Red Hat
Training and Certification

Student Workbook

ACM 2.4 DO480

Multicluster Management with Red Hat OpenShift Platform Plus

Edition 0

DO480-ACM2.4-en-0-f54b122

Red Hat
Learning Community

Join a community dedicated to learning open source

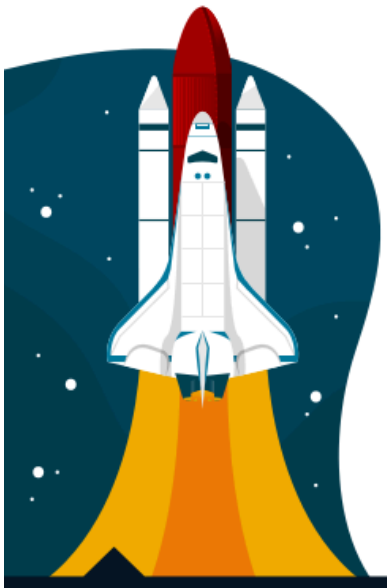
The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.

Network with tens of thousands of comunity 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

Red Hat

Multicluster Management with Red Hat OpenShift Platform Plus

ACM 2.4 DO480
Multicluster Management with Red Hat OpenShift Platform Plus
Edition 0 f54b122
Publication date 20220101

Author: John Smith
Course Architect: Jane Smith
DevOps Engineer: John Smith
Editor: Jane Smith

Contributors: John Doe, Jane Doe

Document Conventions

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

Admonitions

Red Hat Training courses use the following admonitions:

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

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

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

Warning
These should not be ignored. Ignoring these admonitions will most likely cause data loss.

Inclusive Language

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

Introduction

DO480: Multicluster Management with Red Hat OpenShift Platform Plus

Multicluster Management with Red Hat OpenShift Platform Plus teaches the skills required to maintain a diverse portfolio of applications, running across a fleet of OpenShift clusters. Applications follow placement rules determined by capacity and criticality; cluster configurations comply with governance and security policies; all automated according to DevOps principles.

Course Objectives

- Deploy Red Hat Advanced Cluster Management (ACM) in a hub cluster.
- Add a managed cluster to ACM (configure a cluster to be managed by ACM).
- Define and apply cluster configuration policies.
- Detect and correct non-conformance to cluster configuration policies.
- Define and apply application placement policies.
- Deploy and update applications from CI/CD and GitOps workflows.
- Deploy Red Hat Quay in the hub cluster.
- Deploy Red Hat Advanced Cluster Security (ACS) in the hub cluster.
- Integrate Quay and ACS security with ACM management.

Audience

- System Administrators, Developers, SREs, and IT Architects interested in managing and automating the management of a fleet of OpenShift clusters, possibly in different data centers and cloud providers.

Prerequisites

- Red Hat Certified Specialist in OpenShift Administration certification (EX280) or equivalent knowledge for the roles of Red Hat OpenShift cluster administrator or SRE.
- Red Hat Certified Systems Administrator certification (EX200) or equivalent knowledge of Linux system administration is recommended for all roles.
- Basic knowledge of Kubernetes and OpenShift administration skills are recommended.

Orientation to the Classroom Environment

Orientation to the Classroom Environment

In this course, the main computer system used for hands-on learning activities is `workstation`.

The `bastion` system must always be running.

Two other machines are also used by students for these activities:

`servera`, `serverb`, `serverc`, and `serverd`.

All six of these systems are in the `lab.example.com` DNS domain.

Classroom Machines

Machine name	IP addresses
ROLE	bastion.lab.example.com
172.25.250.254	Router that links VMs to central servers
workstation.lab.example.com	172.25.250.9
Graphical workstation for system administration	servera.lab.example.com
172.25.250.10	Managed server "A"
servera.lab.example.com	172.25.250.11
Managed server "B"	servera.lab.example.com
172.25.250.12	Managed server "C"
servera.lab.example.com	172.25.250.13

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
```

Exercises are in two types. The first type, a *guided exercise*, is a practice exercise that follows a course narrative. If a narrative is followed by a quiz, then usually 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 action exercise
```

The *action* is a choice of `start`, `grade`, or `finish`. All exercises support `start` and `finish`. Only end-of-chapter labs and comprehesive 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 required resources to begin an exercise. It might include configuring settings, creating resources, checking prerequisite services, and verifying necessary outcomes from previous exercises. You can take an exercise at any time, even without taking prerequisite exercises.

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 rerun the grade action.

finish
: Formerly `cleanup`. A script's finish logic deletes exercise resources that are no longer necessary. You can take an exercise as many times as you want.

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 are normally displayed 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`, and 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) that is invoked from `workstation` to communicate with server systems by 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

Managing Multicluster Kubernetes Architectures

Goal

Describe multicluster architectures and solve its challenges with Red Hat OpenShift Platform Plus.

Objectives

- Describe common use cases for multiple Kubernetes clusters and examine the challenges of multicluster architectures.
- Identify the tools provided by OpenShift Platform Plus to facilitate end-to-end management, security, and compliance of Kubernetes fleets on hybrid clouds.
- Describe and deploy the RHACM operator from the OperatorHub in an OpenShift hub cluster.

Sections

- Introducing Kubernetes Multicluster and Hybrid Cloud Architectures Landscape (and Quiz)
- Describing Red Hat OpenShift Platform Plus (and Quiz)
- Deploying Red Hat Advanced Cluster Management for Kubernetes (and Guided Exercise)

Lab

Installing the RHACM Operator

Introducing Kubernetes Multicluster and Hybrid Cloud Architectures Landscape

Objectives

After completing this section, you should be able to describe common use cases for multiple Kubernetes clusters and examine the challenges of multicluster architectures.

Interpreting the Background of Kubernetes Multicluster Architectures

Digital transformation is challenging traditional Information Technology (IT) departments and teams.

The wide adoption of the Internet brought new business models to different industries, with new technologies and ways of working. One example is the music and video streaming services that completely changed the entertainment industry. Another example is the growth of online stores that pushed traditional retailers to change how they operated their traditional stores. These disruptions affect other sectors also, such as telecommunications, the automotive business, and financial companies.

Furthermore, these new business models must frequently support operations that serve a huge number of users. An online business with one thousand customers could succeed and grow to hundreds of thousands or millions of users in a few days or weeks.

The following list enumerates some of the challenges that face IT departments:

- Modernizing systems and processes to deliver new software services quickly
- Building new services for customers to adapt to new industry trends and legislation
- Adapting new and existing services to be capable of running on a global scale
- Establishing new methodologies that support the work needed to modernize, build, and adapt their services

Three modern technologies and ways of working are helping IT departments to adapt to digital transformation:

- Cloud computing
- Containers technology
- Automation of processes

Describing Cloud Computing: The Way to the Hybrid Cloud

In IT, a cloud is an environment that abstracts, pools, and shares resources across a network. Cloud computing is the act of running workloads in a cloud environment. The following are some of the main features of a cloud computing IT system:

- You can access all resources in the cloud computing environment through a network.
- The cloud computing environment contains a repository of IT resources, such as applications, software services, and hardware abstractions.
- You can provision and scale the cloud computing environment quickly.

There are different types of clouds, such as public clouds, private clouds, and hybrid clouds. A public cloud provider is a company in charge of maintaining the base hardware, the network, and

the software used to virtualize the servers. Cloud providers also offer service agreements and pay-per-use options for using their services.

The arrival of public cloud providers was a disruptive change in the IT market. Now, an enterprise can run all its software workloads without maintaining on premises data centers or spending on new hardware.

However, not all companies and workloads are suitable for migration to a public cloud. The following list suggests some reasons to limit the use of public clouds:

- The company already has many critical workloads running in its own data centers.
- The cost of migrating to cloud-native developments is very high.
- Some legacy systems are not adapted to run on cloud environments.
- The operational cost of public cloud services can be high, especially if the enterprise grows rapidly.
- Some verticals, such as healthcare or finance, must comply with important regulatory restrictions on where their data is stored. The General Data Protection Regulation law of the European Union is an excellent example of a data location restriction.
- Concerns about security and losing control of the IT infrastructure are prioritized.

There are many other models of interacting with one or multiple clouds, such as between a fully externalized public provider and a private data center. One model is a fully private cloud.

A private cloud is an IT infrastructure hosted in private data centers whose services are requested and served with a model similar to public clouds:

- Self-provisioning on demand
- Auto-scaling resources based on infrastructure usage
- Better resource allocation and utilization over the same physical resources

Private clouds can be isolated from the external world. Ideally, a private cloud cannot and should not communicate with a public cloud. However, in the real world, IT departments use mixed models. Private data centers, hosted data centers, data centers on public clouds, or data centers on private clouds need to share resources between them. A Hybrid Cloud is an IT infrastructure that can communicate and share services between the following resource types:

- One or more data centers with bare metal or virtualized environments owned by the company
- One or more data centers with bare metal or virtualized environments rented to other companies
- One or more private clouds
- One or more public clouds

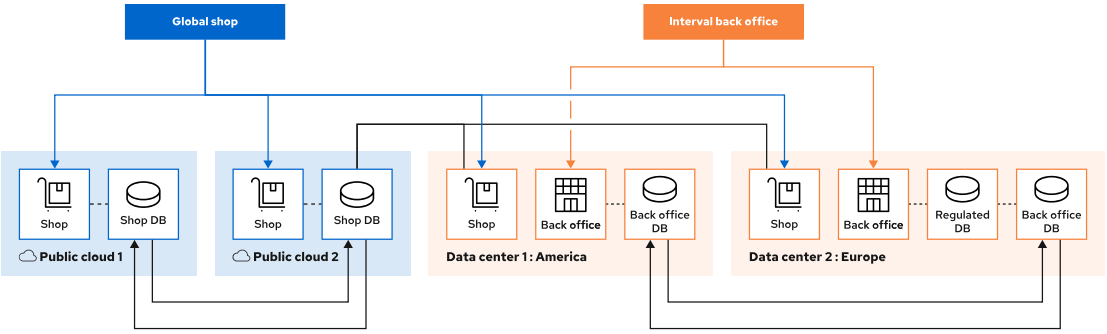
The boundaries between some of these kinds of infrastructures are not always clear, particularly as the technologies evolve. For example, many public cloud providers install some services, and physical resources in some cases, in their customer's private data centers.

Using multiple clouds has the following benefits:

- High availability and failover capabilities: If the primary cloud is down,then other clouds can assume the workload.
- Using specific services from specific clouds: Administrators can use a preferred cloud service from each cloud.
- Geographical proximity: Some services can be faster if they run in close proximity to the user, although this depends on the customer's location.
- Regulatory restrictions: In some cases, data regulatory restrictions can apply.

- Network latency: For some industries like Telco, network traffic latency is a critical concern.

As an example of a hybrid cloud, the following diagram displays the distribution of two software services, an online shop for global access to customers and a back-office application to manage the online shop. The shop service has the benefits of global distribution. Customers always connect to the nearest shop location. The back-office service only runs in the private data centers, whereas the shop service runs in the private data centers and in two public clouds. However, the regulated back-office data is only present in the Europe data center.



Explaining the Need for Containers and Orchestration

As discussed elsewhere in this chapter, software development techniques are evolving quickly. Scale economies and new disruptive players in different industries push to bring new features to their applications as soon as possible. Nevertheless, user applications must still need to be developed with quality, stability, performance, and high availability.

Linux containers are one of the best approaches to providing the infrastructure and development framework to run both cloud-native applications and legacy workloads.

Container technologies use software packaging techniques to make applications fully portable to any hybrid cloud. The packaging and portability of containers make them suitable for every cloud-computing model and in traditional data centers.

The following list describes some of the benefits of using Linux containers:

- Security, storage, and network isolation, similar to virtual machines (VMs)
- Require fewer hardware resources and are quick to start and terminate compared to VMs
- Quick deployment because you do not need to install a complete operating system
- Portable and reuseable applications by using environments

In contrast, the use of Linux containers can introduce some of the following new challenges:

- Managing communications between services provided by containers
- Managing versions of running containers and controlling their deployment
- Scaling containers up and down to respond to the real demands of the service
- Determining the location of the containers in the existing infrastructure
- Limiting and balancing available compute resources for containers

You can address these challenges by using a container orchestration system. Kubernetes is an orchestration system that simplifies the deployment, management, and scaling of containerized applications.

Automation Processes

Companies must reduce the time to market for their new services and offerings but keep the ability to scale their infrastructure up or down quickly if demands change.

By using hybrid clouds and Linux containers orchestration systems, companies can cope with changing demands. Moreover, companies must also increase agility and automate their processes at both the technical and organizational levels.

Modern IT teams are adapting their work to adopt agile models that automate a good part of the software development lifecycle. Methods like CI/CD, DevSecOps, and GitOps are difficult to implement, but are the key to an organization's ability to rapidly bring new business value through software development.

Note

To learn more about CI/CD and GitOps see the GLS course `DO380 - Red Hat OpenShift Administration III : Scaling Kubernetes Deployments in the Enterprise`, chapter 4.

Challenges of Multicluster Architectures

The adoption of automation, containers, and Kubernetes in Hybrid Cloud environments means that companies must find a way to easily create and manage Kubernetes clusters across the hybrid environment.

The following sections discuss the main challenges to adopting a multicluster architecture.

Managing Fleets of Clusters

As software development processes evolve, it might be necessary to use ephemeral Kubernetes clusters. For example, if many DevOps pipelines create a whole cluster to do the quality assurance (QA) end-to-end test, then you need ephemeral Kubernetes clusters. Sometimes, you destroy the cluster after completing the test. Other times, you must keep the cluster temporarily to perform research on it after the tests are done.

When an organization has many development and QA teams belonging to different areas or regions, or in different business units, the number of Kubernetes clusters can grow quickly. The higher the number of clusters, the more difficult it is to locate information about the performance and the status of the fleet.

Managing a high number of clusters is time-consuming and error prone. Administrators must work with multiple consoles and distributed business applications, and sometimes inconsistent security controls across the diverse clusters deployed on premises or in public clouds.

Organizing Compliance and Security of Clusters

IT departments are often under different regulatory compliance requirements. Suppose a company must ensure that one cluster is compliant with standard or custom policies. It it is easy for a team to manually implement the policies on one cluster and ensure that they are correctly applied. On the other hand, manually applying custom policies is impossible if the company has hundreds or thousands of clusters. In such cases, companies need a mechanism to set consistent security policies across diverse environments and to ensure enforcement.

Another concern of IT departments from a security viewpoint is the origin of the software that runs in the Kubernetes clusters. The packaging techniques of containers make it very easy to package software inside a container with security problems or obsolete versions of some dependencies. The developer's freedom to choose the software used in a container has the associated risk of adding layers of defective software to the container image.

Organizing Applications Between Clusters

The placement of the workloads must be automatic and based on the capacity of the clusters, availability zones, or other policies previously defined.

Furthermore, if the applications have dependencies with services in different clusters, having visibility of the network topology is a difficult challenge. Multicluster architectures need repositories of IT artifacts available from different clouds. This need represents another challenge that affects not only the applications but also the deployment of infrastructure in the multicluster environment. In the containers world, many infrastructure components are packaged as container images.

To summarize, the following table relates the new technologies and processes to the challenges that those technologies and new processes present in a multicluster architecture:

Technology or Process	Challenges Introduced by Multicluster Architectures
Cloud computing	- Easily provisioning clusters on all types of clouds - Locating information about objects present in the fleet of clusters - Managing configuration compliance and the fleet of cluster's security - Monitoring the behavior of all clusters and their workloads from performance and scalability points of view
Containers	- Requires a containers orchestration system - Managing communications between services provided by containers, even when they are in different clusters or clouds - Requires a controlled place to store the artifacts and images that containers use - Security scanning of the artifacts and images that containers use
Automation processes	- An easy way for developers to implement CI/CD across a fleet of clusters - An easy way for DevOps teams to implement GitOps across a fleet of clusters

References

Understanding cloud computing
https://www.redhat.com/en/topics/cloud

Types of cloud computing
https://www.redhat.com/en/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud

What is Hybrid Cloud?
https://www.redhat.com/en/topics/cloud-computing/what-is-hybrid-cloud

For more information, refer to the official Documentation of Red Hat Advanced Cluster Management for Kubernetes at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4

Quiz

Introducing Kubernetes Multicluster and Hybrid Cloud Architectures Landscape

Choose the correct answers to the following questions:

1. Which two of the following modern technologies are helping IT departments to adapt to digital transformation? (Choose two.)

a. Cloud computing
b. Distributed computing
c. Linux containers
d. Databases
e. Private data centers

2. Which three of the following advantages are a result of using multiple clouds? (Choose three.)

a. Geographical proximity to the end users
b. Centralized data sources
c. Managing regulatory restrictions per country
d. Ability to use specific cloud services from different providers
e. High availability and service failover between clouds

3. Which two of the following challenges arise from using Linux containers? (Choose two.)

a. Managing communication between services running in containers
b. Easily updating the server RPM packages
c. Allocating resources effectively for the running containers
d. Reverting failed operating system upgrades

4. Which three of the following advantages result from using Linux containers compared to legacy deployments? (Choose three.)

a. Portability and reusability of the applications
b. Better resource utilization
c. Simpler deployment architecture
d. Reduced time to deployment
e. Reduced need to keep software versions updated

5. Which four of the following challenges arise from using a Kubernetes multicluster architecture? (Choose four.)

a. Managing multiple clusters effectively
b. Keeping high availability of the running applications
c. Distributing the applications efficiently among multiple clusters
d. Ensuring the security and compliance of the fleet of clusters
e. Monitoring the fleet of clusters from a single pane of glass
f. Failover when a cloud environment is unavailable

Solution

Introducing Kubernetes Multicluster and Hybrid Cloud Architectures Landscape

Choose the correct answers to the following questions:

1. Which two of the following modern technologies are helping IT departments to adapt to digital transformation? (Choose two.)

a. Cloud computing
b. Distributed computing
c. Linux containers
d. Databases
e. Private data centers

2. Which three of the following advantages are a result of using multiple clouds? (Choose three.)

a. Geographical proximity to the end users
b. Centralized data sources
c. Managing regulatory restrictions per country
d. Ability to use specific cloud services from different providers
e. High availability and service failover between clouds

3. Which two of the following challenges arise from using Linux containers? (Choose two.)

a. Managing communication between services running in containers
b. Easily updating the server RPM packages
c. Allocating resources effectively for the running containers
d. Reverting failed operating system upgrades

4. Which three of the following advantages result from using Linux containers compared to legacy deployments? (Choose three.)

a. Portability and reusability of the applications
b. Better resource utilization
c. Simpler deployment architecture
d. Reduced time to deployment
e. Reduced need to keep software versions updated

5. Which four of the following challenges arise from using a Kubernetes multicluster architecture? (Choose four.)

a. Managing multiple clusters effectively
b. Keeping high availability of the running applications
c. Distributing the applications efficiently among multiple clusters
d. Ensuring the security and compliance of the fleet of clusters
e. Monitoring the fleet of clusters from a single pane of glass
f. Failover when a cloud environment is unavailable

Describing Red Hat OpenShift Platform Plus

Objectives

After completing this section, you should be able to identify the tools provided by OpenShift Platform Plus to facilitate end-to-end management, security, and compliance of Kubernetes fleets on hybrid clouds.

Describing Red Hat OpenShift Platform Plus

Red Hat OpenShift Container Platform (RHOCP) is a certified enterprise Kubernetes distribution. It runs on a wide variety of infrastructure providers: public and private clouds, virtualization software, and bare metal servers. OpenShift Container Platform allows administrators to build a portable, production ready, and hybrid application development container platform.

Red Hat OpenShift Platform Plus is a collection of software tools running on Red Hat OpenShift Container Platform. Red Hat OpenShift Platform Plus allows administratos to manage the software life cycle across Kubernetes and RHOCP clusters. It provides end-to-end visibility and control over multiple Kubernetes clusters from a single pane of glass. Furthermore, OpenShift Platform Plus brings Kubernetes-native security capabilities to protect the software supply chain, infrastructure, and workloads. Finally, OpenShift Platform Plus also provides a globally-distributed and scalable registry.

The capabilities mentioned previously are provided by the following Red Hat products, included in the Red Hat OpenShift Platform Plus subscription, and distributed and installed as Kubernetes operators in OpenShift Container Platform.

Red Hat Advanced Cluster Management for Kubernetes

Red Hat Advanced Cluster Management for Kubernetes provides multi cluster and application control, along with built-in security policies.

By using Red Hat Advanced Cluster Management for Kubernetes, administrators can perform the following tasks:

- Create, update, or delete RHOCP or Kubernetes clusters across multiple private and public clouds.
- Find and modify any Kubernetes resource across the entire domain using the built-in search engine.
- Troubleshoot and resolve issues using the built-in search engine.
- Automate and apply tasks in the clusters through the integration with Red Hat Ansible Automation Platform.
- Enforce compliance policies across the fleet of managed clusters.
- Manage lifecycle of applications across datacenters and hybrid clouds.
- Visualize cluster metrics from a centralized monitoring stack.
- Create and manage alerts generated across all the managed clusters.

Red Hat Advanced Cluster Security for Kubernetes

Powered by StackRox technology, Red Hat Advanced Cluster Security for Kubernetes helps administrators to enforce DevOps and security best practices.

By using Red Hat Advanced Cluster Security for Kubernetes, administrators get the following advantages:

- Visibility – Centralized view of your deployments, traffic in all clusters, and critical system level events in each running container.
- Vulnerability management – Image vulnerability scanning, vulnerability correlation to running deployments, and policy enforcing at build, deploy, and run time.
- Compliance – Assessment for CIS Benchmarks, payment card industry (PCI), Health Insurance Portability and Accountability Act (HIPAA), and NIST SP 800-190. Centralized compliance dashboard with evidence export for auditors, and detailed view compliance to pinpoint specific clusters, nodes, or namespaces.
- Network segmentation – Visualization of allowed and active traffic, simulation of network policy changes, network policy recommendations, and network enforcement capabilities.
- Risk profiling – Deployment raking based on security risk calculation, and security tracking to validate changes in configuration.
- Configuration management – Pre-built DevOps and security policies, Kubernetes role-based access control (RBAC) analysis, audit control for access to Kubernetes Secrets, and configuration policies at build time for CI/CD integration and deploy time.
- Runtime detection and response – System level event monitoring, automatic allowing of process activity, pre-built policies to detect crypto mining, privilege escalation, and various exploits, and system level data collection using external Berkeley Packet Finder (eBPF) or other Linux kernel modules.
- Integration – API and prebuilt plugins to integrate with DevOps systems, CI/CD tools, image scanners, registries, container runtimes, security integration event management (SIEM) solutions, and notification tools.

Red Hat Quay

Red Hat Quay is a scalable container registry to serve as a single source of truth for container images. Administrators can use Red Hat Quay to automate container builds with integration with GitHub, Bitbucket, and more. Red Hat Quay also provides vulnerability scanning for the container images.

By using Red Hat Quay, administrators get the following advantages:

- Time machine, with the ability to rollback image changes to a previous state.
- Geographic replication, for improved performance and availability.
- Continuous garbage collection, removing unused images automatically.
- Advanced access control management, including support for mapping teams and organizations integrating with an existing identity infrastructure.
- TLS security encryption between Red Hat Quay and your servers.
- Security scanning integration with vulnerability detectors like Clair [https://www.redhat.com/en/topics/containers/what-is-clair].

- Notifications for alerting about detected vulnerabilities.
- Integrating with CI/CD pipelines using build triggers, git hooks, and robot accounts.
- Auditing of CI and API actions.
- Integrated prometheus metrics export.

Challenges Addressed by Red Hat OpenShift Platform Plus

The following table relates the challenges of multicluster architectures, as discussed in the *Challenges of Multicluster Architectures* section, to the Red Hat OpenShift Platform Plus software collection.

Challenges Introduced by Multicluster Architectures	Red Hat OpenShift Platform Plus software addressing it
Easily provisioning clusters on all types of clouds	Red Hat Advanced Cluster Management for Kubernetes
Locating information about objects present in a fleet of clusters	Red Hat Advanced Cluster Management for Kubernetes search engine
Managing configuration compliance and the fleet of cluster's security	Red Hat Advanced Cluster Management for Kubernetes governance engines
Monitoring the behavior of all clusters and their workloads from performance and scalability points of view	Red Hat Advanced Cluster Management for Kubernetes observability engine
Requires a containers orchestration system	Red Hat OpenShift Container Platform or Kubernetes
Managing communications between services provided by containers, even when they are in different clusters or clouds	Red Hat Advanced Cluster Management for Kubernetes Submariner services
Requires a controlled place to store the artifacts and images that containers use	Red Hat Quay
Security scanning of the artifacts and images that containers use	Red Hat Quay for static scanning and Red Hat Advanced Cluster Security for Kubernetes for dynamic scanning
Automation processes	Red Hat Advanced Cluster Management for Kubernetes and its integration with the Ansible Automation Platform Resource Operator from Red Hat OpenShift Container Platform
An easy way for developers to implement CI/CD across a fleet of clusters	Red Hat Advanced Cluster Management for Kubernetes applications engine
An easy way for DevOps teams to implement GitOps across a fleet of clusters	Red Hat Advanced Cluster Management for Kubernetes applications engine

References

Red Hat OpenShift Container Platform
https://www.redhat.com/en/technologies/cloud-computing/openshift/container-platform

Red Hat OpenShift Platform Plus
https://www.redhat.com/en/technologies/cloud-computing/openshift/platform-plus

Red Hat Advanced Cluster Management for Kubernetes
https://www.redhat.com/en/technologies/management/advanced-cluster-management

Red Hat Advanced Cluster Security for Kubernetes
https://www.redhat.com/en/resources/advanced-cluster-security-for-kubernetes-datasheet

Red Hat Quay
https://www.redhat.com/en/technologies/cloud-computing/quay

Red Hat Quay datasheet: Private container registry
https://www.redhat.com/en/resources/quay-datasheet

Quiz

Describing Red Hat OpenShift Platform Plus

Choose the correct answers to the following questions:

1. **Which three of the following products include Red Hat OpenShift Platform Plus? (Choose three.)**
   a. Red Hat Advanced Cluster Management for Kubernetes
   b. Red Hat Satellite
   c. Red Hat Quay
   d. Red Hat Advanced Cluster Security for Kubernetes
   e. Red Hat Identity Management

2. **Which three of the following tasks can administrators perform using Red Hat Advanced Cluster Management for Kubernetes? (Choose three.)**
   a. Create, update, or delete RHOCP clusters across multiple private and public clouds.
   b. Create virtual networks to connect clusters across multiple private and public clouds.
   c. Enforce compliance policies across a fleet of managed clusters.
   d. Create custom ISO files to deploy Red Hat Enterprise Linux CoreOS (RHCOS).
   e. Visualize cluster metrics from a centralized monitoring stack.

3. **Which three of the following advantages arise from using Red Hat Quay? (Choose three.)**
   a. Integrate with CI/CD pipelines using build triggers, git hooks, and robot accounts.
   b. Preserve a private source code control system for infrastructure as a service (IaaS) repositories.
   c. Rollback container image changes to a previous state.
   d. Receive alerts about detected vulnerabilities in the hosted container images.
   e. Deploy RHOCP clusters from the Red Hat Quay dashboard.

4. **Which four of the following advantages arise from using Red Hat Advanced Cluster Security for Kubernetes? (Choose four.)**
   a. Assessment for CIS benchmarks
   b. Simulation of changes in the network policies
   c. Configuration policies at build time for CI/CD integration
   d. Automatic application updates to avoid recent vulnerabilities
   e. Centralized vulnerability management with correlation to running deployments
   f. Automatic cluster updates to avoid recent vulnerabilities

Solution

Describing Red Hat OpenShift Platform Plus

Choose the correct answers to the following questions:

1. **Which three of the following products include Red Hat OpenShift Platform Plus? (Choose three.)**
   a. Red Hat Advanced Cluster Management for Kubernetes
   b. Red Hat Satellite
   c. Red Hat Quay
   d. Red Hat Advanced Cluster Security for Kubernetes
   e. Red Hat Identity Management

2. **Which three of the following tasks can administrators perform using Red Hat Advanced Cluster Management for Kubernetes? (Choose three.)**
   a. Create, update, or delete RHOCP clusters across multiple private and public clouds.
   b. Create virtual networks to connect clusters across multiple private and public clouds.
   c. Enforce compliance policies across a fleet of managed clusters.
   d. Create custom ISO files to deploy Red Hat Enterprise Linux CoreOS (RHCOS).
   e. Visualize cluster metrics from a centralized monitoring stack.

3. **Which three of the following advantages arise from using Red Hat Quay? (Choose three.)**
   a. Integrate with CI/CD pipelines using build triggers, git hooks, and robot accounts.
   b. Preserve a private source code control system for infrastructure as a service (IaaS) repositories.
   c. Rollback container image changes to a previous state.
   d. Receive alerts about detected vulnerabilities in the hosted container images.
   e. Deploy RHOCP clusters from the Red Hat Quay dashboard.

4. **Which four of the following advantages arise from using Red Hat Advanced Cluster Security for Kubernetes? (Choose four.)**
   a. Assessment for CIS benchmarks
   b. Simulation of changes in the network policies
   c. Configuration policies at build time for CI/CD integration
   d. Automatic application updates to avoid recent vulnerabilities
   e. Centralized vulnerability management with correlation to running deployments
   f. Automatic cluster updates to avoid recent vulnerabilities

Deploying Red Hat Advanced Cluster Management for Kubernetes

Objectives

After completing this section, you should be able to describe and deploy the RHACM operator from the OperatorHub in an OpenShift hub cluster.

Introducing Red Hat Advanced Cluster Management for Kubernetes

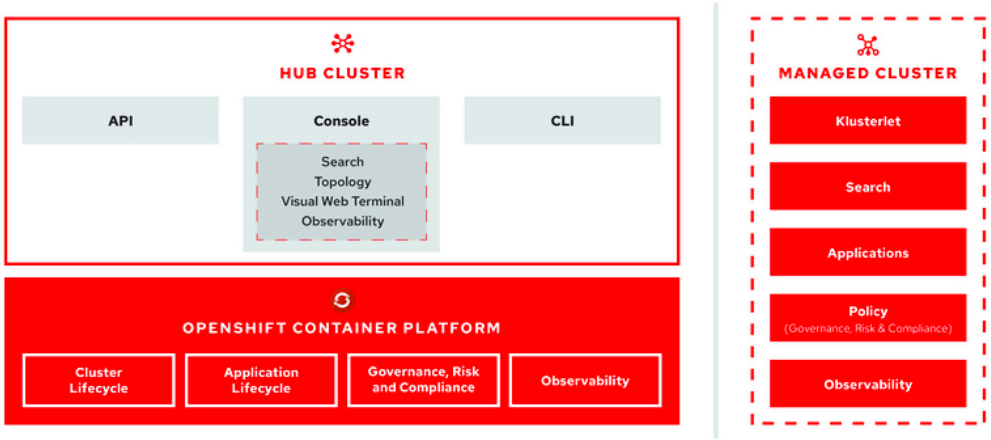
Red Hat Advanced Cluster Management for Kubernetes (RHACM) provides visibility of your OpenShift Container Platform and Kubernetes clusters from a view. RHACM provides built-in governance, cluster lifecycle management, and application lifecycle management, and includes observability features. RHACM is cloud-native by design. It runs on containers, does not use relational persistence, and manages Kubernetes objects directly through their APIs.

The core components of RHACM run in the hub cluster, the central controller that runs the RHACM management APIs, the RHACM web console, and an integrated command-line tool. The `MultiClusterHub` operator, part of the RHACM operator installation, is responsible of managing, upgrading, and installing hub cluster components.

In RHACM, the local cluster is the RHOCP cluster hosting the hub cluster. The hub cluster is also a managed cluster itself unless specified otherwise during the creation of the `MultiClusterHub` object. A managed cluster is an additional cluster managed by the hub cluster.

The agent controlling the connection between clusters is the `klusterlet`. The hub cluster can control managed clusters communicating with the `klusterlet` running in them. The operator creates all the necessary custom resource definitions (CRDs) for the RHACM components.

The following diagram provides a high level overview of an RHACM deployment on RHOCP.



As displayed in the previous diagram, there are three ways to interact with RHACM:

- The RHACM API

- The RHACM web console
- CLI tools such as `oc` and `kubectl`

The web console provides visual, simplified access to the search engine, topology view, visual web terminal, and the observability console among other features.

Apart from the core components running in the hub cluster, every managed cluster runs agents to communicate with the hub cluster. These agents, also called `addons` are:

- the klusterlet
- the search engine addon
- the addon for managing application deployments from RHACM
- the addon for applying policies
- the addon for the observability components

RHACM leverages capabilities from the RHOCP local cluster for cluster lifecycle management, application lifecycle management, applying governance, risk, and compliance policies, and the prometheus and grafana instances running on it for observability.

Installing RHACM

Red Hat recommends administrators to install the Advanced Cluster Management for Kubernetes operator from the **OperatorHub** menu in the RHCOP web console.

Note
A list of supported hub cluster providers is available in the RHACM documentation [https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/supported-clouds#supported-hub-cluster-providers].

The installation requires selecting an **Update channel**. An update channel is used to deliver streams of updates to the operators. Typically, update channels correspond to minor software versions. The installation also requires to choose an **Update approval** strategy, between automatic or manual. Choosing the manual update approval approach requires human intervention to update when a new version of the operator is available in the update stream. Alternatively, the automatic update approval will trigger an update when a new operator version is available.

After the operator is installed, administrators must create the `MultiClusterHub` custom resource to deploy all the RHACM components. You will be required to perform this steps elsewhere in this course.

Administrators can also install the RHACM operator using the CLI. Both installation methods can be performed in connected and disconnected environments.

Note
For more information about all the available installation methods, check the *Red Hat Advanced Cluster Management for Kubernetes Install Guide* at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/install/index

When the `MultiClusterHub` finishes deploying, it creates the `multicloud-console` route to access the RHACM web console. Administrators can find that route using the RHOCP web console, in the **Networking** → **Routes** menu. Alternatively, administrators can use the following command:

```
[user@demo ~]$ oc get routes -n open-cluster-management
NAME                HOST/PORT          ...
multicloud-console  multicloud-console.apps.ocp4.example.com
```

Importing Existing Clusters to RHACM

When the `MultiClusterHub` custom resource installation completes, the next step is to aggregate or import more Kubernetes or RHOCP clusters into RHACM.

Note

A list of supported managed cluster providers is available in the RHACM documentation [https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/supported-clouds#supported-managed-cluster-providers].

The local cluster (the RHOCP cluster where the hub cluster runs) is automatically imported into RHACM, unless specified during the creation of the `MultiClusterHub` object.

There are two ways for importing new clusters: from the web console, and using the CLI.

The following prerequisites are common for both import methods:

1. Internet connectivity between the hub and the managed cluster
2. The `oc` or `kubectl` CLI tools
3. The `base64` utility
4. In case of importing a Kubernetes cluster, a `multiclusterhub.spec.imagePullSecret` defined containing the `cloud.redhat.com` pull secret
5. In case of importing a Red Hat OpenShift Dedicated cluster, the hub cluster running in it, and `cluster-admin` permissions to use RHACM.

Importing Existing Clusters Using the RHACM Web Console

To import an existing cluster from the RHACM web console, administrators must follow this steps:

1. From the RHACM web console, navigate to the **Infrastructure** → **Clusters** menu.
2. Click **Add a cluster**.
3. Click **Import an existing cluster**. Provide a name for it.
4. Add any desired **Additional labels** (optional).
5. Click **Save import and generate code**.
6. Click **Copy command** to copy the generated command and token to the clipboard.
7. Use the `oc` CLI tool to log into the cluster you want to import.

8. Paste and run the copied command.

9. Return to the **Infrastructure** → **Clusters** menu.

The newly imported cluster appears immediately in the **Infrastructure** → **Clusters** menu. It takes a while to see all the cluster details. This is because RHACM is installing all the add-ons in the managed cluster.

Note
For detailed steps and expanded information, review the *Chapter 8. Importing a target managed cluster to the hub cluster* section of the *Red Hat Advanced Cluster Management for Kubernetes Clusters Guide* at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/index#importing-a-target-managed-cluster-to-the-hub-cluster

Importing Existing Clusters Using the CLI

The steps for importing an existing cluster using the CLI are different from the ones using the web console.

The following steps are a summary of the actions to be performed in each cluster, for a better understanding of the process.

Note
For detailed instructions on how to import clusters into RHACM using the CLI, visit https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/index#importing-a-managed-cluster-with-the-cli

In the hub cluster:

1. Create a project with the name of the cluster to import, for instance: `imported-cluster`

2. Add the label `cluster.open-cluster-management.io/managedCluster=`*`imported-cluster`* to the `imported-cluster` namespace

3. Create a new `ManagedCluster` CR with the name `imported-cluster`

4. Create a new `KlusterletAddonConfig` CR with klusterlet configuration file.

In the hub cluster, administrators must extract existing secrets and generate YAML files containing:

1. The klusterlet CRD. For instance in a file named `klusterlet-crd.yaml`.

2. The secret for importing new clusters. For instance in a file named `import.yaml`.

Then, in the managed cluster to import, administrators must use the YAML files generated in the previous step to create the necessary objects.

1. `oc create -f klusterlet-crd.yaml`

2. `oc create -f import.yaml`

Finally, validate the status of the imported cluster.

In the hub cluster, validate the `JOINED` and `AVAILABLE` status of the `managedcluster` named `imported-cluster`.

In the target cluster, validate the status of the pods of all the addons running on the target cluster namespace `open-cluster-management-agent-addon`.

Removing Imported Clusters from RHACM

To remove an imported cluster from the RHACM console, navigate to the **Clusters** page and clik **Actions** → **Detach cluster** from the options of the cluster to remove.

To remove an imported cluster using the CLI, log in to the hub cluster, locate the correspondent `managedcluster` object and use `oc` to delete it. Because RHACM is removing all the add-ons and other RHACM components from the managed cluster, the detach process takes a few minutes.

References

For more information, refer to the *About* guide in the *Red Hat Advanced Cluster Management for Kubernetes* documentation at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/about/index

For more information, refer to the *Install* guide in the *Red Hat Advanced Cluster Management for Kubernetes* documentation at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/install/index

For more information, refer to the *Clusters* guide in the *Red Hat Advanced Cluster Management for Kubernetes* documentation at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/index

Guided Exercise

Deploying Red Hat Advanced Cluster Management for Kubernetes

In this exercise you will install the RHACM operator and import an existing cluster.

Then, you will remove the imported cluster and uninstall the RHACM operator and all its components.

Outcomes

You should be able to:

- Install the RHACM operator from OperatorHub
- Create the `MultiClusterHub` object
- Import an existing cluster into RHACM
- Remove an imported cluster
- Uninstall the RHACM operator and all its components

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 RHACM operator is not already installed.

```
[student@workstation ~]$ lab start multicluster-acm
```

Note

The hub cluster must be installed in the cluster `ocp4.example.com`.

Instructions

1. Using OperatorHub, install the Advanced Cluster Management for Kubernetes operator in the `ocp4.example.com` cluster. The web console URL is `https://console-openshift-console.apps.ocp4.example.com`.

   1.1. From the `workstation` machine, use Firefox to navigate to the RHOCP 4 web console at `https://console-openshift-console.apps.ocp4.example.com`.

   When prompted, select **htpasswd_provider**

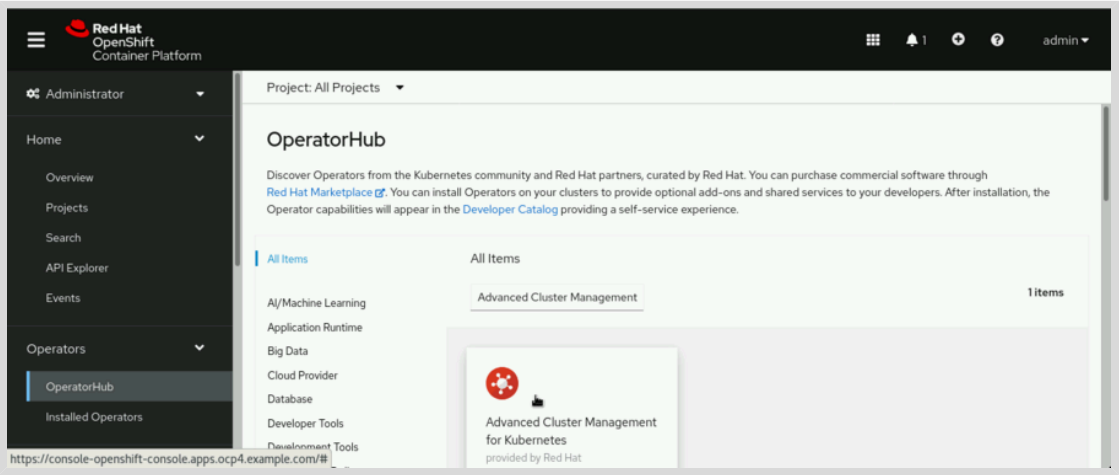
   Type the credentials:

   - Username: `admin`
   - Password: `redhat`

Click **Log in**.

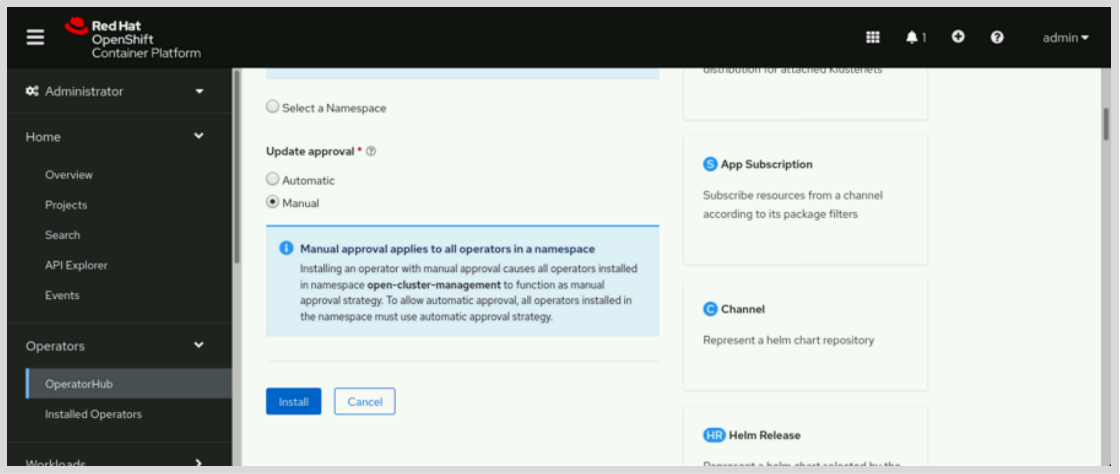
1.2. Install the Advanced Cluster Management for Kubernetes operator from OperatorHub.

Navigate to **Operators** → **OperatorHub** and type `Advanced Cluster Management` in the **Filter by keyword** text field.



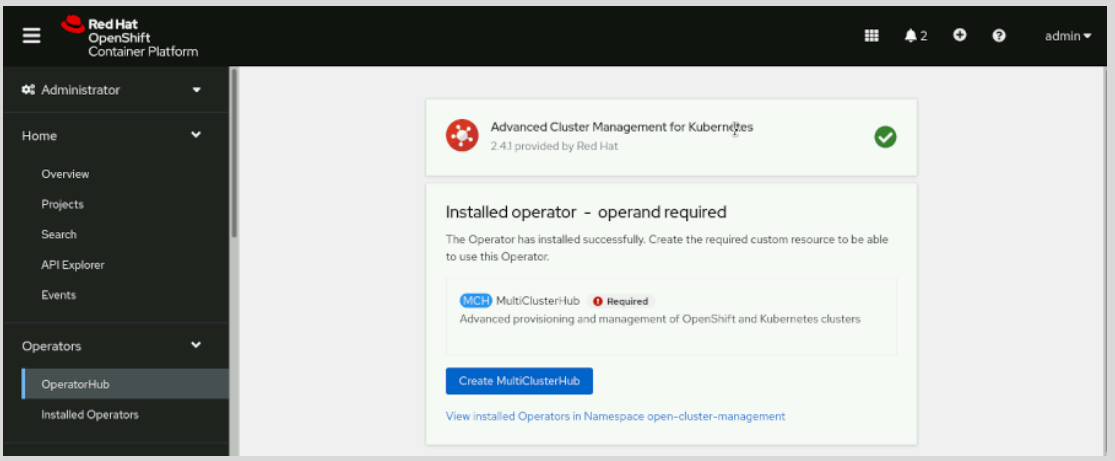
Click **Advanced Cluster Management for Kubernetes**, and then click **Install**.

In the **Update Channel**, ensure that the `release-2.4` radio button is selected. Select **Manual** as the Approval strategy. Then, click **Install**.



Next, you must approve the installation or updates to the RHACM operator manually. Click **Approve** in the next step. The installation can take a few minutes to complete.

When the operator is installed, you see the following message:



Click **Create MulticlusterHub**.

On the `Create MulticlusterHub` page, leave the default values and click **Create**. You will be redirected to the `MultiClusterHubs` tab.

Initially, the `Status` of the `multiclusterhub` object will be `Phase: Installing`.

After some minutes, the `Status` column will show `Phase: Running`.

2. Verify the deployment of the `MultiClusterHub`.

2.1. Open a terminal and log in to the `ocp4.example.com` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

2.2. Check the status of all the objects in the `open-cluster-management` namespace.

```
[student@workstation ~]$ oc get all -n open-cluster-management
...output omitted...
[student@workstation ~]$
```

The output is quite long. Review the status of the pods, services, deployments, replica sets, and stateful sets.

2.3. Retrieve the route to the RHACM web console, named `multicloud-console`.

```
[student@workstation ~]$ oc get route multicloud-console -n
 open-cluster-management
NAME                HOST/PORT      ...
multicloud-console  multicloud-console.apps.ocp4.example.com
```

Copy the URL `multicloud-console.apps.ocp4.example.com` for the next step.

3. Test the RHACM deployment by importing the `ocp4-mng.example.com` cluster from the RHACM web console.

3.1. In Firefox, open a new tab and log in to the RHACM web console with the URL `multicloud-console.apps.ocp4.example.com`.

When prompted, select **htpasswd_provider**.

Type the credentials:

- User name: `admin`
- Password: `redhat`

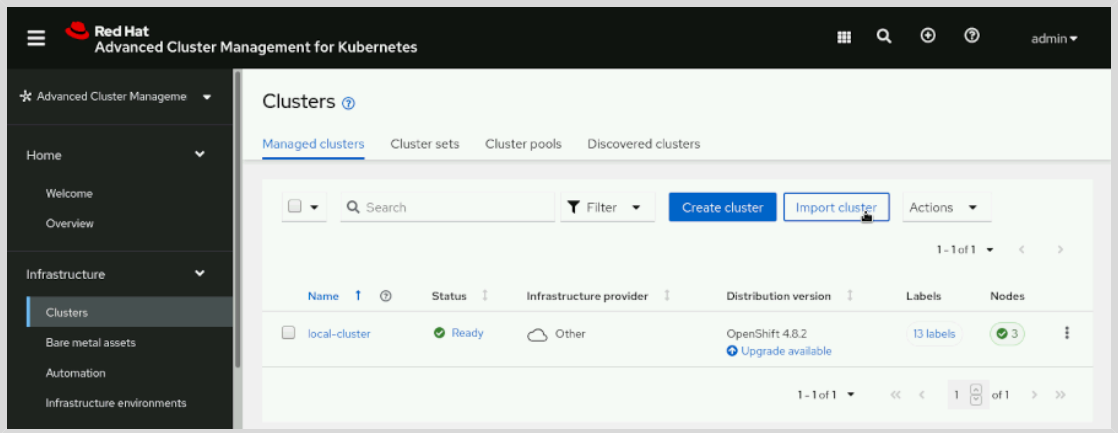
Click **Log in**.

3.2. Explore the **Infrastructure** → **Clusters** menu.

From the **Infrastructure** menu, click **Clusters**. Scroll down to locate the managed clusters. Notice that the `local-cluster` `ocp4.example.com` cluster, where the hub cluster runs, is automatically managed.

3.3. Generate the code to import an existing cluster.

Click **Import Cluster**



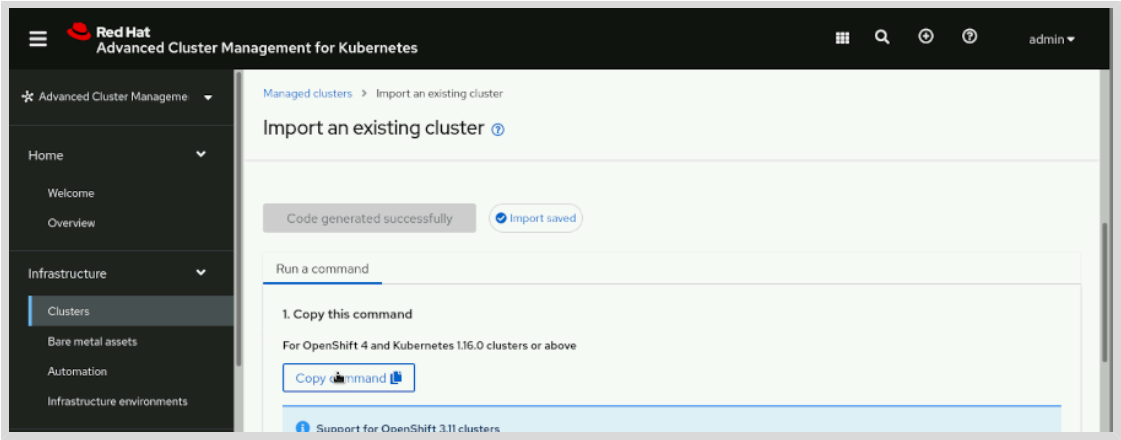
In the Import an existing cluster screen, indicate:

- **Name**: `managed-cluster`

Leave the rest of the values unchanged and click **Save import and generate code**.

The **Save import and generate code** button now displays the `Code generated successfully` message.

Click **Copy command**.



Note

The previous steps copies a command to the clipboard to be executed in a terminal after logging in the managed cluster `ocp4-mng.example.com`.

3.4. From the terminal, log in to the `ocp4-mng.example.com` cluster as the `admin` user. The APIServer URL is `https://api.ocp4-mng.example.com:6443`.

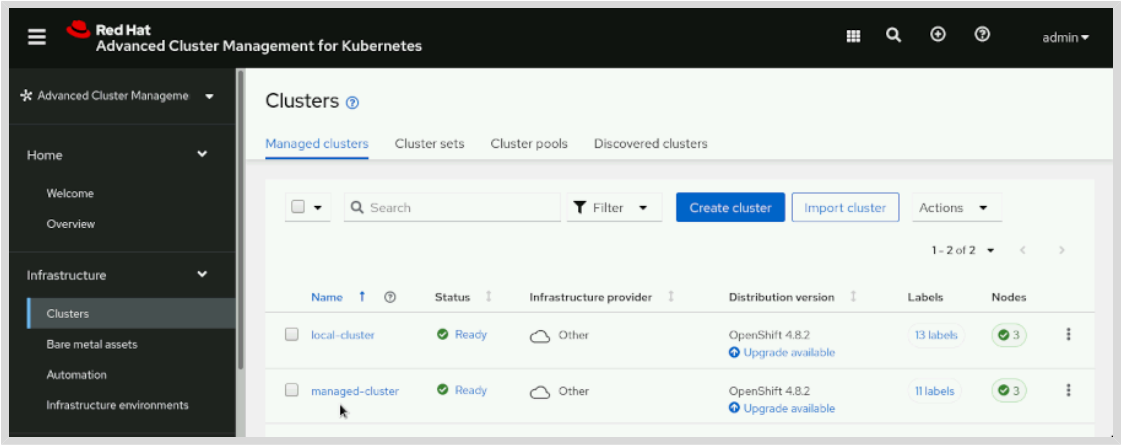
```
[student@workstation ~]$ oc login -u admin -p redhat https://api.ocp4-
mng.example.com:6443
Login successful.
...output omitted...
[student@workstation ~]$
```

3.5. Paste and run the import code in to the terminal and press `Enter`. You can use the keys `Ctrl+Shift+V` or your preferred method for pasting content from the clipboard. The paste command is quite long, and most of it is encoded in `base64`.

```
[student@workstation ~]$ echo "Ci0tLQph.....G9ydCBhZ2Fpbi4=" | base64 -d
customresourcedefinition.apiextensions.k8s.io/klusterlets.operator.open-cluster-
management.io created
namespace/open-cluster-management-agent created
serviceaccount/klusterlet created
secret/bootstrap-hub-kubeconfig created
clusterrole.rbac.authorization.k8s.io/klusterlet created
clusterrole.rbac.authorization.k8s.io/open-cluster-management:klusterlet-admin-
aggregate-clusterrole created
clusterrolebinding.rbac.authorization.k8s.io/klusterlet created
deployment.apps/klusterlet created
klusterlet.operator.open-cluster-management.io/klusterlet created
[student@workstation ~]$
```

3.6. Verify the import of the managed cluster.

Return to the RHACM web console in Firefox. Navigate to **Infrastructure** → **Clusters**. The `managed-cluster` is now listed in the **Managed clusters** list.

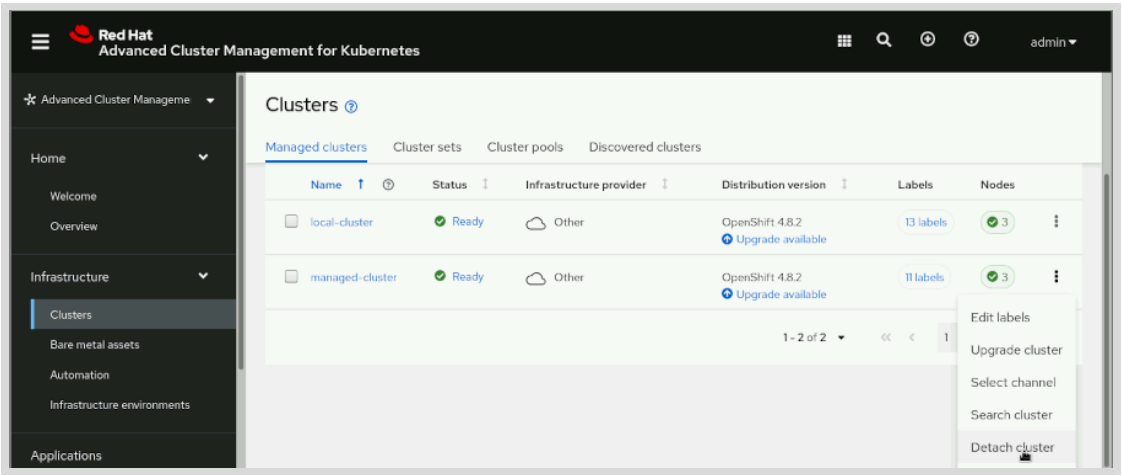


The `managed-cluster` appears in the RHACM web console almost immediately after executing the import code. Initially, you will see limited information about the cluster. After a few minutes, all the details of the cluster will become available.

4. Remove the imported cluster.

4.1. From the RHACM web console, detach the imported cluster.

In the **Infrastructure** → **Clusters** pane, click **Options** button on the right side of the `managed-cluster`. Then, click **Detach cluster**.



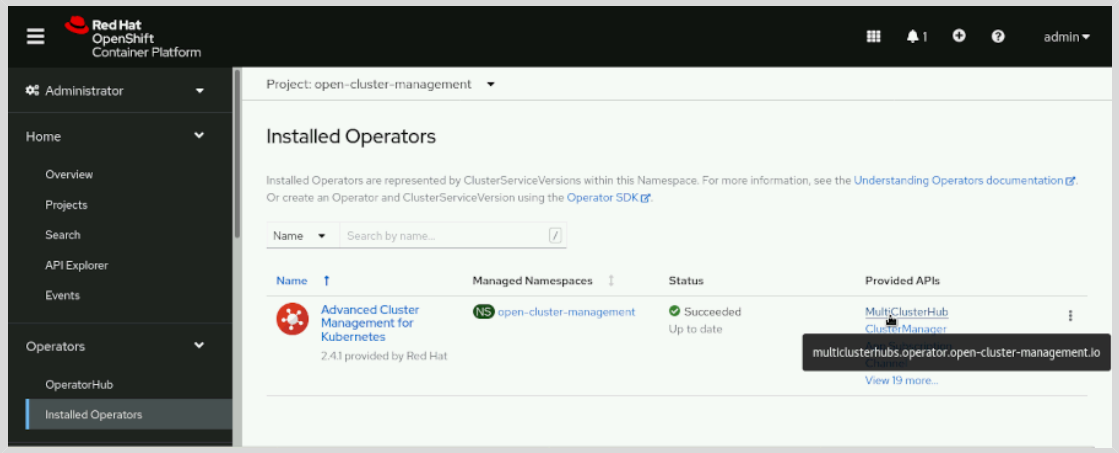
Then, type the cluster name `managed-cluster` in the text field and click **Detach**.

The `Status` column changes to `Detaching`. After a few minutes, the `managed-cluster` is completely detached.

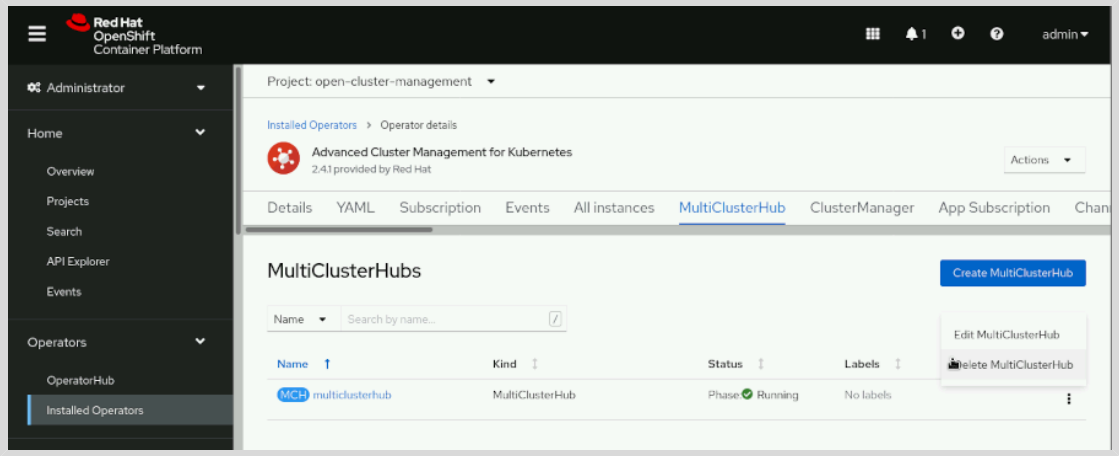
5. Uninstall the RHACM operator and all of its components.

5.1. Delete the `multiclusterhub` object.

From the RHOCP web console, navigate to **Operators** → **Installed Operators** and click **MultiClusterHub** in the `Provided APIs` column.



Click **Options** to the right side of the `multiclusterhub` object. Then, click **Delete MultiClusterHub**.



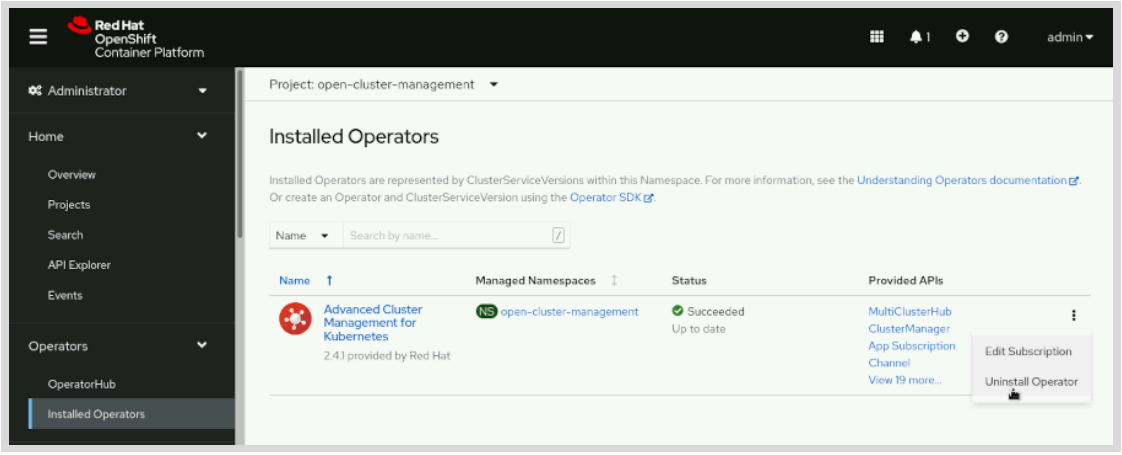
When prompted, click **Delete** to confirm. The `Status` column changes to `Phase: Uninstalling`. After a few minutes, the `multiclusterhub` object is completely removed and the page displays the `No operands found` message.

Note

It takes several minutes to remove the `multiclusterhub` object. Do not continue with the next step until the `multiclusterhub` object is completely removed.

5.2. Uninstall the Advanced Cluster Management for Kubernetes operator.

From the RHOCP web console, navigate to **Operators** → **Installed Operators**. Click **Options** on the right side of the `Advanced Cluster Management for Kubernetes` operator. Then, click **Uninstall Operator**.



When prompted, click **Uninstall** to confirm. After a few minutes, the operator is completely uninstalled and the page displays the `No Operators found` message.

5.3. Remove the `open-cluster-management` namespace.

From the terminal, run the following command to remove the namespace and any remaining objects in the `open-cluster-management` namespace:

```
[student@workstation ~]$ oc delete project open-cluster-management
project.project.openshift.io "open-cluster-management" deleted
[student@workstation ~]$
```

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 finish multicluster-acm
```

This concludes the guided exercise.

Lab

Installing the RHACM Operator

In this lab, you will install RHACM and import an existing cluster in the lab environment using the command-line interface.

Outcomes

You should be able to perform the following tasks by using the `oc` command-line tool:

- Install the RHACM operator.
- Create the `MultiClusterHub` object.
- Import an existing cluster into RHACM.

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 RHACM operator is not already installed.

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

Note

The hub cluster must be installed in the cluster `ocp4.example.com`.

The `managed-cluster` URL is `ocp4-mng.example.com`.

Instructions

1. Install the Advanced Cluster Management for Kubernetes operator on the `ocp4.example.com` cluster.
2. Create the RHACM `MultiClusterHub` object.
3. Prepare RHACM to import a cluster using the name `managed-cluster`.
4. Generate the files to import the `managed-cluster` into RHACM.
5. Import the RHCOP cluster, `ocp4-mng.example.com`, available in the lab environment into RHACM. Use `managed-cluster` as the name of the cluster to import.
6. Log back in to the hub cluster and verify the status of the `managed-cluster`.

Evaluation

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

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

Finish

Do not make any other changes to the lab environment until the next guided exercise. You will continue using this environment in upcoming exercises.

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise.

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

This concludes the lab.

Solution

Installing the RHACM Operator

In this lab, you will install RHACM and import an existing cluster in the lab environment using the command-line interface.

Outcomes

You should be able to perform the following tasks by using the `oc` command-line tool:

- Install the RHACM operator.
- Create the `MultiClusterHub` object.
- Import an existing cluster into RHACM.

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 RHACM operator is not already installed.

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

Note

The hub cluster must be installed in the cluster `ocp4.example.com`.

The `managed-cluster` URL is `ocp4-mng.example.com`.

Instructions

1. Install the Advanced Cluster Management for Kubernetes operator on the `ocp4.example.com` cluster.

1.1. From the `workstation` machine, open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

1.2. Create a project named `open-cluster-management`.

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

1.3. Create an operator group for the `open-cluster-management` namespace. First, create a file named `operator-group.yaml` with the following contents:

```
[student@workstation ~]$ vi operator-group.yaml
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: acm-operator-group
spec:
  targetNamespaces:
  - open-cluster-management
```

Save the changes and close the editor.

Then, create the `OperatorGroup` object by using the `oc` command.

```
[student@workstation ~]$ oc create -f operator-group.yaml
operatorgroup.operators.coreos.com/acm-operator-group created
[student@workstation ~]$
```

1.4. Create a subscription to the Advanced Cluster Management for Kubernetes operator.

First, create a file named `subscription.yaml` with the following contents:

```
[student@workstation ~]$ vi subscription.yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: acm-operator-subscription
spec:
  sourceNamespace: openshift-marketplace
  source: redhat-operators
  channel: release-2.4
  installPlanApproval: Manual
  name: advanced-cluster-management
```

Save the changes and close the editor.

Then, create the `Subscription` object using the `oc` command.

```
[student@workstation ~]$ oc create -f subscription.yaml
subscription.operators.coreos.com/acm-operator-subscription created
```

1.5. Approve the installation of the operator.

Because you created the subscription with the `Manual` approval strategy, you must approve the installation manually.

First, retrieve the installation plans.

```
[student@workstation ~]$ oc get installplan
NAME            CSV                                  APPROVAL   APPROVED
install-4k2q8   advanced-cluster-management.v2.4.1   Manual     false
```

As you can see in the preceding output, the installation is not approved. Approve the installation plan *install-4k2q8* by running the following command.

```
[student@workstation ~]$ oc patch installplan install-4k2q8 --type merge \
  --patch '{"spec":{"approved":true}}'
installplan.operators.coreos.com/install-4k2q8 patched
```

After the installation plan is approved, the cluster creates a `ClusterServiceVersion` object.

1.6. Watch the status of the available `ClusterServiceVersion` objects to verify the status of the installation. Wait until the PHASE column shows the message: `Succeeded`.

```
[student@workstation ~]$ watch oc get csv
...output omitted...
NAME                                 DISPLAY
 VERSION ... PHASE
advanced-cluster-management.v2.4.1   Advanced Cluster Management for Kubernetes
 2.4.1   ... Succeeded
```

Exit the watch command wby pressing `Ctrl+C`.

When the PHASE column shows the message: `Succeeded`, the installation of the operator is complete.

2. Create the RHACM `MultiClusterHub` object.

2.1. From the terminal, create a file named `mch.yaml` containing the definition of the `MultiClusterHub` object.

```
[student@workstation ~]$ vi mch.yaml
apiVersion: operator.open-cluster-management.io/v1
kind: MultiClusterHub
metadata:
  name: multiclusterhub
  namespace: open-cluster-management
spec: {}
```

Save the changes and close the editor.

2.2. Use the `oc` command to create the `MultiClusterHub` object from the `mch.yaml` file.

```
[student@workstation ~]$ oc create -f mch.yaml
multiclusterhub.operator.open-cluster-management.io/multiclusterhub created
```

Note
It takes around 5 minutes to create all the resources of the MultiClusterHub object.

2.3. Wait until the MultiClusterHub object creates all its components. Use the `watch` command to monitor the status.

```
[student@workstation ~]$ watch oc get multiclusterhub
...output omitted...
NAME              STATUS       AGE
multiclusterhub   Installing   3m21s
```

When the STATUS column displays `Running`, exit the watch command with `Ctrl+C`.

The creation of the `MultiClusterHub` object is complete.

3. Prepare RHACM to import a cluster using the name `managed-cluster`.

3.1. In the hub cluster, create the `managed-cluster` namespace.

```
[student@workstation ~]$ oc new-project managed-cluster
...output omitted...
```

3.2. Label the namespace with the cluster name to be used by RHACM.

```
[student@workstation ~]$ oc label namespace managed-cluster \
  cluster.open-cluster-management.io/manage dCluster=managed-cluster
namespace/managed-cluster labeled
...output omitted...
```

3.3. Create a file named `mngcluster.yaml` with the following contents:

```
[student@workstation ~]$ vi mngcluster.yaml
apiVersion: cluster.open-cluster-management.io/v1
kind: ManagedCluster
metadata:
  name: managed-cluster
spec:
  hubAcceptsClient: true
```

Save the changes and close the editor.

3.4. Create the `ManagedCluster` object using the previous file.

```
[student@workstation ~]$ oc create -f mngcluster.yaml
managedcluster.cluster.open-cluster-management.io/managed-cluster created
```

3.5. Create a `klusterlet` add-on configuration file named `klusterlet.yaml` with the following contents:.

```
[student@workstation ~]$ vi klusterlet.yaml
apiVersion: agent.open-cluster-management.io/v1
kind: KlusterletAddonConfig
metadata:
  name: managed-cluster
  namespace: managed-cluster
spec:
  clusterName: managed-cluster
  clusterNamespace: managed-cluster
  applicationManager:
    enabled: true
  certPolicyController:
    enabled: true
  clusterLabels:
    cloud: auto-detect
    vendor: auto-detect
  iamPolicyController:
    enabled: true
  policyController:
    enabled: true
  searchCollector:
    enabled: true
  version: 2.4.0
```

Save the changes and close the editor.

3.6. Use the `klusterlet.yaml` file to create the `klusterlet` configuration in the hub cluster.

```
[student@workstation ~]$ oc create -f klusterlet.yaml
klusterletaddonconfig.agent.open-cluster-management.io/managed-cluster created
```

At this point, the `ManagedCluster-Import-Controller` will generate a secret named managed-cluster-import. This secret contains the `import.yaml` file containing the secret to be used in the cluster to import.

4. Generate the files to import the `managed-cluster` into RHACM.

4.1. Obtain the necessary files to import the `managed-cluster` cluster.

First, run the following command to obtain the `klusterlet-crd.yaml` file. This file is used to create the `klusterlet` in the `managed-cluster`.

```
[student@workstation ~]$ oc get secret managed-cluster-import -n managed-cluster \
  -o jsonpath={.data.c rds\\.yaml} | base64 --decode > klusterlet-crd.yaml
```

Second, obtain the `import.yaml` file by running the following command:

```
[student@workstation ~]$ oc get secret managed-cluster-import -n managed-cluster \
  -o jsonpath={.data.i mport\\.yaml} | base64 --decode > import.yaml
```

This file contains the necessary secrets to import the `managed-cluster` into RHACM.

5. Import the RHCOP cluster, `ocp4-mng.example.com`, available in the lab environment into RHACM. Use `managed-cluster` as the name of the cluster to import.

5.1. Use the terminal to log in to the `ocp4-mng` cluster as the `admin` user. The APIServer URL is `https://api.ocp4-mng.example.com:6443`.

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

5.2. Use the `klusterlet-crd.yaml` file to create the `klusterlet` custom resource definition.

```
[student@workstation ~]$ oc create -f klusterlet-crd.yaml
customresourcedefinition.apiextensions.k8s.io/klusterlets.operator.open-cluster-
management.io created
```

5.3. Now, use the `import.yaml` file to create the rest of the resources necessary to import the cluster into RHACM.

```
[student@workstation ~]$ oc create -f import.yaml
namespace/open-cluster-management-agent created
serviceaccount/klusterlet created
secret/bootstrap-hub-kubeconfig created
clusterrole.rbac.authorization.k8s.io/klusterlet created
clusterrole.rbac.authorization.k8s.io/open-cluster-management:klusterlet-admin-
aggregate-clusterrole created
clusterrolebinding.rbac.authorization.k8s.io/klusterlet created
deployment.apps/klusterlet created
klusterlet.operator.open-cluster-management.io/klusterlet created
```

5.4. Verify the status of the pods running in the `open-cluster-management-agent` namespace.

```
[student@workstation ~]$ oc get pod -n open-cluster-management-agent
NAME                                          READY   STATUS    RESTARTS   AGE
klusterlet-bfb4cd68f-j2fdt                    1/1     Running   0          53s
klusterlet-registration-agent-5f749f5cf9-8xvp6   1/1     Running   0          36s
klusterlet-registration-agent-5f749f5cf9-pnkj6   1/1     Running   0          36s
klusterlet-registration-agent-5f749f5cf9-slk8j   1/1     Running   0          36s
klusterlet-work-agent-7d848f496b-7mnfw        1/1     Running   0          36s
klusterlet-work-agent-7d848f496b-bwkfv        1/1     Running   1          36s
klusterlet-work-agent-7d848f496b-wn2vd        1/1     Running   0          36s
```

5.5. Finally, use the `watch` command to validate the status of the agent pods running in the `open-cluster-management-agent-addon` namespace.

```
[student@workstation ~]$ watch oc get pod -n open-cluster-management-agent-addon
NAME                                                READY   STATUS
 RESTARTS   AGE
klusterlet-addon-appmgr-7f69b84c76-kvhp7            1/1     Running   0
        55s
klusterlet-addon-certpolicyctrl-7c97656db8-5s6xk    1/1     Running   0
        54s
```

```
klusterlet-addon-iampolicyctrl-6f8cccf86c-lj2fg              1/1     Running   0
        54s
klusterlet-addon-operator-b479bb446-kpc6t                    1/1     Running   0
        83s
klusterlet-addon-policyctrl-config-policy-769745c7b6-78sw2   1/1     Running   0
        54s
klusterlet-addon-policyctrl-framework-6455bc9558-bz8mc       3/3     Running   0
        54s
klusterlet-addon-search-7b5bb78f98-h2jlg                     1/1     Running   0
        53s
klusterlet-addon-workmgr-5b84cfd6dc-nxwvt                    1/1     Running   0
        52s
```

When all the pods are ready, press `Ctrl+C` to exit the watch command.

6. Log back in to the hub cluster and verify the status of the `managed-cluster`.

6.1. Log back in to the hub cluster.

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

6.2. Verify the status of the `managed-cluster`.

```
[student@workstation ~]$ oc get managedcluster
NAME              HUB ACCEPTED   ...   JOINED   AVAILABLE   AGE
local-cluster     true           ...   True     True        4h26m
managed-cluster   true           ...   True     True        42m
```

The `JOINED` and `AVAILABLE` status must be `True`.

Evaluation

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

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

Finish

Do not make any other changes to the lab environment until the next guided exercise. You will continue using this environment in upcoming exercises.

As the `student` user on the `workstation` machine, use the `lab` command to complete this exercise.

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

This concludes the lab.

Summary

In this chapter, you learned:

- About the challenges of hybrid cloud environments and multicluster architectures.
- How the Red Hat OpenShift Platform Plus tools help to address the challenges of multicluster architectures.
- How to install RHACM in a hub cluster.
- How to import a managed cluster to RHACM.

Chapter 2

Inspecting Resources from Multiple Clusters Using the RHACM Web Console

Goal

Describe and navigate the RHACM web console. Configure RBAC and search resources across multiple clusters by using the RHACM search engine.

Objectives

- Locate objects across a fleet of managed clusters by using the search engine and enumerate RHACM features through the web console.
- Create different user roles in RHACM and define an authentication model for multicluster management.

Sections

- Navigating the Red Hat Advanced Cluster Management Web Console (and Guided Exercise)
- Configuring Access Control for Multicluster Management (and Guided Exercise)

Lab

Inspecting Resources from Multiple Clusters Using the RHACM Web Console

Navigating the Red Hat Advanced Cluster Management Web Console

Objectives

After completing this section, you should be able to locate objects across a fleet of managed clusters by using the search engine and enumerate RHACM features through the web console.

RHACM Web Console Overview

The RHACM web console represents a single point for access to the management and observation of a fleet of Kubernetes or OpenShift clusters.

The following sections describe the different components of RHACM web console.

Home

The **Home** pane provides information about RHACM and its use cases, along with links to the main product features. The **Home** pane includes the following submenus:

Welcome
Provides information and links to access the main RHACM features.

Overview
Provides a summary and a high-level overview of the details and status of the managed clusters.

Infrastructure

You can use the **Infrastructure** pane to access cluster lifecycle management, bare metal assets management, Ansible automation configuration, and infrastructure environments management. The **Infrastructure** pane includes the following submenus:

Clusters
The following list shows some of the features about clusters that you can use from the **Clusters** pane:

- Creating or upgrading a cluster in many different public cloud providers.
- Importing an existing cluster to the RHACM hub cluster, to manage it.
- Scaling manually the clusters, or enabling autoscaling. The proccess of resizing a cluster is different if you created the cluster by using RHACM, or if the cluster already existed and you imported it to RHACM.
- Using RHACM features such as cluster sets, cluster pools, and discover clusters.

Note

A list of supported hub cluster providers is available in the RHACM documentation [https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/supported-clouds#supported-hub-cluster-providers].

A list of supported managed cluster providers is available in the RHACM documentation [https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/index#supported-managed-cluster-providers].

Bare metal assets

In RHACM. a bare metal asset is a collection of physical or virtual servers that you can configure to run your OpenShift clusters. RHACM uses the Intelligent Platform Management Interface (IPMI) set of specifications to interacting with the bare metal assets by using their Baseboard Management Controller (BMC) microcontrollers. The **Bare metal assets** pane integrates bare metal assets into RHACM via their BMC address. RHACM uses the integration of bare metal assets to deploy and manage clusters hosted the virtual or physical infrastructure.

Automation

In the **Automation** pane, you can create Ansible job templates and run Ansible jobs automatically during different stages of a cluster lifecycle. To create Ansible templates you need the Ansible Automation Platform Resource Operator installed on the RHACM hub cluster. You will also need an Ansible Tower 3.7.3 or a later version available.

Infrastructure environments

In RHACM, an infrastructure environment is a pool of resources that allows you to create hosts, and create clusters on those hosts. The main component for managing infrastructure environments is the Central Infrastructure Management (CIM) service, that you need to enable. In the **Infrastructure environments** pane, you can create infrastructure environments and access to them to add hosts.

Applications

You can use the **Applications** pane to create, deploy, and manage applications across the fleet of clusters.

You can find more information in the *Introducing the RHACM Application Model* chapter elsewhere in this Course.

Governance

Through the **Governance** pane, you can create and manage policies and policy controllers, and apply those to the fleet of clusters.

You can find more information in the *Deploying the Compliance Operator Across the Fleet of Clusters Using the RHACM Compliance Operator Policy* chapter elsewhere in this Course.

Credentials

In RHACM, a credential stores the access information for a cloud provider. Kubernetes secrets is the way to store credentials. Each credential has 2 keys, the cloud provider access information, and

a DNS name within that cloud provider. The following is the list of types of credentials that RHACM uses:

- Cloud provider credentials: for example, Amazon Web Services, Google Cloud Platform, or Microsoft Azure
- Data center credentials: for example, Red Hat OpenStack Platform or bare metal resources
- Automation and other credentials: for example, for access to Red Hat Ansible Automation Platform
- Centrally managed: type of credential for on premises environments

You can use the **Credentials** pane to create and administer credentials for all different cloud providers and systems.

RHACM Search Engine

The RHACM search engine provides visibility of the Kubernetes objects across the fleet of clusters from a single user interface.

The RHACM search engine is always enabled, and you can access to it through the magnifying glass icon in upper area of RHACM web console.

The function of the search engine is to index and store the Kubernetes objects present in the fleet of clusters, and calculates the relationships with other objects.

The RHACM search engine architecture is composed of the following components:

- Collector – It is deployed in each of the clusters of the fleet. In the hub cluster the search collector is deployed in the `open-cluster-management` namespace. In the rest of the managed clusters the collector is deployed in the `open-cluster-management-agent-addon` namespace, as part of the `search-collector` add-on. The collector indexes the information of the Kubernetes objects and computes relationships for objects within the managed clusters.

- Aggregator – It is deployed only in the RHACM hub cluster, in the `open-cluster-management` namespace. The function of the aggregator is to collect the data from multiple collectors in different managed clusters. Then the data is wrote to a Redis database present in the `search-redisgraph` stateful set. The aggregator also computes relationships between objects of different clusters, and tracks the activity from the collectors that send data from the managed clusters.

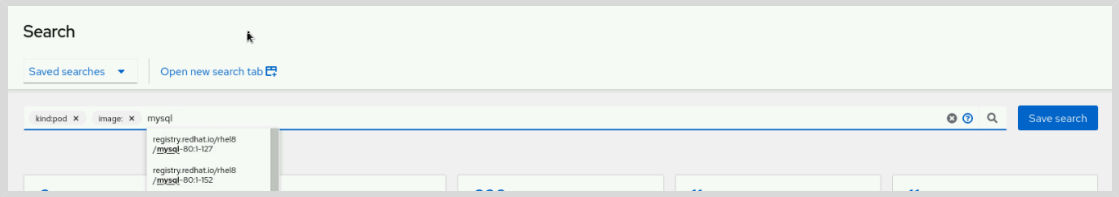
- Search API – Provides an API to access the data in the search index, enforcing role-based access control (RBAC). The search API uses the RBAC of each managed cluster. So, if you are using the RHACM web console you can only search for objects in managed cluster where you already have authorization.

- Search UI – It is deployed as part of th RHACM `MultiClusterHub`.

Using the Search User Interface

You can type free text in the Search box of the user interface. The search engine uses that text to locate it in the Redis database of indexes, and show results for any object that contains the literal. The UI orders the results in a table with a row for each type of Kubernetes or OpenShift object, represented by the `Kind: ` filter.

You can refine each search adding more filters to the query. While you type new filters the UI displays the values that are stored for that index.

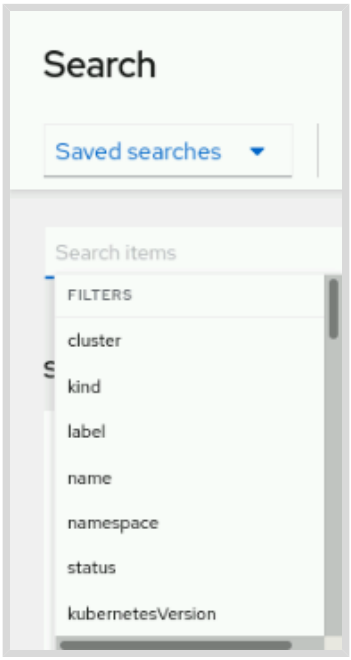


Some of the filters allow aritmethic comparators for fields of numeric nature, as `cpu:`, `replicas:`, `capacity:`, or `memory:`.

You can use use the search UI also for editing the objects. Once you click in the object displayed in a result, you see the YAML definition of the object and the **Edit** button.

What is indexed

The RHACM search engine uses many different filters to classify the objects present in the fleet of clusters. Those filters are the keys of the indexing process. You can see all the filters in the search user interface of the RHACM web console, as displayed in the next image.



You can use any of the filters to perform search, and you can also make free text searchs. But you can not make a search refining by all the fields of the object. If the field is not part of a filter, it is no indexed for search.

Note

RHACM search engine uses always the `label:` filter to index every Kubernetes object. It is a good practice to set labels during the different phases of CI/CD to the objects, to make more accurate and quicker searches.

References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Web Console* guide at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/web_console/index

For more information about cluster management, refer to the *Red Hat Advanced Cluster Management for Kubernetes Clusters* guide at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/clusters/index

For more information about configuring and tuning the search engine, refer to the *Red Hat Advanced Cluster Management for Kubernetes Web Console* guide at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/web_console/index#search-customization

For more information about credentials, refer to the *Red Hat Advanced Cluster Management for Kubernetes Credentials* guide at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/credentials/index

Guided Exercise

Navigating the Red Hat Advanced Cluster Management Web Console

In this exercise you will list the failed deployments across the fleet of clusters to diagnose and fix a failed MySQL deployment. You will also identify the most used database server. Then, you will ensure that there is no running containers using a specific MySQL image with known vulnerabilities.

Outcomes

You should be able to:

- Use the saved searches to quickly identify the failing deployments across a fleet of clusters in the RHACM web console
- Locate kubernetes and OpenShift objects across a fleet of clusters in the RHACM web console
- Edit Kubernetes objects across a fleet of clusters in the RHACM web console

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command creates 15 namespaces in each managed cluster and populate them with 9 different database servers.

Warning

The preparation of this lab will change screen resolution to 1920x1080 for the most comfortable visualization for RHACM console. When the lab is finished resolution is reverted to the existing before the exercise.

```
[student@workstation ~]$ lab start features-console
```

Instructions

1. Identify any failed database server deployments across the fleet of clusters using the RHACM web console feature of saved searches.

   1.1. From workstation, use Firefox to navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`.

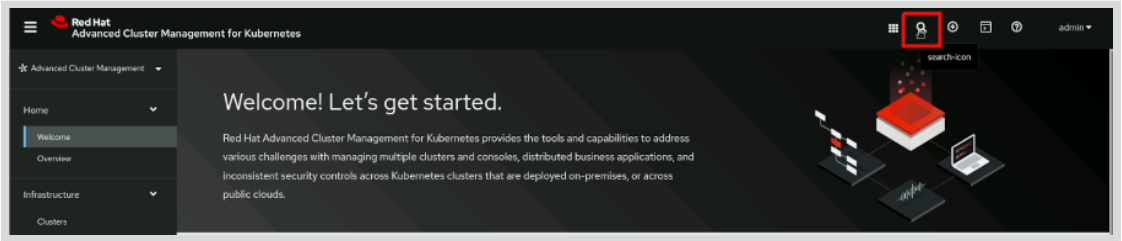
   When prompted, select **htpasswd_provider**

   Type the credentials:

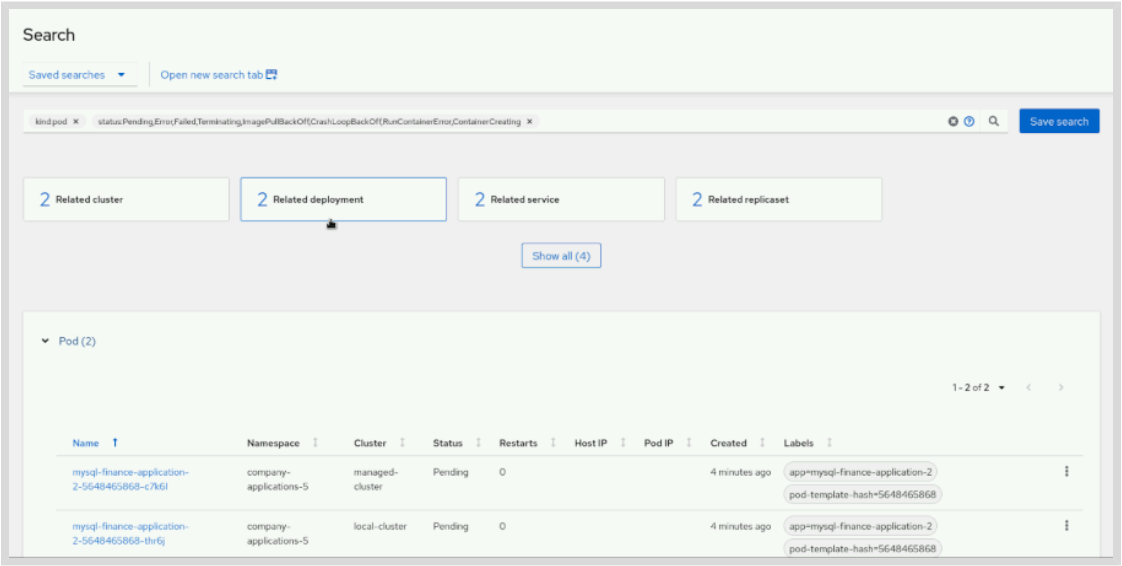
   - Username: `admin`
   - Password: `redhat`

   And then click **Log in**

1.2. Click the glass icon to navigate to the RHACM search web interface.

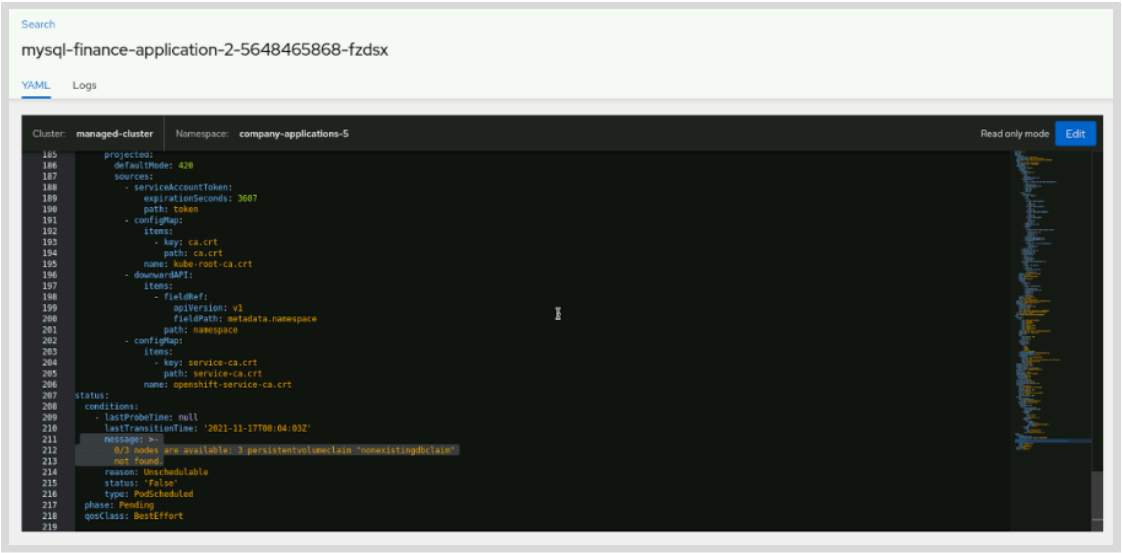


Then click the predefined search **Unhealthy pods**. Observe that the syntax of the search is based on the status of the pods.



There are 2 failing pods with names that start with `mysql-<ID_OF_THE_POD>`. The 2 failing pods are in the namespace called `company-applications-5`. There is a namespace `company-applications-5` in each managed cluster.

1.3. Click the name of one of the MySQL pods in the `Pending` status. Scroll down to the `status` section of the YAML file to identify the cause of the failure.



The pods are pending because there is a reference to a Persistent Volume Claim (PVC) that does not exit.

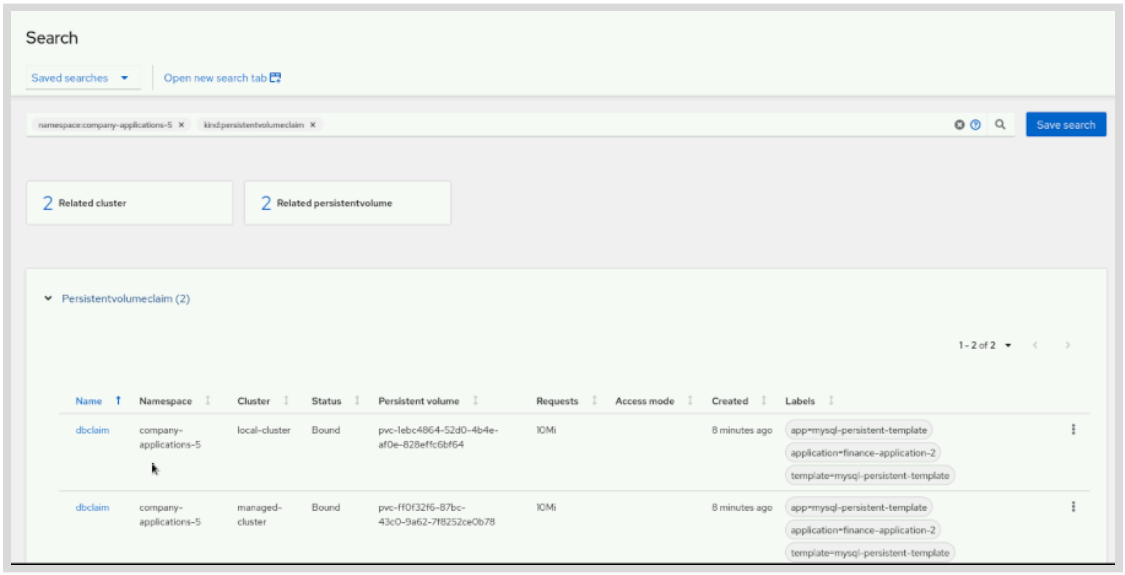
Note

The search results can show pods not related to MySQL deployments. Ignore those results and use the ones related to the MySQL deployments.

1.4. Navigate back to the predefined search **Unhealthy pods** and click the **related deployment** button. The deployment that contains the pods is `mysql-finance-application-2`. The deployment is present in all the managed clusters.

1.5. Click again the glass icon to perform a new search for locating the correct name of the existing PVCs.

Type `namespace:company-applications-5 kind:persistentvolumeclaim` in the search field.

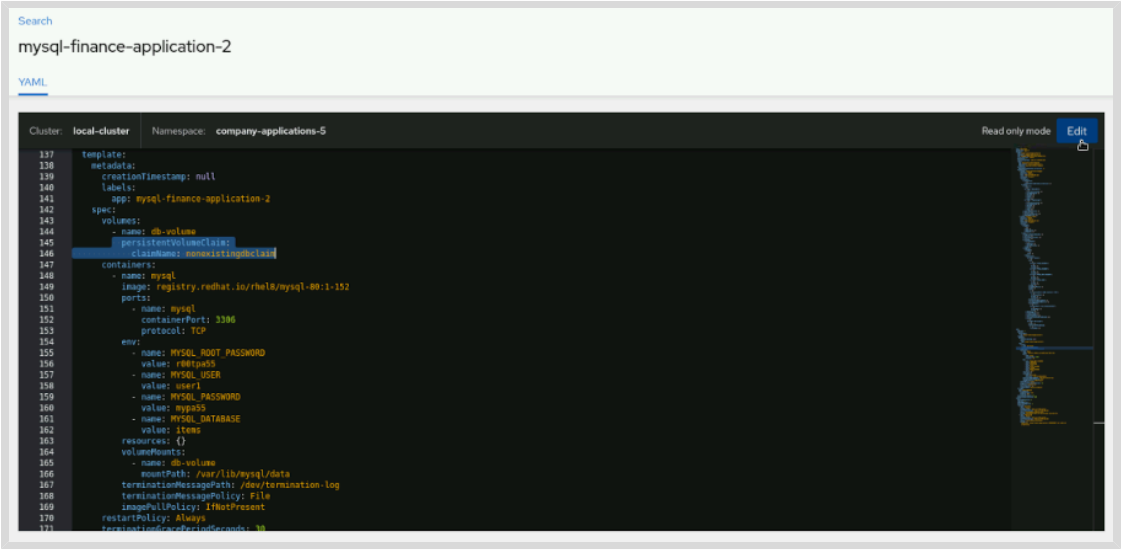


The name of the existing PVC is `dbclaim`.

1.6. Click again the glass icon to make a new search.

Type `namespace:company-application-5 kind:deployment` in the search field.

1.7. Click the name of the first deployment in the **Deployment** results. Edit the yaml file to set `dbclaim` as the PVC name.



Then click the **Save** button to trigger a redeployment automatically using the right name of the PVC.

1.8. Repeat the operation to fix the failing deployment in the other cluster.

1.9. Navigate back to the saved searches and review the **Unhealthy pods** saved search button to verify that there are no more MySQL pods failing.

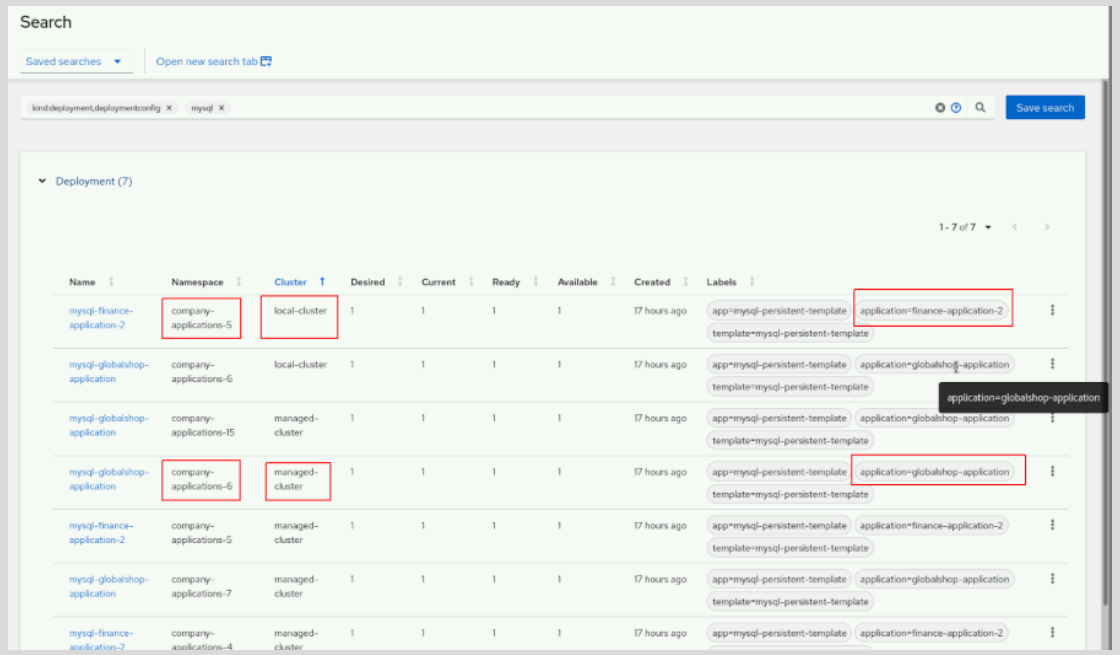
2. Search for any database software deployed across the fleet of managed clusters.

2.1. In the search field type `namespace:` and look at the different namespaces across the fleet of clusters offered by the search engine. You can see that the name of the namespaces are always prepended by `company-applications-`.

2.2. Clear the search field and type `kind:deployment`. Then add `kind:deploymentconfig`. Notice that the RHACM console merges the filters using the syntax `kind:deployment,deploymentconfig`.

Finally add the free text "mysql".

Notice that the MySQL instances are deployed as Kubernetes `Deployment` objects. The label `application` indicates the name of the application. There are 2 different applications using MySQL containers: `globalshop-application`, and `finance-application-2`.



2.3. In the search field, remove the free text "mysql" and replace it with "mariadb". Do not remove the `kind:deployment,deploymentconfig` filter.

Notice that the MariaDB instances are deployed as OpenShift `DeploymentConfig` objects. Use the label `application` to locate the name. There are 2 different applications using MariaDB containers: `humanresources-application-1`, and `marketing-application-2`.

2.4. In the search field, remove the free text "mysql" and replace it with "postgresql". Do not remove the `kind:deployment,deploymentconfig` filter.

You will see that PostgreSQL is deployed as OpenShift `DeploymentConfig` objects.

Use the label `application` to locate the name. There are 5 different applications using PostgreSQL containers: `finance-application-1`, `finance-application-3` `finance-application-4`,`humanresources-application-2`, and `marketing-application-1`.

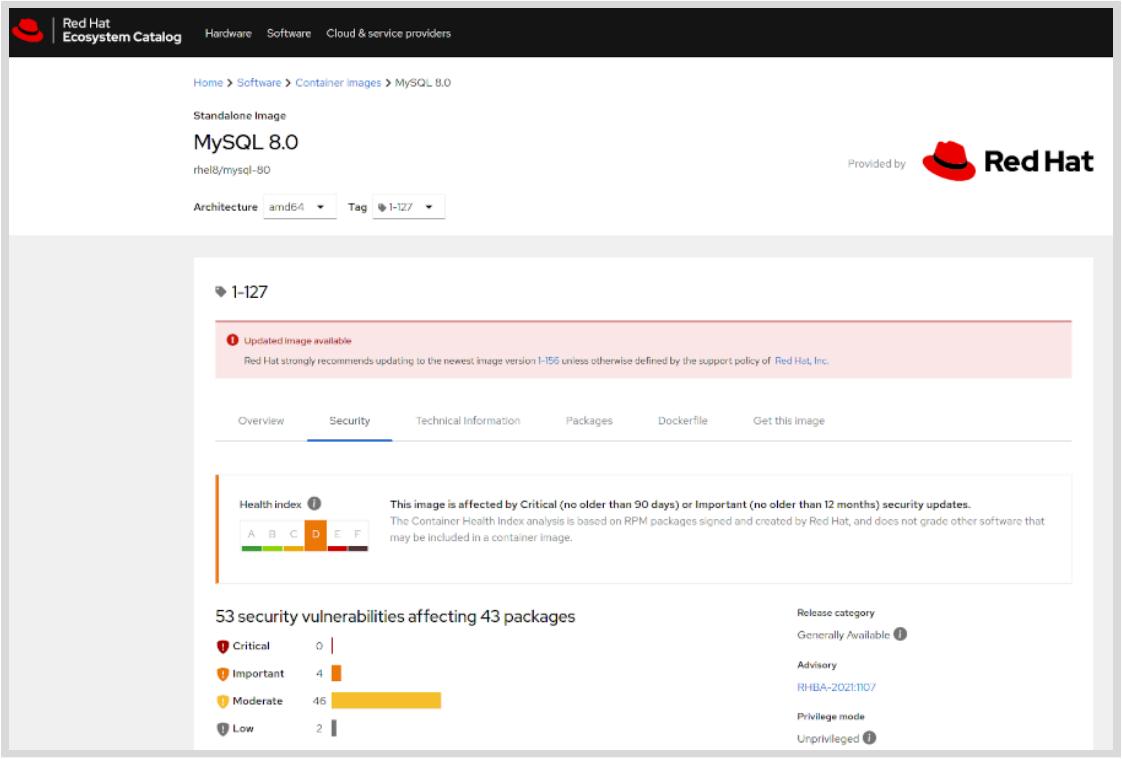
The most used database server is PostgreSQL that is used in 5 applications of existing 9.

3. Using the RHACM web console, locate any running container using a MySQL image with known vulnerabilities. Red Hat Container Catalog states that updated image in use for MySQL 8 must be `registry.redhat.io/rhel8/mysql-80:1`.

Note

You can see the mapping between MySQL version and Red Hat MySQL container image version in https://catalog.redhat.com/software/containers/rhel8/mysql-80/5ba0ad4cdd19c70b45cbf48c

There is a version of MySQL image with the tag `registry.redhat.io/rhel8/mysql-80:1-127` that contains important vulnerabilities and must not be used.



Find the applications using the container image `mysql-80` with the tag `1-127`.

3.1. Type `registry.redhat.io/rhel8/mysql-80` in the search field,

Notice that the results only contain objects of type `pods`.

3.2. Click one of the pods to see the reference to the container image in use in its YAML definition. Most of the MySQL running pods are using `registry.redhat.io/rhel8/mysql-80` with tag `1-152`.

3.3. Navigate to the initial search page of RHACM and add "1-127" in the search field. Do not remove the filter `registry.redhat.io/rhel8/mysql-80`. Inspect the results: you will see two pods running in namespaces with the same name, but in different clusters.

3.4. Click one of the pods to see the image in use in its YAML definition.

3.5. Click the **Logs** tab within the pod page to verify that the running version of MySQL is the old one, 8.0.21, as stated in the Red Hat Container Catalog.

With this information, you can inform the developers which namespaces contain running pods using a vulnerable MySQL container image.

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 finish features-console
```

This concludes the guided exercise.

Configuring Access Control for Multicluster Management

Objectives

After completing this section, you should be able to create different user roles in RHACM and define an authentication model for multicluster management.

Role-based Access Control in RHACM

RHACM access control takes advantage of the Red Hat OpenShift Container Platform authentication layer and roles. Administrators can design a multicluster authentication model to define access for different tenants to different groups of clusters, with fine-grained control.

RHACM Cluster Sets

To create different groups of clusters, RHACM defines the concept of àClusterSe`t. A cluster set tags all the clusters pertaining to it with the `cluster.open-cluster-management.io/clusterset=<managed_clusterset_name>` tag. RHACM uses this tag to identify the clusters belonging to the cluster set, enabling you to take action on all the member clusters at once.

The following table contains the roles created by RHACM to define different access levels to a cluster, all the clusters, or a group of clusters in the same cluster set.

Role	Definition
`open-cluster-management:cluster-manager-admin`	RHACM super user, with full access. Can create a `ManagedCluster` resource.
`open-cluster-management:admin:<managed_cluster_name>`	RHACM administrator access to the `ManagedCluster` resource named `<managed_cluster_name>`.
`open-cluster-management:view:<managed_cluster_name>`	RHACM view access to the `ManagedCluster` resource named `<managed_cluster_name>`.
`open-cluster-management:managedclusterset:admin:<managed_clusterset_name>`	RHACM administrator access to the `ManagedCluster` resource named `<managed_cluster_name>`. The user has also administrator access to the resources of type `managedcluster.cluster.open-cluster-management.io`, `clusterclaim.hive.openshift.io`, `clusterdeployment.hive.openshift.io`, and `clusterpool.hive.openshift.io` with the label `cluster.open-cluster-management.io/clusterset=<managed_clusterset_name>`.

Role	Definition
`open-cluster-management: managedclusterset:view: <managed_clusterset_name>`	RHACM view access to the `ManagedCluster` resource named `<managed_cluster_name>`. The user has also administrator access to the resources of type `managedcluster.cluster.open-cluster-management.io`, `clusterclaim.hive.openshift.io`, `clusterdeployment.hive.openshift.io`, and `clusterpool.hive.openshift.io` with the label `cluster.open-cluster-management.io/clusterset=<managed_clusterset_name>`.
`open-cluster-management: subscription-admin`	Can create Git subscriptions that deploy Kubernetes resources YAML files to multiple namespaces.

RHACM also supports the following default RHOCP roles:

Role	Definition
`cluster-admin`	RHOCP super user, with full access.
`admin, edit, view`	RHOCP default roles. A user with a namespace-scoped binding to these roles has access to open-cluster-management resources in a specific namespace. Cluster-wide binding to the same roles gives access to all of the open-cluster-management resources.

RHACM roles are especially useful for assigning user or group permissions to a cluster set. Apart from using the `oc adm` command, the admin and view cluster set roles can be assigned to users in the **Access management** tab within the clusterset details pane in the RHACM web console. You can assign the `admin` and `view` cluster set roles by using the `oc adm` command or by navigating to the **Access management** tab within the **Cluster sets** details pane in the RHACM web console.

Cluster Selectors and Cluster Sets

A cluster cannot pertain to more than one cluster set simultaneously. If you need to define overlapping groups of clusters, you can use cluster labels acting as selectors.

Combining cluster sets and cluster selectors, administrators can use other tools outside of RHACM to act on a subset of clusters belonging to the same or different cluster sets.

Placement Rules

Using placement rules, administrators can define a subset of clusters pertaining to different cluster sets for placing Kubernetes resources.

Placement rules can use `labelSelector`, `claimSelector`, `clusterSet`, and `numberOfClusters` parameters to determine the clusters on which to deploy the resources.

For example, the following `Placement` uses the `claimSelector` parameter to specify that resources should deploy on clusters from the `us-west-1` region.

```
apiVersion: cluster.open-cluster-management.io/v1alpha1
kind: Placement
metadata:
  name: placement2
  namespace: ns1
spec:
  predicates:
    - requiredClusterSelector:
        claimSelector:
          matchExpressions:
            - key: region.open-cluster-management.io
              operator: In
              values:
                - us-west-1
```

Note

For more information, refer to the *Using ManagedClusterSets with Placement* section at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/clusters/index#placement-managed

Benefits of a Centralized Identity Management Service

One important challenge in multicluster environments is to manage the authentication and authorization mechanisms in a fully controlled manner across different Kubernetes and OpenShift clusters.

As noted previously, RHACM takes advantage of the Red Hat OpenShift Container Platform authentication and authorization layers. The authentication layer of OpenShift uses the OAuth open standard as an authorization framework. By default, OpenShift uses an internal OAuth server. Unfortunately, neither Kubernetes nor OpenShift currently provides a way to federate all the internal OAuth servers of each cluster.

To solve this problem, you can use an external identity manager for central management of the identities of the users that access to your clusters.

Note

You can find the list of supported OAuth providers in the *Supported identity providers* section of the *Red Hat OpenShift Container Platform Authorization and Authentication* guide at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.9/html-single/authentication_and_authorization/index#supported-identity-providers

References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Access Control* guide at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/access_control/index

Guided Exercise

Configuring Access Control for Multicluster Management

In this exercise you will create cluster sets using existing RHOCP clusters. Then you will configure existing users in RHOCP to manage those cluster sets.

Outcomes

You should be able to:

- Create a `ClusterSet`
- Assign a managed cluster to a `ClusterSet`
- Assign predefined RHACM roles to different users
- Remove a `ClusterSet`

Before You Begin

Sample prerequisites

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

The lab environment includes a Red Hat Identity Management server. This command ensures that the necessary users and roles exist in the Red Hat Identity Management server. It also configures an additional LDAP identity provider in the hub cluster.

The following table summarizes the users and groups available in the Red Hat Identity Management server for this exercise.

User	Group
prod-admin	production-administrators
stage-admin	stage-administrators

```
[student@workstation ~]$ lab start features-users
```

Note

RHACM is installed in the hub cluster with URL `ocp4.example.com`.

> **Note**
> The route to the RHACM web console is `multicloud-console.apps.ocp4.example.com`.

Instructions

1. From the RHACM web console, create a cluster set named `production` and add the cluster named `local-cluster`.

   1.1. Open Firefox and navigate to the RHACM console in `multicloud-console.apps.ocp4.example.com`.

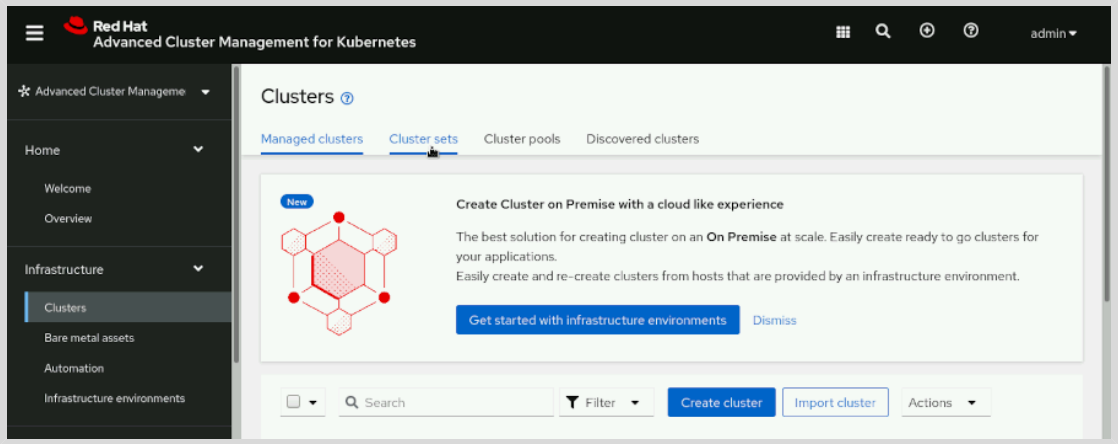
   When prompted, select **htpasswd_provider**

   Type the credentials:

   - Username: `admin`
   - Password: `redhat`

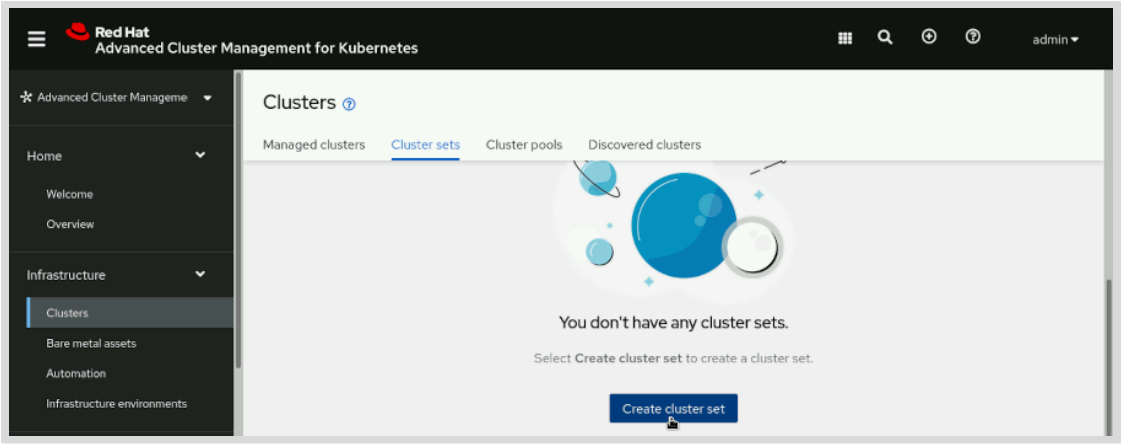
   And then click **Log in**

   1.2. Navigate to **Infrastructure** → **Clusters** and click **Cluster sets**.

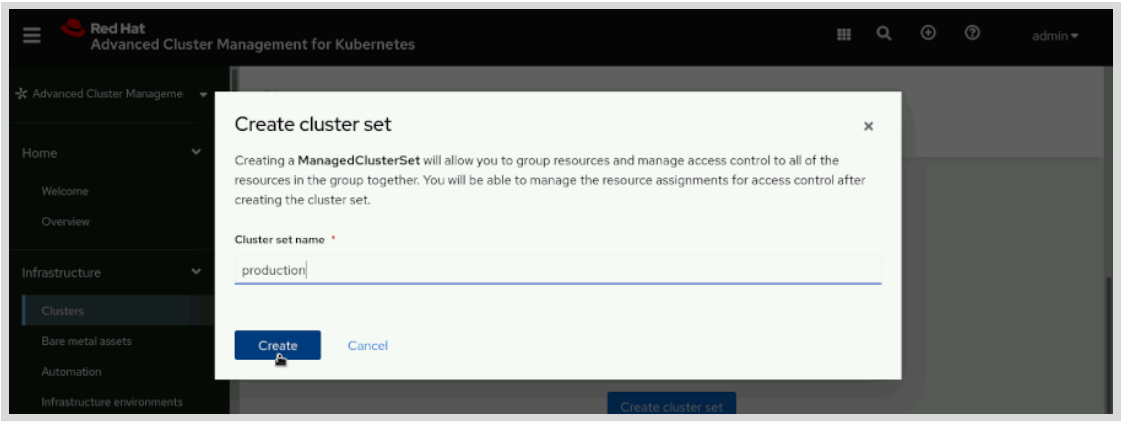


   1.3. Create a cluster set named `production`. Add the `local-cluster` to it.

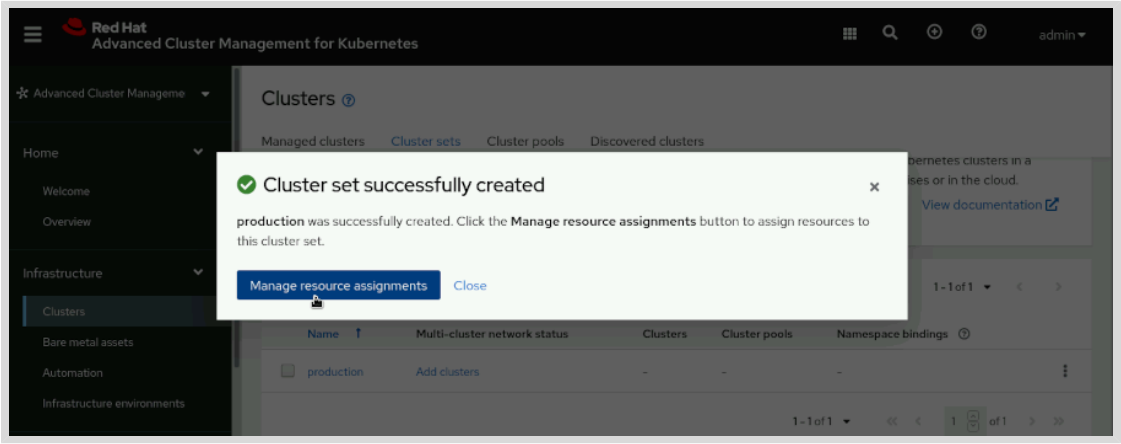
   From within the **Cluster sets** page, scroll down and click **Create cluster set**



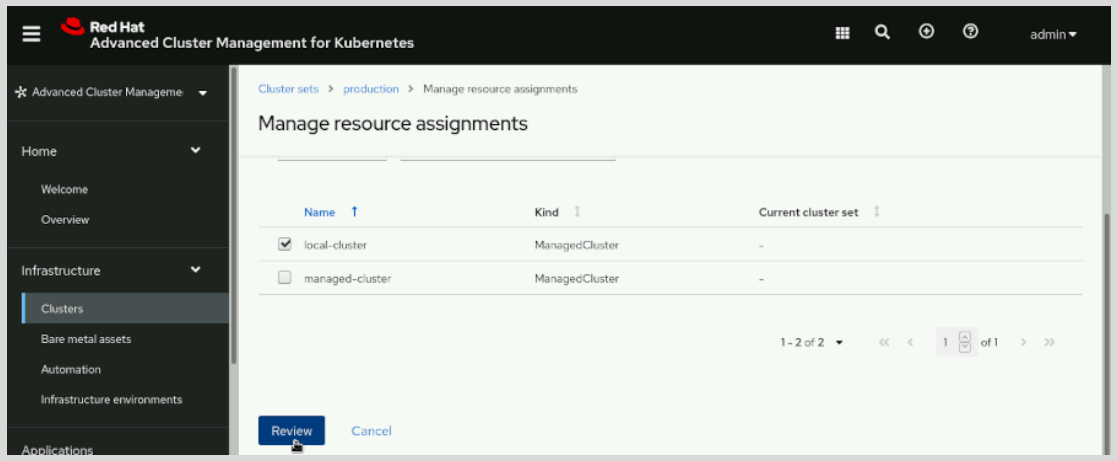
Type `production` in the `Cluster set name` field and click **Create**



In the confirmation menu, click **Manage resource assignments**



Select the cluster named `local-cluster` and click **Review**



In the confirmation screen, click **Save**

Finally, you are redirected to the **Overview** tab of the `production` cluster set.

1.4. Verify the labels corresponding to the `production` cluster set.

From the **Managed clusters** page of the `production` cluster set, click **Managed clusters**

Observe the `cluster.open-cluster-management.io/clusterset=production` label.

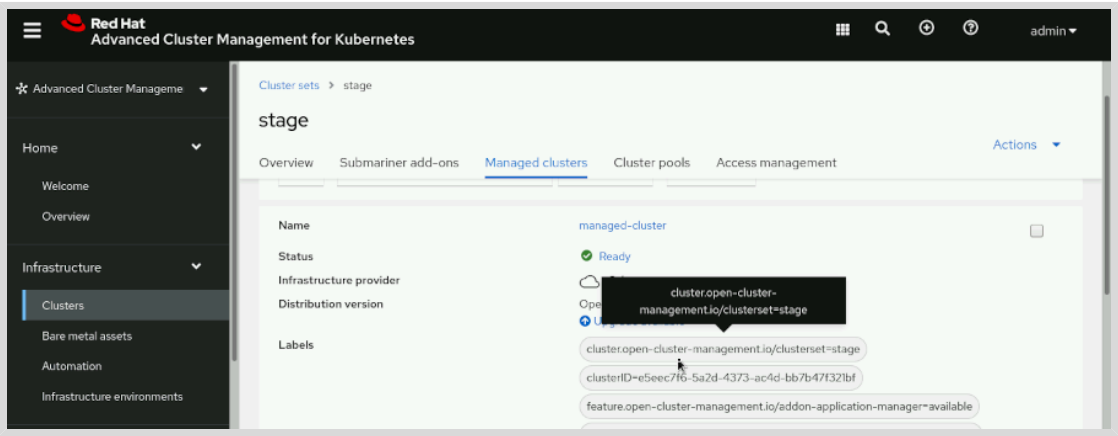
RHACM uses this labels to identify clusters pertaining to a cluster set.

2. From the RHACM web console, create a cluster set named `stage` and add the cluster named `managed-cluster`.

2.1. Navigate to **Infrastructure** → **Clusters** and click **Cluster sets** and repeat all the previous steps to create a `stage` cluster set.

Add the cluster named `managed-cluster` to the `stage` cluster set.

From the **Managed clusters** page of the `stage` cluster set, click **11 labels** to verify that RHACM adds the label `cluster.open-cluster-management.io/clusterset=stage` to the `managed-cluster` cluster.



3. From the terminal, assign the pre-existing group `production-administrators` administrator permissions in the `production` cluster set. Also, assign the group `production-administrators` view permissions in the `stage` cluster set.

Note

RHACM automatically creates default `admin` and `view` cluster roles in the RHOCP hub cluster for each new cluster set. For instance, to grant administrator permissions to the `production` cluster set, use the role `open-cluster-management:managedclusterset:admin:<clusterset_name>`.

To grant view permissions to the `production` cluster set, use the role `open-cluster-management:managedclusterset:view:<clusterset_name>`.

3.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

3.2. Use `oc adm` to assign the cluster role `open-cluster-management:managedclusterset:admin:production` to the existing group `production-administrators`.

Note

The group `production-administrators` already exists in the Red Hat Identity Management server in the lab environment. The `lab start` script triggers a group synchronization between the hub cluster and the Red Hat Identity Management server.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
  open-cluster-management:managedclust erset:admin:production
 production-administrators
clusterrole.rbac.authorization.k8s.io/open-cluster-
management:managedclusterset:admin:production added: "production-administrators"
```

3.3. Use `oc adm` to assign the cluster role `open-cluster-management:managedclusterset:view:stage` to the existing group `production-administrators`.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
  open-cluster-management:managedclust erset:view:stage production-administrators
clusterrole.rbac.authorization.k8s.io/open-cluster-
management:managedclusterset:view:stage added: "production-administrators"
```

4. From the terminal, assign the pre-existing group `stage-administrators` administrator permissions in the `stage` cluster set. Do not assign the `stage-administrators` group any permissions to the `production` cluster set.

4.1. Use `oc adm` to assign the cluster role `open-cluster-management:managedclusterset:admin:stage` to the existing group `stage-administrators`.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
  open-cluster-management:managedclust erset:admin:stage stage-administrators
clusterrole.rbac.authorization.k8s.io/open-cluster-
management:managedclusterset:admin:stage added: "stage-administrators"
```

The following table summarizes the users, groups and roles at this point.

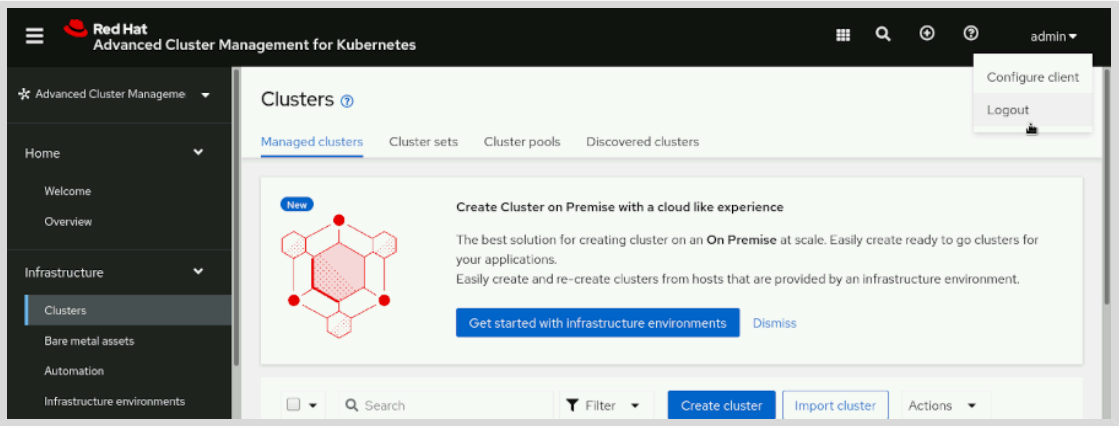
User	Group	Roles
prod-admin	production-administrators	open-cluster-management:managedclusterset:admin:production, open-cluster-management:managedclusterset:view:stage
stage-admin	stage-administrators	open-cluster-management:managedclusterset:admin:stage

5. Log out from RHACM and close Firefox to clean any cached credentials in the browser. Open it again and log in to the RHACM web console with the `prod-admin` user from the Red Hat Identity Management provider. Verify that the `prod-admin` user has administrator permissions on the `production` cluster set, comprised by the cluster named `local-cluster`. Also, verify that the `prod-admin` user has view only permissions on the `stage` cluster set.

Note
The `prod-admin` user is member of the `production-administrators` group.

5.1. Log out from RHACM.

From the RHACM web console, click the **admin** button at the top right corner, and click **Logout**.



5.2. Close all the instances of Firefox. Open a new Firefox window and log in to the RHACM web console with the user `prod-admin` from the Red Hat Identity Management provider.

When prompted, select **Red Hat Identity Management**

Type the credentials:

- Username: `prod-admin`
- Password: `redhat`

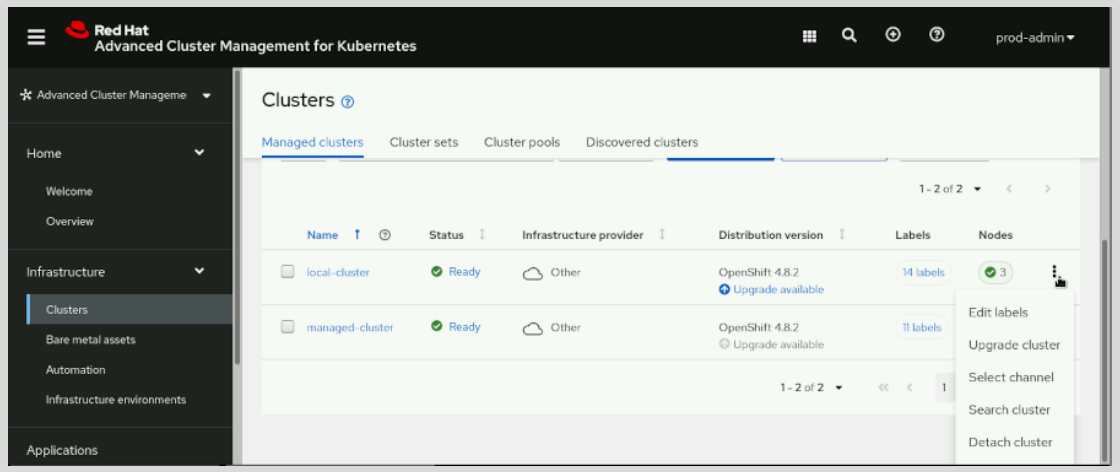
And then click **Log in**

5.3. Navigate to **Infrastructure** → **Clusters**. Note the differences between the two clusters.

The **Upgrade available** button is active for the `local-cluster`, but inactive for the `managed-cluster`. This happens because the `prod-admin` user has administrator privileges in the `production` cluster set, but only view permissions in the `stage` cluster set.

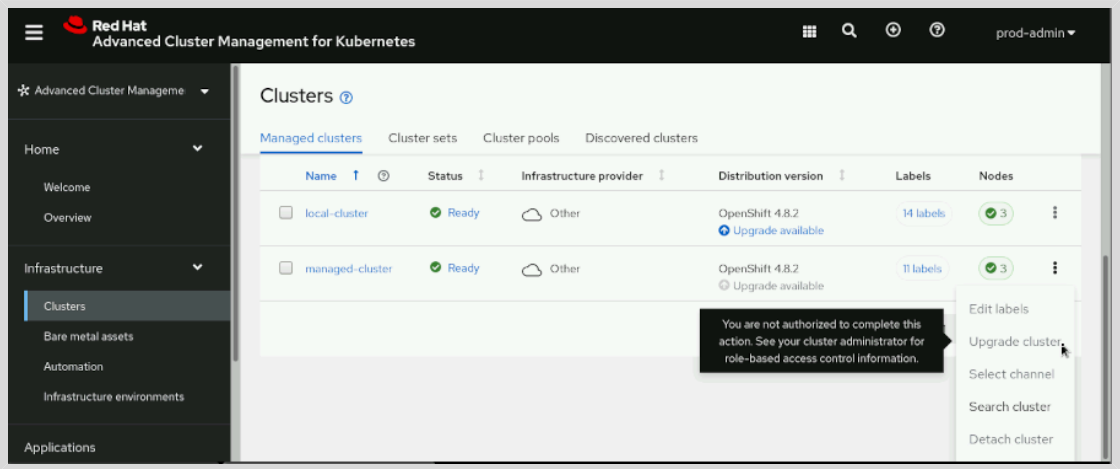
5.4. Click the **Options** button of the `local-cluster`.

Note that as an administrator, all the options are enabled.



A cluster set administrator can perform actions on clusters such as **Edit labels**, **Upgrade cluster**, **Select channel** for updates, **Search cluster** resources, and **Detach cluster**.

5.5. Click the options button of the `managed-cluster`.



Note that as a viewer, only the **Search cluster** option is enabled.

6. Log out from RHACM and close Firefox to clean any cached credentials in the browser. Open it again and log in to the RHACM web console with the `stage-admin` user from the Red Hat Identity Management provider. Verify that the `stage-admin` user has administrator permissions on the `stage` cluster set, comprised by the cluster named `managed-cluster`. Also, verify that the `stage-admin` don't have any permissions on the `production` cluster set.

Note
The `stage-admin` user is member of the `stage-administrators` group.

6.1. Log out from RHACM.

From the RHACM web console, click the **prod-admin** button at the top right corner, and click **Logout**.

6.2. Close all the instances of Firefox. Open a new Firefox window and log in to the RHACM web console with the user `stage-admin` from the Red Hat Identity Management provider.

When prompted, select **Red Hat Identity Management**

Type the credentials:

- Username: `stage-admin`
- Password: `redhat`

And then click **Log in**

6.3. Navigate to **Infrastructure** → **Clusters**. Note that the `stage-admin` user can only see clusters belonging to the `stage` cluster set.

The **Upgrade available** button and all the options from the **Options** button are enabled.

This is because the `stage-admin` user has administrator permissions on the `stage` cluster set.

The cluster `local-cluster`, pertaining to the `production` cluster set does not appear in the **Managed clusters** list.

7. Log out from the RHACM web console and close all the Firefox instances to remove any cached credentials.

7.1. From the RHACM web console, click the **stage-admin** button at the top right corner, and click **Logout**.

Close the Firefox window.

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 finish features-users
```

This concludes the guided exercise.

Lab

Inspecting Resources from Multiple Clusters Using the RHACM Web Console

In this lab, you assign the cluster-wide role `view` to a group of users so that they can perform searches across the fleet of clusters. Then, you log in with a user from that group and search for a deployment wuth few replicas. Finally you log in to RHACM with a user with admin permissions in the managed cluster set and fix the problem.

Outcomes

You should be able to:

- Manage RHACM roles and groups to allow searches across a fleet of clusters.
- Locate Kubernetes and OpenShift objects across a fleet of clusters by using the correct group of users.
- Edit Kubernetes objects across a fleet of clusters using the correct group of users.

Before You Begin

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

This command ensures that RHACM is installed and running and the managed clusters are present. It also checks that the necessary users and roles exist in the Red Hat Identity Management server. Finally, it creates all the deployments across the fleet of clusters.

The following table summarizes the users and groups available in the Red Hat Identity Management server for this exercise.

User	Group
apac-operator	apac-operators
emea-operator	emea-operators
fleet-searcher	fleet-searchers

The following table summarizes the managed cluster sets and the roles of the groups available in the laboratory environment for this exercise.

Managed Cluster Set	Group	Role of the Group
apac	apac-operators	`open-cluster-management:managedclusterset:admin:apac`
emea	emea-operators	`open-cluster-management:managedclusterset:admin:emea`

Managed Cluster Set	Group	Role of the Group
-	fleet-searcher	-

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

Instructions

1. Assign the cluster-wide role `view` to the `fleet-searchers` group to allow searches across all the objects in the fleet of clusters.
2. Locate all Kubernetes objects from the applications that have the `app=finance-application` label and contain fewer than 3 replicas.
3. Fix the number of replicas of the two deployments by logging in as a user belonging to the `emea-operators` group.

Evaluation

Lab scripts to be developed.

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. Do not make any other changes to the lab environment until the next guided exercise. You will continue using it in later guided exercises.

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

This concludes the lab.

Solution

Inspecting Resources from Multiple Clusters Using the RHACM Web Console

In this lab, you assign the cluster-wide role `view` to a group of users so that they can perform searches across the fleet of clusters. Then, you log in with a user from that group and search for a deployment wuth few replicas. Finally you log in to RHACM with a user with admin permissions in the managed cluster set and fix the problem.

Outcomes

You should be able to:

- Manage RHACM roles and groups to allow searches across a fleet of clusters.
- Locate Kubernetes and OpenShift objects across a fleet of clusters by using the correct group of users.
- Edit Kubernetes objects across a fleet of clusters using the correct group of users.

Before You Begin

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

This command ensures that RHACM is installed and running and the managed clusters are present. It also checks that the necessary users and roles exist in the Red Hat Identity Management server. Finally, it creates all the deployments across the fleet of clusters.

The following table summarizes the users and groups available in the Red Hat Identity Management server for this exercise.

User	Group
apac-operator	apac-operators
emea-operator	emea-operators
fleet-searcher	fleet-searchers

The following table summarizes the managed cluster sets and the roles of the groups available in the laboratory environment for this exercise.

Managed Cluster Set	Group	Role of the Group
apac	apac-operators	`open-cluster-management:managedclusterset:admin:apac`
emea	emea-operators	`open-cluster-management:managedclusterset:admin:emea`

Managed Cluster Set	Group	Role of the Group
-	fleet-searcher	-

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

Instructions

1. Assign the cluster-wide role `view` to the `fleet-searchers` group to allow searches across all the objects in the fleet of clusters.

1.1. From the `workstation` machine, open a terminal and log in to the `ocp4` cluster as the `admin` user. The password is `redhat`. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

1.2. Verify the roles assigned to the `apac-operators` group.

```
[student@workstation ~]$ oc get clusterrolebindings.authorization \
  -ocustom-columns=role:.roleRef.name ,groups:.groupNames.* | grep apac-operators

open-cluster-management:managedclusterset:admin:apac    apac-operators
open-cluster-management:admin:local-cluster
 system:masters,system:cluster-admins,apac-operators
open-cluster-management:view:local-cluster              system:cluster-
readers,system:cluster-admins,system:masters,apac-operators
[student@workstation ~]$
```

1.3. Verify the roles assigned to the `fleet-searchers` group.

```
[student@workstation ~]$ oc get clusterrolebindings.authorization \
  -ocustom-columns=role:.roleRef.name ,groups:.groupNames.* | grep fleet-searchers
[student@workstation ~]$
```

This group is not assigned any role.

1.4. Use the `oc adm` command to assign the `view` cluster role to the `fleet-searchers` group.

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

1.5. Reverify the `fleet-searchers` group roles to identify the additional roles assigned by the RHACM.

```
[student@workstation ~]$ oc get clusterrolebindings.authorization \
  -ocustom-columns=role:.roleRef.name ,groups:.groupNames.* | grep fleet-searchers
open-cluster-management:view:local-cluster       system:cluster-
readers,system:cluster-admins,system:masters,apac-operators,fleet-searchers
open-cluster-management:view:managed-cluster     system:cluster-
readers,system:cluster-admins,system:masters,emea-operators,fleet-searchers
view                                             fleet-searchers
[student@workstation ~]$
```

2. Locate all Kubernetes objects from the applications that have the `app=finance-application` label and contain fewer than 3 replicas.

2.1. From the `workstation` machine, use Firefox to navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`.

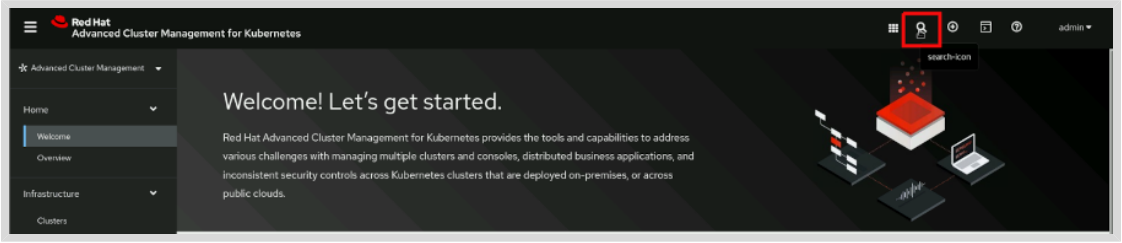
When prompted, select **Red Hat Identity Management**

Type the following credentials:

- Username: `fleet-searcher`
- Password: `redhat`

And then click **Log in**.

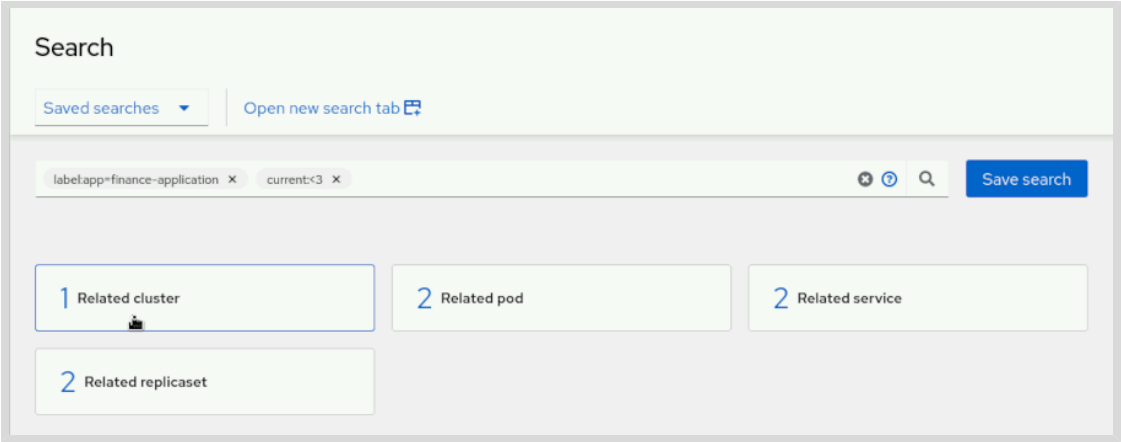
2.2. Click the magnifying glass icon to navigate to the RHACM search web interface.



Type `label:app=finance-application current:<3` in the search field. The results show 2 different `Deployment` objects. Both deployments are in the `managed-cluster` cluster, but one is within the `company-applications-5` namespace, and the other one is within the `company-applications-7`.

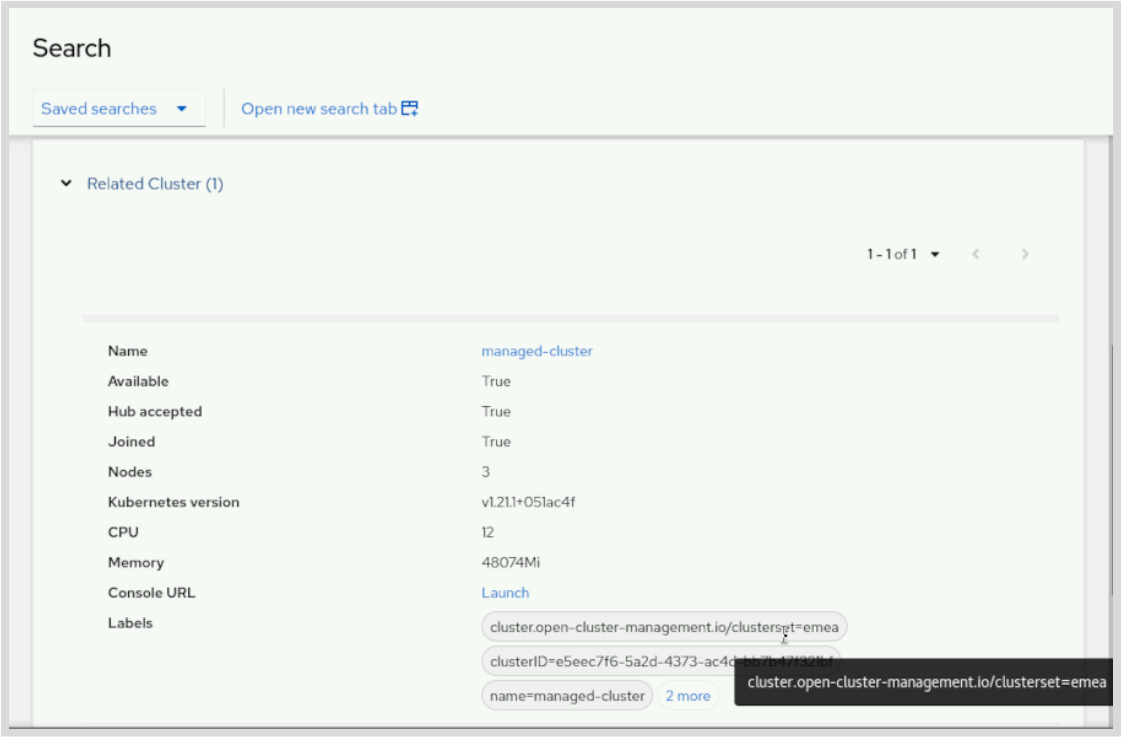
The RHACM console does not allow the `fleet-searcher` user to modify these two deployments.

2.3. Click **Related cluster** in the related objects of the results.



Then expand the related cluster to see all the information about the `managed-cluster`.

2.4. Locate the cluster set that contains the `managed-cluster` by finding the value of the label `cluster.open-cluster-management.io/clusterset=`.



The `managed-cluster` cluster belongs to the `emea` cluster set. The `emea-operators` group can modify the number of replicas.

3. Fix the number of replicas of the two deployments by logging in as a user belonging to the `emea-operators` group.

3.1. Log out from RHACM and close Firefox to clean any cached credentials in the browser. Open Firefox again and log in to the RHACM web console.

When prompted, select **Red Hat Identity Management**

Type the credentials:

- Username: `emea-operator`
- Password: `redhat`

And then click **Log in**.

3.2. Click the magnifying glass icon to navigate to the RHACM search web interface.

Repeat previous search by typing `label:app=finance-application current:<3` in the search field. The same 2 deployments display as when you searched as a member of the fleet-searchers group, but now the members of the emea-operators group can edit the objects.

3.3. Click the name of the deployment in the `company-applications-5` namespace.

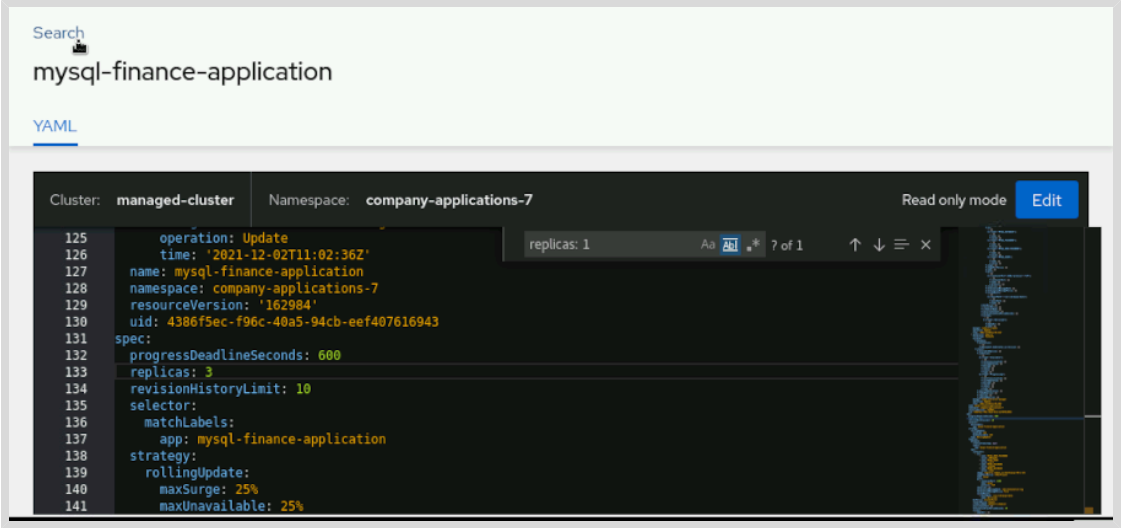
Then click **Edit**.

3.4. Find the text `replicas: 1` in the `spec:` section of the deployment. You can use **crtl +f** to find inside the YAML file.

Then change the value to `replicas: 3` and click **Save**.

3.5. Repeat the preceding operation in the deployment present in the `company-applications-7` namespace.

3.6. Check that the previous search has no more results by clicking the **Search** link.



Evaluation

Lab scripts to be developed.

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. Do not make any other changes to the lab environment until the next guided exercise. You will continue using it in later guided exercises.

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

This concludes the lab.

Summary

In this chapter, you learned:

- How to access the functionalities of RHACM from the web console.
- How to locate Kubernetes objects across de fleet of clusters by using the RHACM search engine.
- How the role-based access control model of RHACM works, and its default roles.
- How to define an authentication model to manage the fleet of clusters classifying the clusters in RHACM clustersets.

Chapter 3

Deploying and Managing Policies for Multiple Clusters with RHACM

Goal

Deploy and manage policies in multicluster environments by using RHACM governance.

Objectives

- Deploy policies on multiple clusters by using the command line and the RHACM Governance Dashboard.
- Deploy the compliance operator policy and view compliance reports from multiple clusters by using the command line and the RHACM Governance Dashboard.
- Deploy the gatekeeper policy and gatekeeper constraints to multiple clusters by using the command line and the RHACM Governance Dashboard.

Sections

- Deploying and Managing Policies with RHACM (and Guided Exercise)
- Deploying and Configuring the Compliance Operator for Multiple clusters using RHACM (and Guided Exercise)
- Integrating RHACM with Other Policy Engines (and Guided Exercise)

Lab

Deploying and Managing Policies with RHACM

Deploying and Managing Policies with RHACM

Objectives

After completing this section, you should be able to deploy policies on multiple clusters by using the command line and the RHACM Governance Dashboard.

RHACM Governance Overview

Introduction

The enterprises must ensure that the security of their workloads and the configuration hosted on clouds meet corporate standards. RHACM Governance provides a policy-based approach for companies to monitor these corporate standards automatically.

Governance Architecture

Provides a summary and componantes of governance architecture

Governance Dashboard

Provides a summary of governance dashboard

RHACM Policy Controllers Overview

The policy controllers monitor if your cluster is compliant with a policy. The policy controller also reports the policy status, which you can see on the RHACM governance dashboard.

List of Policy Controller

- Kubernetes configuration policy controller
- Certificate policy controller
- IAM policy controller
- Integrate third-party policy controllers
- Custom policy controller

RHACM Policy Overview

Provides a summary of the RHACM Policy

Describe Policy YAML File

A policy YAML file has some required and some optional parameters. The optional fields and parameters depend on the policy controller.

```
apiVersion: policy.open-cluster-management.io/v1 (1)
kind: Policy (2)
metadata:
```

```
  name: policy-namespace (3)
  annotations: (4)
    policy.open-cluster-management.io/standards: NIST SP 800-53 (5)
    policy.open-cluster-management.io/categories: CM Configuration Management (6)
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration (7)
spec:
  remediationAction: inform (8)
  disabled: false (9)
  policy-templates: (10)
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-namespace-example
        spec:
          remediationAction: inform # the policy-template spec.remediationAction
 is overridden by the preceding parameter value for spec.remediationAction.
          severity: low
          namespaceSelector:
            exclude: ["kube-*"]
            include: ["default"]
          object-templates:
            - complianceType: musthave
              objectDefinition:
                kind: Namespace # must have namespace 'prod'
                apiVersion: v1
                metadata:
                  name: prod
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: binding-policy-namespace
placementRef:
  name: placement-policy-namespace
  kind: PlacementRule
  apiGroup: apps.open-cluster-management.io
subjects:
- name: policy-namespace
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  name: placement-policy-namespace
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - {key: environment, operator: In, values: ["dev"]}
```

1. *apiVersion:* Required.The value for this parameter is `policy.open-cluster-management.io/v1`
2. *kind:* Required. The value for this parameter is `Policy`
3. *metadata.name:* Required. The parameter sets name of the policy
4. *metadata.annotations:* Optional. This parameter defines standards for policy to validate.
5. *annotations.policy.open-cluster-management.io/standards:* This parameter defines the name of the security standard.
6. *annotations.policy.open-cluster-management.io/categories:* This parameter defines the name of the security control category.
7. *annotations.policy.open-cluster-management.io/controls:* This parameter defines the name of the security control.
8. *spec.remediationAction:* Optional. The values for this parameter are `enforce` and `inform`.
9. *spec.disabled:* Required. The values for this parameter are `true` and `false`.
10. *spec.policy-templates:* This parameter is use for policy creation.

Stable and Community Policies

RHACM can use stable policies, supported by Red Hat, and also can use policies from the community. You can see the most of the stable policies in RHACM Dashboard.

Deploying policy on multicluster with RHACM

- Deploy policy from governance dashboard
- Deploy policy from command line

References

For more information, refer to the _Red Hat Advanced Cluster Management policy yaml structure at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/governance/index#policy-yaml-structure

Guided Exercise

Deploying and Managing Policies with RHACM

In this exercise, you will create a policy in all the clusters to verify the expiration of the certificates in the `openshift-console` and `openshift-ingress` namespaces. Then, you will analyze the status of the policy, apply a fix, and verify the final status after the remediation.

Outcomes

You should be able to:

- Deploy a certificate policy from the RHACM governance dashboard
- Verify the policy status from the RHACM governance dashboard
- Verify the policy status after applying the remediation

Before You Begin

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

```
[student@workstation ~]$ lab start policy-governance
```

Instructions

1. Log in to the Hub OpenShift cluster and create a `policy-governance` project.

   1.1. Open the `terminal` application on the `workstation` machine. Log in to the Hub 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 new project `policy-governance`.

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

2. Log in to the RHACM web console and create the certificate policy.

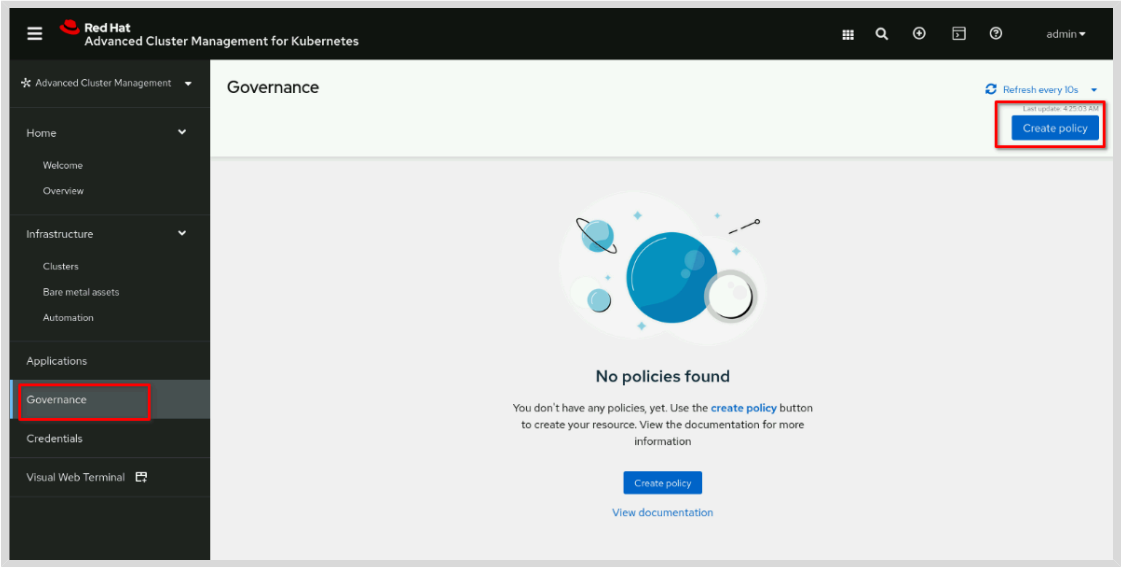
   2.1. From the `workstation` machine, open Firefox and access `https://multicloud-console.apps.ocp4.example.com`.

   2.2. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.

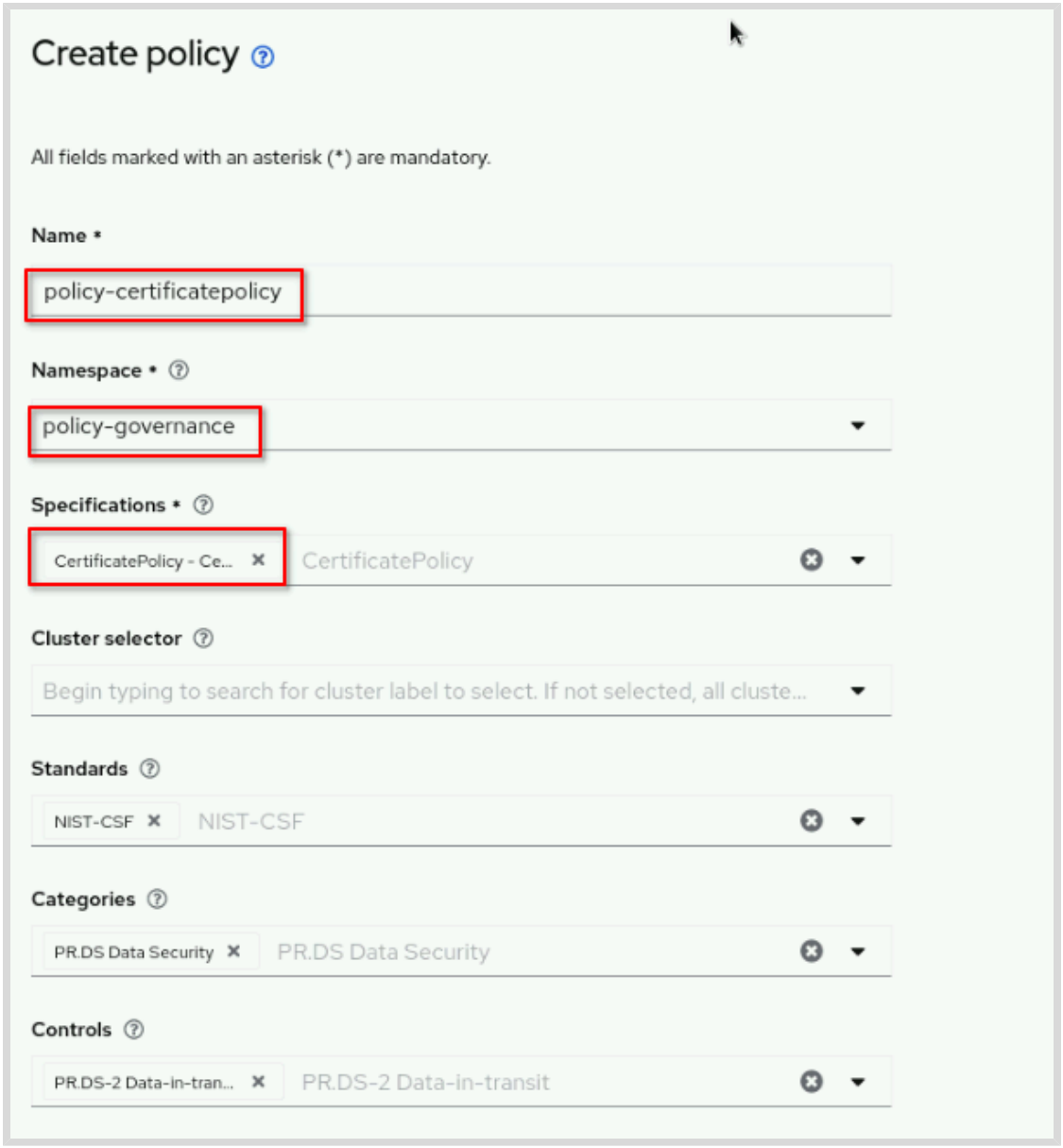
3. Create the certificate policy with the following parameters for `openshift-console` and `openshift-ingress` namespace:

Field Name	Value
`Name`	policy-certificatepolicy
`Namespace`	policy-governance
`Specifications`	CertificatePolicy - Certificate management expiration
`Remediation`	Inform

3.1. Click **Governance** tab on the left pane to navigate to the governance dashboard. Then, click **Create policy** button. The `Create policy` page appears.



3.2. Fill in the fields as follows, leaving rest of the fields unchanged. Do **not** click **Create** button yet.



3.3. On the right side of the `Create policy` page, edit the YAML code as follows:

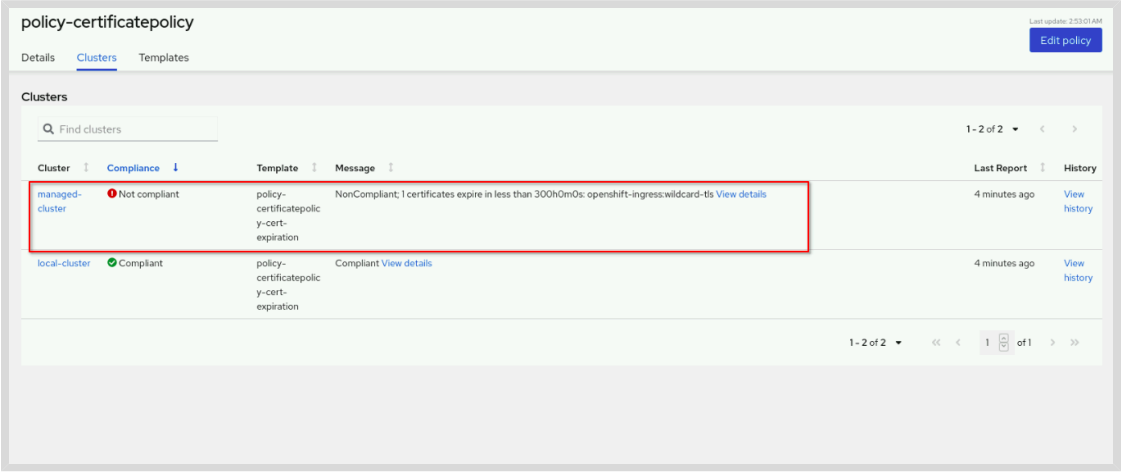
```
spec:
  remediationAction: inform
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: CertificatePolicy
        metadata:
          name: policy-certificatepolicy-cert-expiration
        spec:
          namespaceSelector:
            include:
              - default
```

```
      - openshift-console
      - openshift-ingress
    exclude:
      - kube-*
  remediationAction: inform
  severity: low
  minimumDuration: 300h
```

With that customization the policy applies to the `default`, `openshift-console`, and `openshift-ingress` namespaces.

Click **Create** button.

3.4. In the `Governance` page, scroll down and click the `policy-certificatepolicy` policy name from the list of policies. Click **Clusters** tab to review the policy details.



The status of the policy is `Not compliant` for the cluster named `managed-cluster`.

> **Note**
> `Not compliant` policy status indicates that one certificate has already expired or has less than 300h time left in the `openshift-ingress` namespace.

4. Renew the ingress controller wildcard certificate of the `managed-cluster`.

4.1. Open the `terminal` application on the `workstation` machine and change to the `~/DO480/labs/policy-governance/` directory.

```
[student@workstation ~]$ cd ~/DO480/labs/policy-governance/
```

4.2. Log in to the `managed-cluster` as the `admin` user. The APIServer URL is `https://api.ocp4-mng.example.com:6443`.

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

4.3. 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.

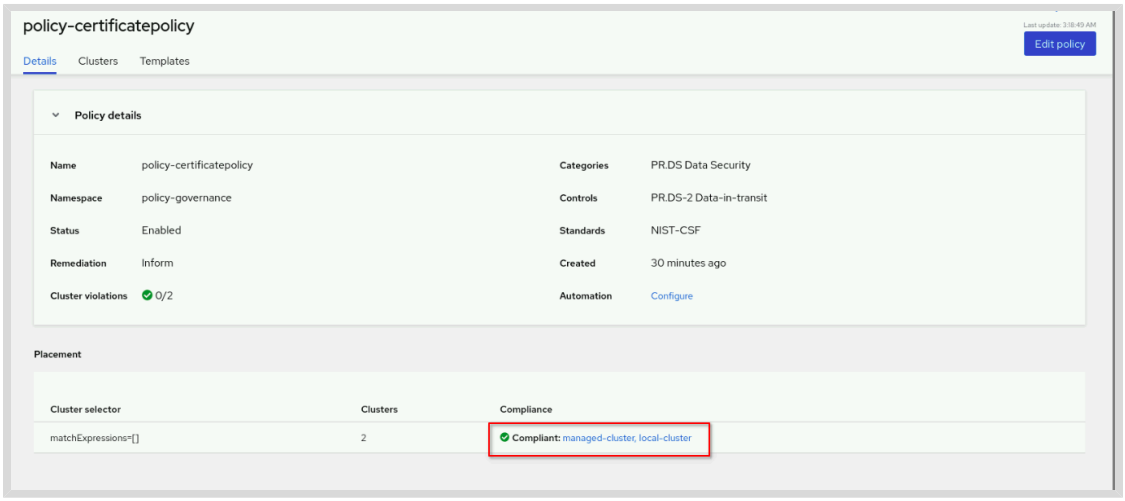
```
[student@workstation policy-governance]$ ./renew_wildcard.sh
...output omitted...
TASK [Create a combined certificate] ************************************
ok: [localhost]
TASK [Create a hard-link to the combined certificate] *********************
ok: [localhost]
PLAY RECAP **************************************************************
localhost      : ok=19     changed=7     unreachable=0     failed=0
skipped=3     rescued=0     ignored=0

configmap/wildcard-bundle data updated
secret/wildcard-tls data updated
```

4.4. Move to the home directory.

```
[student@workstation policy-governance]$ cd ~
[student@workstation ~]$
```

5. Review the status of the `policy-certificatepolicy` policy.

5.1. Click the **Governance** → **policy-certificatepolicy** policy name from the list of policies.



The Policy status is now `Compliant` for both clusters.

> **Note**
> It takes around a minute before the router pods restart and use the updated certificate. Also, it takes a few seconds to change the policy status from `Not complaint` to `Complaint`.

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 finish policy-governance
```

This concludes the guided exercise.

Deploying and Configuring the Compliance Operator for Multiple clusters using RHACM

Objectives

After completing this section, you should be able to deploy the compliance operator policy and view compliance reports from multiple clusters by using the command line and the RHACM Governance Dashboard.

Definition of Compliance

Compliance management is a continuous process to ensure that the IT systems comply with different kinds of policies and requirements. The following list shows some of this kinds of requirements:

- Industry standards
- Security standards
- Corporate policies
- Regulatory policies

Describing the RHOCP Compliance Operator

- You can use the Compliance Operator to verify the compliance state of a cluster.
- The Compliance Operator also verifies the compliance of the Kubernetes objects, as well as the nodes running the cluster.
- The Compliance Operator scans and enforce the security policies you provides using OpenSCAP, a NIST-certified tool.

Explaining the Compliance Operator Components

- Namespace
- Operatorgroup
- Subscription

Deploying the Compliance Operator Across the Fleet of Clusters Using the RHACM Compliance Operator Policy

You can use RHACM console to deploy the policy that deploys RHOCP Compliance Operator in all the managed clusters.

Performing Scans with a Scan Policy

You can use RHACM console to perform scans using scan policies across the fleet of managed clusters.

Verifying Conformance of a Scan Policy

You can use RHACM console to perform scans using scan policies across the fleet of managed clusters.

- Scan and scan binding. Example Essential 8 (E8).

Inspecting Reports of a Scan Policy

You can use RHACM console to verify conformance of a scan policy

References

What is compliance management?
https://www.redhat.com/en/topics/management/what-is-compliance-management

For more information, refer to the *Compliance operator policy* chapter in the *Red Hat Advanced Cluster Management Governance Guide* at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/governance/index#compliance-operator-policy

Guided Exercise

Deploying and Configuring the Compliance Operator for Multiple clusters using RHACM

In this exercise, you will deploy the compliance operator on all the clusters in the Asia-Pacific (APAC) location to verify conformance to the Essential 8 (E8) standard. You will also review the compliance report on the RHACM web console.

Outcomes

You should be able to:

- Install the compliance operator from the RHACM governance dashboard
- Deploy the E8-scan policy
- Check the compliant scan result from `E8 scan`

Before You Begin

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

```
[student@workstation ~]$ lab start policy-compliance
```

Instructions

1. Log in to the Hub OpenShift cluster and create a `policy-compliance` project.

    1.1. Open the `terminal` application on the `workstation` machine. Log in to the Hub 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 new project named `policy-compliance`.

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

2. Log in to the RHACM web console and install the compliance operator.

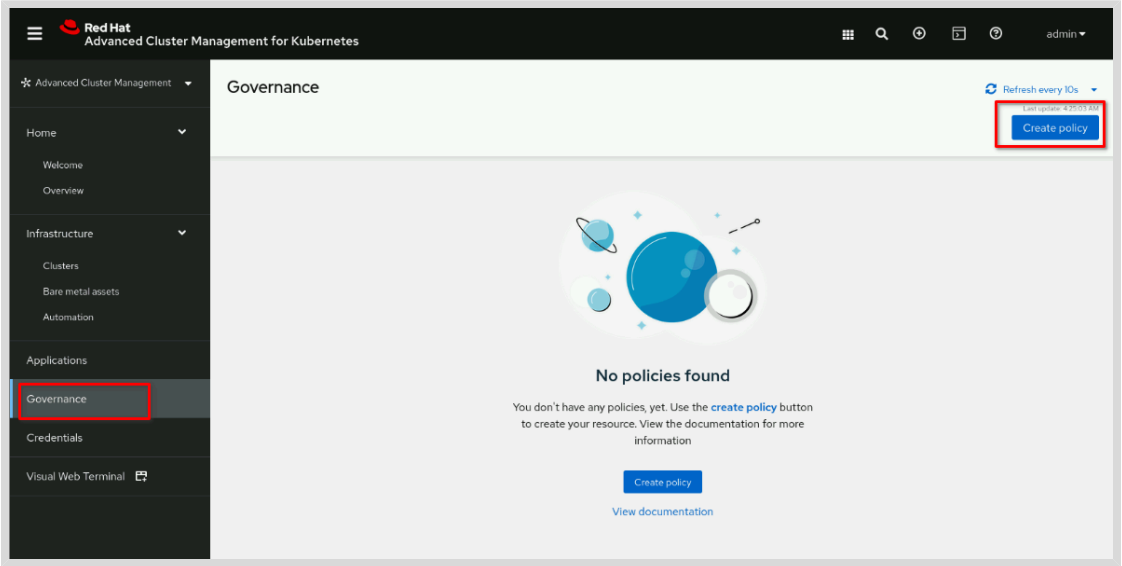
    2.1. From the `workstation` machine, open Firefox and access `https://multicloud-console.apps.ocp4.example.com`.

2.2. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.

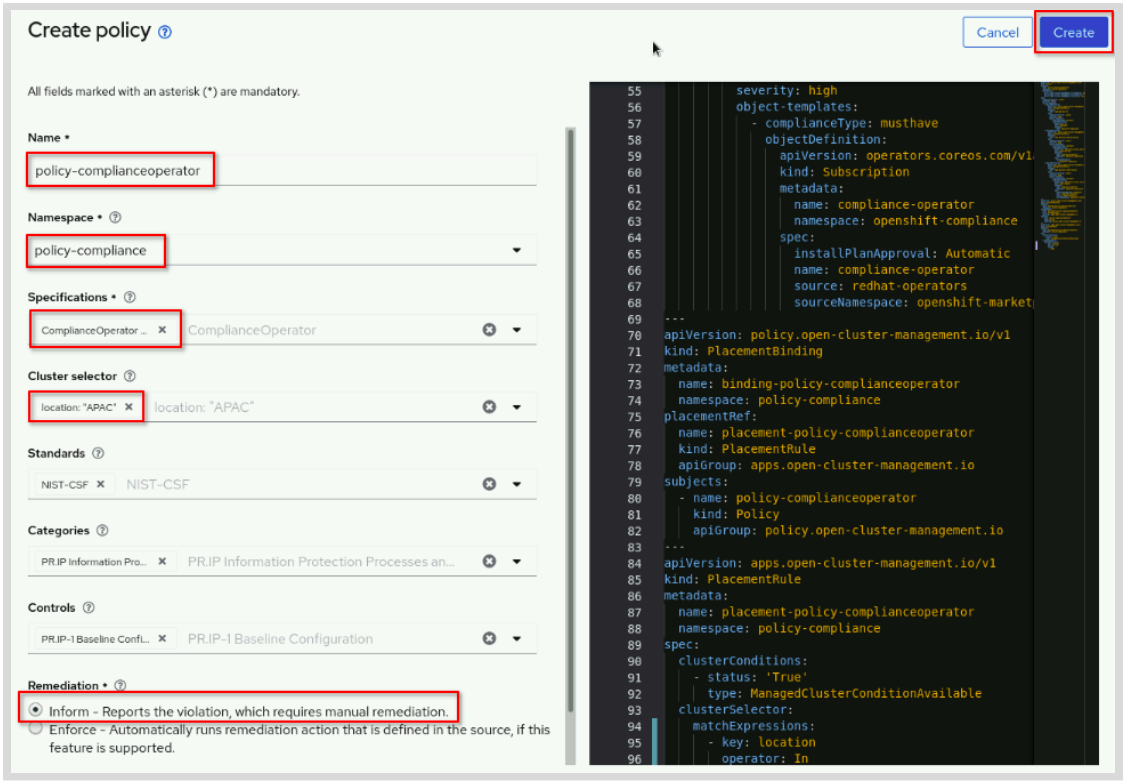
3. Create the compliance operator policy with the following parameters:

Field Name	Value
`Name`	policy-complianceoperator
`Namespace`	policy-compliance
`Specifications`	ComplianceOperator - Install the Compliance operator
`Cluster selector`	location: "APAC"
`Remediation`	Inform

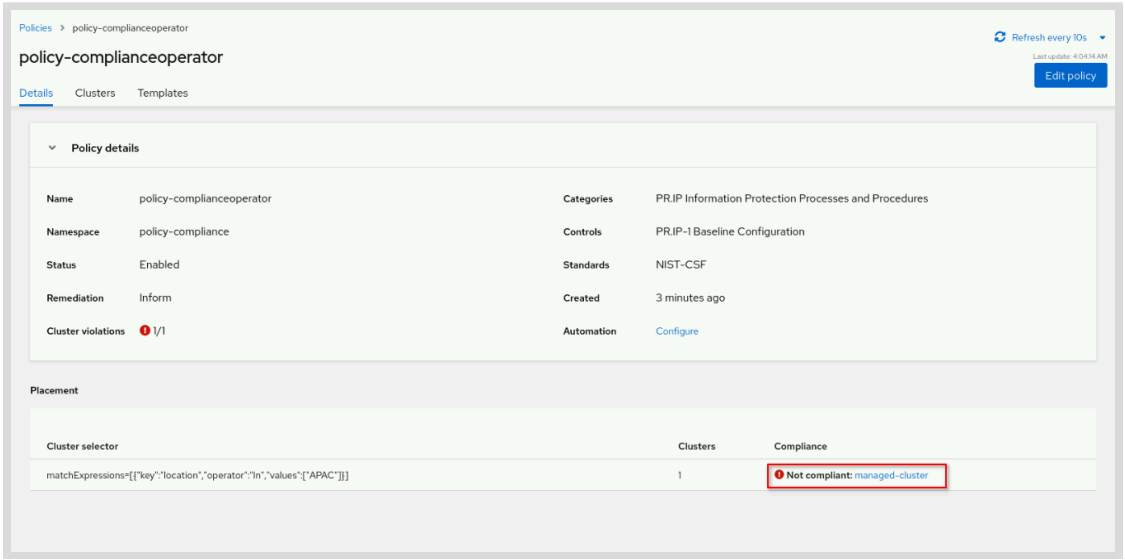
3.1. On the left pane, click **Governance** tab to display the governance dashboard, and then click **Create policy** to create the policy. The `Create policy` page is displayed.



3.2. Complete the page with the following details, leaving the other fields unchanged. Then, click **Create**.



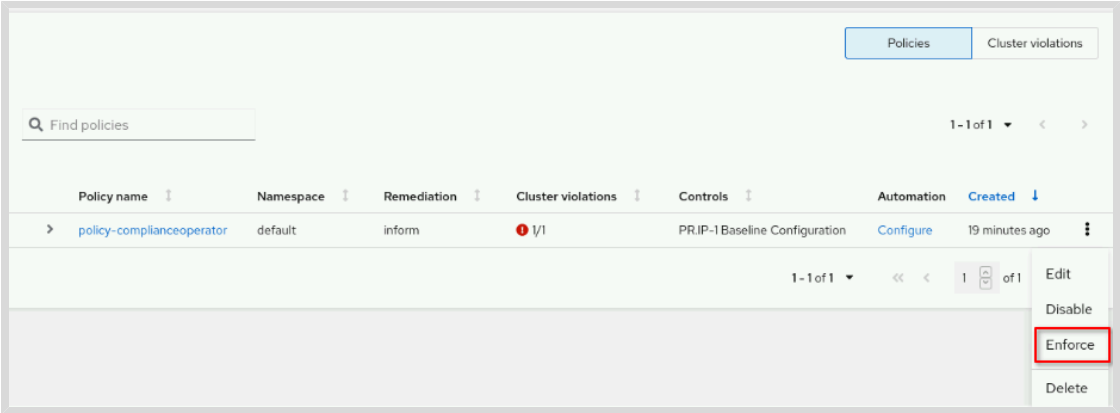
3.3. Click the `policy-complianceoperator` policy name from the list of policies to check the policy details from the governance dashboard.



The Policy status shows `Not compliant` for the cluster named `managed-cluster`. The `Not compliant` policy status indicates that the compliance operator is not installed on the `managed-cluster`.

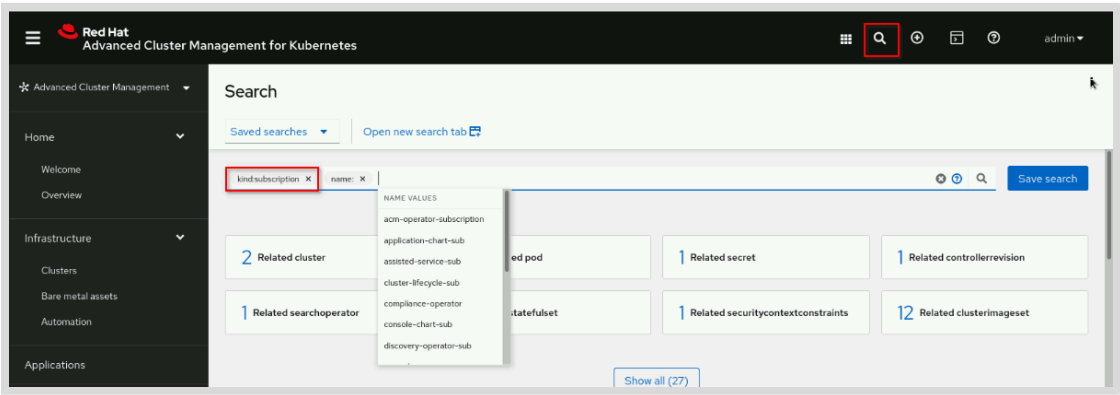
3.4. On the left pane, click **Governance** tab. The dashboard shows the `policy-complianceoperator` policy. Click vertical elipses button to the right of the policy field, and then click **Enforce**. Click **Enforce** again in the confirmation window to change policy mode from `Inform` to `Enforce`.

`Enforce` mode installs compliance operator to the `managed-cluster`, and the `policy-complianceoperator` changes to a `Compliant` status.



4. As the `admin` user, verify the compliance operator installation by using the RHACM web console.

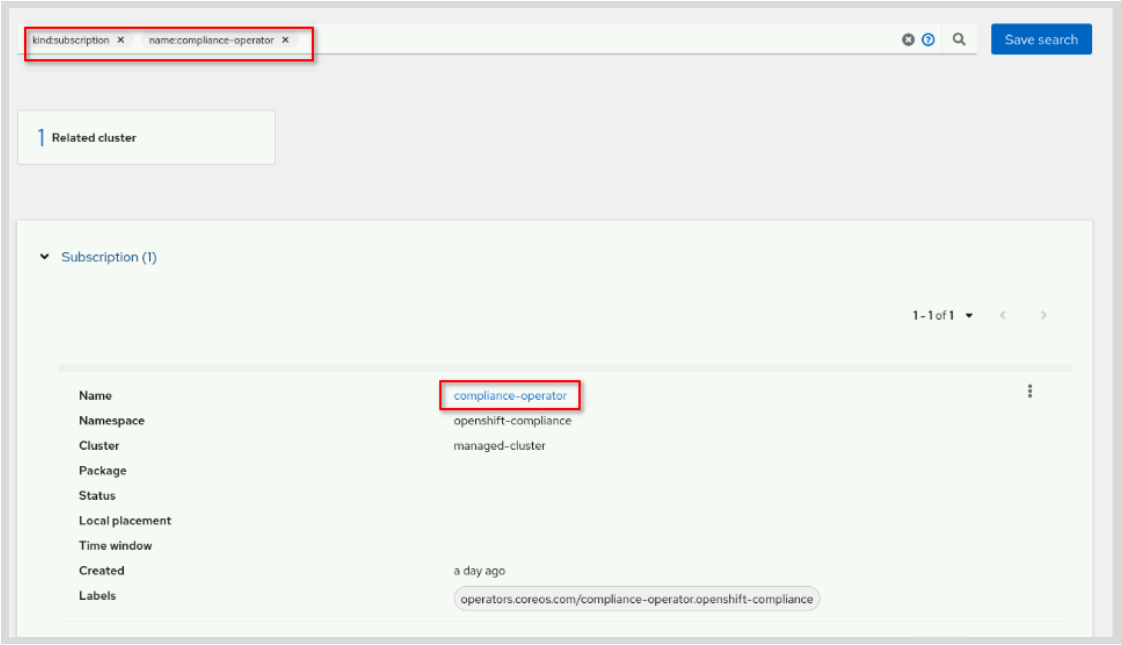
4.1. Click the search icon on the top right of the RHACM web console.



4.2. Type the following parameters in the search bar to search the compliance-operator subscription.

Field Name	Value
`kind`	subscription
`name`	compliance-operator

The search result shows the compliance-operator subscription present in the managed-cluster.



Note
APAC location has one cluster, named `managed-cluster`.

5. Clone the `policy collection` repository. The policy-collection repository has policy examples for Open cluster management.

   5.1. Open a `terminal` application on the `workstation` machine. Run the following command to clone the policy-collection repository.

```
[student@workstation ~]$ git clone \
 https://github.com/open-cluster-management/policy-collection.git
Cloning into 'policy-collection'...
...output omitted...
```

   5.2. Enter your Git repository clone directory.

```
[student@workstation ~]$ cd policy-collection
```

6. Deploy the Essential 8 (E8) scan policy on the APAC location.

   6.1. Edit the `policy-compliance-operator-e8-scan.yaml` YAML file to change the following parameters:

Field Name	Value
`remediationAction`	enforce
`key`	location
`values`	APAC

Note

The policy-collection repository has both stable and community policy examples. This course uses the E8 scan policy that is available under the **stable** → **CM-Configuration-Management** directory.

For more information, please visit https://github.com/open-cluster-management/policy-collection/blob/main/README.md.

```
[student@workstation policy-collection]$ cd stable/CM-Configuration-Management
[student@workstation CM-Configuration-Management]$ vim \
policy-compliance-operator-e8-scan.yaml
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-e8-scan
  annotations:
    policy.open-cluster-management.io/standards: NIST SP 800-53
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-6 Configuration Settings
spec:
  remediationAction: enforce
  disabled: false
...output omitted...

...output omitted...
spec:
  clusterConditions:
  - status: "True"
    type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions:
      - {key: location, operator: In, values: ["APAC"]}
```

6.2. Log in to Hub OpenShift cluster as the `admin` user, and then switch to the `policy-compliance` project.

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

...output omitted...
[student@workstation CM-Configuration-Management]$ oc project policy-compliance
...output omitted...
```

6.3. Deploy the E8-scan policy.

```
[student@workstation CM-Configuration-Management]$ oc create -f \
policy-compliance-operator-e8-scan.yaml
policy.policy.open-cluster-management.io/policy-e8-scan created
placementbinding.policy.open-cluster-management.io/binding-policy-e8-scan created
placementrule.apps.open-cluster-management.io/placement-policy-e8-scan created
```

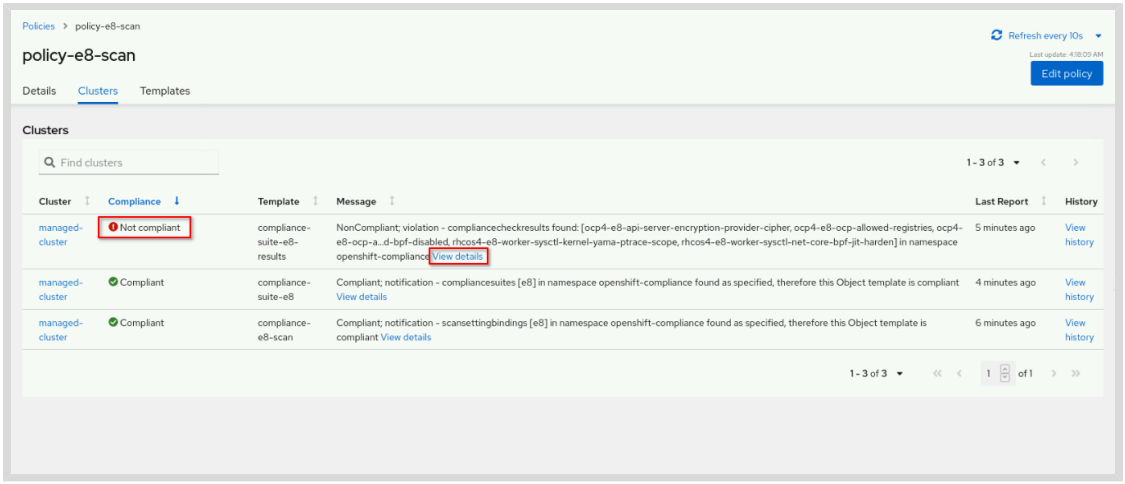
6.4. Move to the home directory.

```
[student@workstation CM-Configuration-Management]$ cd ~
[student@workstation ~]$
```

7. Check the compliance operator e8 scan results.

7.1. On the left pane, click **Governance** tab. The dashboard shows the `policy-e8-scan` policy. Click **policy-e8-scan** and then select the **Clusters** tab to check the status.

7.2. The clusters tab shows three resources: `compliance-e8-scan`, `compliance-suite-e8`, and `compliance-suite-e8-result`. `Compliance-suite-e8-result` will show `Not compliant`.

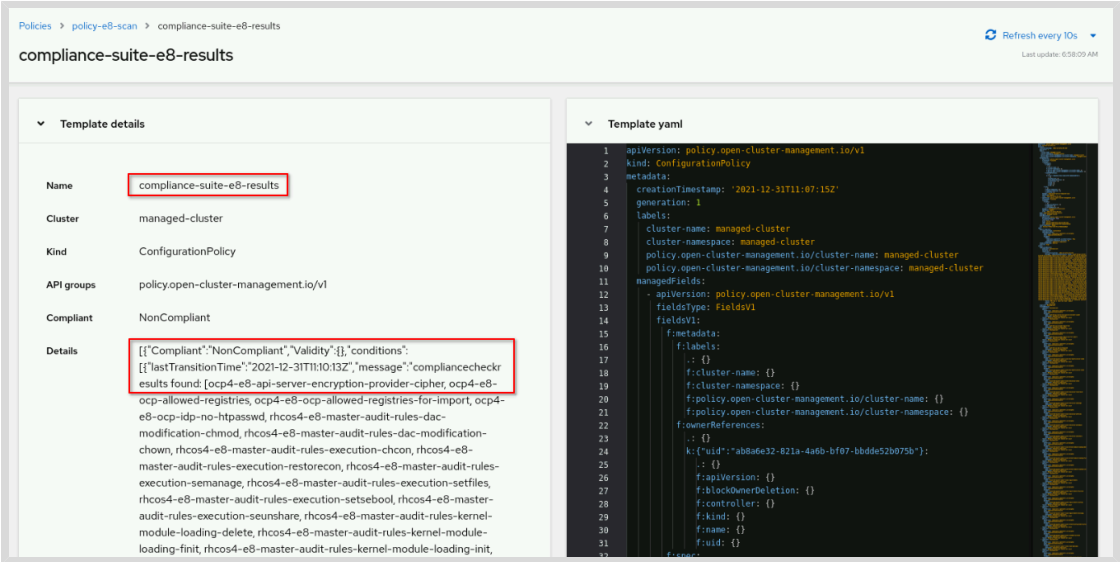


Note

It takes 2-3 minutes for `Compliance-suite-e8-result` to show `Not compliant` status.

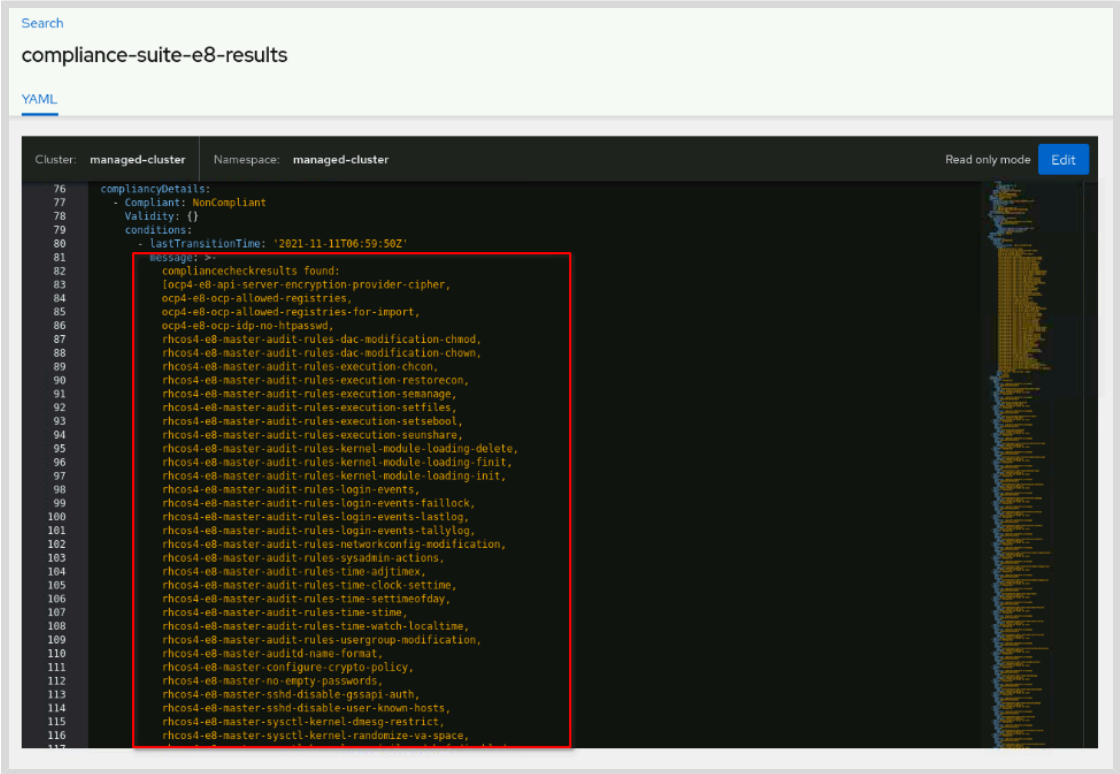
The `compliance-suit-e8-result` object is non-compliant because the policy applies the `mustnothave` condition to `thecompliancecheckresult` object.

7.3. Click **View details** and review the details. On the details page, you can see the name of non-compliant objects.



7.4. To list the compliance check results, click the **search** icon on the top right side of the RHACM web console and type `kind:configurationpolicy`, and then click **search**.

7.5. Click `compliance-suite-e8-result` and check the status. It will show a list of `NonCompliant` objects.



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 finish policy-compliance
```

This concludes the guided exercise.

Integrating RHACM with Other Policy Engines

Objectives

After completing this section, you should be able to deploy the gatekeeper policy and gatekeeper constraints to multiple clusters by using the command line and the RHACM Governance Dashboard.

Open Policy Agent vs Open Policy Agent Gatekeeper

Open Policy Agent (OPA)
Provides a summary of OPA

OPA Gatekeeper
Provides a summary of OPA Gatekeeper

OPA Gatekeeper Integration with RHACM

Introduction

Gatekeeper Integration with RHACM

Gatekeeper operator installation via RHACM Governance

Install the Gatekeeper operator via RHACM in-built policy.

Deploy Constraint with gatekeeper policy

Components of gatekeeper policy

- `ConstraintTemplate` and `Constraint`
- Audit template
- Admission template

References
For more information, refer to the _Red Hat Advanced Cluster Management policy yaml structure at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/governance/index#policy-yaml-structure

For more information, refer to the Managing gatekeeper operator policies section at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/governance/index#managing-gatekeeper-operator-policies

Guided Exercise

Integrating RHACM with Other Policy Engines

Outcomes

You should be able to:

- Deploy a gatekeeper policy from the RHACM governance dashboard.
- Deploy a gatekeeper policy to exclude given namespace for all constraints.
- Deploy a gatekeeper policy to enforce containers not to use images with the latest tag.

Before You Begin

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

```
[student@workstation ~]$ lab start policy-gatekeeper
```

Instructions

1. Log in to the Hub OpenShift cluster and create a `policy-gatekeeper` project.

   1.1. Open the `terminal` application on the `workstation` machine. Log in to the Hub 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 new project named `policy-gatekeeper`.

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

2. Log in to the RHACM web console and install the gatekeeper operator.

   2.1. From the `workstation` machine, open Firefox and access `https://multicloud-console.apps.ocp4.example.com`.

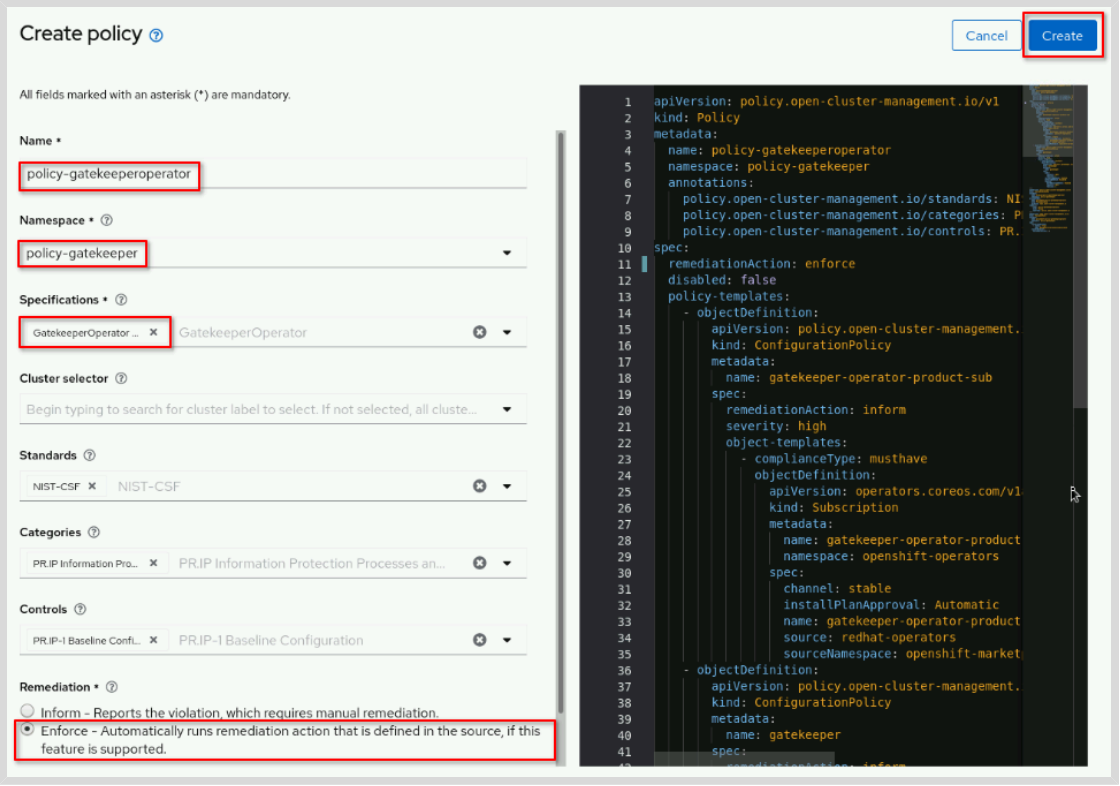
   2.2. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.

3. Create the gatekeeper operator policy with the following parameters:

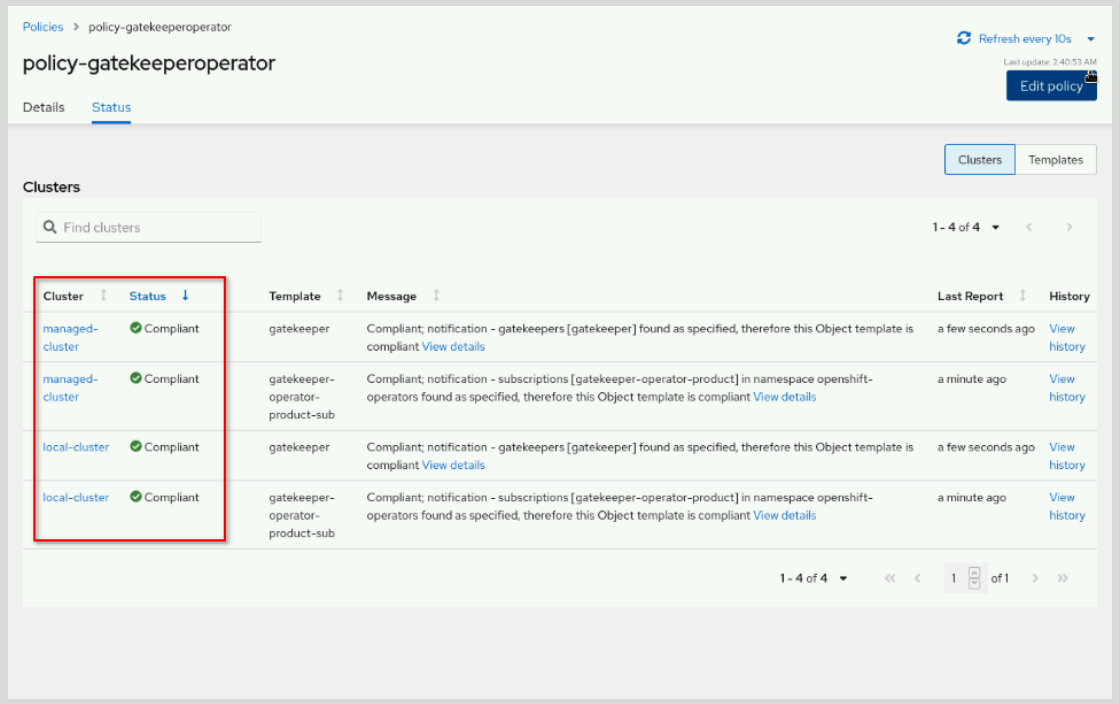
Field Name	Value
`Name`	policy-gatekeeperoperator
`Namespace`	policy-gatekeeper
`Specifications`	GatekeeperOperator - Install the Gatekeeper operator
`Remediation`	Enforce

3.1. On the left pane, click the **Governance** tab to display the governance dashboard, and then click **Create policy** to create the policy. The `Create policy` page is displayed.

3.2. Complete the page with the following details, leaving the other fields unchanged. Then, click **Create**.



3.3. On the `Governance` page, scroll down and click the `policy-gatekeeperoperator` policy name from the list of policies. Click the **Status** tab to review the policy details.



The the policy status is `Compliant` for all the clusters.

Note

It takes 2-3 minutes to install the gatekeeper operator. The policy status changes from `Not complaint` to `Complaint` after gatekeeper installation.

4. Deploy a gatekeeper policy in the `policy-gatekeeper` namespace to exclude `app-stage` namespace for all constraints in the stage environment.

   4.1. Open the `terminal` application on the `workstation` machine and change to the `~/DO480/labs/policy-gatekeeper/` directory.

```
[student@workstation ~]$ cd ~/DO480/labs/policy-gatekeeper/
```

   4.2. Edit the `policy-gatekeeper-config-exclude-namespaces.yaml` YAML file to change the following parameters:

Field Name	Value
`excludedNamespaces`	app-stage
`key`	environment
`values`	stage

Note

The policy-collection repository has both stable and community policy examples. This policy example is from the **community** → **CM-Configuration-Management** directory.

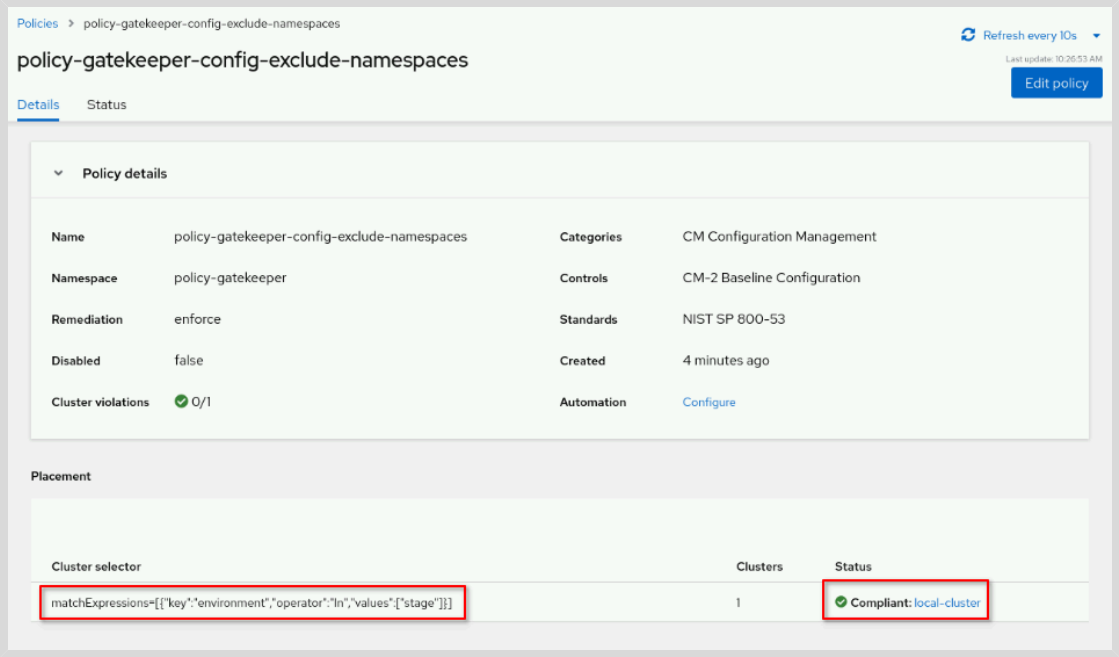
For more information, visit https://github.com/open-cluster-management/policy-collection/blob/main/README.md.

```
[student@workstation policy-gatekeeper]$ vim \
policy-gatekeeper-config-exclude-namespaces.yaml
...output omitted...
apiVersion: config.gatekeeper.sh/v1alpha1
                kind: Config
                metadata:
                  name: config
                  namespace: openshift-gatekeeper-system
                spec:
                  match:
                    - excludedNamespaces:
                        - hive
                        - kube-node-lease
                        - kube-public
                      ...output omitted...

                       ...output omitted...
                        - openshift-user-workload-monitoring
                        - openshift-vsphere-infra
                        - app-stage
                      processes:
                        - '*'
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
...output omitted...

...output omitted...
  clusterSelector:
    matchExpressions:
      - {key: environment, operator: In, values: ["stage"]}
```

4.3. Deploy the gatekeeper policy to exclude `app-stage` namespace for all constraints in the stage environment.

```
[student@workstation policy-gatekeeper]$ oc create -f \
policy-gatekeeper-config-exclude-namespaces.yaml
policy.policy.open-cluster-management.io/policy-gatekeeper-config-exclude-
namespaces created
placementbinding.policy.open-cluster-management.io/binding-policy-gatekeeper-
config-exclude-namespaces created
placementrule.apps.open-cluster-management.io/placement-policy-gatekeeper-config-
exclude-namespaces created
```

4.4. On the `Governance` page, scroll down and click the `policy-gatekeeper-config-exclude-namespaces` policy name from the list of policies.



The the policy status is `Compliant` for `stage` environment.

> **Note**
> The stage environment has one cluster, named `local-cluster`. The production environment has one cluster, named `managed-cluster`.

5. Deploy a gatekeeper policy in the `policy-gatekeeper` namespace to enforce containers not to use images with the latest tag for both `stage` and `production` environments.

5.1. Edit the `policy-gatekeeper-container-image-latest.yaml` YAML file to remove `clusterSelector` values.

```
[student@workstation policy-gatekeeper]$ vim \
policy-gatekeeper-container-image-latest.yaml
...output omitted...
spec:
  clusterConditions:
    - status: "True"
      type: ManagedClusterConditionAvailable
  clusterSelector:
    matchExpressions: []
```

Note

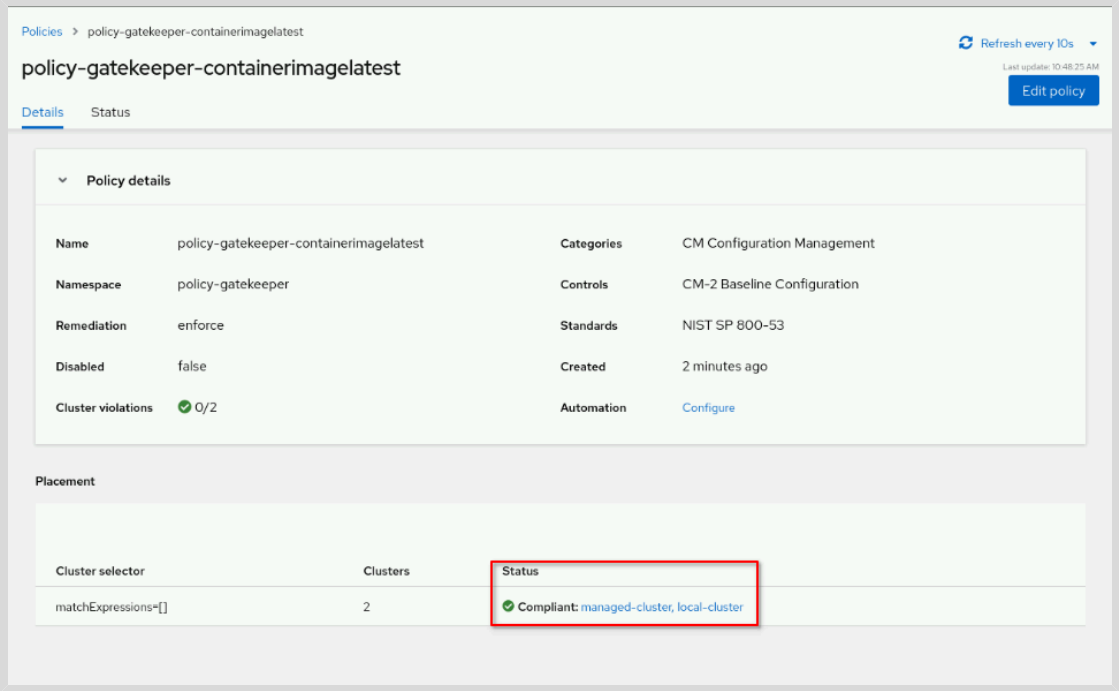
The policy-collection repository has both stable and community policy examples. This policy example is from the **community** → **CM-Configuration-Management** directory.

For more information, visit https://github.com/open-cluster-management/policy-collection/blob/main/README.md.

5.2. Deploy the gatekeeper policy to enforce containers not to use images with the latest tag for both stage and production environment.

```
[student@workstation policy-gatekeeper]$ oc create -f \
policy-gatekeeper-container-image-latest.yaml
policy.policy.open-cluster-management.io/policy-gatekeeper-containerimagelatest
 created
placementbinding.policy.open-cluster-management.io/binding-policy-gatekeeper-
containerimagelatest created
placementrule.apps.open-cluster-management.io/placement-policy-gatekeeper-
containerimagelatest created
```

5.3. On the `Governance` page, scroll down and click the `policy-gatekeeper-containerimagelatest` policy name from the list of policies.



The the policy status is `Compliant` for both `stage` and `production` environments.

> **Note**
> It takes 2-3 minutes to for policy to check all object templates. The policy status changes from `Not complaint` to `Complaint` after 2-3 minutes.

6. Deploy the `stage-hello` application to the `stage` environment by using the `~/DO480/labs/policy-gatekeeper/stage-hello.yaml` file.

   6.1. Log in to the `local-cluster` as the `admin` user.

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

...output omitted...
```

   6.2. A `stage-hello` application file is provided to create the namespace, deployment, service, and route. Review the `stage-hello.yaml` file. This application uses an image with the `latest` tag

```
[student@workstation policy-gatekeeper]$ cat stage-hello.yaml
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: app-stage
spec: {}
---
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: stage-hello
  labels:
    app: stage-hello
    name: stage-hello
  namespace: app-stage
spec:
  replicas: 1
  selector:
    matchLabels:
      app: stage-hello
      name: stage-hello
  template:
    metadata:
      labels:
        app: stage-hello
        name: stage-hello
    spec:
      containers:
      - name: stage-hello
        image: quay.io/redhattraining/do480-hello-app:latest
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: stage-hello
    name: stage-hello
  name: stage-hello
  namespace: app-stage
spec:
  ports:
  - port: 8080
  selector:
    name: stage-hello
---
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: stage-hello
  labels:
    app: stage-hello
    name: stage-hello
  namespace: app-stage
spec:
  host: stage-hello.apps.ocp4.example.com
  subdomain: ''
  to:
    kind: Service
    name: stage-hello
    weight: 100
  port:
    targetPort: 8080
  wildcardPolicy: None
```

6.3. Deploy the `stage-hello` application.

```
[student@workstation policy-gatekeeper]$ oc create -f stage-hello.yaml
project.project.openshift.io/app-stage created
deployment.apps/stage-hello created
service/stage-hello created
route.route.openshift.io/stage-hello created
```

6.4. Test the `stage-hello` application deployment.

```
[student@workstation policy-gatekeeper]$ curl stage-hello.apps.ocp4.example.com
Hello Application!
Image version : latest
```

7. Review the `policy-gatekeeper-containerimagelatest` policy status.

7.1. On the `Governance` page, scroll down and click the `policy-gatekeeper-containerimagelatest` policy name from the list of policies. Click the **Status** tab to review the policy details.

The the policy status is `Compliant` for all the clusters.

Note

The `policy-gatekeeper-containerimagelatest` policy status is `Compliant` despite using the latest tag because the `policy-gatekeeper-config-exclude-namespaces` policy excludes the `app-stage` namespace for all constraints in the stage environment.

8. Deploy the `prod-hello` application to the `production` environment by using the `~/DO480/labs/policy-gatekeeper/prod-hello.yaml` file.

8.1. Log in to the `managed-cluster` as the `admin` user.

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

...output omitted...
```

8.2. A `prod-hello` application file is provided to create the namespace, deployment, service, and route. Review the `prod-hello.yaml` file. This application uses an image with the `latest` tag

```
[student@workstation policy-gatekeeper]$ cat prod-hello.yaml
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: app-prod
spec: {}
---
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: prod-hello
  labels:
    app: prod-hello
    name: prod-hello
  namespace: app-prod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prod-hello
      name: prod-hello
  template:
    metadata:
      labels:
        app: prod-hello
        name: prod-hello
    spec:
      containers:
      - name: prod-hello
        image: quay.io/redhattraining/do480-hello-app:latest
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prod-hello
    name: prod-hello
  name: prod-hello
  namespace: app-prod
spec:
  ports:
  - port: 8080
  selector:
    name: prod-hello
---
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: prod-hello
  labels:
    app: prod-hello
    name: prod-hello
  namespace: app-prod
spec:
  host: prod-hello.apps.ocp4-mng.example.com
  subdomain: ''
  to:
    kind: Service
    name: prod-hello
    weight: 100
  port:
    targetPort: 8080
  wildcardPolicy: None
```

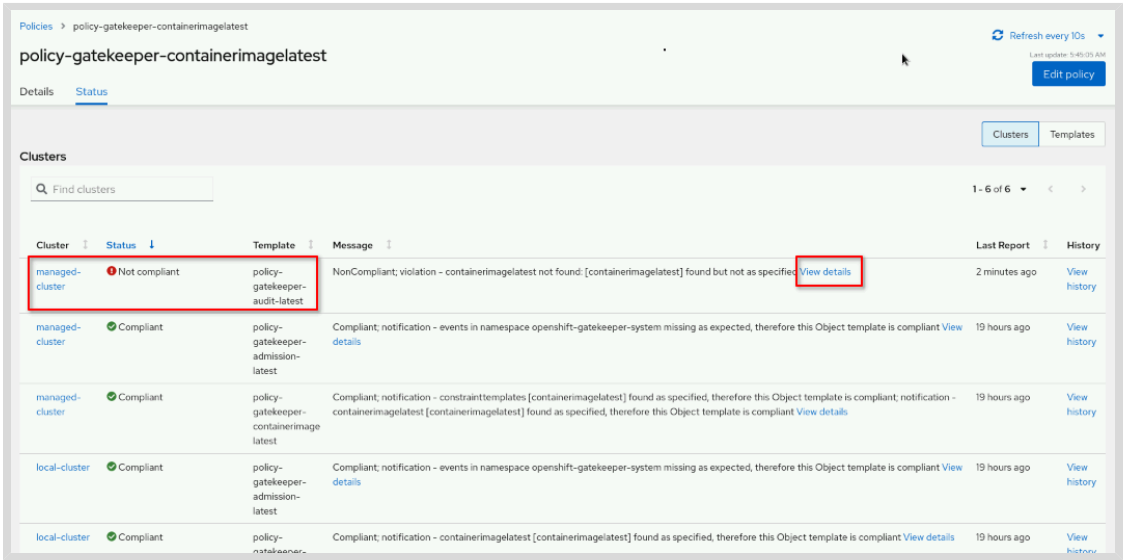
8.3. Deploy the `prod-hello` application.

```
[student@workstation policy-gatekeeper]$ oc create -f prod-hello.yaml
project.project.openshift.io/app-prod created
service/prod-hello created
route.route.openshift.io/prod-hello created
Error from server ([containerimagelatest] Deployment/prod-hello: container 'prod-
hello' is using the latest tag for its image (quay.io/redhattraining/do480-hello-
app:latest), which is an anti-pattern.
[containerimagelatest] Deployment/prod-hello: container 'prod-hello' is using the
 latest tag for its im age (quay.io/redhattraining/do480-hello-app:latest), which
 is an anti-pattern.): error when creating "prod-hello.yaml": admission webhook
 "validation.gatekeeper.sh" denied the request: [containerimagelatest] Deployment/
prod-hello: container 'prod-hello' is using the latest tag for its image (quay.io/
redhattraining/do480-hello-app:latest), which is an anti-pattern.
[containerimagelatest] Deployment/prod-hello: container 'prod-hello' is using the
 latest tag for its image (quay.io/redhattraining/do480-hello-app:latest), which
 is an anti-pattern.
```

We are unable to deploy the application because the deployment is using the latest image tag.

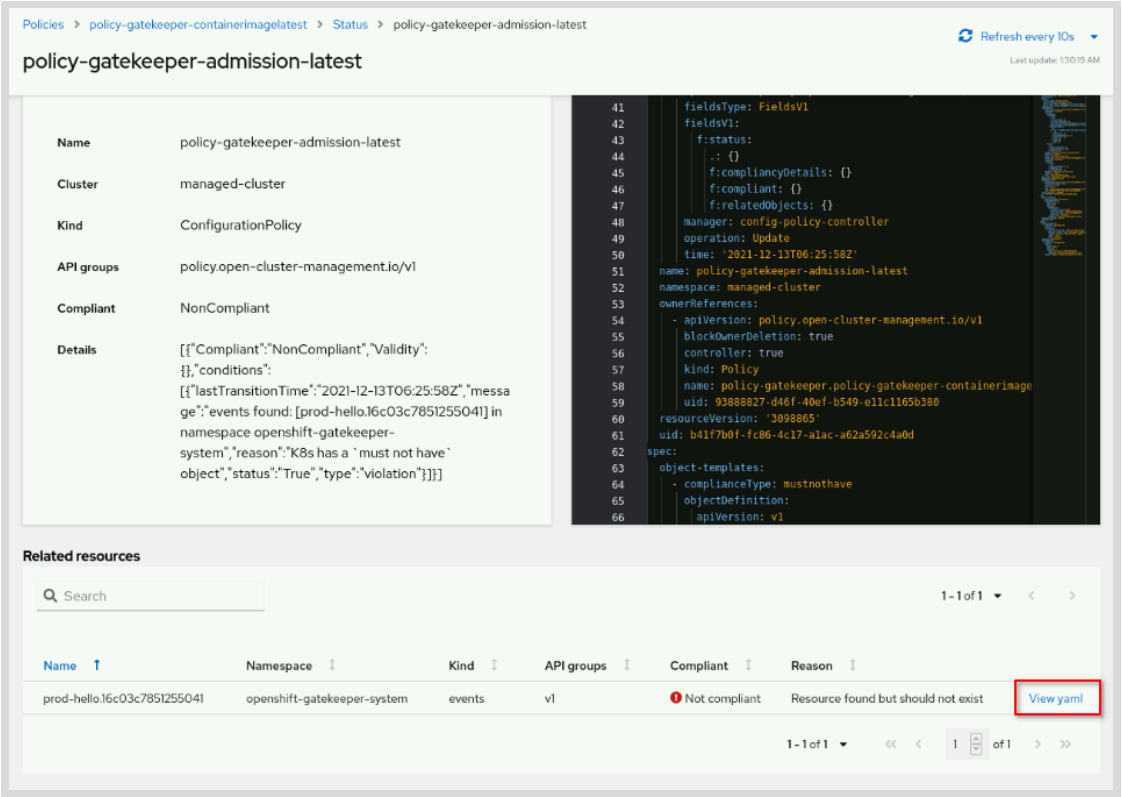
9. Review the `policy-gatekeeper-containerimagelatest` policy status.

9.1. On the `Governance` page, scroll down and click the `policy-gatekeeper-containerimagelatest` policy name from the list of policies. Click the **Status** tab to review the policy details.

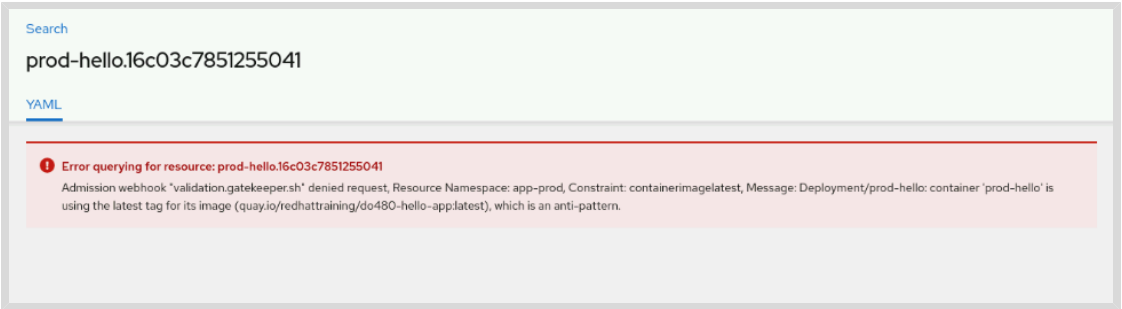


The status of the `policy-gatekeeper-admission-latest` template is `Not compliant` for `managed-cluster`.

9.2. Click **View details** and review the details. On the details page, scroll down and click **View yaml** for event objects.



9.3. The YAML file displays an error message. The error message indicates that `Deployment/prod-hello` is using the latest tag.



10. Replace the `latest` image tag with the `v1.0` version. Deploy and test the `hello-prod` application.

10.1. Log in to the `managed-cluster` as the `admin` user, and then remove the `app-prod` project.

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

...output omitted...
[student@workstation policy-gatekeeper]$ oc delete project app-prod
project.project.openshift.io "app-prod" deleted
```

10.2. Edit the `prod-hello` yaml file to use `v1.0` image tag.

```
[student@workstation policy-gatekeeper]$ vim prod-hello.yaml
...output omitted...
    spec:
      containers:
      - name: prod-hello
        image: quay.io/redhattraining/do480-hello-app:v1.0
...output omitted...
```

10.3. Deploy the `prod-hello` application.

```
[student@workstation policy-gatekeeper]$ oc create -f prod-hello.yaml
project.project.openshift.io/app-prod created
deployment.apps/prod-hello created
service/prod-hello created
route.route.openshift.io/prod-hello created
```

10.4. Test the `prod-hello` application deployment.

```
[student@workstation policy-gatekeeper]$ curl prod-hello.apps.ocp4-mng.example.com
Hello Application!
Image version : v1.0
```

10.5. Move to the home directory.

```
[student@workstation policy-gatekeeper]$ cd ~
[student@workstation ~]$
```

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 finish policy-gatekeeper
```

This concludes the guided exercise.

Lab

Deploying and Managing Policies with RHACM

Outcomes

You should be able to:

- Deploy a namespace policy from the RHACM governance dashboard.
- Deploy an IAM policy from the RHACM governance dashboard.

Before You Begin

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

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

Instructions

1. Log in to the Hub OpenShift cluster and create a `policy-review` project.
2. Log in to the RHACM web console and deploy a namespace policy in the `policy-review` namespace to ensure that the `test` namespace does not exist in the production environment.
3. Deploy an IAM policy in the `policy-review` namespace to limit the number of cluster administrators to 2 for all clusters.

Evaluation

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

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

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 finish policy-review
```

This concludes the lab.

Solution

Deploying and Managing Policies with RHACM

Outcomes

You should be able to:

- Deploy a namespace policy from the RHACM governance dashboard.
- Deploy an IAM policy from the RHACM governance dashboard.

Before You Begin

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

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

Instructions

1. Log in to the Hub OpenShift cluster and create a `policy-review` project.

1.1. Open the `terminal` application on the `workstation` machine. Log in to the Hub 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 new project named `policy-review`.

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

2. Log in to the RHACM web console and deploy a namespace policy in the `policy-review` namespace to ensure that the `test` namespace does not exist in the production environment.

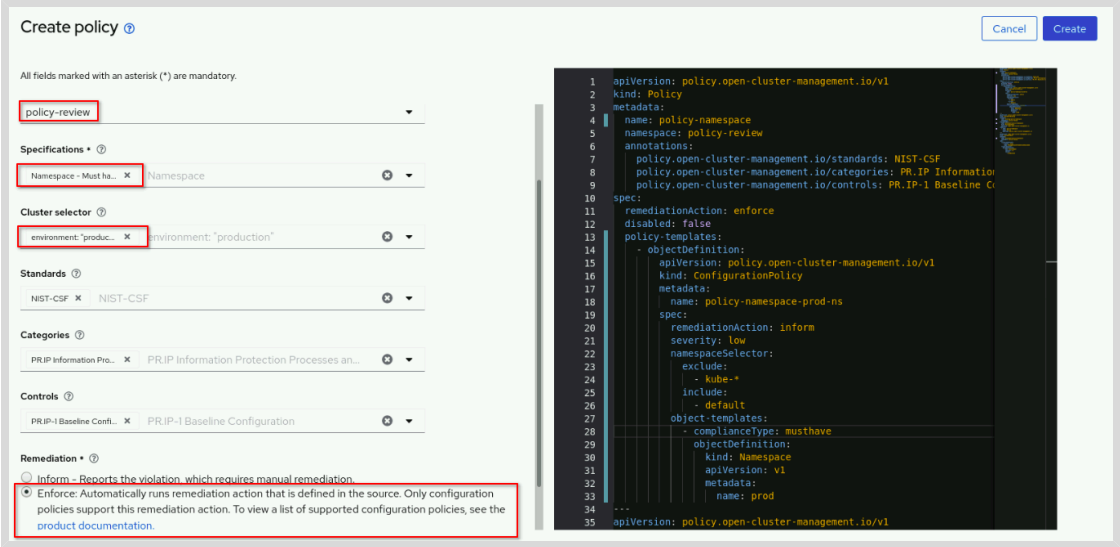
2.1. From the `workstation` machine, open Firefox and access `https://multicloud-console.apps.ocp4.example.com`.

2.2. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.

2.3. On the left pane, click the **Governance** tab to display the **Governance** dashboard, and then click **Create policy** to create the policy. The **Create policy** page is displayed.

2.4. Complete the page with the following details, leaving the other fields unchanged. Do **not** click **Create** button yet.

Field Name	Value
`Name`	policy-namespace
`Namespace`	policy-review
`Specifications`	Namespace - Must have namespace `prod`
`Cluster selector`	environment: "production"
`Remediation`	Enforce



2.5. Use the YAML editor on the right of the **Create policy** page to make the following edits to the YAML file:

```
spec:
  remediationAction: enforce
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: policy-namespace-prod-ns
        spec:
          remediationAction: inform
          severity: low
          namespaceSelector:
            exclude:
              - kube-*
            include:
              - default
          object-templates:
            - complianceType: mustnothave
              objectDefinition:
```

```
                kind: Namespace
                apiVersion: v1
                metadata:
                  name: test
```

Click **Create**.

2.6. On the `Governance` page, scroll down and click the `policy-namespace` policy name from the list of policies. Click the **Clusters** tab to review the policy details.

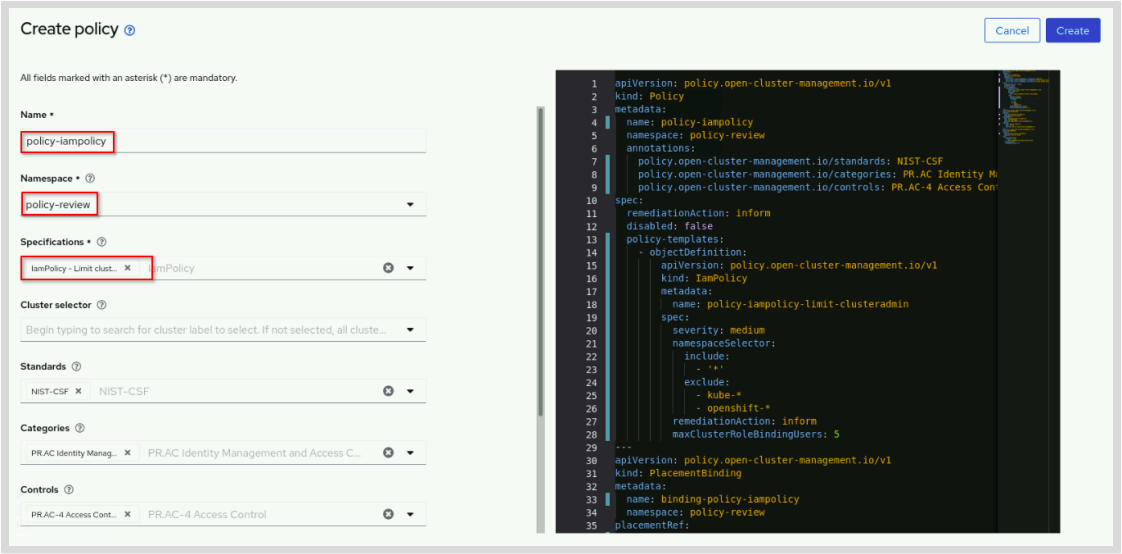
The the policy status is `Compliant` for the `managed-cluster`.

3. Deploy an IAM policy in the `policy-review` namespace to limit the number of cluster administrators to 2 for all clusters.

3.1. On the left pane, click the **Governance** tab to display the **Governance** dashboard, and then click **Create policy** to create the policy. The **Create policy** page is displayed.

3.2. Complete the page with the following details, leaving the other fields unchanged. Do **not** click **Create** button yet.

Field Name	Value
`Name`	policy-iampolicy
`Namespace`	policy-review
`Specifications`	IamPolicy - Limit clusteradmin roles



3.3. Use the YAML editor on the right of the **Create policy** page to make the following edits to the YAML file:

```
spec:
  remediationAction: inform
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: IamPolicy
        metadata:
          name: policy-iampolicy-limit-clusteradmin
        spec:
          severity: medium
          namespaceSelector:
            include:
              - '*'
            exclude:
              - kube-*
              - openshift-*
          remediationAction: inform
          maxClusterRoleBindingUsers: 2
```

Click **Create** button.

3.4. On the `Governance` page, scroll down and click the `policy-iampolicy` policy name from the list of policies. Click the **Clusters** tab to review the policy details.

The status of the `policy-iampolicy-limit-clusteradmin` template is `Not compliant` for `local-cluster` and `Compliant` for `managed-cluster`.

Evaluation

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

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

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 finish policy-review
```

This concludes the lab.

Summary

Note
This section is under development.

Chapter 4

Configuring the Observability Stack in RHACM

Goal

Gain insight into the fleet of managed clusters by using RHACM observability components.

Objectives

- Describe the architecture of the observability stack and the benefits of observability in a multicluster environment.
- Customize the RHACM observability stack.

Sections

- Introducing the RHACM Observability Stack (and Guided Exercise)
- Customizing the RHACM Observability Stack (and Guided Exercise)

Lab

Installing and Customizing the RHACM Observability Stack

Introducing the RHACM Observability Stack

Objectives

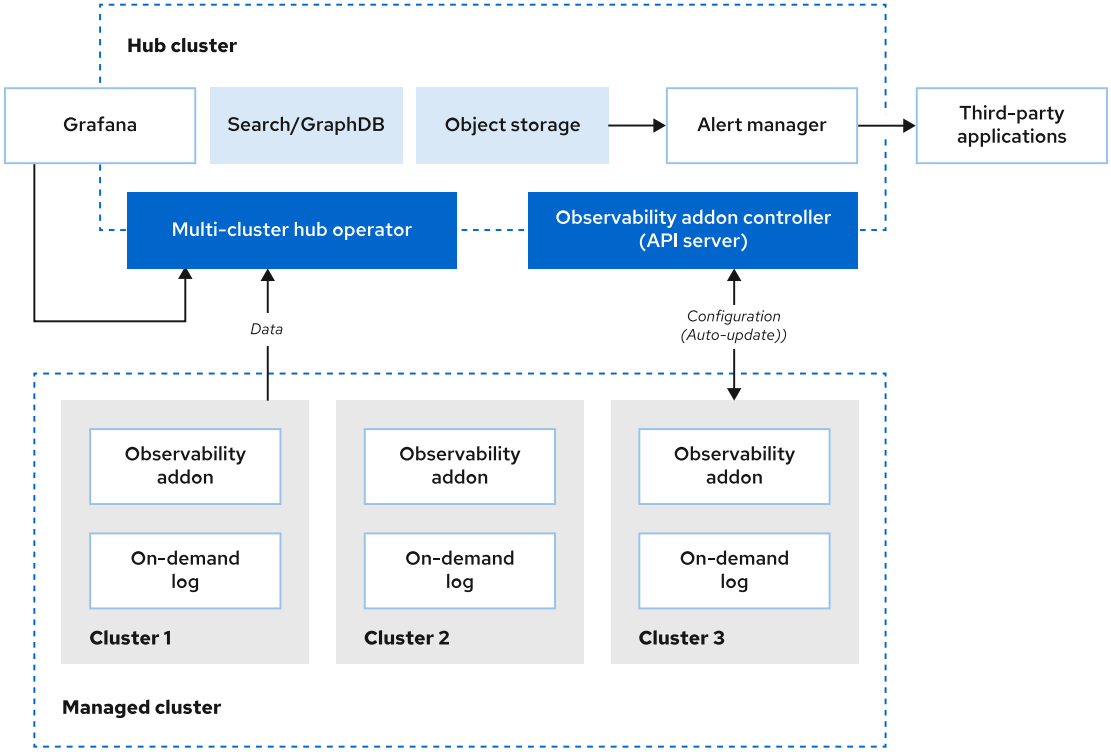
After completing this section, you should be able to describe the architecture of the observability stack and the benefits of observability in a multicluster environment.

Multicluster Observability in RHACM

The observability service in RHACM provides administrators a view across the fleet of clusters from a single window.

In production environments, monitoring, processing, and storing metrics from the fleet of clusters is crucial to anticipating issues, improving the performance of the clusters, and conducting post-mortem analysis.

The following diagram provides an architecture overview of the observability components in RHACM.



When you enable the observability service, the `multicluster-observability-operator` deploys Grafana and Alertmanager instances in the hub cluster. If not specified otherwise, it also deploys the observability addon and on-demand log components in each managed cluster.

The Grafana and Alertmanager instances deployed in the hub cluster receive, process, and display alerts and metrics from all the managed clusters with the observability addon activated.

You can add custom dashboards to Grafana. Also, as explained elsewhere in this chapter, you can create custom recording rules and alerting rules. The Alertmanager instance manages alert forwarding to third-party applications.

Differences Between the RHOCP Monitoring Stack and the RHACM Observability Service

The Grafana and Alertmanager instances provided by RHACM compliment the RHOCP cluster-level metrics.

The RHOCP monitoring stack provides out-of-the-box monitoring for core platform components. Starting with RHOCP version 4.6, you can also monitor your own projects.

The RHACM observability service is designed to provide cluster-level metrics, receiving those from the fleet of clusters. You can use the Grafana instance of the RHACM observability service to monitor cluster-level metrics from across the fleet of clusters.

Enabling the RHACM Observability Service

Due to the requirement for persistent storage, the RHACM observability service is not enabled by default. The observability service supports the following S3 compatible object stores:

- Amazon S3
- Red Hat Ceph
- Google Cloud Storage
- Azure Storage
- Red Hat OpenShift Container Storage
- Red Hat OpenShift on IBM (ROKS)

You must create a `MultiClusterObservability` custom resource in the `open-cluster-management-observability` namespace to enable the observability service. The `MultiClusterObservability` object requires a secret containing the credentials of the S3 bucket storage.

Note

For detailed instructions about enabling the observability service, refer to the *Enable observability service* section at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/observability/index#enable-observability

References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Observability Guide* at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/observability/index

Guided Exercise

Introducing the RHACM Observability Stack

In this exercise you will deploy the observability stack in all the managed clusters in RHACM and verify the installation in the hub cluster and the managed clusters. Then, you will explore the Grafana dashboard provided by the RHACM observability stack. Finally, you will disable the metrics collection in a cluster used for the stage enviromnent.

Outcomes

You should be able to:

- Install the observability stack in RHACM
- Verify the installation in all the managed clusters
- Access the RHACM Grafana dashboard
- Disable the observability agent in the `managed-cluster`

Before You Begin

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

This command ensures that RHACM is deployed in the hub cluster, and the `managed-cluster` exists in RHACM.

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

Instructions

1. Enable the observability service from the terminal.

    1.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

    1.2. Create the namespace `open-cluster-management-observability`.

```
[student@workstation ~]$ oc create namespace open-cluster-management-observability
namespace/open-cluster-management-observability created
```

1.3. Extract the pull-secret from the `openshift-config` namespace. Store it in a variable.

```
[student@workstation ~]$ DOCKER_CONFIG_JSON=`oc extract secret/pull-secret \
  -n openshift-config --to=-`
# .dockerconfigjson
[student@workstation ~]$
```

1.4. Create the pull-secret in the namespace `open-cluster-management-observability`.

```
[student@workstation ~]$ oc create secret generic \
  multiclusterhub-operator-pull-secret \
  -n open-cl uster-management-observability \
  --from-literal=.dockerconfigjson="$DOCKER_CONFIG_JSON" \
  --type=kubernetes.io/dockerconfigjson
secret/multiclusterhub-operator-pull-secret created
[student@workstation ~]$
```

1.5. Use your text editor to create a YAML file with the definition of a new object bucket claim (OBC).

```
[student@workstation ~]$ vi obc.yaml
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: thanos-bc
  namespace: open-cluster-management-observability
spec:
  storageClassName: openshift-storage.noobaa.io
  generateBucketName: observability-bucket
```

Save the file and close the editor.

Now, create the `ObjectBucketClaim` using the `oc` command.

```
[student@workstation ~]$ oc create -f obc.yaml
objectbucketclaim.objectbucket.io/thanos-bc created
```

1.6. Extract the storage configuration details and the secret.

```
[student@workstation ~]$ oc extract configmap/thanos-bc \
  -n open-cluster-management-observability --to=-
# BUCKET_HOST
s3.openshift-storage.svc
# BUCKET_NAME
observability-bucket-eabe4aca-0d93-49b2-a3dc-ce8e902ec48b
# BUCKET_PORT
443
# BUCKET_REGION

# BUCKET_SUBREGION
```

```
[student@workstation ~]$ oc extract secret/thanos-bc \
  -n open-cluster-management-observability --to=-
# AWS_SECRET_ACCESS_KEY
2wNow7IHpJsSxUR9c+OU1CmCIJVkChSe0ZoFrT7V
# AWS_ACCESS_KEY_ID
T6mbdR5Hg6Gf7U2PpXzE
```

1.7. Create a secret containining the `thanos.yaml` configuration file with the information from the previous step.

```
[student@workstation ~]$ vi secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: s3
    config:
      bucket: observability-bucket-eabe4aca-0d93-49b2-a3dc-ce8e902ec48b
      endpoint: s3.openshift-storage.svc
      insecure: true
      access_key: T6mbdR5Hg6Gf7U2PpXzE
      secret_key: 2wNow7IHpJsSxUR9c+OU1CmCIJVkChSe0ZoFrT7V
```

Use the oc command to create the secret in RHOCP.

```
[student@workstation ~]$ oc create -f secret.yaml
secret/thanos-object-storage created
```

1.8. Create the multicluster observability object to enable the observability service.

```
[student@workstation ~]$ vi mcobs.yaml
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  observabilityAddonSpec: {}
  storageConfig:
    metricObjectStorage:
      name: thanos-object-storage
      key: thanos.yaml
```

Use the oc command to create the multicluster observability instance in RHOCP.

```
[student@workstation ~]$ oc create -f mcobs.yaml
multiclusterobservability.observability.open-cluster-management.io/observability
 created
```

At this point, the observability components deployment starts. You can check the progress of the installation using the following command:

```
[student@workstation ~]$ oc describe mco observability
Name:         observability

...output omitted...

Status:
  Conditions:
    Last Transition Time:  2021-11-09T13:34:59Z
    Message:               Installation is in progress
    Reason:                Installing
    Status:                True
    Type:                  Installing

...output omitted...
```

2. Verify the deployment of the observability components in all the managed clusters.

   2.1. Verify the completion of the installation in the `ocp4` cluster, named `local-cluster` in RHACM.

   After a few minutes installing, the status of the observability components transitions to `Ready`.

```
[student@workstation ~]$ oc describe mco observability
Name:         observability

...output omitted...

Status:
  Conditions:
    Last Transition Time:  2021-11-09T13:34:59Z
    Message:               Installation is in progress
    Reason:                Installing
    Status:                True
    Type:                  Installing
    Last Transition Time:  2021-11-09T13:36:32Z
    Message:               Observability components are deployed and running
    Reason:                Ready
    Status:                True
    Type:                  Ready
Events:                    <none>
```

   2.2. Verify the status of the components of the `observability` addon in the cluster named `managed-cluster` in RHACM.

   First, log in to the `ocp4-mng` cluster.

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

Then, retrieve the objects of the namespace `open-cluster-management-addon-observability`.

```
[student@workstation ~]$ oc get all -n open-cluster-management-addon-observability
NAME                                                  READY   STATUS    RESTARTS
  AGE
pod/endpoint-observability-operator-7dd7595ff8-hn8qc   1/1     Running   0
  13m
pod/metrics-collector-deployment-7c5cf96599-cfh5s      1/1     Running   0
  12m

NAME                                             READY   UP-TO-DATE   AVAILABLE
 AGE
deployment.apps/endpoint-observability-operator   1/1     1            1
 13m
deployment.apps/metrics-collector-deployment      1/1     1            1
 12m

NAME                                                        DESIRED   CURRENT
 READY   AGE
replicaset.apps/endpoint-observability-operator-7dd7595ff8   1         1         1
       13m
replicaset.apps/metrics-collector-deployment-546d85d5c6      0         0         0
       12m
replicaset.apps/metrics-collector-deployment-74d5469859      0         0         0
       12m
replicaset.apps/metrics-collector-deployment-7c5cf96599      1         1         1
       12m
replicaset.apps/metrics-collector-deployment-868984fb6d      0         0         0
       12m
```

Note

If the status of the pods is not **Active**, look for error messages in the logs of the pods, or the events in the namespace.

```
[student@workstation ~]$ oc logs pod/metrics-collector-deployment-7c5cf96599-
cfh5s -n open-cluster-management-addon-observability
```

```
[student@workstation ~]$ oc get events -n open-cluster-management-addon-
observability
```

3. Navigate to the Grafana dashboard available in the **Home** → **Overview** menu of the RHACM console. Analyze the capacity and utilization data provided about CPU and memory of the clusters.

Note

After enabling the observability components, RHACM shows direct access to the Grafana dashboard from within the RHACM console.

3.1. Open Firefox and navigate to the RHACM console in `multicloud-console.apps.ocp4.example.com`.

When prompted, select **htpasswd_provider**

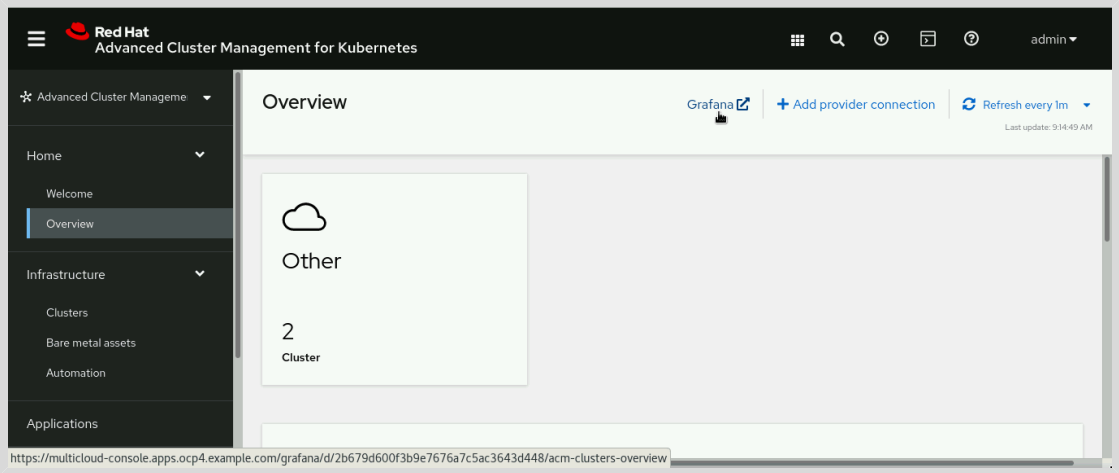
Type the credentials:

- Username: `admin`
- Password: `redhat`

And then click **Log in**

From within the RHACM web console, click **Home** → **Overview**.

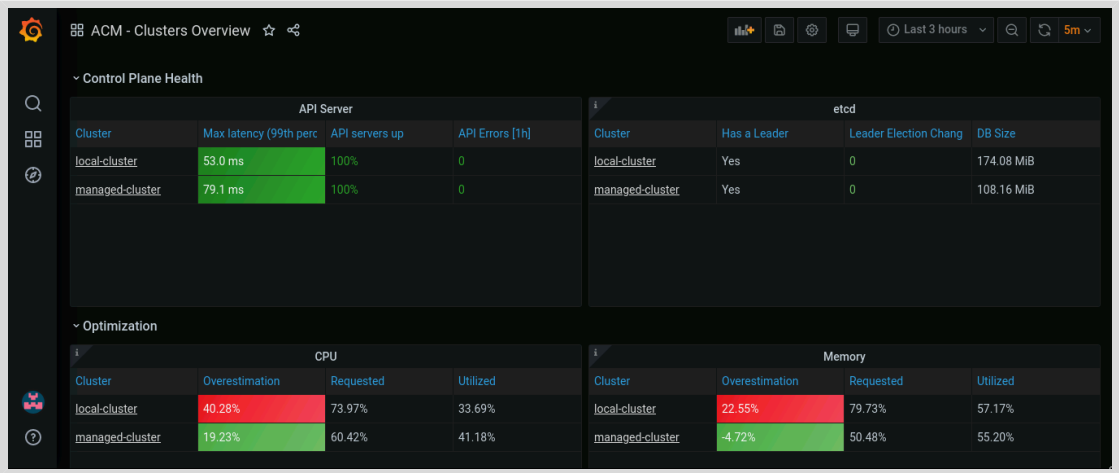
Finally, click **Grafana**



Note

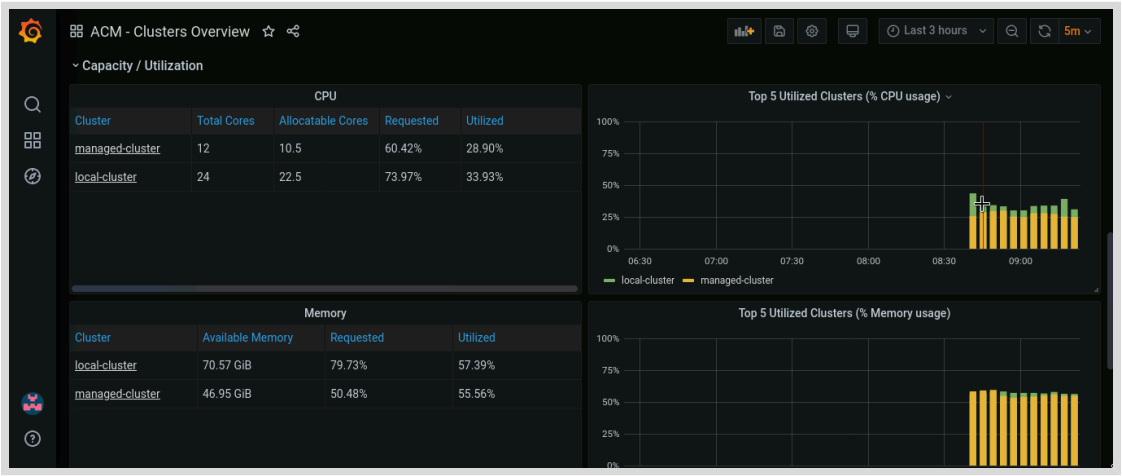
Reload the Grafana dashboard if the panels show any error message.

Finally, you can see the Grafana dashboard with the default configuration.



The Grafana dashboard of RHACM contains information of all the clusters in the fleet. The observability addon, deployed in all the managed clusters, sends information to the central Grafana instance deployed in the hub cluster.

3.2. From within the Grafana dashboard, scroll down to find the `Capacity / Utilization` table.



Grafana gathers information about CPU and memory utilization across all the clusters in the fleet. This way, a system administrator has a single pane of glass to monitor all the custers from the same dashboard.

4. Disable the metrics collection from the `managed-cluster`. This is useful to avoid overloading the observability service with metrics from clusters used for stage environments.

4.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

4.2. Add the label `observability=disabled` to the `managedcluster` object named `managed-cluster`.

```
[student@workstation ~]$ oc label managedcluster managed-cluster \
  observability=disabled -n open-cluster-management
managedcluster.cluster.open-cluster-management.io/managed-cluster labeled
```

After applying this label, RHACM removes the objects in the `open-cluster-management-addon-observability` namespace.

4.3. Verify that the changes in the `managed-cluster` are applied. Log in to the `ocp4-mng` cluster as the `admin` user. The APIServer URL is `https://api.ocp4-mng.example.com:6443`.

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

4.4. Retrieve the objects in the `open-cluster-management-addon-observability` namespace.

```
[student@workstation ~]$ oc get all \
  -n open-cluster-management-observability-addon
No resources found in open-cluster-management-observability-addon namespace.
```

The expected result is that there are not resources found in the `open-cluster-management-observability-addon` namespace.

Note

The Grafana dashboard can show cached information about the `managed-cluster` after disabling the observability addon. After some time, all the information about the `managed-cluster` dissapers.

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 finish observability-install
```

This concludes the guided exercise.

Customizing the RHACM Observability Stack

Objectives

After completing this section, you should be able to customize the RHACM observability stack.

Observability Service Customizations in RHACM

To benefit from the RHACM observability stack, you can add customizations to the Grafana and Alertmanager instances.

Prometheus recording rules

Recording rules allow you to precalculate expensive expressions. The result is saved as a new set of time series that you can use to create custom Grafana dashboards.

Note

For more information, visit the Prometheus recording rules documentation at https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/

Prometheus alerting rules

Alerting rules allow you to receive alerts based on determined conditions.

Note

For more information, visit the Prometheus alerting rules documentation at https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

Prometheus custom metrics

You can create additional custom metrics to be collected from all the managed clusters.

Note

For more information, review the *Adding custom metrics* section at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/observability/index#adding-custom-metrics

Forwarding Metrics to External Services

Typically, organizations have a centralized alerting engine to receive alerts from different systems.

To facilitate this centralized alerting, you can configure extra alert receivers as explained in the Prometheus Alertmanager documentation at https://prometheus.io/docs/alerting/latest/configuration/.

References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Observability Guide* at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.3/html-single/observability/index

Guided Exercise

Customizing the RHACM Observability Stack

In this exercise you will configure an existing RHACM observability service to handle a higher load of monitoring data from the managed clusters. Then, you will create a custom alerting rule that fires when the CPU requests of a cluster are above 70% of the total allocatable CPU. Finally, you will disable the RHACM observability service.

Outcomes

You should be able to:

- Increase the replicas of the observability metrics receiver pods
- Verify that the number of pods increases
- Create a prometheus custom alerting rule
- Verify that the alerts are firing
- Remove the observability components from RHACM

Before You Begin

Before attempting this guided exercise, ensure you have completed the guided exercise `Introducing the RHACM Observability Stack`.

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

This command ensures that RHACM is deployed in the hub cluster, the `managed-cluster` exists in RHACM, and the observability service is enabled.

```
[student@workstation ~]$ lab start observability-customize
```

Instructions

1. From the terminal, increase to six the replicas of the observability metrics receiver pods.

1.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

1.2. Use the `oc` command to edit the `multiclusterobservability` object in the namespace `openshift-multicluster-observability`. In the `spec` section, add the highlighted snippet to set the number of replicas of the receiver pods to six.

```
[student@workstation ~]$ oc edit multiclusterobservability \
  -n openshift-multicluster-observability
...output omitted...
spec:
  advanced:
    receive:
      replicas: 6
  enableDownsampling: true
```

Save the file and close the editor.

Note

After exiting the editor, the following message displays:

```
multiclusterobservability.observability.open-cluster-management.io/
observability edited
```

1.3. Use the `watch` command to verify that there are six pods of the `observability-thanos-receive-default` statefulset.

```
[student@workstation ~]$ watch oc get pods \
  -n open-cluster-management-observability
...output omitted...
NAME                                                  READY   STATUS
 RESTARTS   AGE
...output omitted...
observability-thanos-receive-default-0                1/1     Running   0
      55m
observability-thanos-receive-default-1                1/1     Running   0
      54m
observability-thanos-receive-default-2                1/1     Running   0
      54m
observability-thanos-receive-default-3                1/1     Running   0
      2m56s
observability-thanos-receive-default-4                1/1     Running   0
      2m51s
observability-thanos-receive-default-5                1/1     Running   0
      2m46s
...output omitted...
```

Notice that three of the pods are created recently, and the other three were already running.

Note

Increasing the replicas of the receiver pods is necessary in production environments when the number of managed clusters increases.

2. Create a Prometheus alerting rule so that cluster administrators receive alerts when the CPU requests of any cluster are above 70% of the available compute capacity.

2.1. From the terminal, create a file named `custom_rule.yaml` containing the instructions in PromQL.

```
[student@workstation ~]$ vi custom-rules.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: thanos-ruler-custom-rules
data:
  custom_rules.yaml: |
    groups:
      - name: cluster-health
        rules:
        - alert: ClusterCPUReq-70
          annotations:
            summary: Notify when CPU requests on a cluster are greater than the
 defined utilization limit
            description: "The cluster {{ $labels.cluster }} has an elevated
 percentage of CPU requests: {{ $labels.clusterID }}."
          expr: |
            sum(namespace_cpu:kube_pod_container_resource_requests:sum) by
 (clusterID, cluster, prometheus) /
 sum(kube_node_status_allocatable{resource="cpu"}) by (clusterID, cluster,
 prometheus) > 0.7
          for: 5s
          labels:
            cluster: "{{ $labels.cluster }}"
            prometheus: "{{ $labels.prometheus }}"
            severity: critical
```

2.2. Create the `thanos-ruler-custom-rules` configmap in the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc apply -f custom-rules.yaml \
  -n open-cluster-management-observability
configmap/thanos-ruler-custom-rules created
```

The creation of the configmap triggers a restart of the `observability-thanos-rule-`*X* pods in the `open-cluster-management-observability` namespace.

2.3. Verify that the new configuration applies. You can monitor the restart of the pods in the `open-cluster-management-observability` namespace with the `watch` command.

```
[student@workstation ~]$ watch oc get pods \
  -n open-cluster-management-observability
...output omitted...
NAME                                                     READY   STATUS
 RESTARTS   AGE
...output omitted...
observability-thanos-rule-0                              2/2     Running   0
      1m
observability-thanos-rule-1                              2/2     Running   0
      1m
observability-thanos-rule-2                              2/2     Running   0
      2m
...output omitted...
```

Exit the `watch` command with `Ctrl+C`.

After the `observability-thanos-rule-X` pods restart, the alerting rule is visible from the RHACM Grafana dashboard.

3. Verify that the custom alert is firing for the cluster named `local-cluster`.

3.1. Use Firefox to navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`.

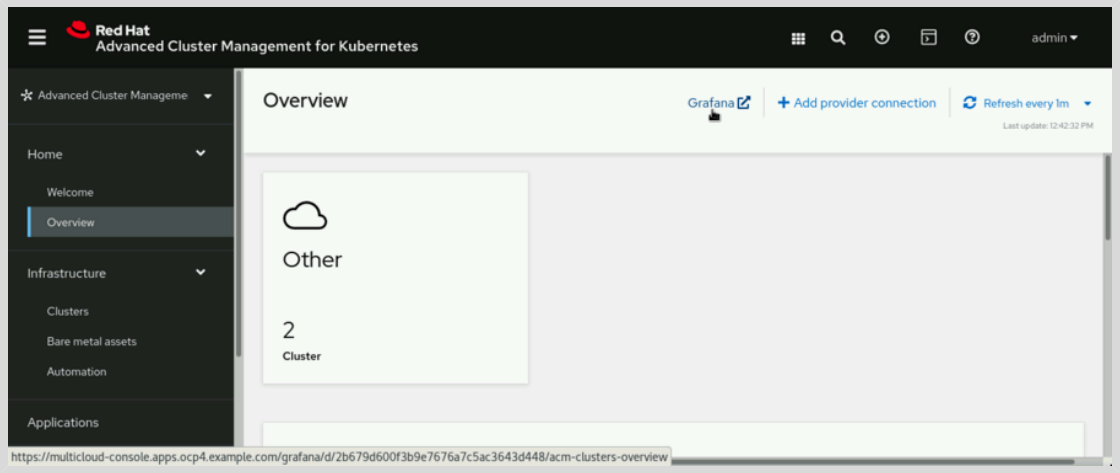
When prompted, select **htpasswd_provider**

Type the credentials:

- Username: `admin`
- Password: `redhat`

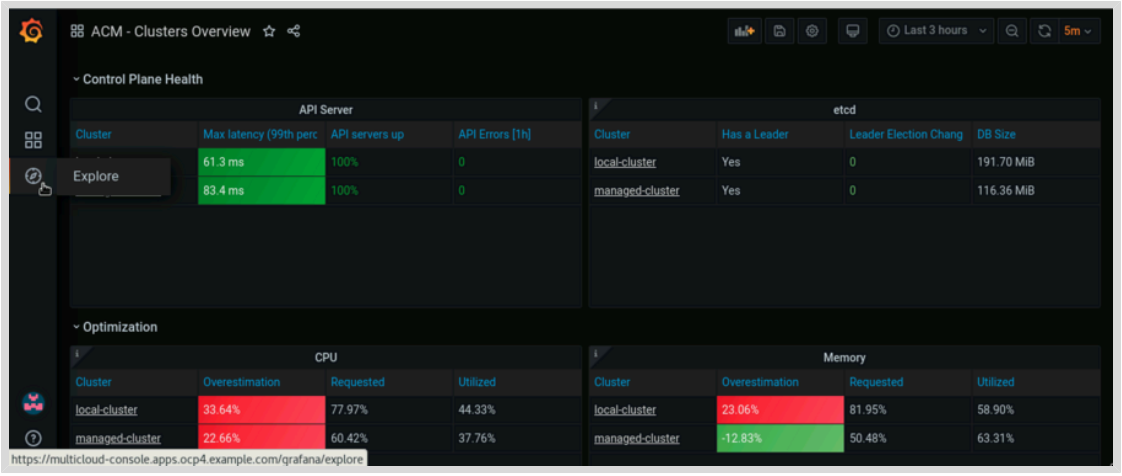
And then click **Log in**

3.2. Navigate to **Home** → **Overview** and click **Grafana**.

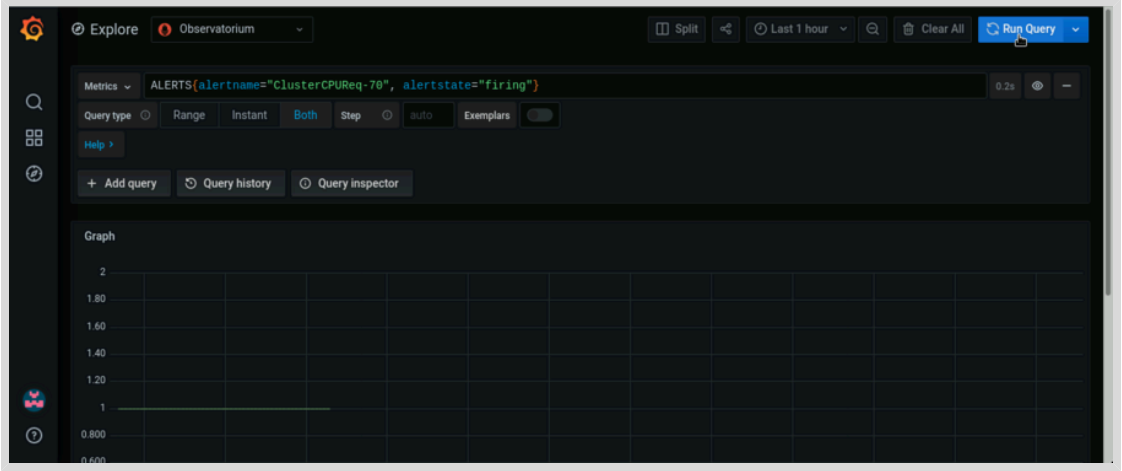


A new tab showing the Grafana dashboard opens in the browser.

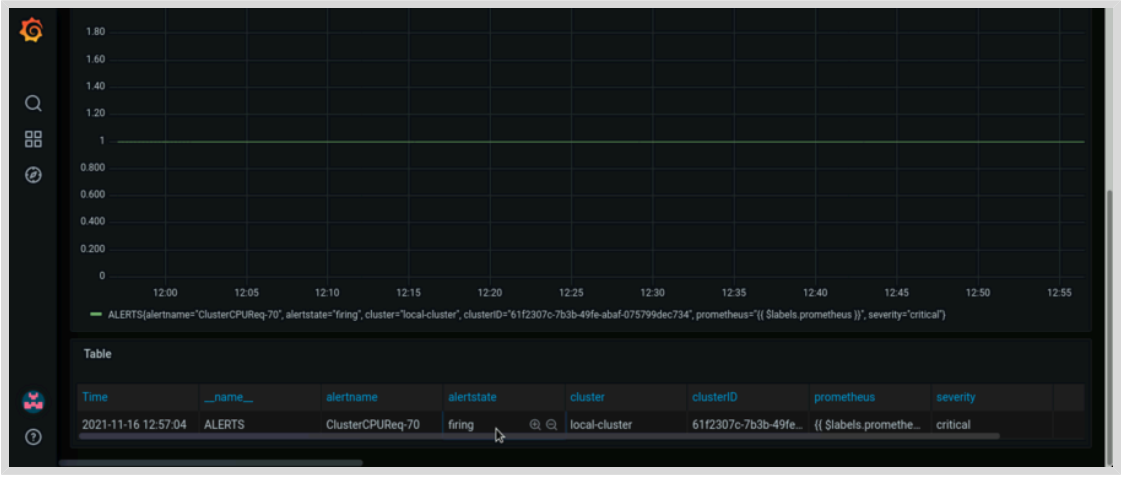
3.3. On the left menu of the Grafana dashboard, click **Explore**



3.4. In the `Metrics` text field, type `ALERTS{alertname="ClusterCPUReq-70", alertstate="firing"}` and click **Run Query**.



Scroll down to the `Table` field and verify that the alert with name `ClusterCPUReq-70` is `firing` in the cluster named `local-cluster`.



4. Disable the RHACM observability service.

4.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

4.2. Use the `oc` to remove the `MultiClusterObservability` object named `observability`.

```
[student@workstation ~]$ oc delete multiclusterobservability \
  observability
multiclusterobservability.observability.open-cluster-management.io "observability"
 deleted
```

4.3. Remove the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc delete namespace \
  open-cluster-management-observability
namespace "open-cluster-management-observability" deleted
```

Note

The deletion of the `open-cluster-management-observability` namespace can take a while.

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 finish observability-customize
```

This concludes the guided exercise.

Lab

Installing and Customizing the RHACM Observability Stack

In this lab, you will enable the RHACM observability service in all the managed clusters. Then, you will create a custom alerting rule that fires a warning when the memory usage in a cluster exceeds 50%. You will verify that the alert is firing for every cluster that meets the condition. Finally, you will disable the observability service.

Outcomes

You should be able to:

- Install the observability service in RHACM
- Create a Prometheus custom alerting rule
- Verify that the alert is firing in the Grafana dashboard
- Remove the observability components from RHACM

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 RHACM is deployed in the hub cluster, the `managed-cluster` exists in RHACM, and the observability service is not enabled.

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

Instructions

1. Enable the observability service from the terminal.
2. Create a Prometheus alerting rule so that cluster administrators receive alerts when the memory usage of a cluster exceeds 50%.
3. From Grafana, verify that the custom alert is firing for the clusters `local-cluster` and `managed-cluster`.
4. Disable the RHACM observability service.

Evaluation

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

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

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 finish observability-review
```

This concludes the lab.

Solution

Installing and Customizing the RHACM Observability Stack

In this lab, you will enable the RHACM observability service in all the managed clusters. Then, you will create a custom alerting rule that fires a warning when the memory usage in a cluster exceeds 50%. You will verify that the alert is firing for every cluster that meets the condition. Finally, you will disable the observability service.

Outcomes

You should be able to:

- Install the observability service in RHACM
- Create a Prometheus custom alerting rule
- Verify that the alert is firing in the Grafana dashboard
- Remove the observability components from RHACM

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 RHACM is deployed in the hub cluster, the `managed-cluster` exists in RHACM, and the observability service is not enabled.

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

Instructions

1. Enable the observability service from the terminal.

   1.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

   1.2. Create the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc create namespace open-cluster-management-observability
namespace/open-cluster-management-observability created
```

1.3. Extract the pull secret from the `openshift-config` namespace. Store it in a variable.

```
[student@workstation ~]$ DOCKER_CONFIG_JSON=`oc extract secret/pull-secret \
  -n openshift-config --to=-`
# .dockerconfigjson
[student@workstation ~]$
```

1.4. Create the pull secret in the namespace `open-cluster-management-observability`.

```
[student@workstation ~]$ oc create secret generic \
  multiclusterhub-operator-pull-secret \
  -n open-cl uster-management-observability \
  --from-literal=.dockerconfigjson="$DOCKER_CONFIG_JSON" \
  --type=kubernetes.io/dockerconfigjson
secret/multiclusterhub-operator-pull-secret created
[student@workstation ~]$
```

1.5. Use your text editor to create a YAML file with a new object bucket claim (OBC) definition.

```
[student@workstation ~]$ vi obc.yaml
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:

  name: thanos-bc
  namespace: open-cluster-management-observability
spec:
  storageClassName: openshift-storage.noobaa.io
  generateBucketName: observability-bucket
```

Save the file and close the editor.

Now, create the `ObjectBucketClaim` object by using the `oc` command.

```
[student@workstation ~]$ oc create -f obc.yaml
objectbucketclaim.objectbucket.io/thanos-bc created
```

1.6. Extract the storage configuration details and the secret.

```
[student@workstation ~]$ oc extract configmap/thanos-bc \
  -n open-cluster-management-observability --to=-
# BUCKET_HOST
s3.openshift-storage.svc
# BUCKET_NAME
observability-bucket-eabe4aca-0d93-49b2-a3dc-ce8e902ec48b
# BUCKET_PORT
443
# BUCKET_REGION

# BUCKET_SUBREGION
```

```
[student@workstation ~]$ oc extract secret/thanos-bc \
  -n open-cluster-management-observability --to=-
# AWS_SECRET_ACCESS_KEY
2wNow7IHpJsSxUR9c+OU1CmCIJVkChSe0ZoFrT7V
# AWS_ACCESS_KEY_ID
T6mbdR5Hg6Gf7U2PpXzE
```

1.7. Create a secret containing the `thanos.yaml` configuration file with the information from the previous step.

```
[student@workstation ~]$ vi secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: s3
    config:
      bucket: observability-bucket-eabe4aca-0d93-49b2-a3dc-ce8e902ec48b
      endpoint: s3.openshift-storage.svc
      insecure: true
      access_key: T6mbdR5Hg6Gf7U2PpXzE
      secret_key: 2wNow7IHpJsSxUR9c+OU1CmCIJVkChSe0ZoFrT7V
```

Use the oc command to create the secret in RHOCP.

```
[student@workstation ~]$ oc create -f secret.yaml
secret/thanos-object-storage created
```

1.8. Create the multicluster observability object to enable the observability service.

```
[student@workstation ~]$ vi mcobs.yaml
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  observabilityAddonSpec: {}
  storageConfig:
    metricObjectStorage:
      name: thanos-object-storage
      key: thanos.yaml
```

Use the oc command to create the multicluster observability instance in RHOCP.

```
[student@workstation ~]$ oc create -f mcobs.yaml
multiclusterobservability.observability.open-cluster-management.io/observability
 created
```

The observability components deployment begins. Verify the status of the installation by using the following command:

```
[student@workstation ~]$ oc describe mco observability
Name:         observability

...output omitted...

Status:
  Conditions:
    Last Transition Time:  2021-11-09T13:34:59Z
    Message:               Installation is in progress
    Reason:                Installing
    Status:                True
    Type:                  Installing

...output omitted...
```

When the observability components are ready, the output of the previous command displays the following message:

```
Name:         observability
...output omitted...
Status:
  Conditions:
Last Transition Time:  2021-11-25T08:29:07Z
    Message:               Observability components are deployed and running
    Reason:                Ready
    Status:                True
    Type:                  Ready
...output omitted...
```

2. Create a Prometheus alerting rule so that cluster administrators receive alerts when the memory usage of a cluster exceeds 50%.

   2.1. From the terminal, create a file named `custom_rule.yaml` containing the ConfigMap object with the instructions in Prometheus Querying Language (PromQL).

```
[student@workstation ~]$ vi custom-rules.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: thanos-ruler-custom-rules
data:
  custom_rules.yaml: |
    groups:
      - name: cluster-health
        rules:
        - alert: MemoryUsage-50
          annotations:
            summary: Notify when the memory usage in idle conditions is greater
 than 50%.
            description: "The cluster {{ $labels.cluster }} has more than 50% of
 the memory in use."
```

```
          expr: |
            cluster:memory_usage:ratio > 0.5
          for: 5s
          labels:
            cluster: "{{ $labels.cluster }}"
            severity: warning
```

2.2. Create the `thanos-ruler-custom-rules` ConfigMap object in the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc apply -f custom-rules.yaml \
  -n open-cluster-management-observability
configmap/thanos-ruler-custom-rules created
```

Creating the ConfigMap object triggers a restart of the `observability-thanos-rule-`*X* pods in the `open-cluster-management-observability` namespace.

2.3. Verify that the new configuration applies. You can monitor the restart of the pods in the `open-cluster-management-observability` namespace with the `watch` command.

```
[student@workstation ~]$ watch oc get pods \
  -n open-cluster-management-observability
...output omitted...
NAME                                                  READY   STATUS
 RESTARTS   AGE
...output omitted...
observability-thanos-rule-0                           2/2     Running   0
      1m
observability-thanos-rule-1                           2/2     Running   0
      1m
observability-thanos-rule-2                           2/2     Running   0
      2m
...output omitted...
```

Exit the `watch` command by pressing `Ctrl+C`.

After the `observability-thanos-rule-`*X* pods restart, the alerting rule is visible from the RHACM Grafana dashboard.

3. From Grafana, verify that the custom alert is firing for the clusters `local-cluster` and `managed-cluster`.

3.1. Use Firefox to navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`.

When prompted, select **htpasswd_provider**.

Type the credentials:

- Username: `admin`
- Password: `redhat`

Then, click **Log in**.

3.2. Navigate to **Home** → **Overview** and click **Grafana**.

A new tab showing the Grafana dashboard opens in the browser.

On the Grafana dashboard left menu, click **Explore**.

In the `Metrics` text field, type `ALERTS{alertname="MemoryUsage-50", alertstate="firing"}` and click **Run Query**.

Scroll down to the `Table` field and verify that the alert with name `MemoryUsage-50` is `firing` in both clusters.

4. Disable the RHACM observability service.

4.1. Open a terminal and log in to the `ocp4` cluster as the `admin` user. The APIServer URL is `https://api.ocp4.example.com:6443`.

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

4.2. Use the `oc` command to remove the `MultiClusterObservability` object named `observability`.

```
[student@workstation ~]$ oc delete multiclusterobservability \
  observability
multiclusterobservability.observability.open-cluster-management.io "observability"
 deleted
```

4.3. Remove the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc delete namespace \
  open-cluster-management-observability
namespace "open-cluster-management-observability" deleted
```

Note

Deleting the `open-cluster-management-observability` namespace can take a while.

Evaluation

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

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

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 finish observability-review
```

This concludes the lab.

Summary

Note
This section is under development.

Chapter 5

Managing Applications Across Multiple Clusters with ACM

Goal

Deploy and manage applications in a multicluster environment with ACM GitOps

Objectives

- Describe and define GitOps concepts and the resources of the RHACM application model.
- Deliver applications into multiple clusters by using RHACM GitOps.
- Describe Kustomize concepts, features, and deploy new customizations with the Kustomize command-line tool.

Sections

- Introducing RHACM GitOps and the Application Model (and Quiz)
- Managing Multicluster Application Resources with RHACM (and Guided Exercise)
- Customizing Resources with Kustomize for RHACM (and Guided Exercise)

Lab

Managing Applications Across Multiple Clusters with RHACM

Introducing RHACM GitOps and the Application Model

Objectives

After completing this section, you should be able to describe and define GitOps concepts and the resources of the RHACM application model.

Introducing GitOps Concepts

GitOps is a framework that consists of procuring the very best practices from Devops and applying them to Infrastructure as Code (IaC). DevOps organizations can deploy code to production clusters effectively and efficiently twenty four hours a day. GitOps implements similar practices such as version control, code review, and CI/CD to automate the process of managing and provisioning infrastructures. GitOps strives to be highly organized maintaining resource files in a repository, such as Git. The resource files located in the Git repository are referred to as the *source of truth*. In other words, the Git repository contains the resources files that generate the same infrastructure environment every time it's deployed, similar to source code used to build binaries. In today's rapidly growing infrastructures, it is essential to have the capability to manage cloud resources that are required for continuous deployments and integration.

GitOps Principles

Implementing and maintaining GitOps principles are essential to your multicluster environment and consists of the following:

- **Declarative desired state**

Systems are required to use declarative data to export the desired state of the system. The data must be in a format that can be written and read by both machines and humans alike.

- **Immutable desired state versions**

The Git version control system acts as a *single source of truth* and all declarative states are stored in Git. In Git, the system infrastructure changes are available chronologically thus assisting users with troubleshooting, auiditing, and rollbacks.

GitOps Components

- **Infrastructure as code**

Infrastructure as code is the practice of storing all configurations for managing and provisioning infrastructure in a repository. This ensures that the same environment is provisioned every time. Similar to any software source code, all files under version control can be modified and then deployed. IaC is used for automating infrastructure provisioning and thereby removes the requirement for developers to manually provision and manage resources such as storage and operating systems.

- **Merge request**

Merge Requests are used effectively as a catalyst for all the infrastructure updates. Merge request enable the project developers to collaborate and review IaC prior to deployment to production systems.

- **Continuous integration**

Continuous Integration (CI) is the discipline of **integrating changes in the main branch as often as possible**. Developers use short-lived branches or small change sets and integrate them frequently into the main branch, ideally several times a day. This speeds up the integration, makes code review easier, and reduces potential problems.

Continuous Integration normally entails validation of integrated changes by using automated integration tests. After the changes have been integrated and validated, the team can then decide when to deploy a new release. By itself, Continuous Integration does not involve automating the release process.

Continuous Integration Tools

At a high level, you need the following tools to implement Continuous Integration:

- **A version control system**, which stores your source code in a repository. The repository uses a *main* or *trunk* branch to keep track of the main development line. Feature development occurs in separate branches, which after review and validation, are integrated into the main branch. The most popular version control system is Git.

- **A CI automation service**. This is usually a server or a daemon that watches the repository for changes. If the repository changes, the CI automation service checks out the new code and verifies that everything is correct. RHACM accomplishes this by deploying the subscription operator to monitor Git for respository changes.

GitOps Workflow

GitOps workflows use Git as the version control system which contains the configurations for managing and provisioning infrastructures. GitOps and IaC use a declarative approach as opposed to an imperative approach to configuration management. A declarative model in programming focuses on writing code to describe what an application should do, while an imperative model focus on how it should do it.

A typical GitOps workflow consists on the following:

1. The GitOps team stores all IaC resource files Git.

2. A branch is created from the *main* repository which will contain the updated resource file.

3. After the branch is updated, the administrator pushes the change to Git and creates a pull request.

4. A GitOps colleague reviews the code to ensure proper configuration.

5. The updated code is then approved and merged.

6. The updated branch is then deployed via the subscription operator in RHACM.

Introducing the RHACM Application Model

The foundation of the application model is the ability to subscribe to channel repositories that contain the resources for managing deployments. The application resources definitions are created, managed, and updated using YAML file spec sections. The following inumerates resources available for the application model:

Subscriptions

The subscription (subscription.apps.open-cluster-management.io) enables clusters to subscribe to a source repository, also known as a `channel`. The subscription type can be a `Git` repository, `Helm` release registry, or `Object storage` repository. Both the hub and managed cluster can use subscriptions. Subscriptions are associated with the channel and identifies new or updated resource templates.

The subscription operator pulls from the source repository and deploys to the targeted managed clusters without checking the hub cluster first. The subscription operator can then monitor for new or updated resources. For example, the following displays a typical subscription YAML file.

```
apiVersion: apps.open-cluster-management.io/v1 (1)
kind: Subscription (2)
metadata:
  name: web-app-cluster1 (3)
  namespace: web-app (4)
  labels:
    deployment: hello
  annotations:
    apps.open-cluster-management.io/github-branch: main (5)
    apps.open-cluster-management.io/github-path: mysql (6)
spec:
  channel:  web-app/web-app-channel (7)
  placement:
    placementRef:
      kind: PlacementRule (8)
      name: cluster1 (9)
```

(1) The `API version` is a required value and set to `apps.open-cluster-management.io/v1`.

(2) The `Subscription` is a required value and indicates the `kind` or type of resource.

(3) The `name` is a required value used for identifying the subscription.

(4) The `namespace` is a required value and sets the namespace resource to use for the subscription.

(5) `github-branch` specifies the branch that contains the deployment.

(6) `github-path` specifies the path to the resource configuration files.

(7) An Optional value that sets namespace name `Namespace/Name` and defines the channel for the subscription.

(8) The `PlacementRule` is a required value and indicates that the resource is a placement rule

(9) The `name` is a required value that indicates the name of the cluster where the rule is placed.

Channels

The application channels *channel.apps.open-cluster-management.io* are source definitions of repositories that the cluster can access. The channels can be a Github repository, Helm charts, objects stores,and deployable resource namespaces on the hub cluster. With the exception of

Github, all channels require individual namespaces. The following examples illustrates a typical channel configuration YAML file:

```
apiVersion: apps.open-cluster-management.io/v1 #API name of the object.
kind: Channel
metadata:
  name: web-app-channel
  namespace: web-app
spec:
  pathname: 'https://github.com/redhattraining/do480-apps' (1)
  type: GitHub (2)
```

(1) The `pathname` is a required value that specifies the URL to the repository that contains the IaC resource files.

(2) The `type` is a required value thats sets the type of repository selected to contain the IaC resource files.

Placement Rules

Placement rules *placementrule.apps.open-cluster-management.io* are used to ensure that the application subscription is placed on the correct cluster in your infrastructure. Placement rules assist in managing a multicluster deployment and can be shared across multiple subscriptions. For example the following illustrates a typical placement rule YAML file.

```
apiVersion: apps.open-cluster-management.io/v1 #API name of the object
kind: PlacementRule
metadata:
  name: cluster1
  namespace: web-app
spec:
  clusterSelector: (1)
    matchLabels: (2)
      clusterid: cluster1
```

(1) The `clusterSelector` is an optional value and identifies the target clusters.

(2) The `matchLabel` is an optional value that indicates the labels that must exist on the target clusters.

Applications

Applications *application.app.k8s.io* in Red Hat Advanced Cluster Management for Kubernetes, are used for grouping Kubernetes resources that build an application. For example the following illustrates a typical application YAML file.

```
apiVersion: app.k8s.io/v1 #API name of the object
kind: Application
metadata:
  name: web-app
  namespace: web-app (1)
spec:
```

```
  selector:
    matchLabels:  (2)
      app: web-app
```

(1) The `namespace` resource identifies the namespace to use for the application.

(2) The `matchLabel` resource is optional and specifies the label that must exist on the target clusters.

References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Applications guide* at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4

Quiz

Introducing RHACM GitOps and the Application Model

1. **Which one of the following statements describing GitOps best practices is the most correct?**
   a. GitOps is a combination of both an internal registry and automated processes.
   b. GitOps implements the best practices such as version control, code review, and CI/CD to manually provision and monitor pods.
   c. GitOps implements the best practices such as version control, code review, and CI/CD to automate the process of managing and provisioning infrastructures.
   d. GitOps implements the best practices such as version control, code review, and CI/CD to virtualize the process of managing and provisioning infrastructures.

2. **Which of the following statements describes best practices for maintaining GitOps resource YAML files?**
   a. Resource YAML files are stored on multiple systems, similar to block chains.
   b. Resource YAML files are stored in a multicluster Postgres database.
   c. Resource YAML files are stored on the local storage of infrastructure nodes.
   d. Resource YAML files are stored in a respository such as Git.

3. **Which of the following sentences list the three major GitOps components?**
   a. Infrastructure as code (IaC), Centralized Processes, Continuous Delivery/Continuous Integration (CD/CI).
   b. Continuous Delivery/Continuous Integration (CD/CI), Pull Requests, Peer Review.
   c. Infrastructure as code (IaC), Merge Request, Continuous Delivery/Continuous Integration (CD/CI).
   d. Declarative Configurations, Resource YAML files, Infrastructure as code (IaC).
   e. Application channels lists repositories that the clusters can access.
   f. Application channels lists are only stored in Helm charts.
   g. Application channels ensure the application is deployed on a specific cluster.
   h. Application channels group resources together to build a deployment.

4. **Which one of the following statements about GitOps workflows is correct?**
   a. GitOps workflows use an imperative programming model.
   b. GitOps workflows use both Git and SVN as version control system.
   c. GitOps workflows use a declarative approach to configuration management.
   d. GitOps workflows use a systematic approach to infrastructure deployment.

5. Which one of the following statements best describes Infrastructure as code (IaC)?

a. Infrastructure as code (IaC) is the practice of storing all configurations for managing and provisioning infrastructure on each OpenShift node.

b. Infrastructure as code (IaC) ensures that the same infrastructure is not provisioned every time.

c. Infrastructure as code (IaC) enables and encourages administrators to to manually provision and manage resources.

d. Infrastructure as code (IaC) is used for automating provisioning.

Solution

Introducing RHACM GitOps and the Application Model

1. **Which one of the following statements describing GitOps best practices is the most correct?**
   a. GitOps is a combination of both an internal registry and automated processes.
   b. GitOps implements the best practices such as version control, code review, and CI/CD to manually provision and monitor pods.
   c. GitOps implements the best practices such as version control, code review, and CI/CD to automate the process of managing and provisioning infrastructures.
   d. GitOps implements the best practices such as version control, code review, and CI/CD to virtualize the process of managing and provisioning infrastructures.

2. **Which of the following statements describes best practices for maintaining GitOps resource YAML files?**
   a. Resource YAML files are stored on multiple systems, similar to block chains.
   b. Resource YAML files are stored in a multicluster Postgres database.
   c. Resource YAML files are stored on the local storage of infrastructure nodes.
   d. Resource YAML files are stored in a respository such as Git.

3. **Which of the following sentences list the three major GitOps components?**
   a. Infrastructure as code (IaC), Centralized Processes, Continuous Delivery/Continuous Integration (CD/CI).
   b. Continuous Delivery/Continuous Integration (CD/CI), Pull Requests, Peer Review.
   c. Infrastructure as code (IaC), Merge Request, Continuous Delivery/Continuous Integration (CD/CI).
   d. Declarative Configurations, Resource YAML files, Infrastructure as code (IaC).
   e. Application channels lists repositories that the clusters can access.
   f. Application channels lists are only stored in Helm charts.
   g. Application channels ensure the application is deployed on a specific cluster.
   h. Application channels group resources together to build a deployment.

4. **Which one of the following statements about GitOps workflows is correct?**
   a. GitOps workflows use an imperative programming model.
   b. GitOps workflows use both Git and SVN as version control system.
   c. GitOps workflows use a declarative approach to configuration management.
   d. GitOps workflows use a systematic approach to infrastructure deployment.

5. **Which one of the following statements best describes Infrastructure as code (IaC)?**
   a. Infrastructure as code (IaC) is the practice of storing all configurations for managing and provisioning infrastructure on each OpenShift node.
   b. Infrastructure as code (IaC) ensures that the same infrastructure is not provisioned every time.
   c. Infrastructure as code (IaC) enables and encourages administrators to to manually provision and manage resources.
   d. Infrastructure as code (IaC) is used for automating provisioning.

Managing Multicluster Application Resources with RHACM

Objectives

After completing this section, you should be able to deliver applications into multiple clusters by using RHACM GitOps.

Introducing RHACM GitOps

RHACM provides GitOps capabilities as part of the application deployment lifecycle. RHACM strives to implement GitOps components and adhere to the GitOps principles. RHACM combines Infrastructure as Code (IaC), merge requests, and continuous integration with the GitOps `push` pattern or method. The GitOps `push` pattern uses a CI/CD pipeline to push changes to the multicluster applications. The CI/CD pipeline is triggered by a merge commit in Git. The subscription operator monitors the channel for new or updated resources. After a merge commit, the subscription operator downloads the updated resources directly from the storage location and deploys them to the targeted managed clusters. This GitOps pattern is agent-less so there isn't a concern with running agents in each infrastructure component. This GitOps pattern uses well documented CI/CD tooling that is familiar to a large segment of the administrators and thereby enables a quick start to projects with a large pool for collaboration. The deployments are standardized and use the same deployment methodology.

Organizing a Git Repository for RHACM GitOps

The RHACM application lifecycle can consume multiple types of directory structures. The directory structure provides the logical implementation and deployment of the resource YAML files for one or multiple directories. For example, the `mysql` application contains two folders and one sub-folder named `mysql-app`, `subscriptions`, and `mysql-sub` respectively. The `subscription` folder targets the `mysql-sub` folder that contains a single subscription and is downloaded and applied to the hub cluster. From the `mysql-sub` folder, subscriptions and policies are applied to the managed clusters based on the placement rule. Placement rules determine which managed-clusters are affected by each subscription.

```
├── mysql-app
└── subscriptions
    └── mysql-sub
```

The directory structure is created for the following subscription flow: subscription > mysql > mysql-apps.

The subscription in the mysql-sub folder is applied from the CLI terminal to the hub cluster. Subscriptions and policies are then downloaded and applied to the hub cluster from the mysql-sub folder. The subscriptions and policies in the mysql-sub folder then run on the managed clusters based on the placement rule. Placement rules determine which managed-clusters are affected by each subscription. The subscriptions or policies determine what is applied to the clusters that match their placement. Moreover, the `subscriptions` content from the mysql-sub folder is applied to all managed clusters, common applications, and common configurations that match the placement rule.

Managing Git Repository Applications

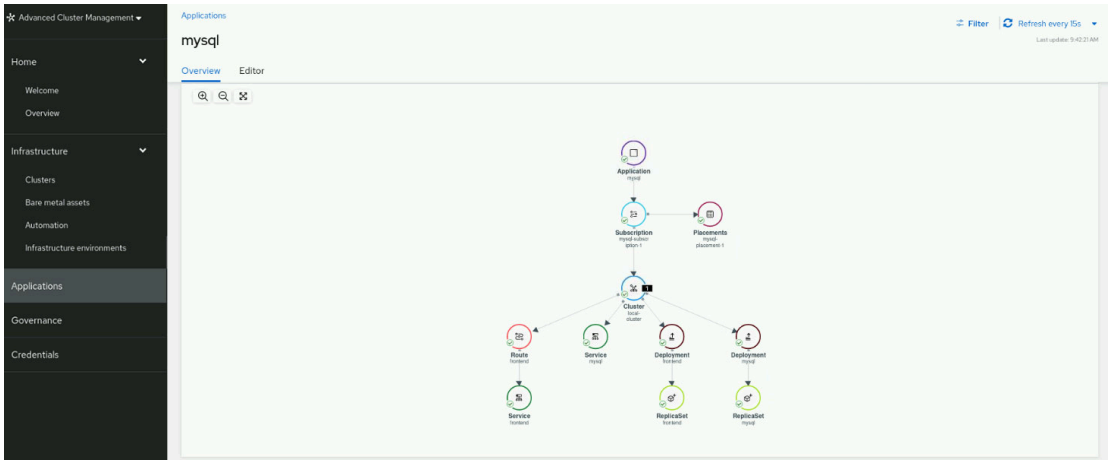
RHACM provides application management functions with concise options for building and deploying applications and updates. Applications are managed across multicluster environments through implementing subscription-based and channel automation. All the application resources are located in the Git repository. Git repositories, Helm repositories, and Object storage repositories are supported, however this course focuses on implementing a Git repository.

Application Console

The application console dashboard manages the application lifecycle. The dashboard includes capabilities to create, manage, and view the application status.

Resource topology

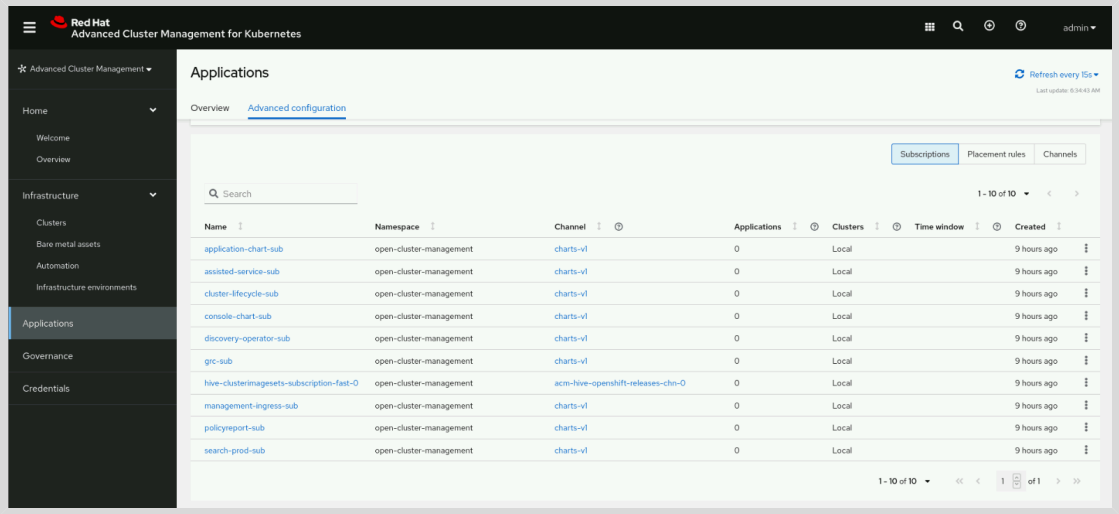
RHACM provides a resource topology page that displays a visual representation of the application and corresponding resources. The visual representation of the state of the resources assists in troubleshooting and confirming application issues.



Advanced configuration

The advanced configuration tab provides the capability to view all application tables and terminology. The Advance configuration tab also includes access to filter for the following resources:

- **Subscription**
- **Placement rules**
- **Channels**



Deploying an application using git repository from RHACM

In preparation for deploying an application using git repository from RHACM, you should define Kubernetes resources for the application in advance, and add them in a Git repository. The RHACM HUB will use these resources to create the desired application architecture in the managed clusters.

To deploy and manage an application with RHACM, you will need to create some custom resources. After creating the resource YAML file, you can verify that the resource definition creates without an error using the `kubectl` command.

Namespace

There must be a namespace resource per deployed application. The following example namespace definition displays the namespace resource created for a MySql application.

```
apiVersion: v1
kind: Namespace
metadata:
  name: mysql
```

Use `kubectl apply -f` to verify the resource definition.

```
[student@workstation base]$ kubectl apply -f namespace.yaml
namespace/mysql created
```

Subscriptions

Create the subscriptions resource YAML file for the MySql application. The following subscription resource is an example of the subscription resource created for a MySql application. The `apiVersion` and full group, *apps.open-cluster-management.io* must be specified in the resource YAML file. By default, the subscription operator subscribes to the `master` branch and you want to use `main`. You can subscribe to a different branch by specifying the branch name annotation in the subscription. You can also specify the path directory, *mysql*, that is used to access the custom resources for deployment.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: mysql-development-subscription
  labels:
    app: mysql
  annotations:
    apps.open-cluster-management.io/github-path: mysql
    apps.open-cluster-management.io/github-branch: main
```

Use `kubectl apply -f` to verify the resource definition.

```
[student@workstation base]$ kubectl apply -f subscription.yaml
subscription/mysql-development-subscription created
```

Channels

Create the channel resource YAML file. Channels define the source repositories that a cluster can subscribe to with a subscription. The following example channel definition displays an example of a `Git` channel for the `mysql` application. Within the hub cluster, the channel uses the `mysql` namespace. The Channel also specifies the `https://github.com/redhattraining/do480-apps` pathname, which points to the storage location of resources used in the deployment.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Channel
metadata:
  name: mysql-channel
  namespace: mysql
spec:
  pathname: 'https://github.com/redhattraining/do480-apps'
  type: GitHub
```

Use `kubectl apply -f` to verify the resource definition.

```
[student@workstation base]$ kubectl apply -f channel.yaml
channel.apps.open-cluster-management.io/mysql-channel created
```

Placementrule

Create the placementrule resource YAML file. Placementrules describe and specify the target clusters where subcriptions are deployed. The following example placementrule definition specifies the `mysql` namespace and that the application can only occupy the local cluster *hub-cluster*.

```
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  labels:
    app: mysql
  name: mysql-placement-1
  namespace: mysql
spec:
```

```
  clusterSelector:
    matchLabels:
      'local-cluster': 'true'
```

Use `kubectl apply -f` to verify the resource definition.

```
[student@workstation base]$ kubectl apply -f channel.yaml
placementrule.apps.open-cluster-management.io/mysql-channel created
```

Services

Create the service resource YAML file. Services are used for pod communication in the network. The following example placementrule definition specifies the service targets pods based on the selector specified label. The service requires a selector to identify it works with any pods with the label `app: todonodejs`

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: todonodejs
    name: mysql
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
    name: mysql
```

Use `kubectl apply -f` to verify the resource definition.

```
[student@workstation base]$ kubectl apply -f service.yaml
service.apps.open-cluster-management.io/mysql created
```

Routes

Create the route resource YAML file. Routes are used by OpenShift to expose a Service outside the cluster. These routes must have a resource YAML file as part of the deployment or the administrators would have to create the route manually.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: route-frontend
  name: frontend
  namespace: mysql
spec:
  host: todo.apps.ocp4.example.com
  path: "/todo"
  to:
    kind: Service
```

```
    name: frontend
    weight: 100
  wildcardPolicy: None
```

Use `kubectl apply -f` to verify the resource definition.

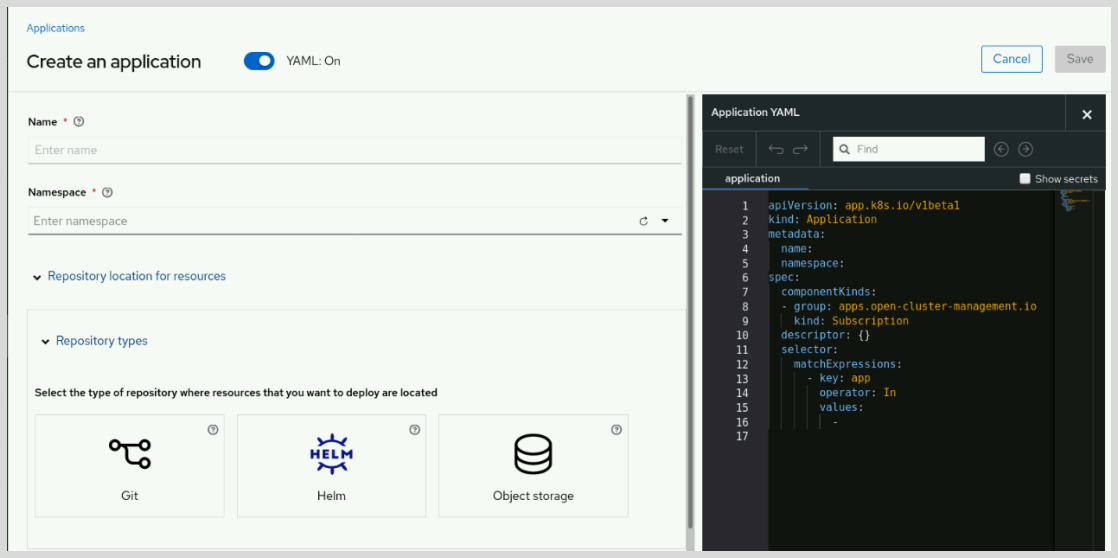
```
[student@workstation base]$ kubectl apply -f channel.yaml
route.route.openshift.io/frontend created
```

Deploying an application using git repository from RHACM

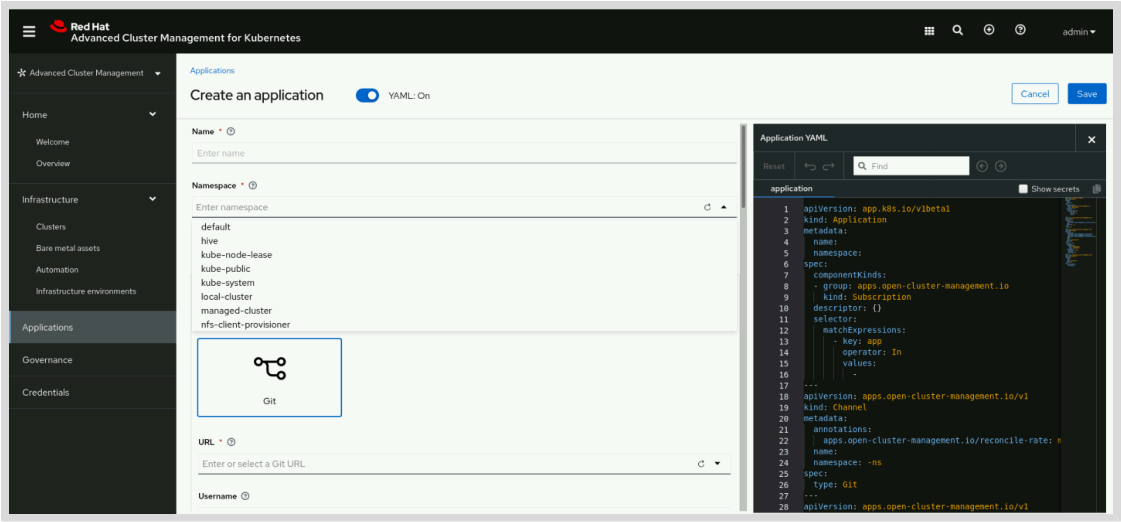
After creating the required resource YAML files for the application and pushing them to a Git repository, you are ready to deploy the application with RHACM.

Log in to the console.

Navigate to the left menu and select **Applications** and click **Create application**. RHACM provides a YAML editor that is built directly into the application console. The YAML web UI enables administrators to view, configure, and update the resources in the YAML file directly without leaving the console. The YAML web UI updates and displays in real time as the administrator configures the application.



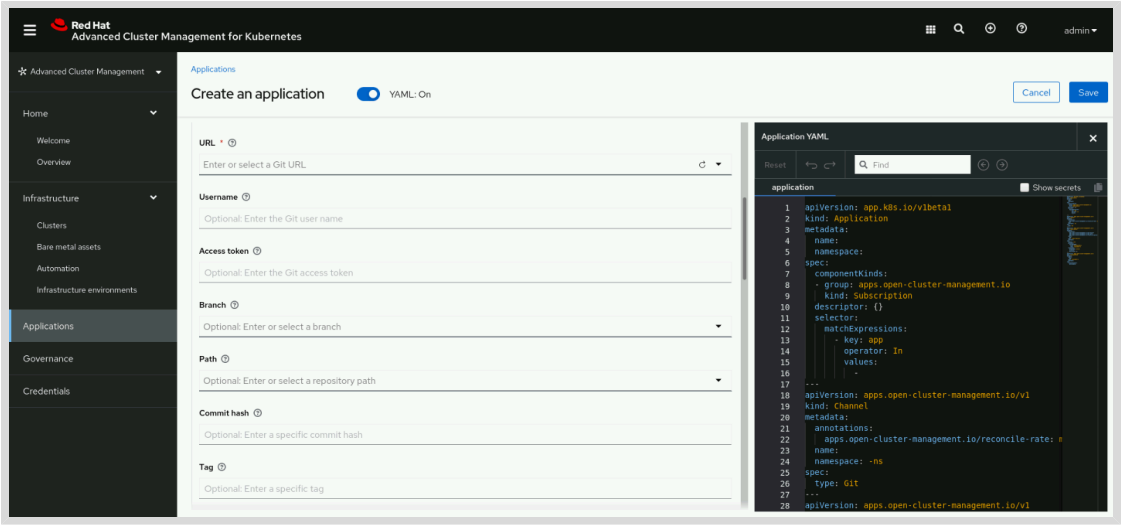
Enter a valid Kubernetes name and namespace for the application. The drop-down menu provides a list of current namespaces based on your access role.Administrators can select a namespace from drop-down menu or create a new one if they have been assigned the correct access role.



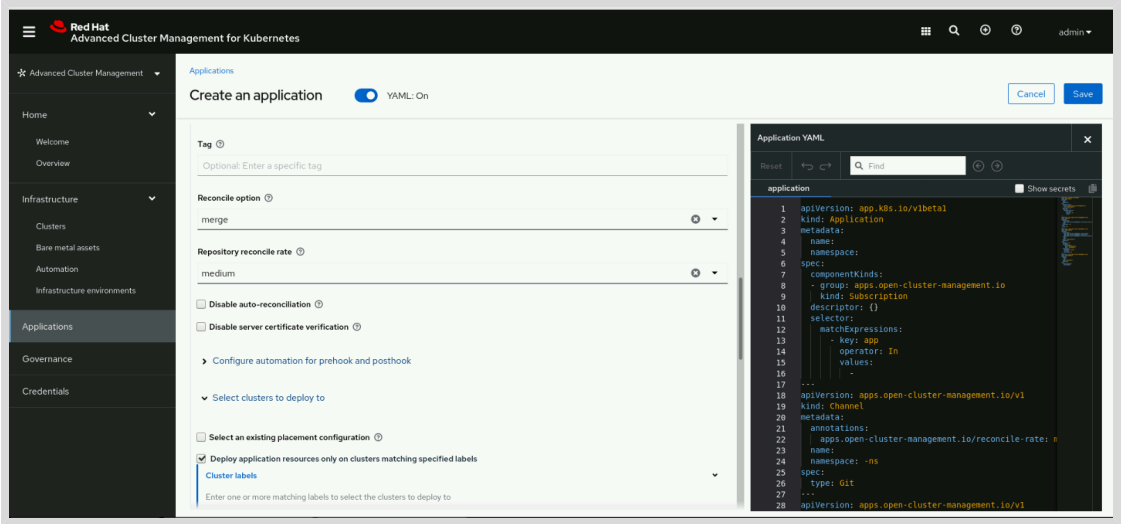
The `Location for respository resources` menu contains three repository types Git, Helm, and Object storage. Selecting the Git repository displays the following fields:

Git Repository Fields.

Field	Value
`URL`	Git repository location path.
`Branch`	The ranch that contains the applicable resource files.
`Path`	The path to directory that contains the applicable resource files.
`Commit hash`	To subscribe to a specific Git commit.
`Tag`	To subscribe to a specific tag.
`Reconcile`	The default is to reconcile on a commit `merge`. The other option, `replace`, replaces the existing resource with the Git resource.
`Pre/Post deployment task`	Set the Ansible Tower secret for jobs that you want to run before or after the subscription deploys the application resources. The Ansible Tower tasks must be placed within prehook and posthook folders in this repository.

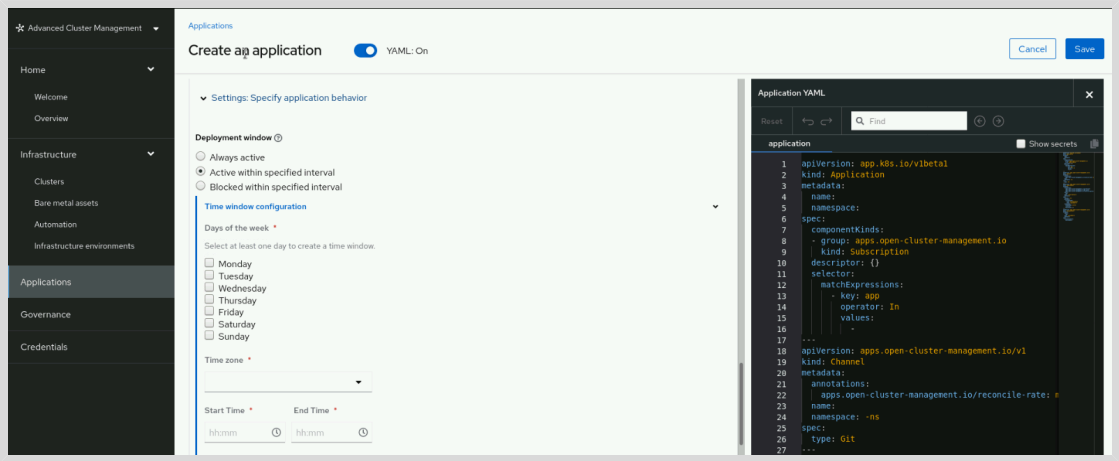


The `Select clusters to deploy` field allows you to define and update the placementrule allocated for the application. You have the option to deploy the placementrule to the local *hub* cluster, to the online *managed* clusters and the local cluster, or only deploy to clusters that match a specified label. You also have the option to `Select existing placement configuration` if you want to use a previously defined placementrule for an application in an existing namespace.



From settings, you can specify the application behavior for the `Deployment window` and `Time window`. The `Time window` allows resource subscriptions to begin deployments only during specific days and hours. You can define the subscription `Deployment window` type as `Always active`, `active within specified interval`, or `blocked within specified interval` for the specified time window. For example, you could define an `active` time window between `3:00 PM` and `3:30 PM` every Monday for web application updates. This `Time window` would allow the application deployments to occur every Monday between 3:00 PM and 3:30 PM. In the event that a deployment begins during a defined time window and is still running when the time window elapses, the deployment will still continue until the deployment is completed. If you were to use `blocked` instead of `active` with the previous example, application deployments could `not` begin between `3:00 PM` and `3:30 PM`, but could begin at any other time. The subscription operator continues to monitor regardless of the defined deployment windows. The following is an example `Time window` YAML definition that defines an active, 10:00 PM to 12:00 PM, eastern time zone time window that occurs every Monday.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
..._output omitted_...
spec:
  channel:
  placement:
    placementRef:
      kind: PlacementRule
      name: mysql-placement-1
  timewindow:
    windowtype: active
    location: "America/New_York"
    daysofweek: ["Monday"]
    hours:
      - start: "10:00PM"
        end: "12:00AM"
```



References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Applications guide* at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4

Guided Exercise

Managing Multicluster Application Resources with RHACM

In this exercise you will use an `RHACM GitOps` workflow to deploy a MySQL task-list application to the `Development` clusters based on an active time window interval.

Outcomes

You should be able to:

- Deliver applications into multiple clusters by using `RHACM GitOps`.
- Implement application placement rules.
- Define active time interval for application deployments.

Before You Begin

A GitHub account is required to complete this general exercise.

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

This command ensures that the `RHACM` is installed and that the `Development` clusters are ready and available.

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

Instructions

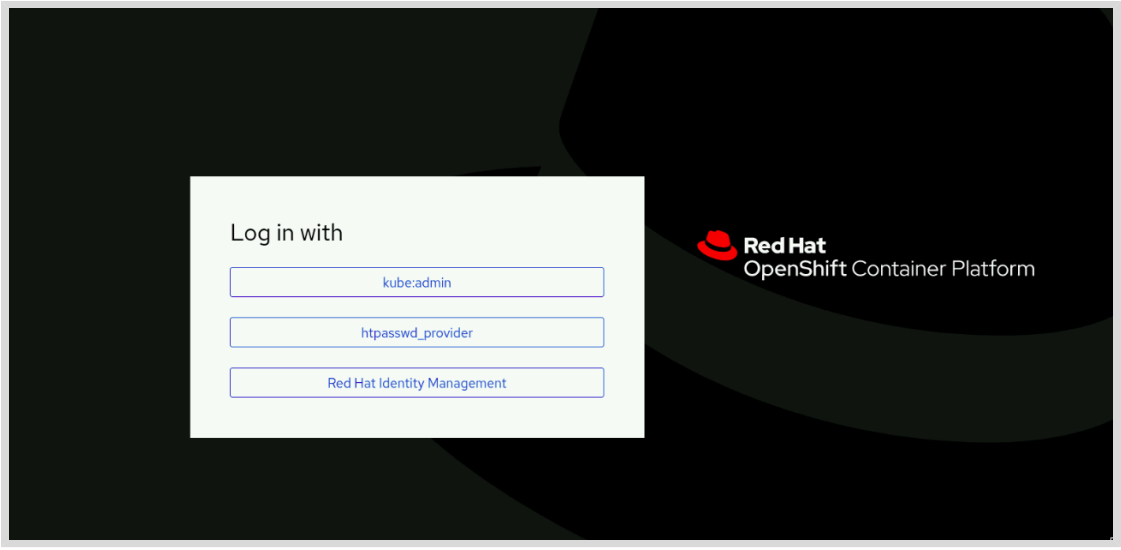
1. Fork the `do480-apps` course **GitHub** repository at `https://github.com/redhattraining/do480-apps/`. This repository contains the YAML files required to build the application.

   1.1. From `workstation`, use Firefox to navigate to the `do480-apps` repository at `https://github.com/redhattraining/do480-apps/`.

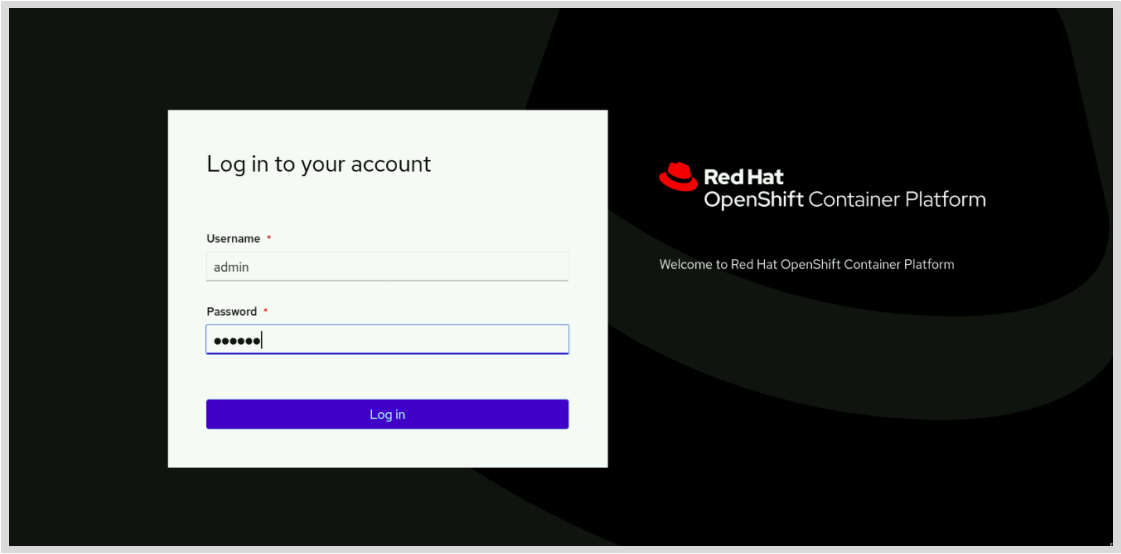
   1.2. In the top-right corner of the **GitHub** page, click **Fork** to create a fork for your repository.

2. Access the RHACM web console at `https://multicloud.console.apps.ocp4.example.com`

   2.1. From the `workstation` machine, open Firefox and access `https://multicloud-console.apps.ocp4.example.com`.



2.2. Click **htpasswd_provider** and log in as the `admin` user with the `redhat` password.

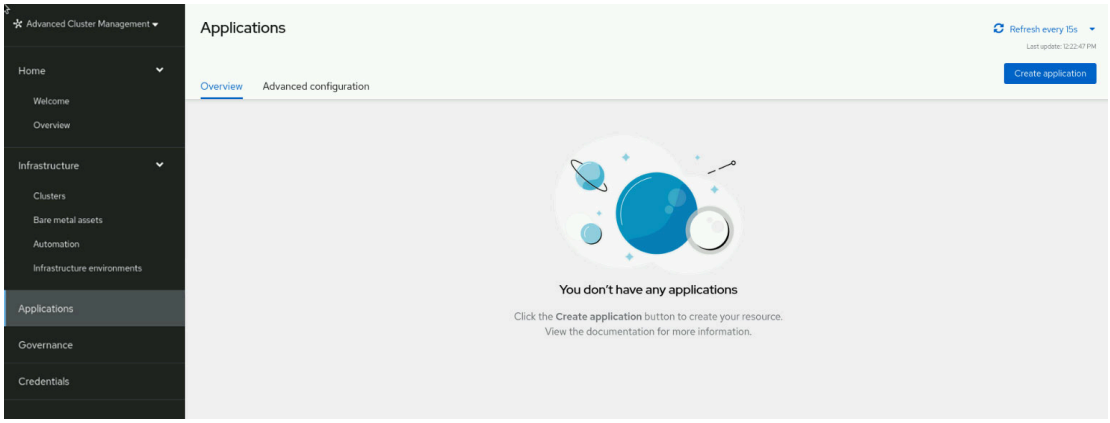


3. Use ACM Git Ops to create a new `MySQL` application based on the following criteria.

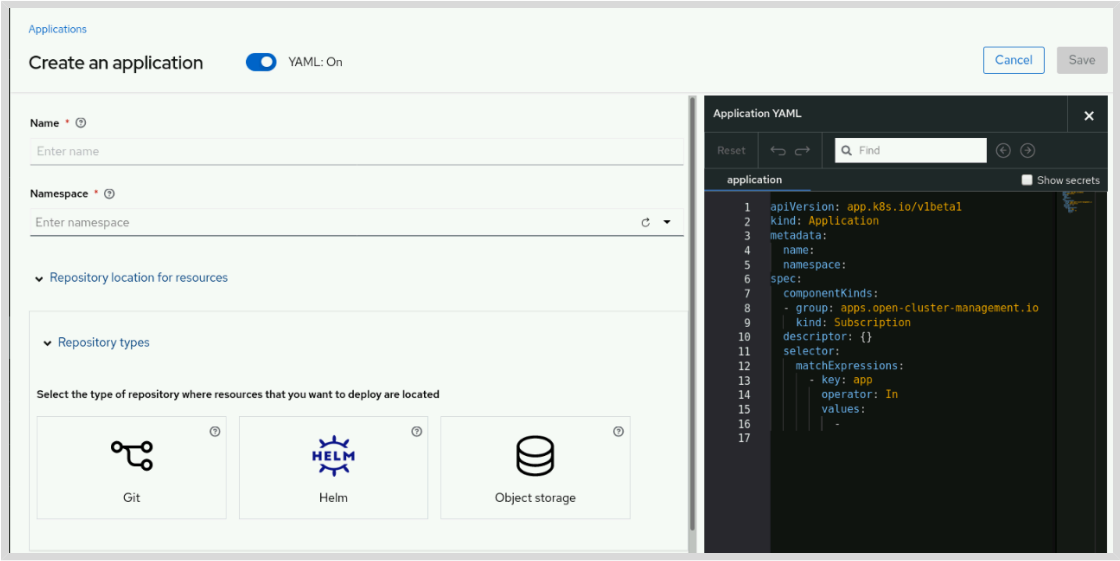
MySQL for Development Clusters.

Field	Value
`Name`	mysql
`Namespace`	mysql
`Repository types`	GitHub
`URL`	github.com/<your fork>/do480-apps/
`Branch`	main
`Path`	mysql

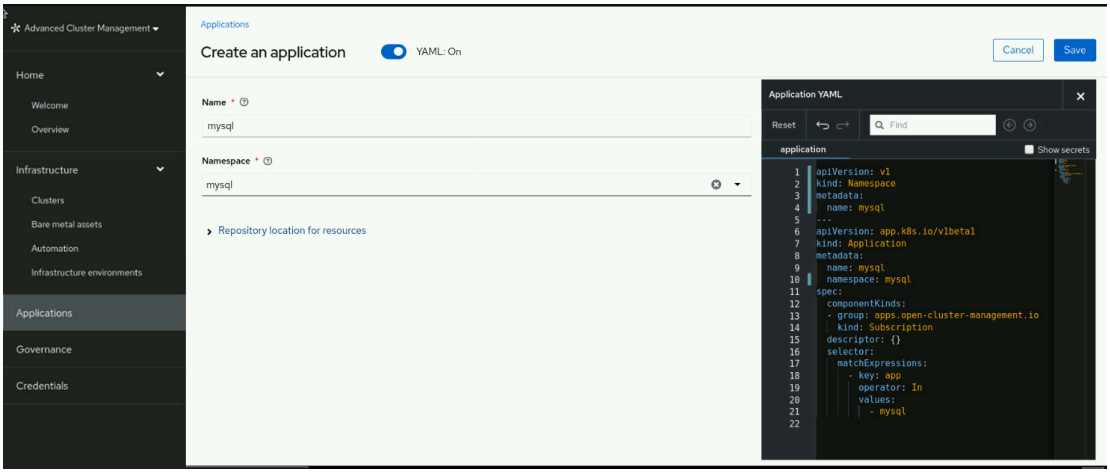
3.1. Navigate to the left menu and select **Applications** and click **Create application**.



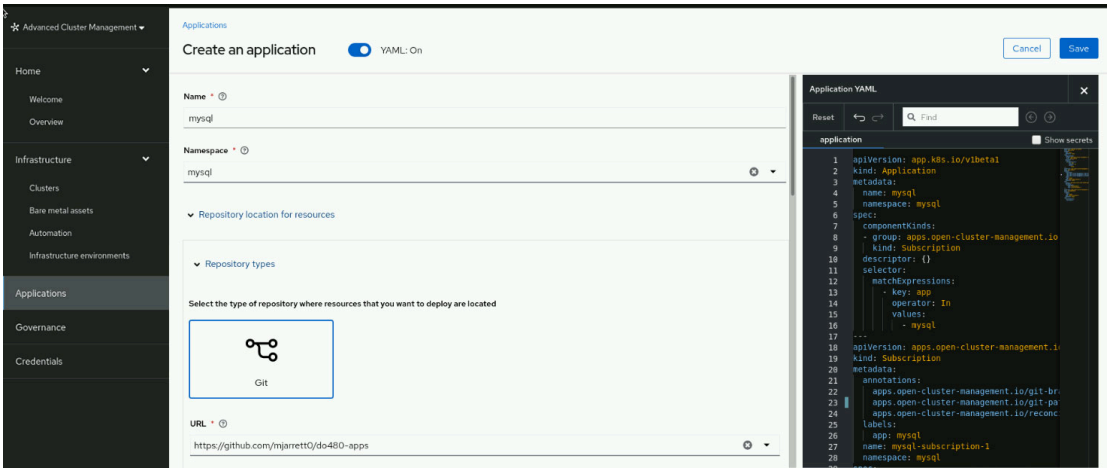
3.2. Select **YAML: On** to view the YAML in the console as you create the application.



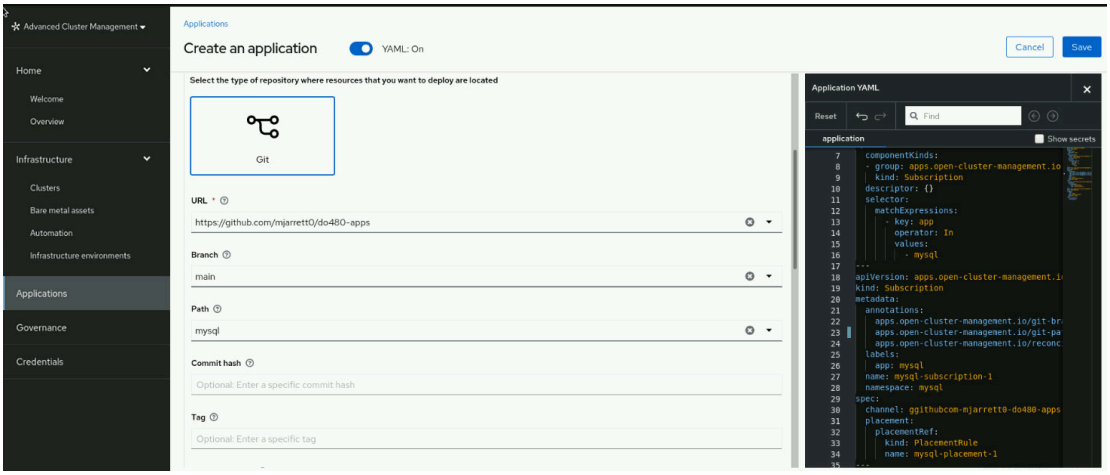
3.3. In both the `Name` and `namespace` fields, type `mysql`. Applications from the local hub-cluster require a namespace for every application to contain the resources that represent the application in the `managed-clusters`.



3.4. Select `Git` for the `Repository type` to specify the location of your deployable resources. Input the URL of your forked do480-apps repository for the channel source.



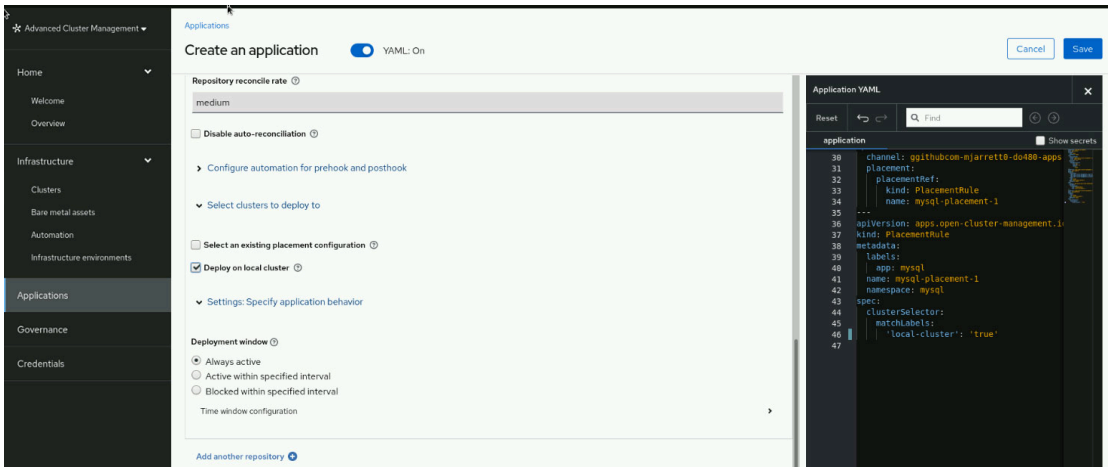
3.5. Type `main` for the branch and `mysql` for the path respectively.



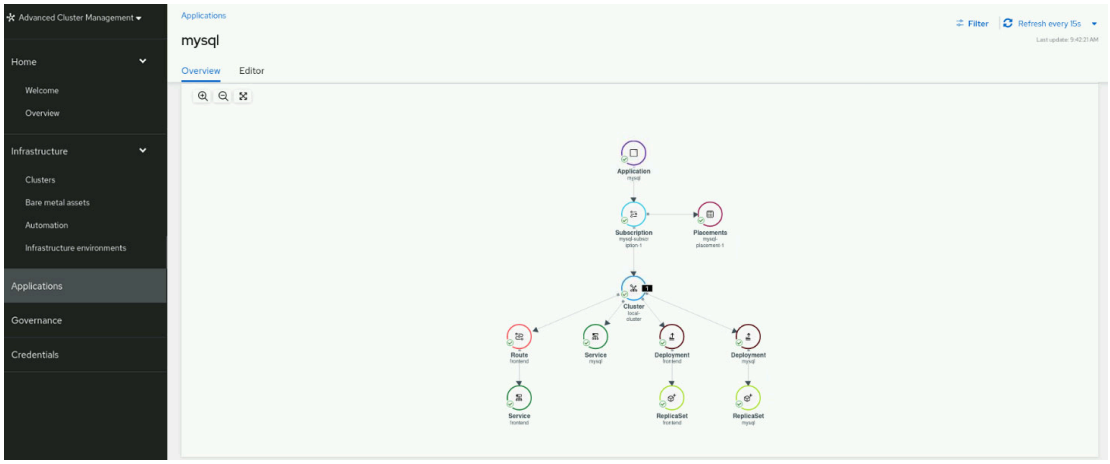
View the subscription and channel configuration in web yaml UI.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  annotations:
    apps.open-cluster-management.io/git-branch: main
    apps.open-cluster-management.io/git-path: mysql
    apps.open-cluster-management.io/reconcile-option: merge
  labels:
    app: mysql
  name: mysql-subscription-1
  namespace: mysql
spec:
  channel: ggithubcom-mjarrett0-do480-apps-ns/ggithubcom-mjarrett0-do480-apps
```

3.6. Scroll down and select **Select clusters to deploy to**to configure a placement rule for the application. Check the box located to the left of `Deploy on local cluster` which creates a placment rule to only deploy the application on the local cluster.



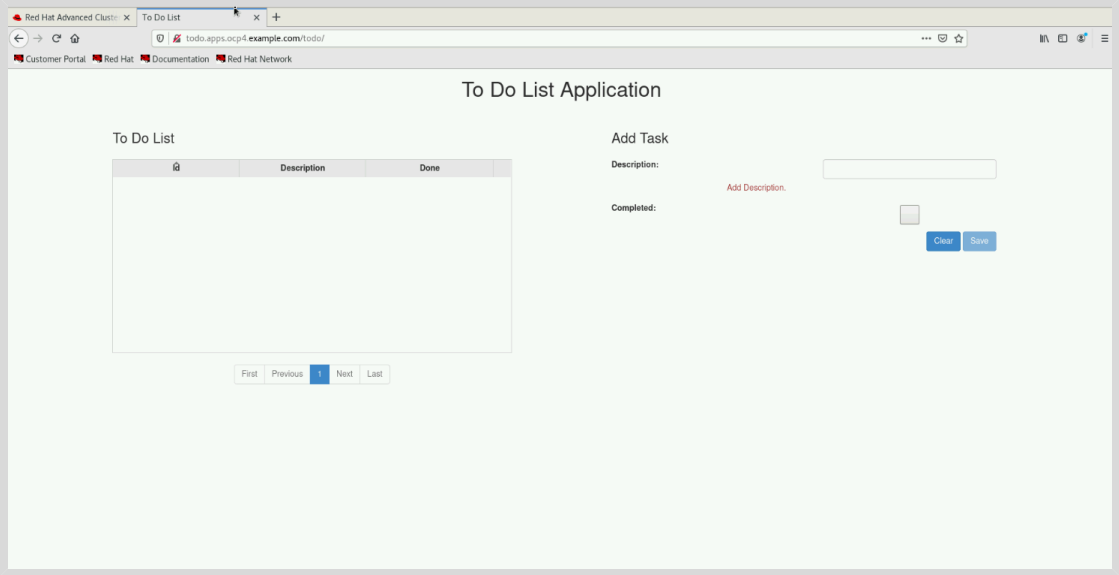
3.7. Click the **Save** button, then you are redirected to the applications **Topology** page.



4. Verify you can access the `MySQL` application frontend for the `Development` clusters.

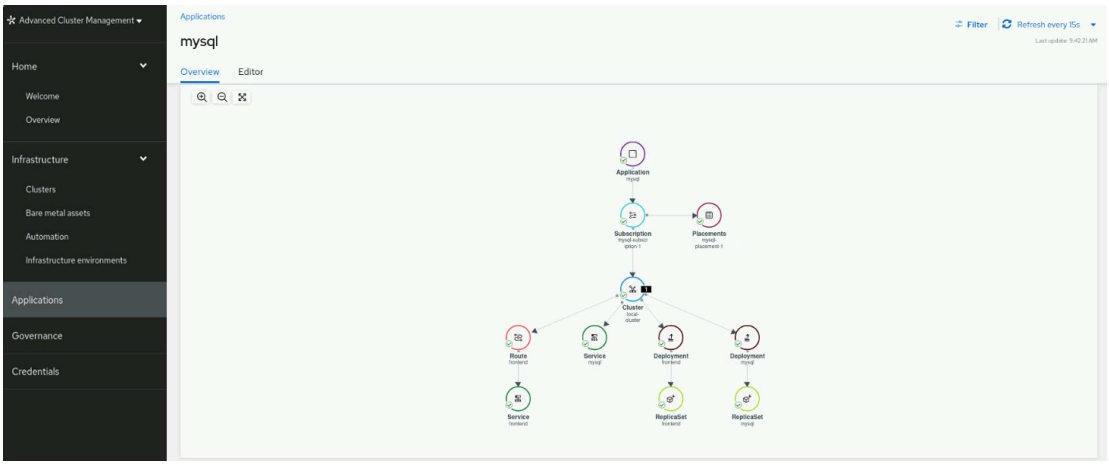
4.1. In the **Topology** page, select the `router pod`, and click the url located under `Location` to view the application frontend on the `Development` cluster.

4.2. In the subsequent web page, append `/todo/` to the url to display the application.

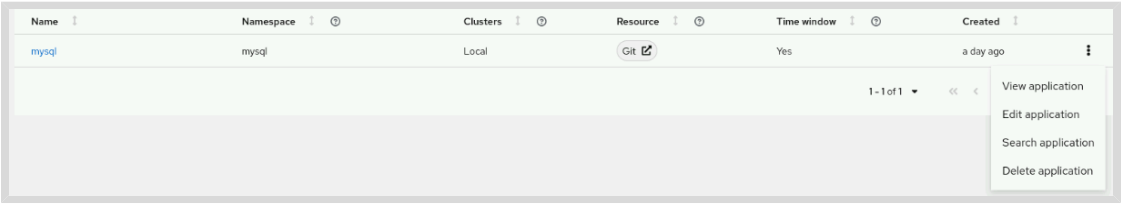


5. Define an active time window to the deploy updated versions of application resources.

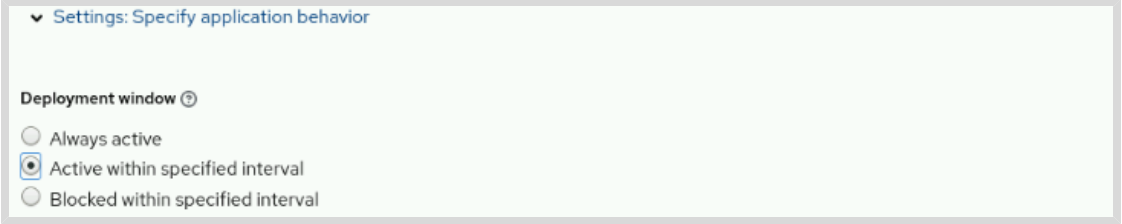
5.1. Click the `Applications` link at the top of the page.



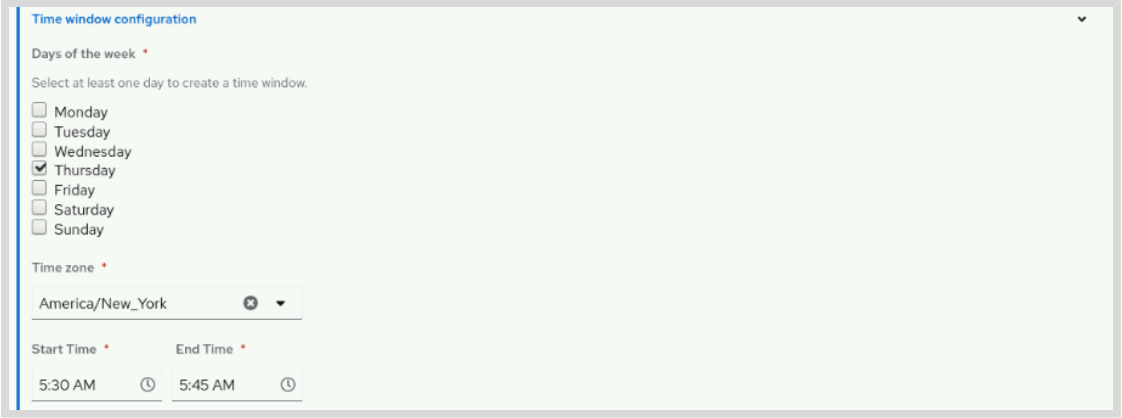
5.2. Select the menu at the end of the `mysql` application row and click **edit application** to update the application resources.



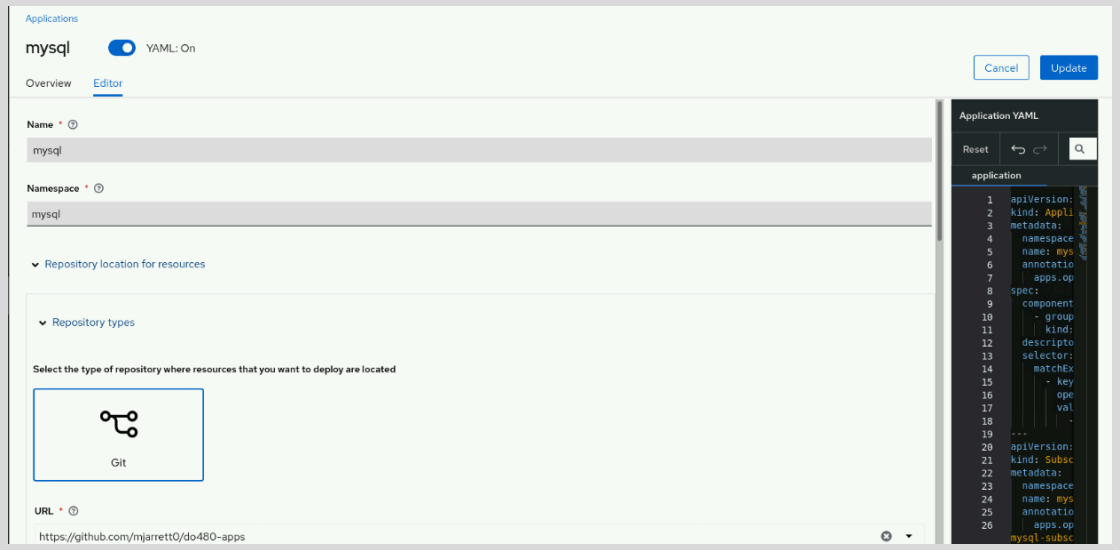
5.3. Scroll down and click **Settings: Specify application behavior**. Select the **Active with specified interval** radio button which displays the `Time window configuration`



5.4. Specify the day of the week for the time interval. Select the day you are performing this general exercise. For example, today is Thursday and the current time is 05:30. The active time window is defined for between 05:30 and 05:45.



5.5. Click the **Update** button and you are redirected to the applications **Topology** page.



6. Navigate to your forked `do480-apps` repository in the `main` branch and edit the `do480-apps/mysql/deployment.yaml` file. Update the container image to `mysql-80:152`.

    6.1. In your github fork, ensure you are in the `main` branch. Click `mysql` and then click the `deployment.yaml` file link to access the file. Next, click the pencil icon to edit the YAML file.

    6.2. In the YAML web editor, update the current image tag to `mysql-80:1-156`. Then, scroll to the bottom of the page and click the **Commit changes** button to update the `main` branch.

```
spec:
      containers:
      - image: registry.redhat.io/rhel8/mysql-80:1-152
        name: mysql
        env:
```

7. Return to the RHACM console and view the applications **Topology** page to verify the new pod was created.

    7.1. In the **Topology** page, select the `mysql ReplicaSet` pod. View the `Created` time for the pod as the resource updates and creates a new pod. After a couple minutes, you will see the newly created `mysql ReplicaSet` pod.

8. Verify resource updates cannot occur outside the active time window. For example, the current time is 5:35, the timezone is US/NY and the time window intervel is defined to end at 5:45

8.1. After the time window interval expires, in your github fork, ensure you are in the `main` branch. Click `mysql` and then click the `deployment.yaml` file link to access the file. Next, click the pencil icon to edit the YAML file.

8.2. In the GitHub YAML editor, update the current image tag to `mysql-80:latest`. Then, scroll to the bottom of the page and click the **Commit changes** button to update the `main` branch.

```
spec:
      containers:
      - image: registry.redhat.io/rhel8/mysql-80:latest
        name: mysql
        env:
```

8.3. Return to the RHACM console and view the applications **Topology** page to verify the new pod was created.

8.4. In the **Topology** page, select the `mysql ReplicaSet` pod. The pod has not been re-created due to the expiration of the active time window interval.

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 finish applications-resources
```

This concludes the guided exercise.

Customizing Resources with Kustomize for RHACM

Objectives

After completing this section, you should be able to describe Kustomize concepts, features, and deploy new customizations with the Kustomize command-line tool.

Introducing Kustomize

Kustomize is a configuration management solution that allows an administrator to make declarative changes to application configurations and components while preserving the original base YAML files. Kustomize overlays declarative YAML artifacts, or `patches` that specifically override the general settings without actually modifying the original base files. For example, you could have an multiphase application for multiple departments. The application configuration is predominately identical amongst the different departments, however there are a few settings that are specific to each department and therefore require some customization. Kustomize implements the `kustomization.yaml` file that contains specific settings that override the parameters of the original base configuration. Kustomize enables the GitOps team to consume base YAML file updates while maintaining use-case specific customization overrides.

Kustomize implements the concepts of `bases` and `overlays`. Bases are essentially the base minimum YAML files that contain the settings shared among the application target environments. For example, you could have a base set a files to deploy a database. Overlays, inherit from either one or more bases and are used patch manifests for specific environments or clusters. In reference to the previous example, administrators could have a database with base files for deploying PostGreSql and an overlay that patches the base manifest for an environment to use a specific type of storage.

Kustomize CLI

The Kustomize CLI Kustomize was embedded in Kubernetes since the March 2019 release of version 1.14. and also maintains a standalone version.

Kustomize benefits

Reusability

Kustomize allows administrators to reuse the `base` YAML files across multiple environments.

Templateless

Kustomize does not have a templating language and uses the same standard YAML syntax as Kubernetes.

Debug isolation

Kustomize `overlays` contains the non common or environment specific setting. Due to the isolation of these problematic settings, troubleshooting issues becomes significantly less cumbersome as you only need to focus on the `overlays` performance and compare that to the performance of the base.

Kustomize files

Kustomize breaks down application YAML files structure in to `base` YAML files and `overlay` YAML files for environmental specific settings. After the customization parameters have been defined, Kustomize rebuilds and deploys a new manifest with `kubectl kustomize` and `kubectl apply` respectively. Kustomize consists of the following file types:

- **Base Yaml files**

The base YAML files contain the application settings that are identical or common multiple environments.

- **Overlay YAML files**

The patch overlay YAML files contain the application settings that are unique and will override the base file parameters.

- **Kustomization files**

The kustomization YAML files define how the settings in the overlay YAML files should interact with the base YAML files to build the new manifest. The new manifest builds consisting of the base YAML files and their overlays and are called `variants`. Generally, in a multicluster infrastructure, each environment has a corresponding kustomization.yaml file that contains specific settings for different environments such as `production` and `development`.

Kustomize Features

The following table lists a subset of the features available from Kustomize:

Kustomize Features

=== | **namePreffix** | prepends a specific prefix to resource names. | **commonLabels** | replaces label values for a specific keyword. | **images** | replaces the image with a customized image name. | **configMapGenerator** | creates K8s ConfigMap resources used as environment variables in the pod definition. | **patchesStrategicMerge** | merges specific configuration segments into to the common base configuration. ===

The infrastructure consists of two specific environments, named `development` and `production`, that run mysql applications on their respective clusters. Although both environments consists mostly of the same configuration settings, there are a few differences.

First the administrator procures all of the required resource YAML files.

The following is an example of the directory structure:

```
── base
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── dev
    │   └── kustomization.yaml
    ├── production
    │   ├── kustomization.yaml
    │   ├── pvc.yaml
```

The base directory must contain a `kustomization.yaml` file that specifies the resource YAML files that are required to build a new manifest for the application.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- deployment.yaml
- service.yaml
```

The base directory contains the resource files for the application and must be built with the `kubectl kustomize` command. This command takes the YAML source (via a path or URL) and creates a new YAML that can be piped into the `kubectl create` command.

```
[student@workstation base]$ kubectl kustomize .
```

Patch `overlays` are created to facilitate the customization of both the `production` and `development` environments.

namePrefix

The namePrefix directive is used to add a prefixed set of characters to the beginning of resource names. The following kustomization.yaml examples apply the prefix `prod` to all the resource names for the production environment and the prefix `dev` to all the resource names in the development environment.

```
#Production
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

#path to base directory
bases:
- ../../base

#Add name prefix "prod"
namePrefix: prod-
```

```
#Development
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

#path to base directory
bases:
  - ../../base

#Add name prefix "dev"
namePrefix: dev-
```

commonLabels

The production and development cluster sets require a label for the application that references the cluster set environment. The following `kustomization.yaml` file defines the labels

`prod-todolist.js` and `dev-todolist.js` for the keyword `app` on the `production` and `development` cluster sets respectively.

```
#Production
 commonLabels:
  app: prod-todolist.js
```

```
#Development
commonLabels:
  app: dev-todolist.js
```

Note that with `commonLabels` does not append or prepend characters but directly replaces the values.

images

The `production` and `development` cluster sets require an image for the application that references the cluster set environment. The images feature replaces the placeholder `mysql-image` in the deployment.yaml file. The prod and devel cluster sets images are defined in their kustomization.yaml file using `newName` and `newTag`. The following `kustomization.yaml` files defines the `newName` value as `quay.io/example/todo-single` for both the production and development cluster sets. The production cluster sets use the stable version 1.1 that has been tested and secured. However, the development cluster sets implements the latest open source release of the image, v2.0-beta.

```
#Production
images:
  - name: mysql-db-image
    newName: quay.io/example/todo-single
    newTag: v1.1
```

```
#Development
images:
  - name: mysql-db-image
    newName: quay.io/example/todo-single
    newTag: v2.0-beta
```

patchesStrategicMerge

The `development` cluster sets use the default emptyDir setting with ephemeral storage. However, the `production` environment requires persistent volumes and persistent volume claims for storage. The PVCs are defined in a separate YAML file named `pvc.yaml` You can use the patchesStrategicMerge to apply the volumes defined in the `pvc.yaml`. In the `kustomization.yaml` file for the production environment, the `pvc.yaml` is defined under `resources`.

```
#Production
patchesStrategicMerge:
  - pvc.yaml
```

References

For more information, refer to the *Red Hat Advanced Cluster Management for Kubernetes Applications guide* at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4

Guided Exercise

Customizing Resources with Kustomize for RHACM

In this exercise you will use Kustomize to maintain a common set of deployment resources and build overlays. The overlays modify the base deployment resources for specific environments. You will build deployment overlays that modify available storage and replicas on the production clusters.

Outcomes

You should be able to:

- Build a base environment for Kustomize.
- Build overlays to modify the base deployment.
- Configure kustomization.yaml to add resources.
- Deploy an application modified and managed with Kustomize

Before You Begin

Fork the `do480-apps` course GitHub repository at `https://github.com/redhattraining/do480-apps/`. This repository contains the YAML files required to build the 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 `RHACM` is installed and that the `Development` clusters are ready and available.

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

Instructions

1. From `workstation`, use `Kustomize` to build the `base` directory and corresponding `kustomization.yaml` file in the `~/DO480/labs/applications-kustomize` directory. The base directory contents are located at `http://github.com/<your_fork>/do480-apps` in the `kustomize` branch.

    1.1. Open the `terminal` application on the `workstation` machine. Log in to the Hub 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 new project named `mysql`.

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

1.3. Navigate to the `DO480/labs/applications-kustomize` project directory and create the `base` directory which will contain the set of common yaml resource files.

```
[student@workstation ~]$ cd DO480/labs/applications-kustomize
```

```
[student@workstation applications-kustomize]$ mkdir base
```

1.4. Use `Git` to clone your `do480-apps` forked repository to access the common set of yaml resource files

```
[student@workstation applications-kustomize]$ git clone \
 https://github.com/<your_fork>/do480-apps
Cloning into 'do480-apps'...
remote: Enumerating objects: 144, done.
remote: Counting objects: 100% (144/144), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 144 (delta 56), reused 104 (delta 36), pack-reused 0
Receiving objects: 100% (144/144), 18.03 KiB | 18.03 MiB/s, done.
Resolving deltas: 100% (56/56), done.
```

1.5. Navigate to the `do480-apps` directory and change to the `kustomize` branch to access the `mysql` deployment yaml files.

```
[student@workstation applications-kustomize]$ cd do480-apps
```

```
[student@workstation do480-apps]$ git checkout kustomize
Branch 'kustomize' set up to track remote branch 'kustomize' from 'origin'.
Switched to a new branch 'kustomize'
```

1.6. Navigate to the `base` directory, copy the contents of the `kustomize` directory to your `base` directory and list the contents.

```
[student@workstation do480-apps]$ cd ../base/
```

```
[student@workstation base]$ cp ../do480-apps/kustomize//* .
```

```
[student@workstation base]$ ll
total 32
-rw-rw-r--. 1 student student 165 Dec 24 06:10 applications.yaml
-rw-rw-r--. 1 student student 807 Dec 24 06:10 deployment-frontend.yaml
-rw-rw-r--. 1 student student 692 Dec 24 06:10 deployment.yaml
-rw-rw-r--. 1 student student  65 Dec 24 06:10 namespace.yaml
-rw-rw-r--. 1 student student 227 Dec 24 06:10 placementrule.yaml
```

```
-rw-rw-r--. 1 student student 287 Dec 24 06:10 route.yaml
-rw-rw-r--. 1 student student 166 Dec 24 06:10 service-frontend.yaml
-rw-rw-r--. 1 student student 165 Dec 24 06:10 service.yaml
```

1.7. Build the base directory `kustomization.yaml` file which contains the set of resources and associated customizations. Your `kustomization.yaml` file should contain the following:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- applications.yaml
- deployment-frontend.yaml
- deployment.yaml
- service.yaml
- service-frontend.yaml
- namespace.yaml
- placementrule.yaml
- route.yaml
```

1.8. Verify the `kustomization.yaml` file builds the deployment without errors with the `kubectl kustomize` command.

```
[student@workstation base]$ kubectl kustomize .
apiVersion: v1
kind: Namespace
metadata:
  name: mysql
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: todonodejs
    name: frontend
  name: frontend
spec:
  ports:
  - port: 8080
  selector:
    name: frontend
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: todonodejs
    name: mysql
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
```

```
    name: mysql
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: todonodejs
    name: frontend
  name: frontend
  namespace: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: todonodejs
      name: frontend
  template:
    metadata:
      labels:
        app: todonodejs
        name: frontend
    spec:
      containers:
      - env:
        - name: MYSQL_ENV_MYSQL_DATABASE
          value: items
        - name: MYSQL_ENV_MYSQL_USER
          value: user1
        - name: MYSQL_ENV_MYSQL_PASSWORD
          value: mypa55
        - name: APP_PORT
          value: "8080"
        image: quay.io/redhattraining/todo-single:v1.0
        name: todonodejs
        ports:
        - containerPort: 8080
          name: nodejs-http
        resources:
          limits:
            cpu: "0.5"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: todonodejs
    name: mysql
  name: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: todonodejs
      name: mysql
  template:
```

```
    metadata:
      labels:
        app: todonodejs
        name: mysql
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: r00tpa55
        - name: MYSQL_USER
          value: user1
        - name: MYSQL_PASSWORD
          value: mypa55
        - name: MYSQL_DATABASE
          value: items
        image: registry.redhat.io/rhel8/mysql-80:1-156
        name: mysql
        ports:
        - containerPort: 3306
          name: mysql
---
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  labels:
    app: mysql
  name: mysql-placement-1
  namespace: mysql
spec:
  clusterSelector:
    matchLabels:
      local-cluster: "true"
---
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: route-frontend
  name: frontend
  namespace: mysql
spec:
  host: todo.apps.ocp4.example.com
  path: /todo
  to:
```

```
    kind: Service
    name: frontend
    weight: 100
  wildcardPolicy: None
```

Now you have a base set of deployment resources that can be shared amongst both the `Production` and `production` administrators.

2. Examine the `deployment.yaml` in the base directory and note that the storage is not configured. The `production` clusters require that the mysql applications use persistent volumes and persistent volume claims for storage. Build an overlay for the `production` clusters that uses persistent storage.

   2.1. Create the `overlay` directory and the corresponding `production` and `development` subdirectories. Ensure the `overlays` directory is at the same directory level as the `base` directory.

```
[student@workstation base]$ tree ../
base
overlays/
├── development
└── production
```

   2.2. Create the `dbclaim-pv.yaml` file resource, which contains the persistent volume claim, in the `overlays/production/` subdirectory. Your `dbclaim-pv.yaml` file should contain the following:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  namespace: mysql
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
  storageClassName: nfs-storage
```

   2.3. Create the corresponding `kustomization.yaml` file in the `overlay/production/` subdirectory to add the new resource. Your `kustomization.yaml` file should contain the following:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

bases:
  - ../../base

resources:
- dbclaim-pv.yaml
```

```
[student@workstation base]$ cat ../overlays/production/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

bases:
  - ../../base

resources:
- dbclaim-pv.yaml
```

2.4. Use the `kubectl kustomize` command to build the deployment for the production clusters with the new storage resource. Verify the new storage resource has been added to the deployment.

```
[student@workstation base]$ kubectl kustomize ../overlays/production
...output_omitted...
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  namespace: mysql
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
  storageClassName: nfs-storage
...output_omitted...
```

2.5. The output of the previous command can now be applied to the production clusters. Use the `kubectl kustomize` command piped to `kubectl apply -f` to build and deploy the application to the `production` clusters with the new storage resource.

```
[student@workstation base]$ kubectl kustomize ../overlays/production | kubectl
 apply -f -
namespace/mysql created
service/frontend created
service/mysql created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/frontend created
deployment.apps/mysql created
application.app.k8s.io/mysql created
placementrule.apps.open-cluster-management.io/mysql-placement-1 created
route.route.openshift.io/frontend created
```

2.6. Verify the deployment and confirm that the new storage resource is deployed.

```
[student@workstation base]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
frontend-5cbdf5bf85-94qwl   1/1     Running   0          12m
mysql-6c5d98dd9b-q94c8      1/1     Running   0          12m
```

```
[student@workstation base]$ oc get pvc
NAME             STATUS   VOLUME                                     CAPACITY
 ACCESS MODES   STORAGECLASS   AGE
mysql-pv-claim   Bound    pvc-c95583c8-3fea-45d4-bcba-330a33eea6ff   10Mi
 RWO            nfs-storage    5m32s
```

2.7. From `workstation`, use Firefox to navigate to the `mysql` application at `http://todo.apps.ocp4.example.com/todo/` to verify the application.

2.8. Use the `kubectl kustomize` command to confirm that the base yaml files have not been altered.

```
[student@workstation base]$ kubectl kustomize .
apiVersion: v1
kind: Namespace
metadata:
  name: mysql
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: todonodejs
    name: frontend
  name: frontend
spec:
  ports:
  - port: 8080
  selector:
    name: frontend
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: todonodejs
    name: mysql
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
    name: mysql
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
    labels:
      app: todonodejs
      name: frontend
    name: frontend
    namespace: mysql
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: todonodejs
        name: frontend
    template:
      metadata:
        labels:
          app: todonodejs
          name: frontend
      spec:
        containers:
        - env:
          - name: MYSQL_ENV_MYSQL_DATABASE
            value: items
          - name: MYSQL_ENV_MYSQL_USER
            value: user1
          - name: MYSQL_ENV_MYSQL_PASSWORD
            value: mypa55
          - name: APP_PORT
            value: "8080"
          image: quay.io/redhattraining/todo-single:v1.0
          name: todonodejs
          ports:
          - containerPort: 8080
            name: nodejs-http
          resources:
            limits:
              cpu: "0.5"
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    labels:
      app: todonodejs
      name: mysql
    name: mysql
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: todonodejs
        name: mysql
    template:
      metadata:
        labels:
          app: todonodejs
          name: mysql
      spec:
```

```
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: r00tpa55
        - name: MYSQL_USER
          value: user1
        - name: MYSQL_PASSWORD
          value: mypa55
        - name: MYSQL_DATABASE
          value: items
        image: registry.redhat.io/rhel8/mysql-80:1-156
        name: mysql
        ports:
        - containerPort: 3306
          name: mysql
---
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
  labels:
    app: mysql
  name: mysql-placement-1
  namespace: mysql
spec:
  clusterSelector:
    matchLabels:
      local-cluster: "true"
---
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: route-frontend
  name: frontend
  namespace: mysql
spec:
  host: todo.apps.ocp4.example.com
  path: /todo
  to:
    kind: Service
    name: frontend
    weight: 100
  wildcardPolicy: None
```

3. Create an overlay for the `production` clusters to increase the number of mysql database replicas from `one` to `three`.

3.1. Verify the current number of `mysql` replicas.

```
[student@workstation base]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
frontend-5cbdf5bf85-94qwl   1/1     Running   0          12m
mysql-6c5d98dd9b-q94c8      1/1     Running   0          12m
```

3.2. Modify the `kustomization.yaml` file to increase the number of `mysql` replica pods. Your `kustomization.yaml` file should contain the following:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

bases:
  - ../../base

resources:
- dbclaim-pv.yaml

replicas:
- name: mysql
  count: 3
```

```
[student@workstation base]$ cat ../overlays/production/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

bases:
  - ../../base

resources:
- dbclaim-pv.yaml

replicas:
- name: mysql
  count: 3
```

3.3. Use the `kubectl kustomize` command piped to `kubectl apply -f` to build and deploy the application and verify the number of `mysql` replicas.

```
[student@workstation base]$ kubectl kustomize ../overlays/production | kubectl
 apply -f -
namespace/mysql unchanged
service/frontend unchanged
service/mysql unchanged
persistentvolumeclaim/mysql-pv-claim unchanged
deployment.apps/frontend configured
deployment.apps/mysql configured
```

```
application.app.k8s.io/mysql unchanged
placementrule.apps.open-cluster-management.io/mysql-placement-1 unchanged
route.route.openshift.io/frontend unchanged
```

```
[student@workstation base]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
frontend-5cbdf5bf85-94qwl   1/1     Running   0          12m
mysql-6c5d98dd9b-25mss      1/1     Running   0          6s
mysql-6c5d98dd9b-2nj2j      1/1     Running   0          6s
mysql-6c5d98dd9b-q94c8      1/1     Running   0          12m
```

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 finish applications-kustomize
```

This concludes the guided exercise.

Lab

Managing Applications Across Multiple Clusters with RHACM

In this lab, you will use `RHACM` GitOps application management to promote a new version of the `todo` application from the `Development` clusters to the `Production` clusters.

Outcomes

You should be able to:

- Configure and deploy a MySQL application with RHACM Git Ops.
- Assign additional labels to clusters managed with RHACM.
- Promote an application from development to production.

Before You Begin

A GitHub account is required to complete this lab.

The following command ensures that the `RHACM` is installed and that the `Development` and `Production` clusters are ready and available.

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

Instructions

1. Fork the `do480-apps` course GitHub repository at `https://github.com/redhattraining/do480-apps/`. This repository contains the YAML files required to build the application.
2. Use the `RHACM` web console to add the labels `usage=development` and `usage=production` to the `local-cluster` and `managed-cluster` clusters respectively. The web console URL is `https://multicloud.console.apps.ocp4.example.com`
3. Use ACM Git Ops to create a new `MySQL` application based on the following:

Mysql for development clusters.

Field	Value
Name	mysql
Namespace	mysql
Repository types	GitHub
URL	github.com/<your fork>/do480-apps/
Branch	main
Path	mysql
Label name	environment
Label value	development

Add an additional repository.

Mysql for Production clusters.

Field	Value
Repository types	GitHub
URL	github.com/<your fork>/do480-apps/
Branch	production
Path	mysql
Label name	environment
Label value	production

4. Verify you can access the `MySQL` application frontend for the `Development` and `Production` clusters.
5. In Github, use your fork of the `do480-apps` repository to promote the `Development` `MySQL` image `mysql-80:1-156` to `Production`.
6. Return to the `RHACM` web console and refresh the application **Topology** page. Verify the `Mysql` application is working and using the correct image.

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 finish application-review
```

This concludes the lab.

Solution

Managing Applications Across Multiple Clusters with RHACM

In this lab, you will use RHACM GitOps application management to promote a new version of the `todo` application from the `Development` clusters to the `Production` clusters.

Outcomes

You should be able to:

- Configure and deploy a MySQL application with RHACM Git Ops.
- Assign additional labels to clusters managed with RHACM.
- Promote an application from development to production.

Before You Begin

A GitHub account is required to complete this lab.

The following command ensures that the RHACM is installed and that the `Development` and `Production` clusters are ready and available.

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

Instructions

1. Fork the `do480-apps` course **GitHub** repository at `https://github.com/redhattraining/do480-apps/`. This repository contains the YAML files required to build the application.

   1.1. From `workstation`, use Firefox to navigate to the `do480-apps` repository at `https://github.com/redhattraining/do480-apps/`.

   1.2. In the top-right corner of the **GitHub** page, click **Fork** to create a fork for your repository.

2. Use the RHACM web console to add the labels `usage=development` and `usage=production` to the `local-cluster` and `managed-cluster` clusters respectively. The web console URL is `https://multicloud.console.apps.ocp4.example.com`

   2.1. From workstation, use Firefox to navigate to the RHACM web console at `https://multicloud.console.apps.ocp4.example.com`.

   When prompted, select **htpasswd_provider**

   Type the following credentials

   - Username: `admin`
   - Password: `redhat`

   And then click **Log in**

2.2. In the left navigation menu, select **Clusters** to add new labels to the `local-cluster` and `managed-cluster` clusters.

2.3. On the `local-cluster` row, select the drop-down menu at the end of the row, and click **edit label**.

2.4. Type `environment=development` for the new label and click **Save**.

2.5. On the `managed-cluster` row, select the drop-down menu at the end of the row, and click **edit label**.

2.6. Type `environment=production` for the new label and click **Save**.

3. Use ACM Git Ops to create a new `MySQL` application based on the following:

Mysql for development clusters.

Field	Value
Name	mysql
Namespace	mysql
Repository types	GitHub
URL	github.com/<your fork>/do480-apps/
Branch	main
Path	mysql
Label name	environment
Label value	development

Add an additional repository.

Mysql for Production clusters.

Field	Value
Repository types	GitHub
URL	github.com/<your fork>/do480-apps/
Branch	production
Path	mysql
Label name	environment
Label value	production

3.1. Navigate to the left menu and select **Applications** and click **Create application**.

3.2. Create a `MySQL` application using the following:

Do **not** click save.

MySQL for Development Clusters.

Field	Value
Name	mysql
Namespace	mysql
Repository types	GitHub
URL	github.com/<your fork>/do480-apps/
Branch	main
Path	mysql
Label name	environment
Label value	development

3.3. Add an additional repository to the `MySQL` application for the `Production` cluster based on the following and click **Save**:

MySQL for Production Clusters.

Field	Value
Repository types	GitHub
URL	github.com/<your fork>/do480-apps/
Branch	production
Path	mysql
Label name	environment
Label value	production

4. Verify you can access the `MySQL` application frontend for the `Development` and `Production` clusters.

4.1. In the **Topology** page and navigate to **Subscription** → **mysql-subscription-1**. Select the `router pod` and click the url located under `Location` to view the application frontend on the `Development` cluster. In the subsequent web page, append `/todo/` to the url to display the application.

4.2. In the **Topology** page and navigate to **Subscription** → **mysql-subscription-2**. Select the `router pod` and click the url located under `Location` to view the application frontend on the `Production` cluster. In the subsequent web page, append `/todo/` to the url to display the application.

5. In Github, use your fork of the `do480-apps` repository to promote the `Development MySQL` image `mysql-80:1-156` to `Production`.

5.1. In your github fork, switch to the `production` branch. Click **mysql** and then click the **deployment.yaml** link to access the file. Next, click the **pencil** icon to edit the YAML file.

5.2. Scroll down and replace the current image tag with `mysql-80:1-156`. Then scroll to the bottom of the page, select **Create a new branch for this commit and start a pull request**, accept the default branch, and click **Proposed changes**.

5.3. To complete the workflow, click **Create pull request**, **Merge pull request**, and finally **Confirm merge**. After your pull request is successfully merged and closed, click **Delete branch**.

6. Return to the `RHACM` web console and refresh the application **Topology** page. Verify the `Mysql` application is working and using the correct image.

6.1. In the **Topology** page, select the **Subscription** → **mysql-subscription-2** to view the `managed-cluster` topology.

6.2. Select the **mysql ReplicaSet** pod and click **View Pod YAML and Logs** to view the YAML file and verify the image tag.

6.3. In the YAML console UI, scroll down and verify the image is `mysql-80:1-156`.

6.4. In the left navigation menu, select the **Applications** → **mysql** link to view the application.

6.5. In the **Topology** page, select **Subscription** → **mysql-subscription-2** and select the `router pod`. Click **todo.apps.ocp4-mng.example.com/**, located under `Location`, to view the application. In the subsequent web page, append `/todo/` to the url to display the application.

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 finish application-review
```

This concludes the lab.

Summary

In this chapter, you learned:

- About the benefits of implementing the GitOps concepts of `Infrastructure as Code (IaC)`, `Merge Commits`, and `Continuous Integration` and `Continuous Deployment (CI/CD)`.
- How to apply the GitOps principles of `declarative desired state` and `immutable desired state versions` for the application lifecycle in `RHACM`.
- How to deploy a subscription based application with `RHACM`.
- How to leverage `Kustomize` to apply and manage custom settings for multicluster environments.