete
...
```



Warning

Using NFS to back persistent volumes used by core services, such as Prometheus, is not recommended for production environments. Block storage is recommended for Prometheus.

The classroom environment uses NFS storage for simplicity.

- 4.2. Change to the ~/D0380/labs/monitor-storage directory.

```
[student@workstation ~]$ cd ~/D0380/labs/monitor-storage
```

- 4.3. Modify the persistent-storage.yml file.

```
[student@workstation monitor-storage]$ vim persistent-storage.yml
```

- 4.4. Ensure that the file matches the following bold lines, and then save the file.

```
prometheusK8s:
  retention: 15d
  volumeClaimTemplate:
    spec:
      storageClassName: nfs-storage
      resources:
        requests:
          storage: 40Gi
alertmanagerMain:
  volumeClaimTemplate:
    spec:
      storageClassName: nfs-storage
```

```
resources:
  requests:
    storage: 20Gi
```

**Important**

This configuration indicates that each Prometheus pod requests a 40GB persistent volume claim, and each Alertmanager pod requests a 20GB persistent volume claim.

Although the storage class creates each persistent volume claim with the desired size, the classroom environment only allocates 40GB in total to the NFS server (`utility.lab.example.com`).

- 4.5. Apply the configuration using the `oc create` command.

```
[student@workstation monitor-storage]$ oc create configmap \
  cluster-monitoring-config -n openshift-monitoring \
  --from-file config.yaml=persistent-storage.yaml
configmap/cluster-monitoring-config created
```

- ▶ 5. Verify that Prometheus uses persistent storage.

- 5.1. List the persistent volume claims using the `app=prometheus` label in the `openshift-monitoring` namespace. The `nfs-storage` storage class creates the persistent volume claims dynamically, and this process might take up to a minute to complete. After the persistent volume claims display, press `Ctrl+C` to exit the `watch` command.

```
[student@workstation monitor-storage]$ watch oc get persistentvolumeclaims \
  -l app=prometheus -n openshift-monitoring
NAME                               STATUS   VOLUME
prometheus-k8s-db-prometheus-k8s-0   Bound    pvc-d029d039-d26c-4e68-a743-3b3...
prometheus-k8s-db-prometheus-k8s-1   Bound    pvc-e804510d-7e38-4687-b3ba-4c5...
```

- 5.2. Verify that the `prometheus-k8s-db-prometheus-k8s-0` persistent volume claim is mounted to the `/prometheus` directory in the `prometheus-k8s-0` pod. The `/exports` directory on the NFS server (listed under Filesystem) includes the name of the namespace, the name of the persistent volume claim, and the name of the associated volume.

```
[student@workstation monitor-storage]$ oc exec -it prometheus-k8s-0 \
  -c prometheus -n openshift-monitoring -- df -h /prometheus
Filesystem                                Size  Used
Avail Use% Mounted on
192.168.50.254:/exports-ocp4/openshift-monitoring-.../prometheus-db  40G  705M
  40G   2% /prometheus
```

- ▶ 6. Verify that Alertmanager uses persistent storage.

- 6.1. List the persistent volume claims using the `app=alertmanager` label in the `openshift-monitoring` namespace.

```
[student@workstation monitor-storage]$ oc get persistentvolumeclaims \
-l app=alertmanager -n openshift-monitoring
NAME                                     STATUS  VOLUME...
alertmanager-main-db-alertmanager-main-0  Bound   pvc-4f168ac5-2161-4daa-a8...
alertmanager-main-db-alertmanager-main-1  Bound   pvc-51d64a64-4abc-43c8-af...
alertmanager-main-db-alertmanager-main-2  Bound   pvc-404420c4-a7e3-4ff4-b1...
```

- 6.2. Verify that the alertmanager-main-db-alertmanager-main-0 persistent volume claim is mounted to the /alertmanager directory in the alertmanager-main-0 pod. The /exports directory on the NFS server (listed under Filesystem) includes the name of the namespace, the name of the persistent volume claim, and the name of the associated volume.

```
[student@workstation monitor-storage]$ oc exec -it alertmanager-main-0 \
-c alertmanager -n openshift-monitoring -- df -h /alertmanager
Filesystem                                Size  Used
Avail Use% Mounted on
192.168.50.254:/exports-ocp4/openshift-monitoring-.../alertmanager-db  40G  731M
40G  2% /alertmanager
```

- 7. Change to the /home/student directory.

```
[student@workstation monitor-storage]$ cd
```

Finish

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

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

▶ Lab

Managing Cluster Monitoring and Metrics

In this lab, you will configure Alertmanager to send alert notifications and configure persistent storage for Prometheus.

Outcomes

You should be able to:

- Configure Alertmanager to send email notifications.
- Configure Prometheus to use persistent storage.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that:

- The cluster API is reachable.
- The **utility.lab.example.com** host is configured to receive email.
- The **ocp-admins@example.com** email alias sends messages to the **lab** user on **utility.lab.example.com**.
- The cluster contains a firing **TestAlert** alert.
- Prometheus and Alertmanager use ephemeral storage.

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



Note

- Consider using the *Cluster Monitoring* chapter of the Red Hat OpenShift Container Platform Monitoring guide as a reference.

Instructions

1. Configure Alertmanager to send email notifications for all warning alerts. Use the values in the following table to configure Alertmanager:

Setting	Value
smtp_smarthost	192.168.50.254:25
smtp_from	alerts-review@ocp4.example.com
to	ocp-review@example.com
repeat_interval	2m

**Note**

You are not required to configure a secure connection from Alertmanager to the SMTP host.

- Configure Prometheus to use persistent storage. Request 25Gi of storage space from the nfs-storage storage class. Ensure that Prometheus retains the data for ten days.

**Note**

The ~/D0380/labs/monitor-review/persistent-storage.yml file contains a template for persistent storage configuration.

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 monitor-review grade
```

Finish

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

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

► Solution

Managing Cluster Monitoring and Metrics

In this lab, you will configure Alertmanager to send alert notifications and configure persistent storage for Prometheus.

Outcomes

You should be able to:

- Configure Alertmanager to send email notifications.
- Configure Prometheus to use persistent storage.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.

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

This command ensures that:

- The cluster API is reachable.
- The **utility.lab.example.com** host is configured to receive email.
- The **ocp-admins@example.com** email alias sends messages to the **lab** user on **utility.lab.example.com**.
- The cluster contains a firing **TestAlert** alert.
- Prometheus and Alertmanager use ephemeral storage.

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



Note

- Consider using the *Cluster Monitoring* chapter of the Red Hat OpenShift Container Platform Monitoring guide as a reference.

Instructions

1. Configure Alertmanager to send email notifications for all warning alerts. Use the values in the following table to configure Alertmanager:

Setting	Value
smtp_smarthost	192.168.50.254:25
smtp_from	alerts-review@ocp4.example.com
to	ocp-review@example.com
repeat_interval	2m

**Note**

You are not required to configure a secure connection from Alertmanager to the SMTP host.

- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Extract the `alertmanager-main` secret, which contains the current `alertmanager.yaml` configuration file.

```
[student@workstation ~]$ oc extract secret/alertmanager-main \
-n openshift-monitoring --to=/tmp
/tmp/alertmanager.yaml
```

- 1.3. Update the content of the `/tmp/alertmanager.yaml` file with the required configuration. The following configuration is valid:

```
"global":
  "resolve_timeout": "5m"
  smtp_smarthost: 192.168.50.254:25
  smtp_from: alerts-review@ocp4.example.com
  smtp_require_tls: false
"receivers":
- "name": default
- name: email-notification
  email_configs:
    - to: ocp-review@example.com
"route":
  "group_by":
    - "namespace"
  "group_interval": "5m"
  "group_wait": "30s"
  "receiver": default
  "repeat_interval": 2m
  "routes":
    - "match":
```

```
"alertname": "Watchdog"
"receiver": default
- match:
  severity: warning
  receiver: email-notification
```

**Note**

In a previous guided exercise, you preprocess the `alertmanager.yaml` file to remove extraneous quote ("") characters. If you preprocess the file, then its content matches the style of other YAML files in the course.

You do not need to remove the extraneous quote characters. This solution shows only the required changes to the extracted `/tmp/alertmanager.yaml` file.

- 1.4. Update the `alertmanager-main` secret with the content of the local `alertmanager.yaml` file:

```
[student@workstation ~]$ oc set data secret/alertmanager-main \
-n openshift-monitoring --from-file=/tmp/alertmanager.yaml
secret/alertmanager-main data updated
```

- 1.5. Verify that the new Alertmanager configuration loads without any errors:

```
[student@workstation ~]$ oc logs -f -c alertmanager alertmanager-main-0 \
-n openshift-monitoring
...output omitted...
level=info ts=... msg="Loading configuration file" file=.../alertmanager.yaml
level=info ts=... msg="Completed loading of configuration file" ...
```

**Note**

If you see configuration errors in the logs, then modify the `/tmp/alertmanager.yaml` file and apply your changes again.

2. Configure Prometheus to use persistent storage. Request 25Gi of storage space from the `nfs-storage` storage class. Ensure that Prometheus retains the data for ten days.

**Note**

The `~/D0380/labs/monitor-review/persistent-storage.yaml` file contains a template for persistent storage configuration.

- 2.1. Change to the `~/D0380/labs/monitor-review` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/monitor-review
```

- 2.2. Modify the `persistent-storage.yaml` file to match the storage requirements for Prometheus.

```
prometheusK8s:
  retention: 10d
  volumeClaimTemplate:
    spec:
      storageClassName: nfs-storage
      resources:
        requests:
          storage: 25Gi
```

- 2.3. Use the `persistent-storage.yaml` file to create the `cluster-monitoring-config` configuration map.

```
[student@workstation monitor-review]$ oc create cm cluster-monitoring-config \
-n openshift-monitoring --from-file="config.yaml=persistent-storage.yaml"
configmap/cluster-monitoring-config created
```

- 2.4. Verify that persistent volume claims are created for Prometheus. After the persistent volume claims display, press `Ctrl+C` to exit the watch command.

```
[student@workstation monitor-review]$ watch oc get persistentvolumeclaims \
-n openshift-monitoring
NAME                   ...  CAPACITY   ...  STORAGECLASS  AGE
prometheus-k8s-db-prometheus-k8s-0  ...  25Gi       ...  nfs-storage   23h
prometheus-k8s-db-prometheus-k8s-1  ...  25Gi       ...  nfs-storage   23h
```

- 2.5. Change to the `/home/student` directory.

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

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

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

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

Summary

In this chapter, you learned:

- OpenShift 4 installs the cluster monitoring stack by default.
- Prometheus rules generate alerts for potential cluster problems.
- Alertmanager can send alerts to external locations and display alerts in the OpenShift web console.
- Grafana graphically displays metrics collected by Prometheus and you can use it to identify trends and problems for the cluster, nodes, and namespaces.
- How to configure Prometheus and Alertmanager to use persistent storage to prevent data loss.

Chapter 10

Provisioning and Inspecting Cluster Logging

Goal

Deploy and query cluster-wide logging, and diagnose common issues using tools.

Objectives

- Deploy and configure cluster logging.
- Inspect Openshift and application logs using Lucene queries in the Kibana UI.
- Create charts to visualize logs with Kibana.
- Diagnose cluster logging problems with debugging and monitoring tools.

Sections

- Deploying Cluster Logging (and Guided Exercise)
- Querying Cluster Logs with Kibana (and Guided Exercise)
- Visualizing Cluster Logs with Kibana (and Guided Exercise)
- Diagnosing Cluster Logging Problems (and Guided Exercise)

Lab

Provisioning and Inspecting Cluster Logging

Deploying Cluster Logging

Objectives

After completing this section, you should be able to deploy and configure cluster logging.

Discussing Cluster Logging

The main benefit of cluster logging is the aggregation of all the logs from the pods and node of an OpenShift cluster to a centralized location. Centralized logging allows for easier searching, visualizing, and reporting of data.

Cluster Logging Components

logStore

The logStore is the Elasticsearch cluster that:

- Stores the logs into indexes.
- Provides RBAC access to the logs.
- Provides data redundancy.

collection

Implemented with Fluentd, the collector collects node and application logs, adds pod and namespace metadata, and stores them in the logStore. The collector is a DaemonSet, so there is a Fluentd pod on each node.

visualization

The centralized web UI from Kibana displays the logs and provides a way to query and chart the aggregated data.

event routing

The Event Router monitors the OpenShift events API and sends the events to STDOUT so the collector can forward them to the logStore. The events from OpenShift are stored in the `infra` index in Elasticsearch.

Installing the Elasticsearch Operator

The Elasticsearch Operator handles the creation of and updates to the Elasticsearch cluster defined in the Cluster Logging Custom Resource. Each node in the Elasticsearch cluster is deployed with a PVC named and managed by the Elasticsearch Operator. A unique Deployment object is created for each Elasticsearch node to ensure that each Elasticsearch node has a storage volume of its own.

The Elasticsearch Operator must be installed in a namespace other than `openshift-operators` to avoid possible conflicts with metrics from other community operators. Create and use the `openshift-operators-redhat` namespace for the Elasticsearch Operator.

```

apiVersion: v1
kind: Namespace
metadata:
  name: openshift-operators-redhat
  annotations:
    openshift.io/node-selector: ""
  labels:
    openshift.io/cluster-monitoring: "true"

```

Create an `OperatorGroup` object to install the operator in all namespaces and a `Subscription` object to subscribe the `openshift-operators-redhat` namespace to the operator.

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-operators-redhat
  namespace: openshift-operators-redhat
spec: {}

---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: "elasticsearch-operator"
  namespace: "openshift-operators-redhat"
spec:
  channel: "4.6"
  installPlanApproval: "Automatic"
  source: "redhat-operators"
  sourceNamespace: "openshift-marketplace"
  name: "elasticsearch-operator"

```

Create the RBAC objects to grant Prometheus permission to access the `openshift-operators-redhat` namespace.

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: prometheus-k8s
  namespace: openshift-operators-redhat
rules:
  - apiGroups:
    - ""
      resources:
        - services
        - endpoints
        - pods
      verbs:
        - get
        - list
        - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding

```

```

metadata:
  name: prometheus-k8s
  namespace: openshift-operators-redhat
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: prometheus-k8s
subjects:
  - kind: ServiceAccount
    name: prometheus-k8s
    namespace: openshift-operators-redhat
---

```

Verify that operator is available in each namespace.

```

[user@host ~]$ oc get csv -A
NAMESPACENAMESPACE      NAME                           ... PHASE
default      default      default      default      ...
...output omitted...

```

Installing the Cluster Logging Operator

The Cluster Logging Operator creates components detailed in the cluster logging custom resource, and updates the deployment upon any changes to the custom resource. Similar to the Elasticsearch Operator, the Cluster Logging Operator requires: namespace, operator group, and subscription objects.

```

apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging
  annotations:
    openshift.io/node-selector: ""
  labels:
    openshift.io/cluster-logging: "true"
    openshift.io/cluster-monitoring: "true"
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  targetNamespaces:
    - openshift-logging
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging

```

```
namespace: openshift-logging
spec:
  channel: "4.6"
  name: cluster-logging
  source: redhat-operators
  sourceNamespace: openshift-marketplace
---
```

Confirm the installation.

```
[user@host ~]$ oc get csv -n openshift-logging
NAME          ... PHASE
clusterlogging.4.6.0-202108311008   ... Succeeded
elasticsearch-operator.4.6.0-202108311008 ... Succeeded
```

Deploying a Cluster Logging Instance

The cluster logging custom resource defines the configuration for an instance of the cluster logging components and provides the configuration for the components.

Configuring Elasticsearch

Consider the following when configuring the Elasticsearch component.

- For availability and scalability, Elasticsearch requires a minimum of three cluster nodes. The first three nodes of the Elasticsearch cluster are created with master, client, and data roles. Additional nodes are provisioned as data nodes, with client and data roles.
- Elasticsearch uses either persistent or ephemeral storage, but requires persistent storage to prevent data loss. Fast storage, such as SSDs, is preferred over spinning disks. Use dedicated local storage instead of NFS, SMB, or Amazon EBS. Consider the logging requirements of the applications in the OpenShift cluster and the OpenShift nodes when sizing the storage for Elasticsearch.
- The redundancy policy determines how Elasticsearch shards are replicated across data nodes in the cluster. Several Elasticsearch shards combine to make a single Elasticsearch index. Each shard is responsible for a subset of the data that an Elasticsearch index provides.

Redundancy mode	Description
FullRedundancy	Elasticsearch copies the shards for each index to every data node in the cluster. Fully replicated shards provide the most redundancy to protect from accidental data loss.
MultipleRedundancy	The primary shards for each index are replicated to half of the data nodes. This provides a good balance between performance and redundancy.
SingleRedundancy	Each primary shard is copied once to another node. If at least two data nodes remain available, then the logs are recoverable. If the Elasticsearch cluster has 5 or more nodes, SingleRedundancy performs better than MultipleRedundancy.
ZeroRedundancy	Shards are not replicated. Data loss could happen if a node fails.

Example Cluster Logging Instance

```

apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      nodeSelector:
        node-role.kubernetes.io/infra: ''
    storage:
      storageClassName: "local-blk"
      size: 50G
      redundancyPolicy: "MultipleRedundancy"
  visualization:
    type: "kibana"
    kibana:
      nodeSelector:
        node-role.kubernetes.io/infra: ''
      replicas: 1
  collection:
    logs:
      type: "fluentd"
      fluentd: {}

```



Note

All of the components target dedicated infrastructure nodes.

Confirm the cluster logging components statuses using the command `oc get clusterlogging -n openshift-logging instance -o yaml`.

```

[user@host ~]$ oc get clusterlogging -n openshift-logging instance -o yaml
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
...output omitted...
status:
  collection:
    logs:
      fluentdStatus:
        daemonSet: fluentd
        ...output omitted...
      pods:
        failed: []
        notReady: []
        ready:
        - fluentd-b7xqq

```

```

- fluentd-l6dzb
- fluentd-lkd98
- fluentd-pfpqn
- fluentd-sz8rs
- fluentd-x6lbh
- fluentd-xlclg
- fluentd-xlmfd
- fluentd-zjplq
curation: {}
logStore:
  elasticsearchStatus:
    - cluster:
        activePrimaryShards: 8
        activeShards: 10
        initializingShards: 0
        numDataNodes: 2
        numNodes: 2
        pendingTasks: 0
        relocatingShards: 0
        status: green
        unassignedShards: 0
        ...output omitted...
    pods:
      client:
        failed: []
        notReady: []
        ready:
          - elasticsearch-cdm-ua6ezqs6-1-c7b8cdff4-44mx5
          - elasticsearch-cdm-ua6ezqs6-2-7fb79c755-gnr4f
      data:
        failed: []
        notReady: []
        ready:
          - elasticsearch-cdm-ua6ezqs6-1-c7b8cdff4-44mx5
          - elasticsearch-cdm-ua6ezqs6-2-7fb79c755-gnr4f
    master:
      failed: []
      notReady: []
      ready:
        - elasticsearch-cdm-ua6ezqs6-1-c7b8cdff4-44mx5
        - elasticsearch-cdm-ua6ezqs6-2-7fb79c755-gnr4f
  shardAllocationEnabled: all
visualization:
  kibanaStatus:
    - deployment: kibana
      pods:
        failed: []
        notReady: []
        ready:
          - kibana-969c74586-flf5z
  replicaSets:
    - kibana-969c74586
  replicas: 1

```

Installing the Event Router

By default, Kubernetes events are saved in the cluster etcd instance. The event router monitors the events API and prints events to standard output so Fluentd can collect them and send them to Elasticsearch. Elasticsearch stores the events in the `infra` index.

The OpenShift documentation provides a template that you use to install the Event Router. The template defines a service account, deployment, and other RBAC resources for the Event Router.

To install the Event Router:

- Download the template from OpenShift Container Platform documentation.
- Use the `oc process` command to create a manifest file for the Event Router.
- Use the `oc apply` command to apply the manifest resources to the OpenShift cluster.

Creating Kibana Index Patterns

Every log message is converted into a JSON document that Elasticsearch stores in an index. Elasticsearch uses indexes to organize related documents. Elasticsearch applies role-based access control (RBAC) rules to indexes based on OpenShift project permissions. Beginning in OpenShift Container Platform 4.5, log message documents are stored in an index with one of the following prefixes:

`infra-`

An index with this prefix stores pod log messages for infrastructure projects. An infrastructure project includes project names that have a `openshift-` or `kube-` prefix.

`app-`

An index with this prefix stores pod log messages for all projects except infrastructure projects.

`audit-`

An index with this prefix stores audit log messages. Audit logs allow you to review the OpenShift API activity of a user, administrator, or OpenShift component (such as a service account).

To search for log messages in Kibana, you specify an index pattern. An index pattern defines a set of Elasticsearch indexes that you use in a query. An index pattern simplifies an Elasticsearch query because logs might exist in many different indexes.

The first time you log in to Kibana, you must create the Kibana index patterns. Red Hat recommends that you create an index pattern for each of preceding Elasticsearch indexes:

`infra`

Create this index pattern to match any of the `infra-*` Elasticsearch indexes.

`app`

Create this index pattern to match any of the `app-*` Elasticsearch indexes.

`audit`

Create this index pattern to match any of the `audit-*` Elasticsearch indexes.



References

For more information, refer to the *Understanding cluster logging*, *Installing cluster logging*, and *Configuring your cluster logging deployment* chapters in the Red Hat OpenShift Container Platform 4.6 *Logging* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/logging/index

For more information on the Event Router, refer to the *Collecting and storing Kubernetes event* chapter in the Red Hat OpenShift Container Platform 4.6 *Logging* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/logging/index#cluster-logging-eventrouter

Tune for indexing speed

https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-indexing-speed.html#_use_faster_hardware

► Guided Exercise

Deploying Cluster Logging

In this exercise you will deploy and configure cluster logging.

Outcomes

You should be able to:

- Install the Elasticsearch and Cluster Logging Operators.
- Install the Event Router.
- Install and configure cluster logging.

Before You Begin

This exercise depends upon the availability of extra workers. *Guided Exercise: Adding Compute Nodes* describes how to add extra worker nodes.

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. The command ensures that extra OpenShift nodes are available to use.

```
[student@workstation ~]$ lab logging-deploy start
```

Instructions

- 1. Review the provided Elasticsearch Operator, Cluster Logging Operator, cluster logging instance, and event router manifests.
- 1.1. Change to the `~/D0380/labs/logging-deploy` directory.

```
[student@workstation ~]$ cd ~/D0380/labs/logging-deploy
```



Note

The Elasticsearch cluster is provisioned in a minimal setup to accommodate the classroom. A standard deployment would have at least three Elasticsearch nodes, each with at least 16 GB memory.

- 2. Update the cluster logging instance manifest `cl-minimal.yml`.
- 2.1. Set the `redundancyPolicy` to `MultipleRedundancy`
 - 2.2. The updated manifest should match the following.

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
```

```

name: "instance"
namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    retentionPolicy:
      application:
        maxAge: 1d
      infra:
        maxAge: 7d
      audit:
        maxAge: 7d
    elasticsearch:
      nodeCount: 2
      nodeSelector:
        node-role.kubernetes.io/infra: ''
    storage:
      storageClassName: "local-blk"
      size: 20G
    redundancyPolicy: "MultipleRedundancy"
  resources:
    limits:
      memory: "8Gi"
    requests:
      cpu: "1"
      memory: "8Gi"
  visualization:
    type: "kibana"
    kibana:
      nodeSelector:
        node-role.kubernetes.io/infra: ''
      replicas: 1
  curation:
    type: "curator"
    curator:
      schedule: "30 3 * * *"
  collection:
    logs:
      type: "fluentd"
      fluentd: {}

```

► 3. Run the provided playbook to install cluster logging.

3.1. Log in to your OpenShift cluster as the `admin` user.

```
[student@workstation logging-deploy]$ oc login -u admin -p redhat \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...
```

3.2. Execute the playbook.

```
[student@workstation logging-deploy]$ ansible-playbook cluster-logging.yml -v
Using /etc/ansible/ansible.cfg as config file

PLAY [Install Cluster Logging] ****
TASK [Create objects from the manifests] ****
changed: [localhost] => (item=logging-operator.yml) => changed=true
  ansible_loop_var: item
  item: logging-operator.yml
  result:
...output omitted...
PLAY RECAP ****
localhost      : ok=1  changed=1  unreachable=0  failed=0  skipped=0  ...


```

▶ 4. Confirm the installation.

Wait a few moments for the installation of the Elasticsearch Operator to complete in all namespaces.

```
[student@workstation logging-deploy]$ oc get csv -A
NAMESPACE      NAME          ...  PHASE
default        elasticsearch-operator.4.6.0-...  ...  Succeeded
kube-node-lease elasticsearch-operator.4.6.0-...  ...  Succeeded
kube-public     elasticsearch-operator.4.6.0-...  ...  Succeeded
kube-system     elasticsearch-operator.4.6.0-...  ...  Succeeded
...output omitted...
```

Verify the Cluster Logging Operator succeeded in the openshift-logging namespace.

```
[student@workstation logging-deploy]$ oc get csv -n openshift-logging
NAME          ...  PHASE
clusterlogging.4.6.0-...  ...  Succeeded
elasticsearch-operator.4.6.0-...  ...  Succeeded
```

The cluster logging components and the Event Router take a few minutes to deploy. Watch the components to validate that everything deploys.

```
[student@workstation logging-deploy]$ watch oc get deployment,pod,pvc,svc,route \
-n openshift-logging
```

▶ 5. Log in to the Kibana UI.

- 5.1. Use the `oc get routes` command to discover the route, or click the Application Launcher and select **Observability** → **Logging** in the OpenShift web console.

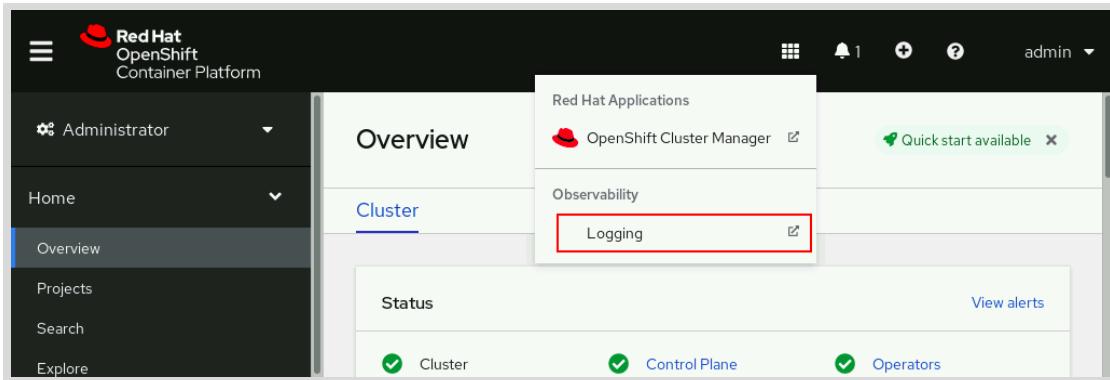


Figure 10.1: Accessing Kibana from the OpenShift web console

- 5.2. If prompted, accept the TLS certificate for the Kibana UI, and then click **Log in with OpenShift**.
- 5.3. Also accept the TLS certificate of the OpenShift OAuth server, if prompted, and then click **htpasswd_provider** to enter the OpenShift log in credentials page.
- 5.4. Log in as admin with the **redhat** password, and then click **Allow selected permissions**.
- ▶ 6. Click the **Create index pattern** to the app index pattern.
 - 6.1. Type **app** in the **Index pattern** text field.
 - 6.2. Click **Next step**.
 - 6.3. Select **@timestamp** from the **Time Filter field name** list.
 - 6.4. Click **Create index pattern**.
- ▶ 7. Create the **infra** index pattern.
 - 7.1. Click again **Create index pattern**.
 - 7.2. Type **infra** in the **Index pattern** text field.
 - 7.3. Click **Next step**.
 - 7.4. Select **@timestamp** from the **Time Filter field name** list.
 - 7.5. Click **Create index pattern**.
- ▶ 8. Change to the **/home/student** directory.

```
[student@workstation logging-deploy]$ cd
```



Note

If the Elasticsearch Cluster does not come up correctly, or the Kibana UI shows errors with Elasticsearch, then run the same playbook with `k8s_state` set to `absent`.

```
$ ansible-playbook cluster-logging.yml -v -e k8s_state=absent
```

The removal of the logging component resources takes several minutes. Use the `oc get` command to validate that logging component resources are removed from the `openshift-logging` namespace before rerunning the playbook to install cluster logging.

Finish

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

```
[student@workstation ~]$ lab logging-deploy finish
```

Querying Cluster Logs with Kibana

Objectives

After completing this section, you should be able to inspect OpenShift and application logs using Lucene queries in the Kibana user interface (UI).

Introducing the Kibana UI

There are two ways to navigate to Kibana:

- Use the `oc` command to discover the route: `oc get routes -n openshift-logging`.
- Click the Application Launcher, and then select **Observability** → **Logging**.

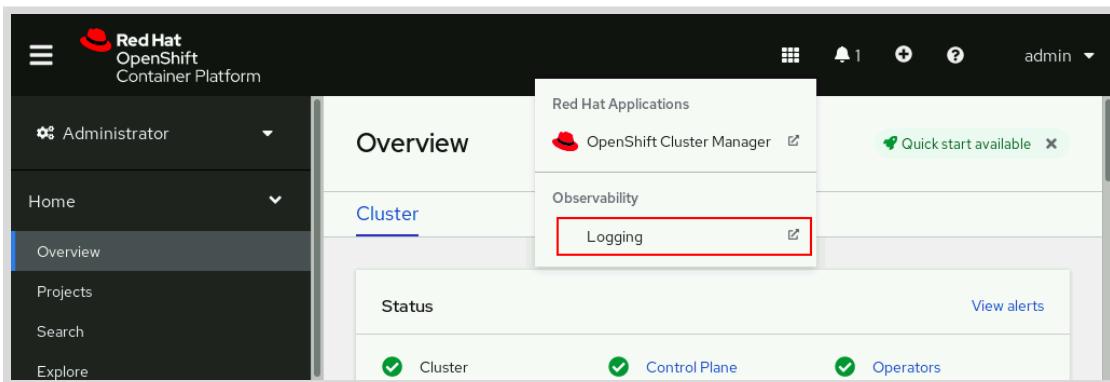
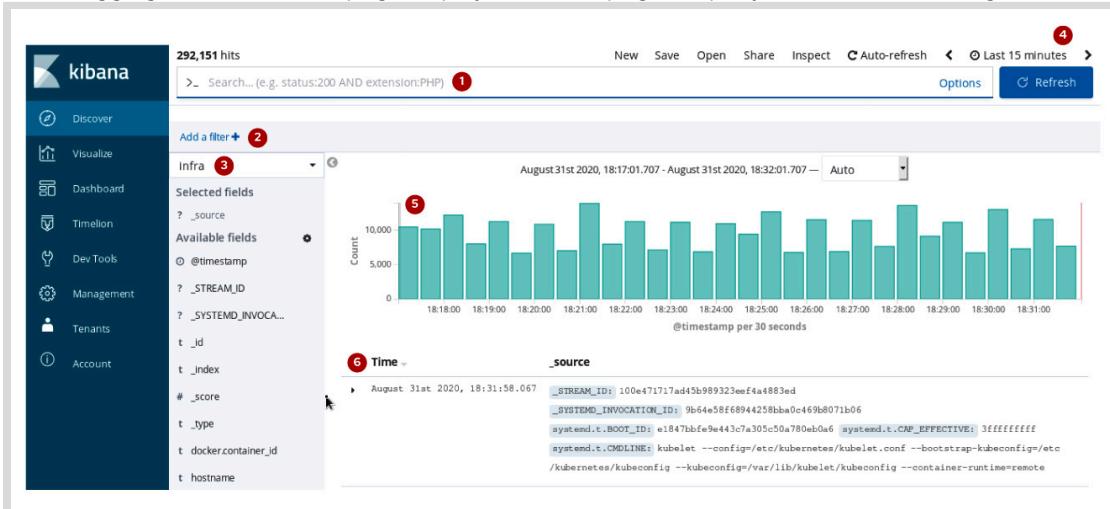


Figure 10.2: Accessing Kibana from the OpenShift web console

After logging in, the **Discover** page displays. Use this page to query Elasticsearch for logs.



- ① Search the logs using a Lucene query.
- ② Add a filter to narrow the search based on the existence or value of a field.

- ③ Select the index pattern to filter the query by project.
- ④ Set the time range to a relative range, such as `Last 15 minutes`, or an absolute time range.
- ⑤ A time series chart displaying a count of the log records that match the search query.
- ⑥ Table of log records that match the search query.

Selecting Indexes

To query for logs in Kibana, first select the correct index pattern for your query. An index pattern specifies a set of Elasticsearch indexes to search. For example, the app index pattern searches for messages in any `app-*` Elasticsearch index.

Selecting Table Fields

Elasticsearch stores OpenShift logs as JSON documents. A JSON document is similar to a record in a relational database table. Each log record defines a set of fields and the associated value for each field.

The default Kibana **Discover** view displays a table of logs with `Time` and `_source` fields. The `Time` field defines a time stamp for the log message. The `_source` field contains the JSON document string for the record.

You can display other fields in the table by either:

Time	_source
September 1st 2020, 09:28:31.866	<pre> { "docker.container_id": "df46d96f70e7208bc3737b4ca99d0dc0e065463330b12a754b93f5cce682c80a", "kubernetes.container_name": "local-storage-provisioner", "kubernetes.namespace_name": "storage-local", "kubernetes.pod_name": "local-disks-local-provisioner-9244p", "kubernetes.container_image": "registry.redhat.io/openshift4/cse-local-storage-static-provisioner@sha256:eddba41ba428a9a23d7f8331c4f51c66fa0585a32f6ea89f2b9f3974d8ce5474" } </pre>
	<pre> { "@timestamp": "2020-09-01T09:28:31.866Z", "id": "Y2QwGMM5MmItNjBhZC00YTJhLTlInjJtYWF1MDVjMnU1ZDEw", "index": "app-000001", "score": null, "type": "log" } </pre>
	<pre> { "host": { "name": "app00001" } } </pre>
	<pre> { "id": "df46d96f70e7208bc3737b4ca99d0dc0e065463330b12a754b93f5cce682c80a", "level": "INFO", "log": "Starting local storage provisioner at /var/run/origin-worker05", "message": "Local storage provisioner starting up", "namespace": "storage-local", "pod": "local-disks-local-provisioner-9244p", "process": "main" } </pre>
	<pre> { "image": "registry.redhat.io/openshift4/cse-local-storage-static-provisioner@sha256:eddba41ba428a9a23d7f8331c4f51c66fa0585a32f6ea89f2b9f3974d8ce5474" } </pre>

- ① Clicking **add** next to an entry in the **Available Fields** list, or
- ② Expanding a log record by clicking the arrow next to it, and then clicking the table icon next to the field to add.

Querying Logs

Elasticsearch is based on the Lucene search engine. To search for log messages in Elasticsearch, you must use Lucene syntax to execute queries.

Introducing Lucene Queries

In Lucene, a set of terms forms a basic query within the selected index pattern.

The following examples demonstrate basic Lucene queries: * dns finds all records that contain the dns text string in any of the indexed text fields. * dns dig finds all records that contain either the dns or dig term in any of the indexed fields. A record must contain only one of the terms, not all of the them.

Use double quotation marks ("") to create a single term from one or more terms. For example, the "openshift api" query finds records with the phrase openshift api in any indexed text field.

Similarly, use double quotation marks ("") to wrap terms that contain symbols, such as -. For example, to search for the phrase openshift-apiserver-operator in any indexed text field, you must use a Lucene query of "openshift-apiserver-operator".

If a query contains multiple terms, then you can use the + symbol to identify terms that must match the record. Similarly, you can use the - symbol to identify terms that must not match the record. For example, the +dns -api query searches for all records that contain the term dns but not the term api.

Filtering Queries

Fluentd adds OpenShift metadata to pod logs to provide fields for querying and filtering. Some of the fields include the following:

hostname

The host name of the OpenShift node for the pod that generated the message.

kubernetes.flat_labels

The field contains key/value pairs in the form of key=value. Each key/value pair corresponds to a label for the pod that generated the log message.

kubernetes.container_name

The name of the container that generated the log message.

kubernetes.namespace_name

The name of the pod project for the pod that generated the log message.

kubernetes.pod_name

The name of the pod that generated the log message.

level

The log level of the message.

message

The actual log message, which corresponds to a single line of output from a pod.

To restrict a term search to specific field, specify the field and term together, separated by a colon (:) character. For example, a level:info Lucene query returns records that have contain a info text string in the value of the level field.

For example, enter the following query to search for all messages that contain the term elected in the openshift-etcd namespace.

```
+kubernetes.namespace_name:"openshift-etcd" +message:elected
```

Optionally, click **Add a filter** to filter the results of a query. A form to filter based on values or the existence of fields displays.

Finding OpenShift Event Logs

The Event Router pod watches OpenShift and sends events to the cluster logging system. Event records are stored in the Elasticsearch `infra` index. You can find event records using the following query under the `infra` index: `kubernetes.event : *`.

Event records define several fields that contain event metadata. These fields begin with `kubernetes.event..`. You can use these fields in queries and filters to retrieve events from event metadata.

Examples of the event fields that you can use include:

`kubernetes.event.involvedObject.name`

This field contains the name of the OpenShift resource associated with the event. A common pattern to deploy an application is to use the same name for all the resources. Using this field, you can find any event that matches a resource name without regard to the resource type.

`kubernetes.event.involvedObject.namespace`

This field contains the project name that corresponds to the OpenShift resource for the event. Use this field to query for any events from a specific project.

`kubernetes.event.reason`

This field contains a string that represents the reason for the event. The values in this field correspond to the values in the `REASON` column that displays in the output of the `oc get events` command.

`kubernetes.event.type`

This field contains the type of message. For example, the `kubernetes.event.type:warning` Lucene query finds warning events.



References

Query string syntax

<https://www.elastic.co/guide/en/elasticsearch/reference/6.8/query-dsl-query-string-query.html#query-string-syntax>

► Guided Exercise

Querying Cluster Logs with Kibana

In this exercise you will query logs from the Kibana graphical user interface.

Outcomes

You should be able to use Lucene queries and filters to find logs.

Before You Begin

Before starting this exercise, you must set up cluster logging by completing the *Guided Exercise: Deploying Cluster Logging*.

As the student user on the **workstation** machine, use the `lab` command to prepare your system for this exercise. The command deploys OpenShift pods that frequently generate logs.

```
[student@workstation ~]$ lab logging-query start
```

Instructions

- ▶ 1. Navigate to the Kibana UI and log in as the **admin** user.
 - 1.1. Navigate to the OpenShift web console in **Firefox**.
 - <https://console-openshift-console.apps.ocp4.example.com>
 - 1.2. Click the Application Launcher and select **Observability** → **Logging** in the OpenShift web console.
- ▶ 2. Query the logs for messages from the `machine-api-operator` container. The `lab start` script restarted this container.
 - 2.1. Click **New**, and then ensure that the `infra` index pattern is selected.
 - 2.2. Type `kubernetes.container_name:"machine-api-operator"` in the query input field, and then click **Update**.
If no results are found, then click the time range selector and pick a larger range, such as `Last 1 hour`.
 - 2.3. In the **Available Fields** list, click **add** next to the **Message** field. The machine API operator logs typically output messages about syncing status.
 - 2.4. Locate the log messages from when the container started, such as `successfully acquired lease openshift-machine-api/machine-api-operator` and `Starting Machine API Operator`.
- ▶ 3. View the event log from the Event Router to see events related to the restarted pod.
 - 3.1. Click **New**, and then ensure that the `infra` index pattern is selected.

- 3.2. Type `kubernetes.event : *` in the query input field and click **Update**.
If no results are returned, then click the time range selector and pick a larger range, such as `Last 1 hour`.
 - 3.3. Click the arrow next to a log result to display the table of fields. Locate a log record where the `kubernetes.event.involvedObject.namespace` field is `openshift-machine-api`, and then click the **Filter for value** magnifying glass icon for that field.
 - 3.4. Click the **Toggle column in table** icon next to the **Message** field.
 - 3.5. Click the arrow next to the log time to collapse result listing.
 - 3.6. Locate the `Created container` and `Started container` log messages. If they are not visible in the listing, then select a larger time range.
- 4. In the `logging-query` project, locate server error logs marked with a 500 status code.
- 4.1. Click **New**, and then select the app index pattern.
 - 4.2. Click **Add a filter**. Create a filter in the `kubernetes.namespace_name` field. Select the `is` operator and a value of `logging-query`. Click **Save** to view the filtered results.
 - 4.3. In the Lucene query text field, type `message:500` to list all messages including the term "500".

Finish

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

```
[student@workstation ~]$ lab logging-query finish
```

Visualizing Cluster Logs with Kibana

Objectives

After completing this section, you should be able to visualize Kibana charts and understand their purpose.

Creating Visualizations

Graphical representations of cluster logs and running applications help administrators to effectively analyze any possible issues within the cluster.

You can create charts from the **Visualize** menu in Kibana to graphically represent any kind of information present in the logs.

There are different types of visualizations, including Area Charts, Data Tables, Heat Maps, Time Series, and Other.

Hover over a visualization type to view a brief description:

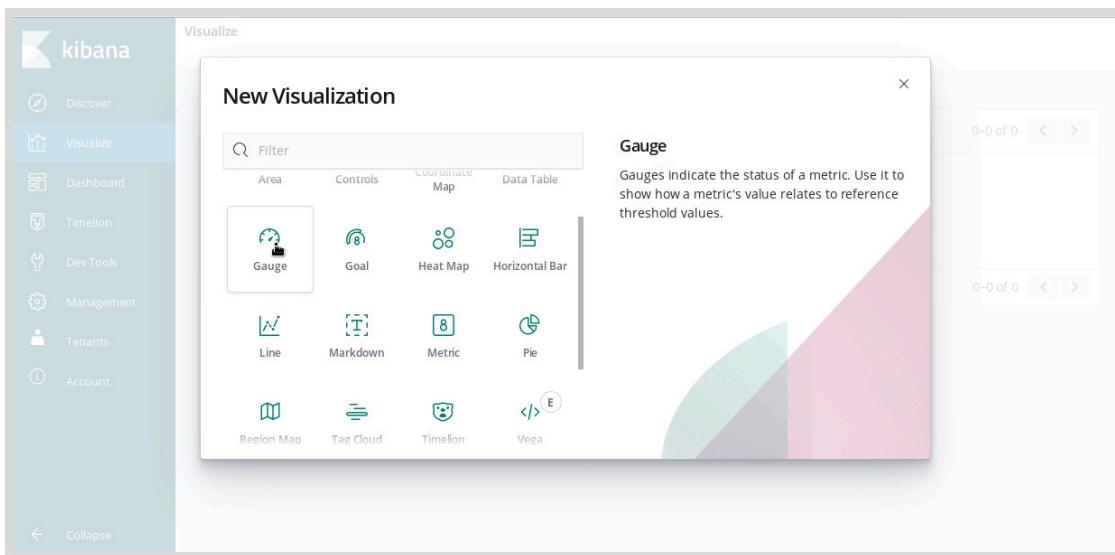


Figure 10.3: Visualization types and a brief description

For most visualizations, after selecting the visualization type you must provide the search source by selecting an index or a saved query.

Describing a Pie Chart

The following image represents a pie chart showing different types of HTTP response codes present in the logs of an application. On the left, you can create different **Filters** (HTTP response codes in this example). The chart on the right is a graphical representation of the **Filters** that you create on the left.

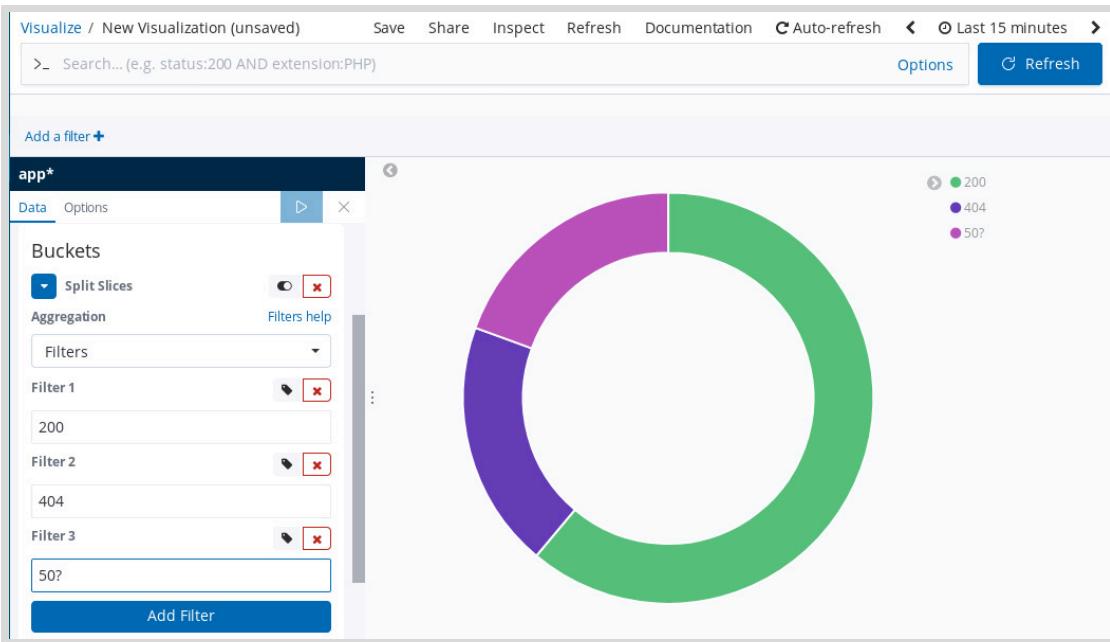


Figure 10.4: The Kibana Visualize pane displaying a pie chart

The previous image shows three **Filters**. Each one represents a slice of the pie chart:

- The **Filter** with value **200** is represented in **green**.
- The **Filter** with value **404** is represented in **purple**.
- The **Filter** with value **50?** is represented in **pink**.



Note

The value **50?** matches any type of 500 HTTP error code, such as **500** (Internal server error) or **503** (Bad Gateway).

Hover over the pie chart to see the number and percentage of logs from each slice, as shown in the following image:

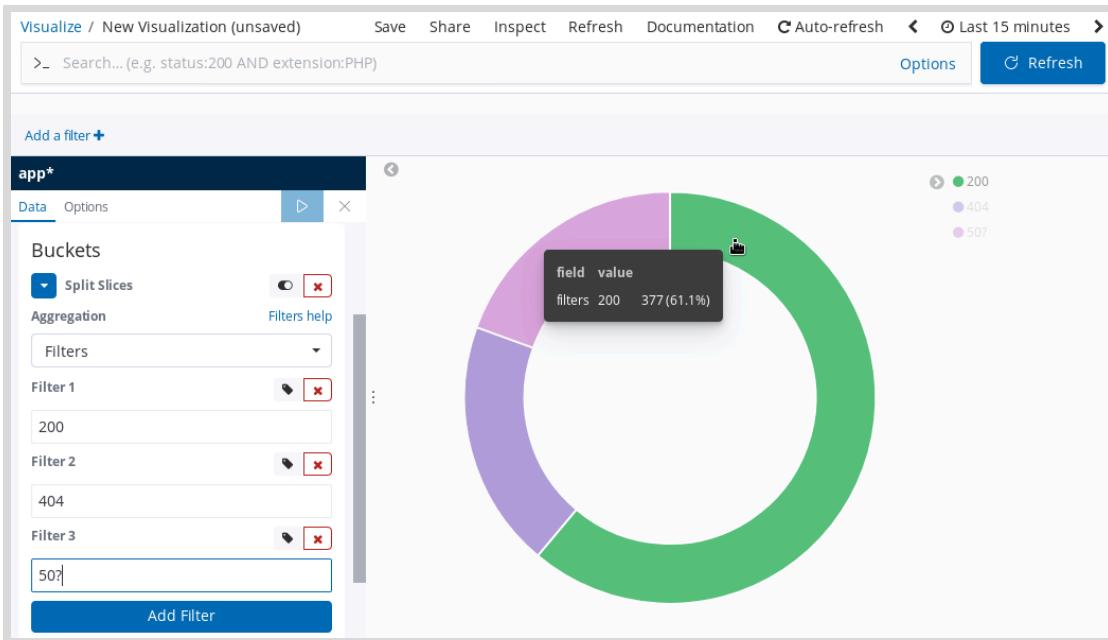


Figure 10.5: A pie chart showing information about the Slice

Select Save on the main menu, type a name in the Title field, and then click Confirm Save.

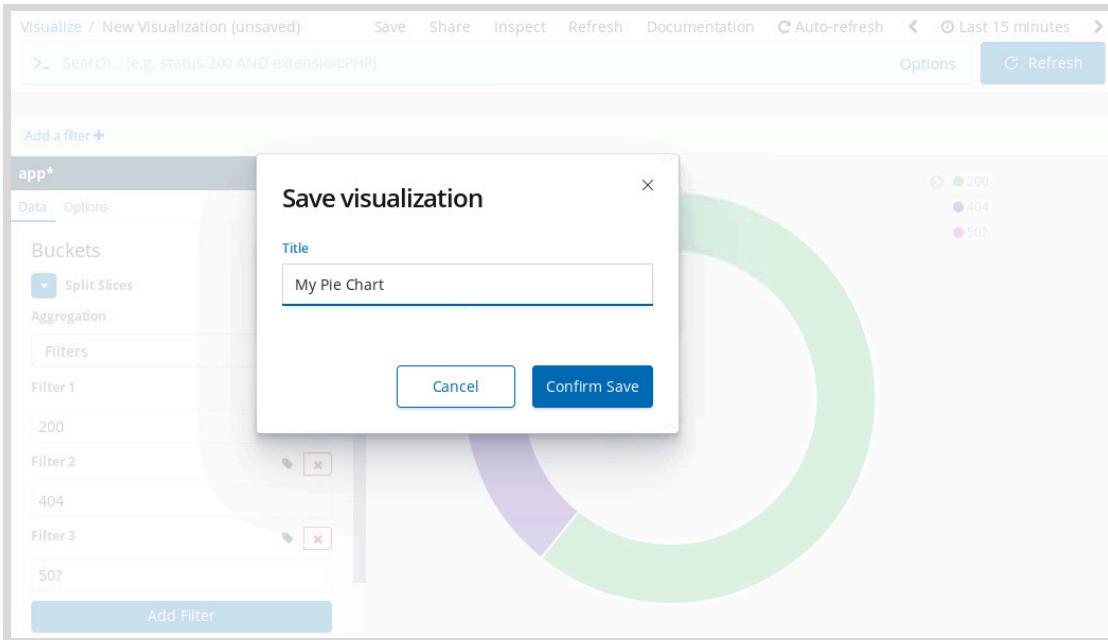


Figure 10.6: Save Visualization window

After saving, the chart is available from the Visualize option on the main menu:

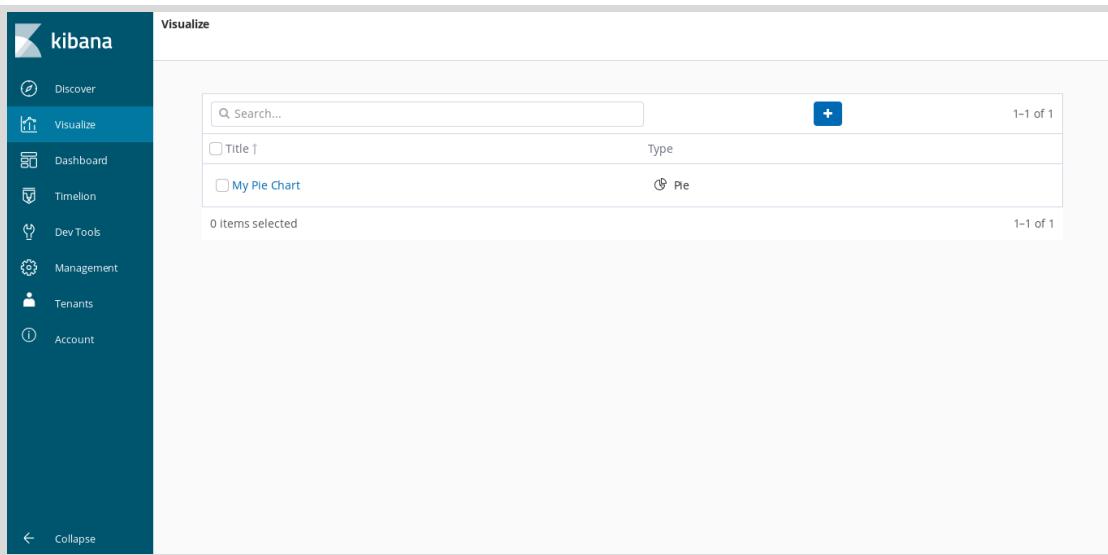


Figure 10.7: List of saved visualization charts

Visualizing Time Series with Timelion

Timelion uses time series data as an input to visualize data over time. Timelion provides functions to pull data from multiple data sources and to transform the data before representing it graphically. To access the Timelion documentation, click the [Jump to the function reference](#) link on the Timelion welcome page.

Describing a Sample Time Series



Important

Timelion queries should be typed on a single line. The queries are presented here as multiline statements for readability.

You can visualize a new Timelion graphical series from the **Timelion** menu in Kibana.

In the text box, you must indicate a valid **Timelion Expression**. Then, click **Play** to see the Time Series graph. The default expression is `.es(*)`, which uses the Elasticsearch function `(.es())` to pass the `*` Lucene query to Elasticsearch. The default query gathers all the data from the default index pattern. You can edit this expression to add a sample expression, such as:

```
.es('+' + kubernetes.namespace_name:logging-query + message:200'),
.es('+' + kubernetes.namespace_name:logging-query + message:404'),
.es('+' + kubernetes.namespace_name:logging-query + message:500')
```

This expression represents the count of logs from the `logging-query` namespace that have either 200, 404, or 500 in them. Click **Apply Changes** to display the time series graph:

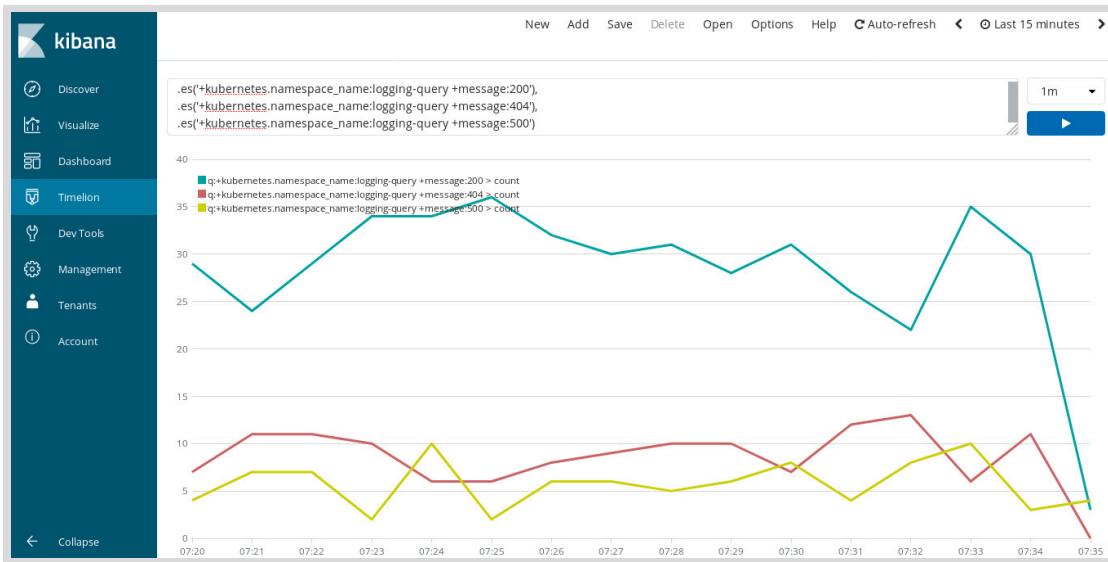


Figure 10.8: Timelion Time Series data chart

**Note**

For better visualization, click on the "Full screen" icon on the generated graph.

You can also choose the resolution of the graph using the list located above the **Play** button. Adjust the resolution to refine time periods where there have been issues with your applications.

You can also click and drag the mouse over the graph to zoom in and view fine-grained details to find the exact time when an issue occurred.

Also, you can change the display of the chart. The following expression shows the count of all logs as points, and the logs that contain 200 as bright red bars.

```
.es(*).points(), .es(200).bars().color(#f33)
```

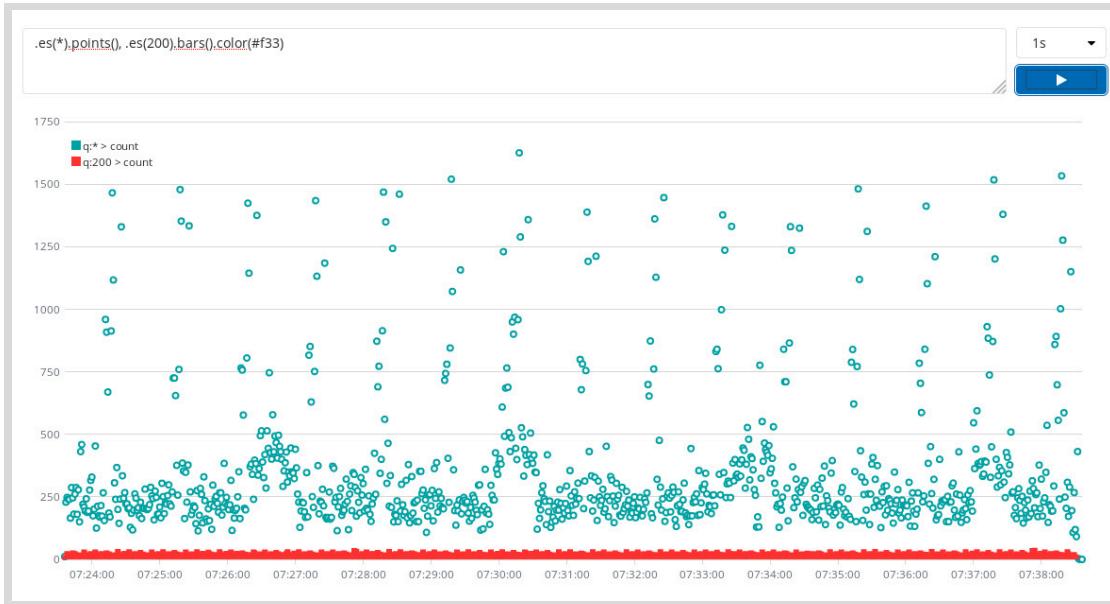


Figure 10.9: Timelion Time Series data chart displayed as dots

As another example, the following expression adds labels and offset data for 10 minutes. The expression also labels the query for `admin` as `current` and the query for `cart` as `previous`.

```
.es(index=project.* , q='admin').label(current),
.es(index=project.* , q=cart, offset=-10m).label('previous')
```

Finally, you can save the charts as either a Timelion sheet or as a Kibana dashboard panel. Click **Save** on the Kibana toolbar, choose a save option, provide a name, and then click **Save**.



References

Timelion Tutorial - From Zero to Hero

<https://www.elastic.co/blog/timelion-tutorial-from-zero-to-hero>

Kibana Guide - Visualizing your data

<https://www.elastic.co/guide/en/kibana/6.8/tutorial-visualizing.html>

► Guided Exercise

Visualizing Cluster Logs with Kibana

In this exercise you will create visualizations in Kibana.

Outcomes

You should be able to:

- Create a simple Kibana chart.
- Create a time series chart using Timelion.

Before You Begin

Before starting this exercise, you must set up cluster logging by completing the *Guided Exercise: Deploying Cluster Logging*.

As the **student** user on the **workstation** machine, use the `lab` command to prepare your system for this exercise. The command ensures that an application that creates access logs containing HTTP status codes is deployed.

```
[student@workstation ~]$ lab logging-visualize start
```

Instructions

- ▶ 1. Navigate to the Kibana UI and log in as the **admin** user.
 - 1.1. Navigate to the OpenShift web console in **Firefox**.
 - <https://console-openshift-console.apps.ocp4.example.com>
 - 1.2. Click the Application Launcher and select **Observability** → **Logging** in the OpenShift web console.
- ▶ 2. Create a bar chart displaying buckets of the status codes.
 - 2.1. Select the **Visualize** option, and then click **Create a visualization**.
 - 2.2. Click the **Horizontal Bar** basic chart.
 - 2.3. Select the **app index** pattern.
 - 2.4. Click **Add a filter**. Create a filter on the `kubernetes.container_name` field. Select the `is` operator and a value of `logger`. Click **Save**.
 - 2.5. Click the **Split Series** bucket type, and then select **Filters** as the **Aggregation**.
 - 2.6. Click **Add Filter** twice. Enter the following values in the **Filter 1**, **Filter 2**, and **Filter 3** inputs respectively: `message:200`, `message:40?`, `message:50?`. The `?` question mark symbol matches any character in that position. For example, `message:40?` will match `400` and `403`.

- 2.7. Click the **Apply changes** arrow icon to view the chart.
- 3. Create a Timelion chart that displays the count of 500 Internal Server Error status codes per minute.
- 3.1. Click **Timelion** to navigate to the timeline chart tool.
 - 3.2. Enter the following Timelion expression to create a graph of logs with a 500 status code compared with the total count of logs.

```
.es('+kubernetes.container_name:logger +message:500'),  
.es('+kubernetes.container_name:logger +message:*)')
```



Important

Timelion queries must be typed in a single line. The queries are presented here as multiline statements for readability.

An error stating "Timelion: Error: in cell #1: input must be a seriesList" displays if there are spaces between function calls.

Press **Enter**, or click the arrow button to render the graph.

- 3.3. Select an interval of **1m** from the interval selection list to the right of the query input field.
 - 3.4. Click the arrow button to rerender the graph with the updated time interval.
- 4. Update the Timelion chart to display the percent of 500 status code messages using the **divide()** and **multiply()** functions.

- 4.1. Modify the Timelion expression to match the following:

```
.es('+kubernetes.container_name:logger +message:500')  
.divide(.es('+kubernetes.container_name:logger +message:*))  
.multiply(100)
```

4.2. Press **Enter** or click the arrow button to render the updated graph.

- 5. Create a Timelion chart that compares now to five minutes ago.

- 5.1. Modify the Timelion expression to match the following:

```
.es('+kubernetes.container_name:logger +message:500').label(current)
```

5.2. Click the time range selector, click **Relative**, update the first set of text fields to read **5 Minutes ago**, and then click **Go**.

5.3. Add a second Elasticsearch es query with a **-5m** offset labeled **previous**.

```
.es('+kubernetes.container_name:logger +message:500').label(current),  
.es(q='+kubernetes.container_name:logger +message:500',  
offset=-5m).label(previous)
```

- 5.4. Press **Enter** or click the arrow button to render the updated graph.

Finish

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

```
[student@workstation ~]$ lab logging-visualize finish
```

Diagnosing Cluster Logging Problems

Objectives

After completing this section, you should be able to diagnose cluster logging problems with debugging and monitoring tools.

Discussing Cluster Logging Alerts

The OpenShift cluster logging infrastructure provides cluster users centralized access that enables application log analysis. If a component of the cluster logging infrastructure fails, then cluster users lose the capability to diagnosis application issues from logs. As a cluster administrator, you must use the monitoring and alerting features of OpenShift to quickly find and fix cluster logging issues.

Fluentd is the component collecting the logs. Prometheus scrapes metrics from Fluentd, such as availability, queue length, and errors. After collecting the logs, Fluentd forwards them to Elasticsearch for storage. Prometheus also scrapes metrics from Elasticsearch, such as health and resource utilization. The cluster logging uses alerting rules on Fluentd and Elasticsearch metrics to notify cluster administrators of any problems.

Common Cluster Logging Alerts

Alert	Description
FluentdErrorsHigh	Fluentd is reporting more than one error per minute.
FluentdQueueLengthIncreasing	The log buffer is growing faster than Fluentd can process.
ElasticsearchNodeDiskWatermarkReached	Available disk space is low.
ElasticsearchProcessCPUHigh	The CPU usage is above 90%.

To display the list of alerting rules, navigate to **Monitoring → Alerting**, and then click **Alerting Rules**. To filter the list of alerting rules, enter a term in the **Filter Alerting Rules by Name** field. For example, enter `fluentd` to find alerting rules that relate to Fluentd. Click an alert, such as `FluentdQueueLengthIncreasing`, to view the alert rule definition.

Each cluster logging alert rule is defined in a `PrometheusRule` resource in the `openshift-logging` namespace. You can use the `oc get prometheusrules` command to list and inspect Prometheus rule objects:

```
[user@host ~]$ oc get prometheusrules -n openshift-logging
NAME          AGE
elasticsearch-prometheus-rules  2d15h
fluentd        2d15h

[user@host ~]$ oc get prometheusrules -n openshift-logging fluentd -o yaml
```

```

kind: PrometheusRule
...output omitted...
spec:
  groups:
    - name: logging_fluentd.alerts
      rules:
        - alert: FluentdNodeDown
  ...output omitted...
        - alert: FluentdQueueLengthIncreasing
          annotations:
            message: In the last 12h, fluentd {{ $labels.instance }} buffer queue
length
              constantly increased more than 1. Current value is {{ $value }}.
            summary: Fluentd file buffer usage issue
          expr: |
            delta(fluentd_output_status_buffer_queue_length[1m]) > 1
          for: 12h
          labels:
            service: fluentd
            severity: critical
  ...output omitted...

```

Observing Cluster Logging Metrics

Cluster logging is a standard Kubernetes workload made of deployments, replica sets, daemon sets, and cron jobs in the `openshift-logging` namespace. The cluster monitoring stack collects metrics on Kubernetes workloads. Use cluster monitoring metrics to observe cluster logging resource utilization.

View key health metrics on the **Monitoring → Dashboards** section in the web console.

To view CPU usage by pod:

1. Select the **Kubernetes > Compute Resources > Namespace (Pods)** dashboard from the **Dashboard** list.
2. Select the `openshift-logging` namespace from the **Namespace** list.
3. Observe the following:
 - The **CPU Usage** time series graph displays CPU usage by pod.
 - The **Memory Usage** time series graph displays memory usage by pod.
 - Fluentd CPU usage rises with increased log frequency, size, and complexity.

To view CPU, memory, network, and disk utilization and saturation time series graphs for a specific host:

1. Select the **USE Method / Node** dashboard from the **Dashboard** list.
2. Select a pod running the cluster monitoring stack from the **Instance** list.

Query the available disk space from the `es_fs_path_available_bytes` Prometheus metric.

If Elasticsearch consumes too much disk space, consider:

- Modifying log retention.

- Extending Elasticsearch storage.

Querying Kibana for Cluster Logging Health

Use the Kibana UI to view cluster logging component logs.

You must use the `infra` index pattern to retrieve logs for cluster logging. Add a `kubernetes.namespace_name:openshift-logging` filter to display logs for the `openshift-logging` project only. Use the `kubernetes.container_name` field in a filter to view logs for a specific component only.

For instance, you can use the Kibana UI to find workloads generating more logs and locate sources of cluster logging performance issues.

Discussing Elasticsearch Troubleshooting Tools

In addition to the standard Elasticsearch commands, the Elasticsearch image provided by Red Hat contains extra tools that are useful for debugging purposes, such as:

`es_util`

Queries the Elasticsearch REST endpoint.

`es_cluster_health`

Returns a JSON summary of the Elasticsearch cluster health.

To use either of these tools, first locate an Elasticsearch pod in the `openshift-logging` namespace.

```
[user@host ~]$ oc get pods -n openshift-logging
NAME
...
...output omitted...
elasticsearch-cdm-giqewkd9-1-5f8b46cdbb-qrqf7 ...
elasticsearch-cdm-giqewkd9-2-56cb8f56bf-pb9kv ...
...output omitted...
```

Next, use the `oc exec` command to execute either tool in the `elasticsearch` container of the pod that you identified. The following example executes the `es_cluster_health` tool in the `elasticsearch` container of an Elasticsearch pod to display the Elasticsearch cluster health metrics:

```
[user@host ~]$ $ oc exec -n openshift-logging -c elasticsearch \
  elasticsearch-cdm-giqewkd9-1-5f8b46cdbb-qrqf7 -- es_cluster_health
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 31,
  "active_shards" : 36,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
```

```
"number_of_in_flight_fetch" : 0,  
"task_max_waiting_in_queue_millis" : 0,  
"active_shards_percent_as_number" : 100.0  
}
```

To display information about Elasticsearch indexes, use the `oc exec` command to execute the `es_util` tool in the `elasticsearch` container of any Elasticsearch pod:

```
[user@host ~]$ oc exec -n openshift-logging -c elasticsearch \  
elasticsearch-cdm-giqewkd9-1-5f8b46cdbb-qrqf7 -- es_util \  
--query=_cat/indices?pretty  
green open app-000001 nk3TFaibRVqkQfL-eiGVDQ 2 0 251 0 407.9kb 407.9kb  
green open audit-000001 W_bkITL8QpW1PAdvS7H2_w 2 0 0 0 522b 522b  
green open .security ygEGPAvQT3q7z7Itc903Yg 1 1 5 0 59.9kb 29.9kb  
green open .kibana_1 i2jbvo9-QveVwXhXr2deyA 1 1 0 0 523b 261b  
green open infra-000001 vC0jma4EQNS4UX2VsI5vrQ 2 0 1064849 0 606.5mb 606.5mb
```



References

For more information about troubleshooting cluster logging, refer to the *Troubleshooting cluster logging* chapter in the Red Hat OpenShift Container Platform 4.6 *Logging* documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/logging/index#troubleshooting-cluster-logging

► Guided Exercise

Diagnosing Cluster Logging Problems

In this exercise you will monitor the health of the cluster logging system and identify a problematic application.

Outcomes

You should be able to:

- Verify the health of the cluster logging system using Prometheus.
- Identify and stop an application that is overwhelming the log collector.

Before You Begin

Before starting this exercise, you must set up cluster logging by completing the *Guided Exercise: Deploying Cluster Logging*.

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. The command creates a `logging-diagnose` project and a `noisy` deployment that outputs multiple log messages per second.

```
[student@workstation ~]$ lab logging-diagnose start
```

Instructions

- ▶ 1. First, observe the steady state logging metrics collected by Prometheus.
 - 1.1. Navigate to the OpenShift web console in Firefox.
 - <https://console-openshift-console.apps.ocp4.example.com>
 - 1.2. Click **Monitoring** → **Dashboards**, and then select the **Kubernetes / Compute Resources / Namespace (Pods)** dashboard and the `openshift-logging` namespace. Review the data, paying particular attention to the **CPU Usage** and **Memory Usage** charts.
 - 1.3. Click **Monitoring** → **Metrics**, and then enter the expression `max_over_time(fluentd_output_status_buffer_queue_length[1m])` and click **Run Queries**. Review the chart that displays the maximum number of logs per minute in the collector buffer.
- ▶ 2. The current deployment has only one replica and generates logs at a reasonable rate (every 100ms). Increase the number of replicas to add load.
 - 2.1. From the `workstation` machine, log in to OpenShift as the `admin` user.

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

You have access to 58 projects, the list has been suppressed. You can list all
projects with 'oc projects'

Using project "default".
```

- 2.2. Edit the deployment to specify ten replicas and a container --delay argument of 1ms.

```
[student@workstation ~]$ oc project logging-diagnose
Now using project "logging-diagnose" on server "https://
api.ocp4.example.com:6443".
```

```
[student@workstation ~]$ oc edit deployment/noisy
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
...output omitted...
  spec:
    containers:
      - args:
          - --delay
          - 1ms
        image: quay.io/redhattraining/logger:v0.5
...output omitted...
```

- 3. Review logging metrics in the monitoring dashboard for changes. The log load is still small and might not be immediately noticeable.
- 3.1. Click **Monitoring** → **Dashboards**, and then review the **Kubernetes / Compute Resources / Namespace (Pods)** dashboard for the **openshift-logging** namespace.
 - 3.2. Click **Monitoring** → **Metrics**, enter the expression **max_over_time(fluentd_output_status_buffer_queue_length[1m])**, and then click **Run Queries**. Review the chart that displays the maximum number of logs per minute in the collector's buffer.
- 4. Scale the noisy deployment to 50 replicas.

```
[student@workstation ~]$ oc scale deployment/noisy --replicas 50
deployment.apps/noisy scaled
```

- ▶ 5. Review the logging metrics in the monitoring dashboard. The monitoring stack takes time to reflect updates in metrics.
- 5.1. Click **Monitoring → Dashboards**, and then review the **Kubernetes / Compute Resources / Namespace (Pods)** dashboard for the **openshift-logging** namespace. Notice the increase in CPU usage.
 - 5.2. Click **Monitoring → Metrics**, enter the expression `max_over_time(fluentd_output_status_buffer_queue_length[1m])`, and then click **Run Queries**. Review the chart that displays the maximum number of logs per minute in the collector buffer. Notice the increasing queue length.
- ▶ 6. Review the log collector alerts. The monitoring stack takes time to reflect updates in alerts.
- 6.1. Click **Monitoring → Alerting**, and then review the alerts in **Pending** and **Firing** states.
 - 6.2. Click **Clear all filters**, select **Name** from the list, and then type **fluentd** in the search text field.
Review the list of preconfigured alerts related to Fluentd.
 - 6.3. Type **elastic** in the search text field, select **Name** from the list, and then review the list of alerts related to Elasticsearch.
- ▶ 7. Query Elasticsearch to discover the namespace containing the application responsible for the problem.
- 7.1. Click the Application Launcher, and then select **Observability → Logging** in the OpenShift web console. If prompted to log in, click **Log in with OpenShift**.
 - 7.2. Select the app index pattern. Click the arrow preceding a log result to display the table of fields and values, and then click the **Toggle column in table** icon for the **kubernetes.namespace_name** field.
 - 7.3. Click the **kubernetes.namespace_name** in the **Selected Fields** list, and then click **Visualize**. Review the chart that displays the number of logs per namespace generated in the last 15 minutes.
 - 7.4. Click **Discover** to return to the log query screen, and then click **New**. Select the app index pattern.
 - 7.5. Create a filter on the **kubernetes.namespace_name** field. Select the **is** operator and a value of **logging-diagnose**. Click **Save** to view the filtered results.
 - 7.6. Click **add** next to the **message** and **kubernetes.container_name** fields to list the log messages and container name in the table.
- ▶ 8. Scale the noisy deployment to one replica.

```
[student@workstation ~]$ oc scale deployment/noisy --replicas 1  
deployment.apps/noisy scaled
```

The log collector will continue to process the log buffer. After a few minutes, the CPU usage will return to baseline levels and logs will be fully available in the Kibana UI.

Finish

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

```
[student@workstation ~]$ lab logging-diagnose finish
```

▶ Lab

Provisioning and Inspecting Cluster Logging

In this lab, you will use the cluster logging infrastructure to analyze the log activity of four applications.

Outcomes

You should be able to:

- Specify Lucene queries and filters in the Kibana UI to find logs.
- Create charts to summarize application log entries.

Before You Begin

Before starting this exercise, you must:

- Add extra worker nodes by completing the *Guided Exercise: Adding Compute Nodes*.
- Set up cluster logging by completing the *Guided Exercise: Deploying Cluster Logging*.

As the **student** user on the **workstation** machine, use the `lab` command to prepare your system for this exercise. The command creates four deployments (`app1`, `app2`, `app3`, and `app4`) in the `logging-review` project.

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

Each deployment generates Apache log entries and defines an `app` label that matches the deployment name.

Use the cluster logging capabilities to answer three questions regarding the characteristics of deployments:

Each application is configured to emit log records for twenty minutes after it starts. Ensure that your analysis of the time range includes activity from the time the application started. Further ensure that the time range includes at least ten minutes of activity.

Record your answers to these questions in the `~/D0380/labs/logging-review/answers.yml` file.

Instructions

1. Navigate to the Kibana UI.
2. Each of the four deployments defines an `app` label that matches the deployment name. Ensure that Kibana indexes the field that you need to query for deployments with an `app` label.
3. Use Lucene queries to answer the first question:
 - Which deployment has the most log records? Which deployment has the fewest?

Update the `question1` section of the `~/D0380/labs/logging-review/answers.yml` answer file.

```
...output omitted...
question1:
  most_total_entries: app1|app2|app3|app4
  least_total_entries: app1|app2|app3|app4
...output omitted...
```

For the `highest_total_entries` key, keep only the deployment name with the most log records. For the `least_total_entries` key, keep the deployment name with fewest log records.

4. Create a pie chart for each deployment. Ensure that one component of each pie chart corresponds to log messages that contain an HTTP server error code. Further ensure that the entire pie chart corresponds to all log messages for the deployment.

Use the pie charts to answer the second question:

- Which deployment has the largest percentage of server errors? Which deployment has the smallest percentage?

Update the `question2` section of the `~/D0380/labs/logging-review/answers.yml` answer file.

```
...output omitted...
question2:
  largest_percentage_of_errors: app1|app2|app3|app4
  smallest_percentage_of_errors: app1|app2|app3|app4
...output omitted...
```

For the `largest_percentage_of_errors` key, keep only the deployment name with the largest percentage of server error log records. For the `smallest_percentage_of_errors` key, keep only the deployment name with the smallest percentage of server error log records.

5. Use Kibana to analyze the log record trends for each deployment. Determine if each deployment has an increasing, decreasing, or steady number of log records per minute.

Modify the `question3` section of the `~/D0380/labs/logging-review/answers.yml` answer file.

```
...output omitted...
question3:
  app1: increasing|decreasing|steady
  app2: increasing|decreasing|steady
  app3: increasing|decreasing|steady
  app4: increasing|decreasing|steady
...output omitted...
```

For each deployment, keep only the value that describes the log record trends for the application:

- **increasing** - The number of log records each minute is increasing.
- **decreasing** - The number of log records each minute is decreasing.
- **steady** - The number of log records each minute is neither increasing, nor decreasing.

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 logging-review grade
```

Finish

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

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

► Solution

Provisioning and Inspecting Cluster Logging

In this lab, you will use the cluster logging infrastructure to analyze the log activity of four applications.

Outcomes

You should be able to:

- Specify Lucene queries and filters in the Kibana UI to find logs.
- Create charts to summarize application log entries.

Before You Begin

Before starting this exercise, you must:

- Add extra worker nodes by completing the *Guided Exercise: Adding Compute Nodes*.
- Set up cluster logging by completing the *Guided Exercise: Deploying Cluster Logging*.

As the **student** user on the **workstation** machine, use the `lab` command to prepare your system for this exercise. The command creates four deployments (`app1`, `app2`, `app3`, and `app4`) in the `logging-review` project.

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

Each deployment generates Apache log entries and defines an app label that matches the deployment name.

Use the cluster logging capabilities to answer three questions regarding the characteristics of deployments:

Each application is configured to emit log records for twenty minutes after it starts. Ensure that your analysis of the time range includes activity from the time the application started. Further ensure that the time range includes at least ten minutes of activity.

Record your answers to these questions in the `~/D0380/labs/logging-review/answers.yml` file.

Instructions

1. Navigate to the Kibana UI.
 - 1.1. Navigate to the OpenShift web console in Firefox.
 - <https://console-openshift-console.apps.ocp4.example.com>
 - 1.2. From the OpenShift web console, click the Application Launcher, and then select **Observability** → **Logging**.



2. Each of the four deployments defines an app label that matches the deployment name. Ensure that Kibana indexes the field that you need to query for deployments with an app label.
 - 2.1. Navigate to **Management** in the Kibana UI, and then click **Index Patterns**.
 - 2.2. Select the app index pattern to list all fields in the app index.
 - 2.3. Enter `kubernetes.flat_labels` in the text field. If a `kubernetes.flat_labels` entry displays in the list, you can use the `kubernetes.flat_labels` field in Lucene queries.

If the filtered list is empty, then click **Refresh field list**.

Name	Type	Format	Search...	Aggrega...	Excluded
No items found					

In the confirmation window that displays, click **Refresh**.

3. Use Lucene queries to answer the first question:
 - Which deployment has the most log records? Which deployment has the fewest?

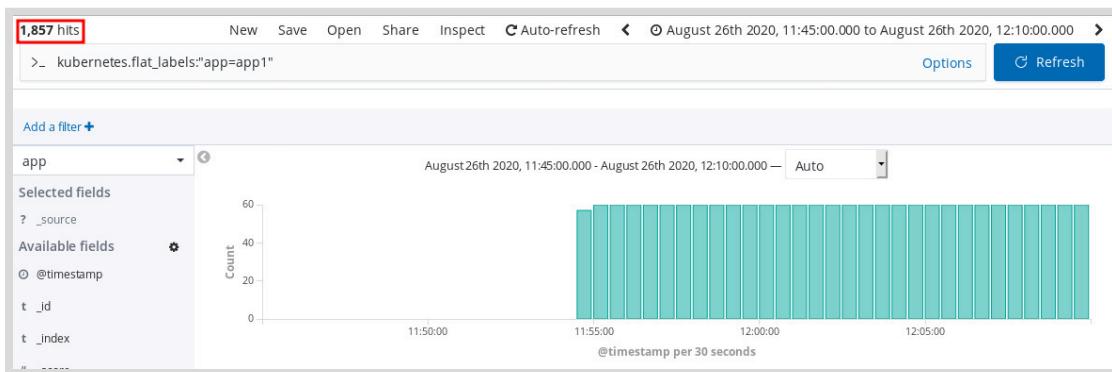
Update the `question1` section of the `~/DO380/labs/logging-review/answers.yml` answer file.

```
...output omitted...
question1:
  most_total_entries: app1|app2|app3|app4
  least_total_entries: app1|app2|app3|app4
...output omitted...
```

For the `highest_total_entries` key, keep only the deployment name with the most log records. For the `least_total_entries` key, keep the deployment name with fewest log records.

- 3.1. Navigate to the **Discover** view, and then click **New**. Select the app index pattern, and then enter `kubernetes.flat_labels:"app=app1"` in the text field.

The results display the total number of log records for the `app1` deployment. In the following example, the query returns 1,857 records (1,857 hits).



Note

Your results will differ from the preceding example, due to several factors including:

- The elapsed time from the start of the lab.
- The configured bucket size for time stamp aggregation (Auto vs. Minute)

If your results include at least ten minutes of application activity, then you can use the results to answer the questions.

- 3.2. Note the number of log records for the `app1` deployment. Repeat the previous step for the `app2`, `app3`, and `app4` deployments and note the total number of log records.

An example results summary follows:

Example Total Records for Each Deployment

Deployment	Total Records	Description
app1	1857	most
app2	1264	
app3	1490	
app4	894	least

- 3.3. Modify the `question1` section of the `~/DO380/labs/logging-review/answers.yml` file to match the following:

```
...output omitted...
question1:
  most_total_entries: app1
  least_total_entries: app4
...output omitted...
```

4. Create a pie chart for each deployment. Ensure that one component of each pie chart corresponds to log messages that contain an HTTP server error code. Further ensure that the entire pie chart corresponds to all log messages for the deployment.

Use the pie charts to answer the second question:

- Which deployment has the largest percentage of server errors? Which deployment has the smallest percentage?

Update the `question2` section of the `~/D0380/labs/logging-review/answers.yml` answer file.

```
...output omitted...
question2:
  largest_percentage_of_errors: app1|app2|app3|app4
  smallest_percentage_of_errors: app1|app2|app3|app4
...output omitted...
```

For the `largest_percentage_of_errors` key, keep only the deployment name with the largest percentage of server error log records. For the `smallest_percentage_of_errors` key, keep only the deployment name with the smallest percentage of server error log records.

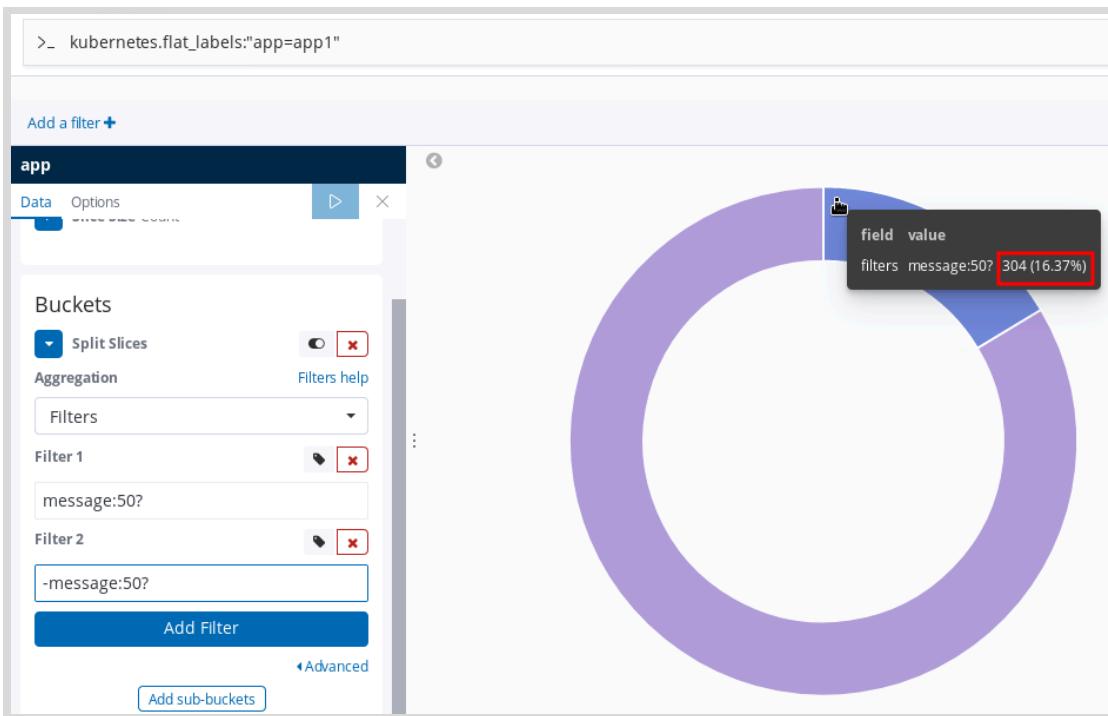
- 4.1. Navigate to **Visualize**, and then click **+** to create a new visualization.

Click **Pie**. On the page that displays, click **app**.

- 4.2. Enter `kubernetes.flat_labels:"app=app1"` in the text field. Click **Split Slices**, and then select **Filters** for the aggregation type. Add two filters:

- `message:50?`
- `-message:50?`

Click the **Apply changes** arrow icon to view the chart.



- 4.3. Use the pie chart to note the percentage of error messages for the app1 deployment. Update the Lucene query text field for each of the remaining deployments, and note the percentage of error messages.

Example results follow:

Example Total Records for Each Deployment

Deployment	Percent Error Messages	Description
app1	16.8%	
app2	10.7%	
app3	25.3%	largest
app4	5.8%	smallest

- 4.4. Modify the question2 section of the `~/D0380/labs/logging-review/answers.yml` file to match the following:

```
...output omitted...
question2:
  largest_percentage_of_errors: app3
  smallest_percentage_of_errors: app4
...output omitted...
```

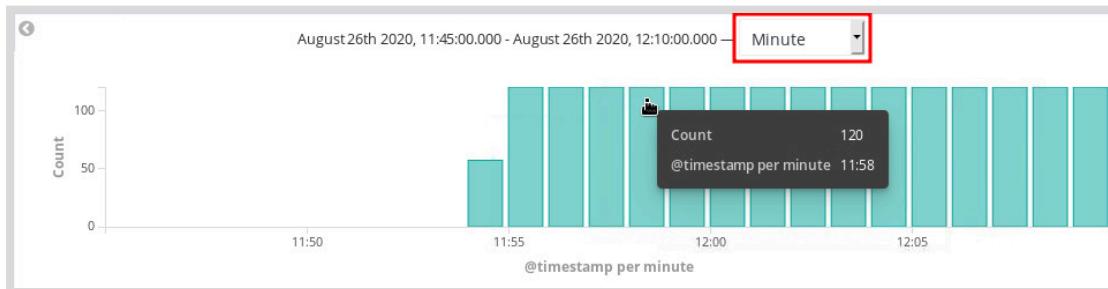
5. Use Kibana to analyze the log record trends for each deployment. Determine if each deployment has an increasing, decreasing, or steady number of log records per minute. Modify the question3 section of the `~/D0380/labs/logging-review/answers.yml` answer file.

```
...output omitted...
question3:
app1: increasing|decreasing|steady
app2: increasing|decreasing|steady
app3: increasing|decreasing|steady
app4: increasing|decreasing|steady
...output omitted...
```

For each deployment, keep only the value that describes the log record trends for the application:

- **increasing** - The number of log records each minute is increasing.
- **decreasing** - The number of log records each minute is decreasing.
- **steady** - The number of log records each minute is neither increasing, nor decreasing.

- 5.1. Navigate to the **Discover** view. Reuse the Lucene query for the app1 deployment. Ensure that records aggregate each minute.



The app1 deployment has approximately 120 log records per minute, every minute. The log record rate does not increase, nor decrease.

- 5.2. Repeat the previous step for each of the remaining deployments. Summarize the log records per minute trend as increasing, decreasing, or steady.

Records per Minute Trend for Deployment

Deployment	Records Per Minute Trend
app1	steady
app2	decreasing
app3	increasing
app4	decreasing

- 5.3. Modify the question3 section of the ~/D0380/labs/logging-review/answers.yml file to match the following:

```
...output omitted...
question3:
app1: steady
app2: decreasing
app3: increasing
app4: decreasing
...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 logging-review grade
```

Finish

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

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

Summary

In this chapter, you learned:

- About the cluster logging components for OpenShift and how they interact with each other.
- How to install and configure the cluster logging components.
- How to use the Kibana UI to query Elasticsearch for logs.
- How to use the Kibana UI to create charts and visualizations of the data that resides in Elasticsearch.
- Diagnostic steps to find problematic deployments using the tools provided by cluster logging.

Chapter 11

Recovering Failed Worker Nodes

Goal

Inspect, troubleshoot, and remediate worker nodes in a variety of failure scenarios.

Objectives

- Profile worker nodes to examine healthy nodes, ensure the node services function properly, and identify adverse performance or degradation.
- Diagnose and troubleshoot capacity issues that occur on worker nodes.

Sections

- Profiling Degraded Worker Nodes (and Guided Exercise)
- Troubleshooting Worker Node Capacity Issues (and Guided Exercise)

Lab

Recovering Failed Worker Nodes

Profiling Degraded Worker Nodes

Objectives

After completing this section, you should be able to profile worker nodes to examine healthy nodes, ensure the node services function correctly, and identify adverse performance or degradation.

Examining Healthy Worker Node Statuses

To properly troubleshoot issues with a worker node, you must understand the typical status and behaviors of healthy worker nodes. Familiarity with the output of commands that report worker node status, running processes, and other critical information about the cluster and worker node health are essential to identifying a degraded environment. This knowledge allows you to identify and troubleshoot issues using valuable details that aid mitigation, such as the taints OpenShift applies to degraded worker nodes that describe the symptom.

Retrieve node status:

```
[user@host ~]$ oc get nodes <NODE>
```

Get information for a node through top output for currently running processes:

```
[user@host ~]$ oc adm top node <NODE>
```

Check for any applied taints for a node:

```
[user@host ~]$ oc describe node <NODE> | grep -i taint
```

Additionally, inspect the worker node through the web console and CLI commands to familiarize yourself with healthy operational statuses and where to find valuable information, such as on the Monitoring page. You can find a link to the OpenShift web console for your cluster using the command:

```
[user@host ~]$ oc whoami --show-console
```

Ensuring Worker Node Service Functionality

Each worker node has two main services, `kubelet` and `cri-o`, that allow for cluster operations. Various probes are configured to monitor worker node performance and provide necessary details during service issues on the Monitoring page. More severe service issues might require direct access using CLI commands on the worker node, such as `systemctl`, to troubleshoot further and restore node health.

- Determine the appropriate method to manage the services on worker nodes:
 - The web console monitoring provides insight into unhealthy worker nodes, but might not give specific details for every scenario.

- Use the `oc` client commands to gather as much information as available for the given symptoms.
- If you cannot correct the issue using other means, then connect directly to the worker node using SSH to mitigate the issue.
- Navigate the web console information alerting administrators to node issues.

Identifying Adverse Worker Node Performance

When OpenShift detects an adverse node condition, OpenShift applies a corresponding taint to the node.

A taint that represents an adverse node condition consists of a key value pair directing cluster interactions with the worker node.

The taint key helps administrators identify the cause of the node degradation. For example, the `node.kubernetes.io/disk-pressure` taint key indicates that a node is low on available disk space.

OpenShift uses the taint effect value to direct pod scheduling preference on the tainted node. The table that follows describes the three taint effects and the corresponding impact on pod scheduling.

OpenShift Taint Effects

Effect value	Description
NoSchedule	<ul style="list-style-type: none">• New pods that do not match the taint are not scheduled on the node.• Pods that are running on the node continue running.
PreferNoSchedule	<ul style="list-style-type: none">• New pods that do not match the taint are only scheduled on the node when no other suitable node can be located.• Pods that are running on the node continue running.
NoExecute	<ul style="list-style-type: none">• New pods that do not match the taint are not scheduled on the node.• Pods that are running on the node are removed and deployed on another suitable node.



References

For more information about using taints in scheduling, refer to the *Controlling pod placement using node taints* section in the *Controlling pod placement onto nodes (scheduling)* chapter in the Red Hat OpenShift Container Platform 4.6 Nodes documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#nodes-scheduler-taints-tolerations

Taints: Expectations vs. Reality

<https://www.openshift.com/blog/taints-expectations-vs.-reality>

► Guided Exercise

Profiling Degraded Worker Nodes

In this exercise, you will profile a degraded worker node and utilize proper corrective measures to restore node functionality.

Outcomes

You should be able to:

- Identify cluster issues resulting from an unhealthy worker node.
- Inspect and isolate the specific cause of a worker node degradation.
- Repair the issue and restore proper worker node functionality.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- Access to the workstation virtual machine and cluster administrator credentials.

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

This command ensures that the cluster API is reachable.

```
[student@workstation ~]$ lab workers-degrade start
```

Instructions

- 1. Log in to your OpenShift cluster as the **admin** user.

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

- 2. From the workstation machine, inspect the OpenShift web console Monitoring page for alerts that provide information regarding cluster performance degradation.

- 2.1. Display the address for the web console by running the command:

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

- 2.2. Open the link provided in the command output in a browser, and then log in as the **admin** user with the **redhat** password.

- ▶ 3. Navigate to the Monitoring section of the web console and inspect the Alerts tab found on the Alerting page. After a few minutes, several alerts with a severity of Warning and a state of Firing are displayed with additional AlertManager notifications. The alerts provide details about the cluster degradation, cluster member absence, and make explicit mention of **worker01** being unready.
- ▶ 4. Utilize CLI commands to profile the status of the worker nodes to determine the appropriate remediation.
 - 4.1. Check the status of all nodes. Notice that one worker node reports a **NotReady** status.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES     AGE      VERSION
master01   Ready     master    ...      ...
master02   Ready     master    ...      ...
master03   Ready     master    ...      ...
worker01   NotReady  worker    ...      ...
...output omitted...
```

- 4.2. Inspect the **worker01** node showing a **Not Ready** status using the **oc describe** command.

```
[student@workstation ~]$ oc describe node/worker01
...output omitted...
Taints:           node.kubernetes.io/not-ready:NoExecute
                  node.kubernetes.io/not-ready:NoSchedule
...output omitted...
Conditions:
Type      Status  ...      Reason                Message
Ready     False   ...      KubeletNotReady        [container runtime is down...
...output omitted...
```

OpenShift is unable to schedule pods for this worker node because the **worker01** node has **NoSchedule** and **NoExecute** taints. The **Conditions** section shows that the container run time (**cri-o**) is down, which is the reason for the taints and **Not Ready** status.

- 4.3. Use the **oc debug** command to log in to the cluster and attempt remediation. The **oc debug** command fails to start the debug pod because the worker node is in a **Not Ready** state. Press **Ctrl+C** to stop the **oc debug** command.

```
[student@workstation ~]$ oc debug node/worker01
Starting pod/worker01-debug ...
To use host binaries, run chroot /host
^C
Removing debug pod ...
```

- ▶ 5. Inspect the OpenShift services and correct the outage for the **crio** service on the worker node.
 - 5.1. Check the status of **crio** service for **worker01** to reveal that the service is not running.

```
[student@workstation ~]$ ssh core@worker01 "sudo systemctl is-active crio"
inactive
```

- 5.2. Start crio service for worker01.

```
[student@workstation ~]$ ssh core@worker01 "sudo systemctl start crio"
```

- 5.3. Check status of crio service for worker01 to confirm that the service is running.

```
[student@workstation ~]$ ssh core@worker01 "sudo systemctl is-active crio"
active
```

- ▶ 6. Verify the functionality of deployments on the worker node, and also that all monitoring alerts are resolved.
 - 6.1. Check the status of all Nodes and verify the node has been properly restored. It may take several minutes for the worker01 node to return to the Ready status.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES      AGE      VERSION
master01   Ready     master     ...      ...
master02   Ready     master     ...      ...
master03   Ready     master     ...      ...
worker01   Ready    worker     ...      ...
...output omitted...
```

- 6.2. Inspect the worker01 node for taints using the oc describe command.

```
[student@workstation ~]$ oc describe node/worker01 | grep -i taints
Taints:        <none>
```

Worker node worker01 is healthy and no longer displays any taints.

Finish

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

```
[student@workstation ~]$ lab workers-degrade finish
```

Troubleshooting Worker Node Capacity Issues

Objectives

After completing this section, you should be able to diagnose and troubleshoot capacity issues that occur on worker nodes.

Causes for Worker Node Storage Exhaustion

The OpenShift environment, and applications running within that environment, can cause storage issues on one or more worker nodes. As a cluster operator or administrator, you have an array of tools, such as configured monitoring probes and scripted cluster health reports, that aid in assessing and remediating worker node storage constraints.

Typical cluster operations, and the various deployments running within the cluster, are often the leading causes of storage exhaustion. You can avoid this problem using honed operational practices, or mitigate it through focused troubleshooting techniques. Continually improving operational practices, such as removing old logs or container images, instills behaviors that result in effective capacity management. Additionally, simulating disaster recovery scenarios and practicing troubleshooting skills directly correlates to overall better cluster health.

When left unmanaged, services that log output to the local disk of a worker node result in depleted disk space that can degrade overall cluster health and performance. Large or numerous container images consume space and require proper footprint management to avoid worker node storage depletion. Additionally, applications that are configured to utilize ephemeral storage allocated directly from the local disk of a worker node deplete available capacity.

One clue that OpenShift provides when worker node storage issues arise is the application of a `disk-pressure:NoSchedule` and/or `disk-pressure:NoExecute` taint. These taints are displayed in the output of the `oc describe node <NODE_NAME>` command.

Other Worker Node Capacity Concerns

Because worker node storage issues commonly arise in active clusters, these are not the main resources that require proper management by cluster administrators. System memory used by running applications is finite, and a large or poorly authored deployment is a candidate for worker node RAM depletion.

Similarly, CPU processing power is a limited resource that requires consideration when deploying applications with high computational requirements.

Lastly, network bandwidth is shared between all applications that run on a worker node, so an application that utilizes heavy traffic throughput results in degradation of all pods running on the worker node.

Best Practices for Managing Worker Node Resources

A best practice for maintaining worker node performance and overall cluster health, is properly utilizing the default configured monitoring. It is also prudent to configure additional monitoring probes for cluster operations, as needed, for both custom operators and deployed applications.

Traditional system administration tools, such as the `logrotate` service, are invaluable for maintaining resource allocations.

Employing quotas to manage application consumption of resources is also prudent in large-scale environments. Curating container images and using good management practices to prune old, stale, or large images reduces storage constraints caused by the growth of the image footprints.

► Guided Exercise

Troubleshooting Worker Node Capacity Issues

In this exercise, you will identify a capacity issue on a worker node and remediate the root cause.

Outcomes

You should be able to:

- Identify cluster capacity issues.
- Inspect and isolate the specific cause of a capacity constraint.
- Repair the problem and restore proper worker node functionality.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The workstation VM and cluster administrator credentials.

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

This command ensures that the cluster API is reachable and downloads the sample job.

```
[student@workstation ~]$ lab workers-capacity start
```

Instructions

► 1. Log in to the OpenShift cluster and create a `workers-capacity` project.

1.1. Log in as the `admin` user.

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

1.2. Create a new project `workers-capacity`.

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

▶ 2. Deploy a job that uses disk space.

- 2.1. Change to the ~/D0380/labs/workers-capacity directory.

```
[student@workstation ~]$ cd ~/D0380/labs/workers-capacity
```

- 2.2. Deploy the job.

```
[student@workstation workers-capacity]$ oc apply -f job.yml  
job.batch/diskuser created
```

▶ 3. Monitor the job execution.

- 3.1. Use the watch command to monitor the job execution. Note that a pod is created, runs for a while, and then is evicted. A new pod is created, but it stays in the Pending state. Press Ctrl+C to exit the watch command.

```
[student@workstation workers-capacity]$ watch oc get job,pod  
NAME          COMPLETIONS DURATION   AGE  
job.batch/diskuser  0/1        107s       107s  
  
NAME          READY  STATUS    RESTARTS  AGE  
pod/diskuser-4cfdd  0/1    Pending   0          7s  
pod/diskuser-ck4df  0/1    Evicted   0          107s
```

**Note**

The job can also create new pods that are evicted immediately.

- 3.2. Use the oc command to see which node executed the evicted pod.

```
[student@workstation workers-capacity]$ oc get pod -o wide  
NAME      READY  STATUS    ...  NODE      ...  
diskuser-4cfdd  0/1    Pending   ...  <none>    ...  
diskuser-ck4df  0/1    Evicted   ...  worker02  ...
```

**Note**

The job has a node selector ensuring it runs on worker02.

- 3.3. Use the oc describe command to inspect the worker02 node. Note that the worker02 node has a disk pressure taint and condition. The disk pressure taint has a NoSchedule effect.

```
[student@workstation workers-capacity]$ oc describe node worker02  
Name:           worker02  
...output omitted...  
Taints:         node.kubernetes.io/disk-pressure:NoSchedule  
...output omitted...  
Conditions:
```

Type	Status	Reason	...
MemoryPressure	False	KubeletHasSufficientMemory	...
DiskPressure	True	KubeletHasDiskPressure	...
PIDPressure	False	KubeletHasSufficientPID	...
Ready	True	KubeletReady	...
<i>...output omitted...</i>			

- 4. Examine the job and remediate the problem.

- 4.1. Delete the `diskuser` job.

```
[student@workstation workers-capacity]$ oc delete job diskuser
job.batch "diskuser" deleted
```

- 4.2. Run the `oc describe` command until the node has no disk pressure taint. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation workers-capacity]$ watch \
  "oc describe node worker02 | grep -i taint"
Taints:           <none>
```



Note

The `diskuser` job runs a pod that consumes all the available ephemeral disk space in the node. The kubelet detects the lack of disk space on the node and starts evicting pods from the node. Then, the kubelet adds a disk pressure taint and condition to the node.

The job can start new pods before the kubelet adds the disk pressure taint. The kubelet evicts the pods immediately because the node has a disk pressure condition.

When the kubelet evicts the `diskuser` pod that consumes disk space, the pod frees the space.

The kubelet takes some time to clear the disk pressure taint. As the job can only run on the `worker02` node, while the node has the disk pressure taint, new pods wait in the pending state until the kubelet clears the taint. If you wait until the kubelet clears the disk pressure taint, then the new pod executes and consumes disk space again, restarting the sequence.

- 4.3. Correct the `job.yml` file. The job tries to create a file of 1 TB that exceeds the node capacity. Change the job to create a 1 MB file. Update the file to match the following.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: diskuser
spec:
  template:
    spec:
      containers:
```

```
- name: diskuser
  image: registry.redhat.io/openshift4/ose-cli:v4.6
  command: ["sh", "-c", "dd if=/dev/zero of=/tmp/big_file bs=1M count=1 || sleep infinity"]
  nodeSelector:
    kubernetes.io/hostname: worker02
  restartPolicy: OnFailure
```



Note

The command: ... infinity"] fragment must be typed as a single line.

- 5. Run the job and monitor the job execution.

- 5.1. Deploy the job.

```
[student@workstation workers-capacity]$ oc apply -f job.yml
job.batch/diskuser created
```

- 5.2. Use the `watch` command to monitor the job execution. The job completes successfully. Press `Ctrl+C` to exit the `watch` command.

```
[student@workstation workers-capacity]$ watch oc get job,pod
NAME          COMPLETIONS   DURATION   AGE
job.batch/diskuser  1/1        3s         14s

NAME          READY   STATUS      RESTARTS   AGE
pod/diskuser-qgtzb  0/1     Completed   0          14s
```

- 6. Remove the project to clean up the exercise.

```
[student@workstation workers-capacity]$ oc delete project workers-capacity
project.project.openshift.io "workers-capacity" deleted
```

- 7. Change to the /home/student/ directory.

```
[student@workstation workers-capacity]$ cd
```

Finish

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

```
[student@workstation ~]$ lab workers-capacity finish
```

► Lab

Recovering Failed Worker Nodes

In this lab, you will identify, troubleshoot, and restore a failing worker node to a healthy status.

Outcomes

You should be able to:

- Profile worker nodes to examine healthy nodes, ensure the node services function properly, and identify adverse performance or degradation.
- Diagnose and troubleshoot issues that occur on worker nodes.
- Verify that proper worker node functionality is restored.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- Administrative access to the CLI and web console.

On the **workstation** machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

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

Instructions

1. Access the cluster CLI and web console and inspect the monitoring alerts.
2. Review the status of all cluster nodes and identify the unhealthy node.
3. Correct the issue with the unhealthy worker node.
4. Verify the worker node health is restored and that the monitoring alerts have subsided.

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 workers-review grade
```

Finish

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

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

► Solution

Recovering Failed Worker Nodes

In this lab, you will identify, troubleshoot, and restore a failing worker node to a healthy status.

Outcomes

You should be able to:

- Profile worker nodes to examine healthy nodes, ensure the node services function properly, and identify adverse performance or degradation.
- Diagnose and troubleshoot issues that occur on worker nodes.
- Verify that proper worker node functionality is restored.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- Administrative access to the CLI and web console.

On the **workstation** machine, use the `lab` command to prepare your system for this exercise.

The command ensures that the cluster API is reachable.

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

Instructions

1. Access the cluster CLI and web console and inspect the monitoring alerts.

- 1.1. Log in to your OpenShift cluster as the `admin` user.

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

...output omitted...
```

- 1.2. Display the address for the web console by running the following command:

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

- 1.3. Open the link provided in the command output in a browser, and then log in as the `admin` user with the `redhat` password.

- 1.4. From the web console, navigate to the **Monitoring** section. Inspect the **Alerts** tab found on the **Alerting** page for alerts that provide information regarding cluster performance degradation. After a few minutes, several alerts with a severity of **Warning** and a state of **Firing** are displayed alongside additional AlertManager notifications. These alerts provide details about the cluster degradation, cluster member absence, and make explicit mention that **worker02** is not ready.
2. Review the status of all cluster nodes and identify the unhealthy node.
 - 2.1. Utilize CLI commands to profile the status of the worker nodes, determine affected worker node, and then formulate an appropriate remediation.

Check the status of all nodes. Notice that one worker node reports a **Not Ready** status.

```
[student@workstation ~]$ oc get nodes
NAME        STATUS   ROLES      AGE     VERSION
master01    Ready    master     14d    ...
master02    Ready    master     14d    ...
master03    Ready    master     14d    ...
worker01    Ready    worker     14d    ...
worker02    Not Ready worker     14d    ...
...output omitted...
```

- 2.2. Inspect the **worker02** node showing a **Not Ready** status using the **oc describe** command.

```
[student@workstation ~]$ oc describe node/worker02
...output omitted...
Taints:           node.kubernetes.io/unreachable:NoExecute
                  node.kubernetes.io/unreachable:NoSchedule
...output omitted...
Conditions:
  Type        Status  ...  Reason
  ----        -----  ...  ...
  MemoryPressure Unknown ...  NodeStatusUnknown ...
  DiskPressure  Unknown ...  NodeStatusUnknown ...
  PIDPressure   Unknown ...  NodeStatusUnknown ...
  Ready        Unknown ...  NodeStatusUnknown ...
...output omitted...
```

OpenShift cannot schedule pods for this worker node because the **worker02** node has an unreachable taint with the **NoExecute** and **NoSchedule** effects. The **Conditions** section shows **NodeStatusUnknown**, and a message of **Kubelet stopped posting node status**. This condition details the reason for the taint and **Not Ready** status.

3. Correct the issue with the unhealthy worker node.
 - 3.1. Use the **oc debug** command to log in to the cluster and attempt remediation. The **oc debug** command fails to start the debug pod because the worker node is in a **Not Ready** state. Press **Ctrl+C** to stop the **oc debug** command.

```
[student@workstation ~]$ oc debug node/worker02
Starting pod/worker02-debug ...
To use host binaries, run chroot /host
^C
Removing debug pod ...
```

- 3.2. Inspect the OpenShift services on the worker02 worker node and correct the outage for the kubelet service.

Check the status of kubelet service for worker02 to reveal that the service is not running.

```
[student@workstation ~]$ ssh core@worker02 "sudo systemctl is-active kubelet"
inactive
```

- 3.3. Start kubelet service for worker02.

```
[student@workstation ~]$ ssh core@worker02 "sudo systemctl start kubelet"
```

- 3.4. Verify the status of the kubelet service for worker02 and confirm that the service is running.

```
[student@workstation ~]$ ssh core@worker02 "sudo systemctl is-active kubelet"
active
```

4. Verify the worker node health is restored and that the monitoring alerts have subsided.

- 4.1. Check the status of all Nodes and verify the node has been properly restored. It can take several minutes for the worker02 node to return to the Ready status.

```
[student@workstation ~]$ oc get nodes
NAME      STATUS    ROLES      AGE      VERSION
master01   Ready     master     14d      ...
master02   Ready     master     14d      ...
master03   Ready     master     14d      ...
worker01   Ready     worker     14d      ...
worker02   Ready     worker     14d      ...
...output omitted...
```

- 4.2. Inspect the worker02 node for taints using the oc describe command.

```
[student@workstation ~]$ oc describe node worker02 | grep -i taints
Taints:        <none>
```

Worker node worker02 is healthy and no longer displays any taints.

- 4.3. Revisit the monitoring section of the web console to verify that the monitoring alerts have cleared. This might take several minutes of additional polling before the status is returned to normal and the alerts clear.

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 workers-review grade
```

Finish

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

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

Summary

In this chapter, you learned:

- Worker nodes can experience a degradation of performance and functionality that can be identified and mitigated through the use of monitoring, CLI commands, and status information provided by OpenShift.
- Proper administration of an OpenShift cluster requires awareness and mitigation of various capacity issues that arise during operation.

Appendix A

Creating a Quay Account

Goal

Describe how to create a Quay account for labs in this course.

Creating a Quay Account

Objectives

After completing this section, you should be able to create a Quay account and create public container image repositories for the labs in this course.

Creating a Quay Account

You need a Quay account to create one or more *public* container image repositories for the labs in this course. If you already have a Quay account, then you can skip the steps to create a new account listed in this appendix.



Important

If you already have a Quay account, then ensure that you only create *public* container image repositories for the labs in this course. The lab grading scripts and instructions require unauthenticated access to pull container images from the repository.

To create a new Quay account, perform the following steps:

1. Navigate to <https://quay.io> using a web browser.
2. Click **Sign in** in the upper-right corner (next to the search bar).
3. On the **Sign in** page, you can log in using your Red Hat, Google or GitHub credentials. Alternatively, click **Register for a Red Hat account** to create a new account.

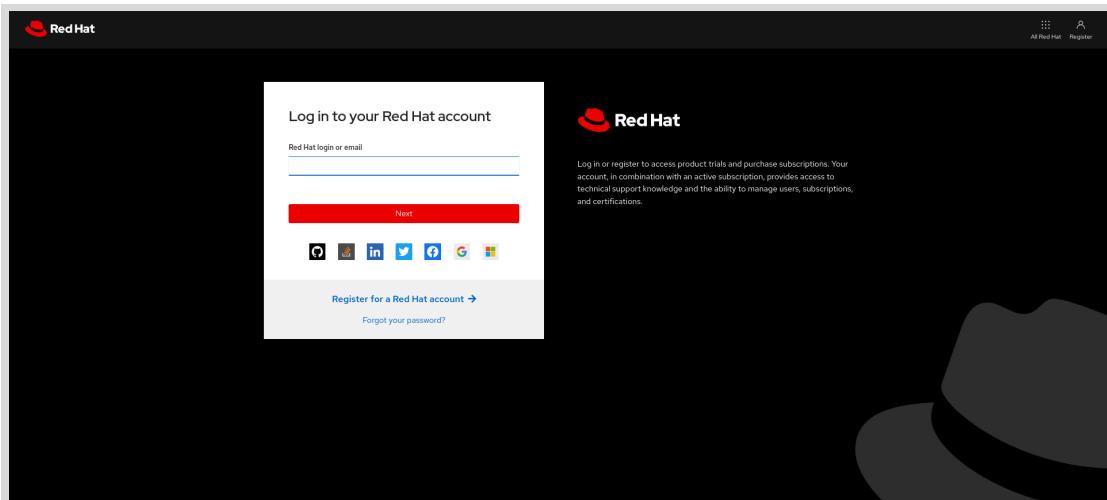
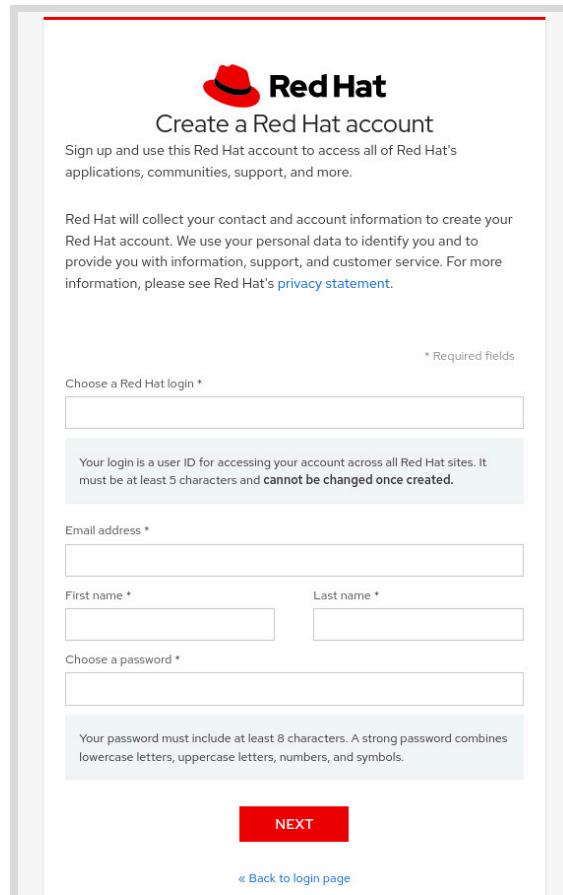


Figure A.1: Sign in using Red Hat, Google or GitHub credentials or create a new account.

4. If you chose to skip the Google or GitHub log-in method and instead opted to create a new account, the **Register for a Red Hat account** link will take you to the **Create a Red Hat account** page, follow the instructions on this page to create your new Quay account.

Appendix A | Creating a Quay Account



The image shows a screenshot of the Red Hat account creation form. At the top, there is a Red Hat logo and the text "Create a Red Hat account". Below this, a note states: "Sign up and use this Red Hat account to access all of Red Hat's applications, communities, support, and more." Another note below it says: "Red Hat will collect your contact and account information to create your Red Hat account. We use your personal data to identify you and to provide you with information, support, and customer service. For more information, please see Red Hat's [privacy statement](#)". The form includes fields for "Choose a Red Hat login *", "Email address *", "First name *", "Last name *", and "Choose a password *". A note next to the password field specifies: "Your password must include at least 8 characters. A strong password combines lowercase letters, uppercase letters, numbers, and symbols." At the bottom right is a red "NEXT" button, and at the bottom left is a link "« Back to login page".

Figure A.2: Sign in using Red Hat, Google or GitHub credentials or create a new account.

5. Verify your email address and then sign in to the Quay website with the username and password you provided during account creation.



References

Quay.io transitioning to Red Hat SSO

<https://access.redhat.com/articles/5925591>

Repositories Visibility

Objectives

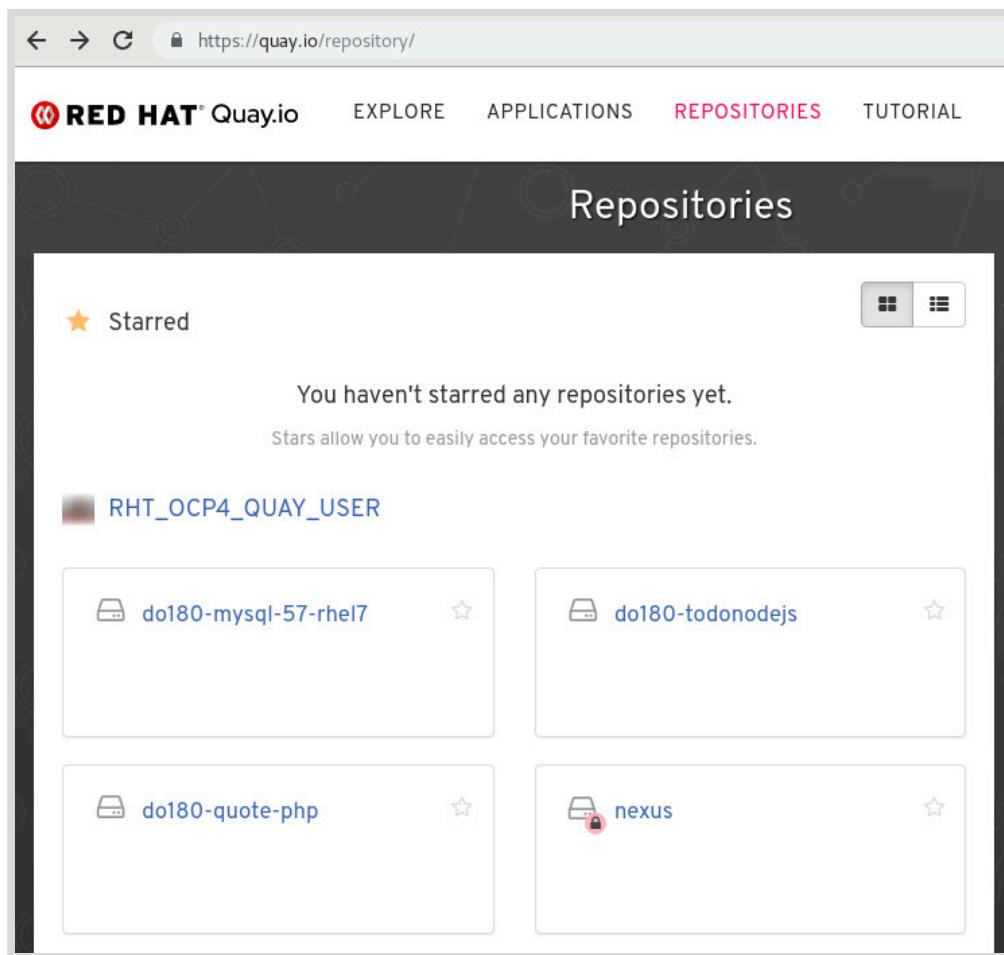
After completing this section, you should be able to control repository visibility on Quay.io.

Quay.io Repositories Visibility

Quay.io offers the possibility of creating public and private repositories. Public repositories can be read by anyone without restrictions, however write permissions must be explicitly granted. Private repositories have both read and write permissions restricted. Nevertheless, the number of private repositories in quay . io is limited depending on the namespace's plan.

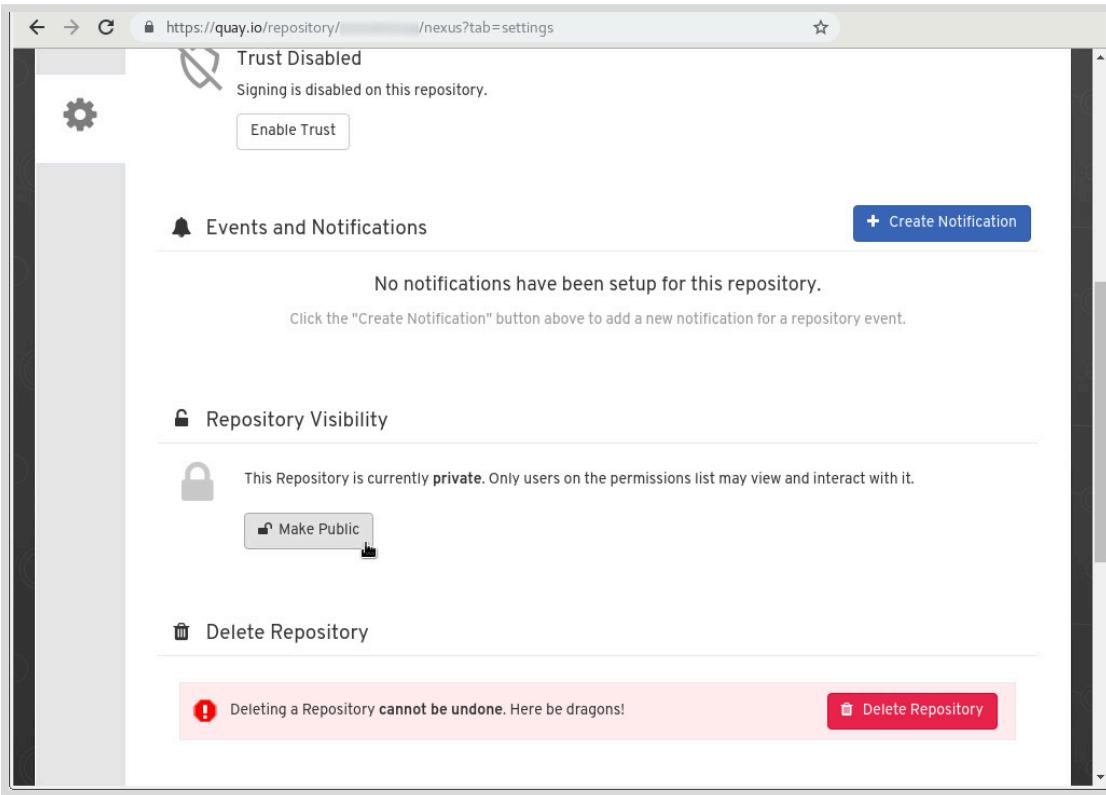
Default Repository Visibility

Repositories created by pushing images to quay . io are private by default. In order for OpenShift (or any other tool) to fetch those images you can either configure a private key in both OpenShift and Quay, or make the repository public, so no authentication is required. Setting up private keys is outside the scope of this document.



Updating Repository Visibility

In order to set repository visibility to public, select the appropriate repository in <https://quay.io/repository/> (log in to your account if needed) and open the **Settings** page by clicking on the gear icon on the left-bottom edge. Scroll down to the **Repository Visibility** section and click the **Make Public** button.



Navigate back to the list of repositories. The lock icon beside the repository name disappears, indicating the repository has become public.

Appendix B

Creating a GitHub Account

Goal

Describe how to create a GitHub account for labs in the course.

Creating a GitHub Account

Objectives

After completing this section, you should be able to create a GitHub account and create public Git repositories for the labs in the course.

Creating a GitHub Account

You need a GitHub account to create one or more *public* Git repositories for the labs in this course. If you already have a GitHub account, you can skip the steps listed in this appendix.



Important

If you already have a GitHub account, ensure that you only create *public* Git repositories for the labs in this course. The lab grading scripts and instructions require unauthenticated access to clone the repository. The repositories must be accessible without providing passwords, SSH keys, or GPG keys.

To create a new GitHub account, perform the following steps:

1. Navigate to <https://github.com> using a web browser.
2. Enter the required details and then click **Sign up for GitHub**.

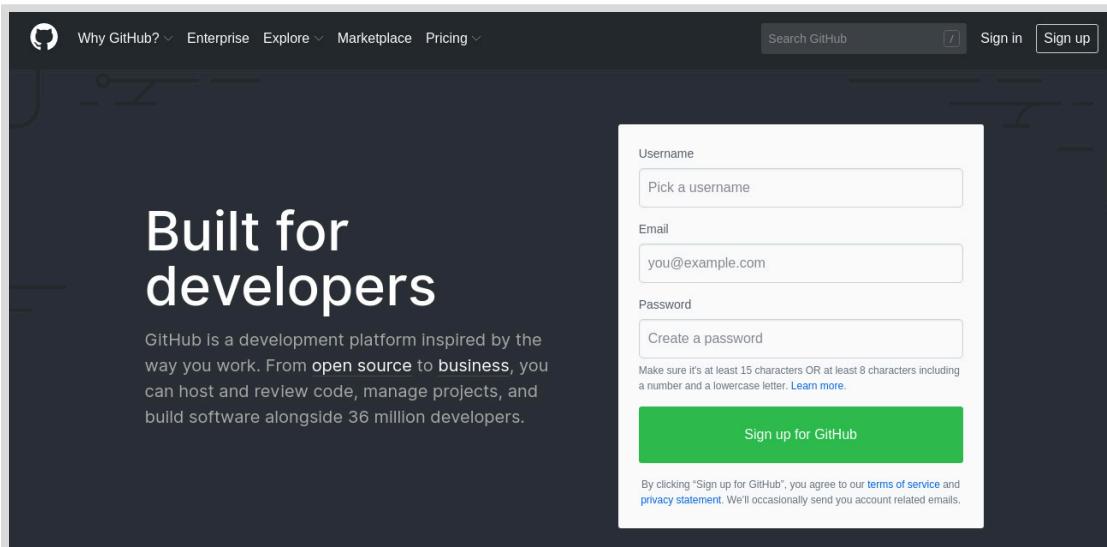


Figure B.1: Creating a GitHub account

3. You will receive an email with instructions on how to activate your GitHub account. Verify your email address and then sign in to the GitHub website using the username and password you provided during account creation.
4. After you have logged in to GitHub, you can create new Git repositories by clicking **New** in the **Repositories** pane on the left of the GitHub home page.

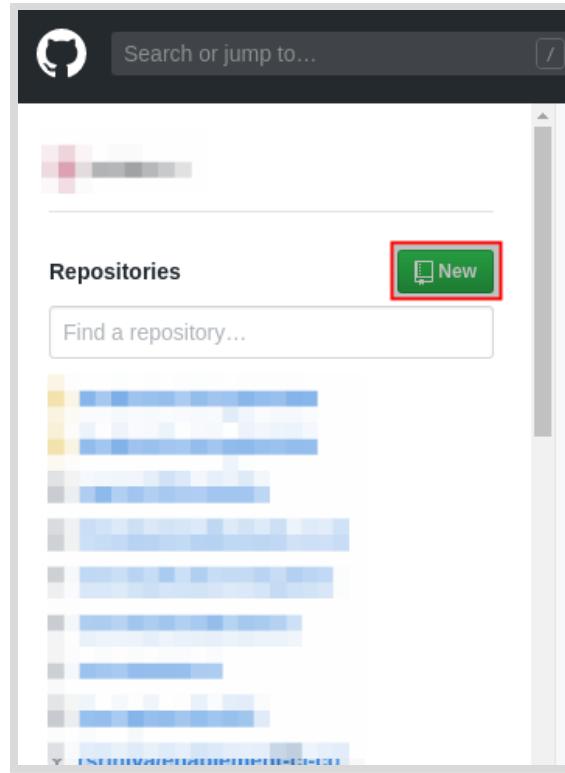


Figure B.2: Creating a new Git repository

Alternatively, click the plus icon (+) in the upper-right corner (to the right of the bell icon) and then click **New repository**.

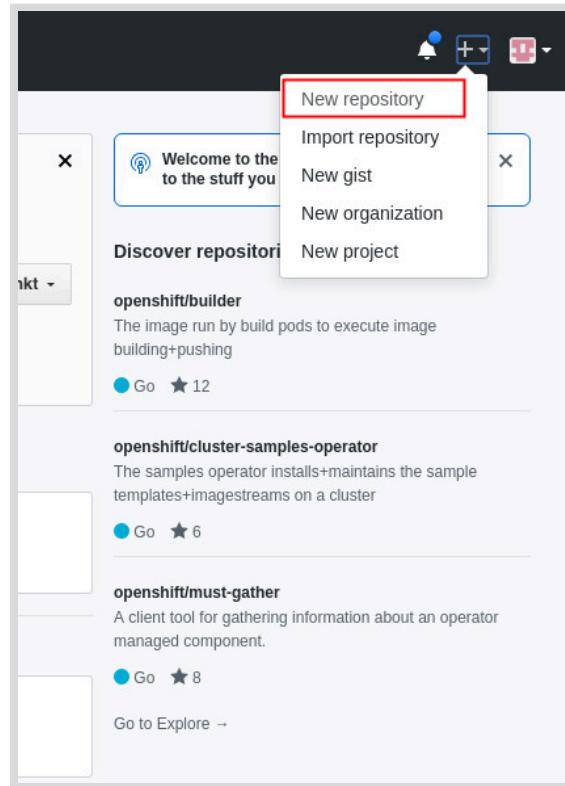


Figure B.3: Creating new Git repository

Creating a GitHub Access Token

1. Navigate to <https://github.com> using a web browser and authenticate.
2. On the top of the page, click your profile icon and select the **Settings** menu.

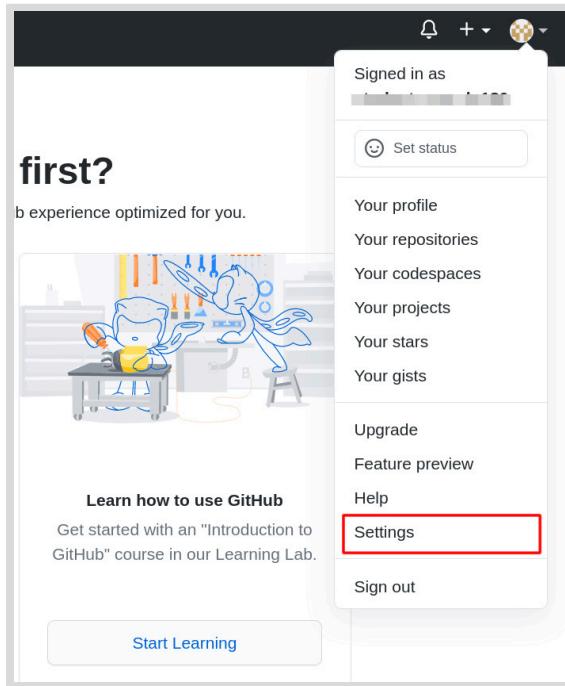


Figure B.4: User menu

3. Select **Developer settings** in the left pane.

A screenshot of the GitHub developer settings page. On the left, there is a sidebar with a list of options: Profile, Account, Appearance (with a "New" badge), Account security, Billing & plans, Security log, Security & analysis, Emails, Notifications, SSH and GPG keys, Repositories, Packages, Organizations, Saved replies, Applications, and Developer settings (which is highlighted with a red box). The main content area contains sections for Name, Public email, Bio, URL, Twitter username, and Company, each with input fields and descriptive text. A "Profile picture" placeholder image is also present.

Figure B.5: Developer settings

Appendix B | Creating a GitHub Account

4. Select the Personal access token section on the left pane. On the next page, create your new token by clicking **Generate new token** and then entering your password when prompted.

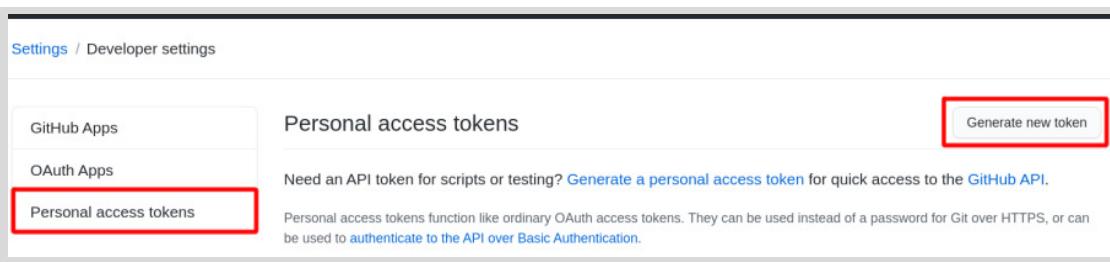


Figure B.6: Personal access token pane

5. Write a short description about your new access token in the **Note** field.
6. Select the **public_repo** option only. Create your new access token by clicking **Generate token**.

The screenshot shows the 'New personal access token' configuration page. A red box highlights the 'Note' input field containing 'Course DO380'. Another red box highlights the 'Select scopes' section, specifically the 'public_repo' checkbox which is checked.

Scopes	Description
<input type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events

Figure B.7: Personal access token configuration

7. Your new personal access token is displayed in the output. Using your preferred text editor, create a new file in the student's home directory named **token** and ensure you paste in your generated personal access token. The personal access token can not be displayed again in GitHub.

The screenshot shows the GitHub 'Personal access tokens' page. At the top, there are buttons for 'Generate new token' and 'Revoke all'. Below this, a message says 'Tokens you have generated that can be used to access the [GitHub API](#)'. A note below says 'Make sure to copy your new personal access token now. You won't be able to see it again!'. A red box highlights a specific token entry: '✓ ghp_kgYGzWcGE1CrdovkzuzeLTWvYY6eBX2l0vck' with a copy icon, and a 'Delete' button to its right. Below the token list, a note states: 'Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API](#) over Basic Authentication'.

Figure B.8: Generated access token

8. On the workstation machine execute the `git config credential.helper` command with the `cache` parameters to store in cache memory your credentials for future use. The `--global` parameter applies the configuration to all of your repositories.

```
[student@workstation ~]$ git config credential.helper 'cache --timeout=3600'
```



Important

During this course, if you are prompted for a password while using Git operations on the command line, use your access token as the password.



References

[Signing up for a new GitHub account](https://help.github.com/en/articles/signing-up-for-a-new-github-account)

<https://help.github.com/en/articles/signing-up-for-a-new-github-account>

[Create a GitHub access token](https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token)

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>