





Join a community dedicated to learning open source

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



Network with tens of thousands of community members



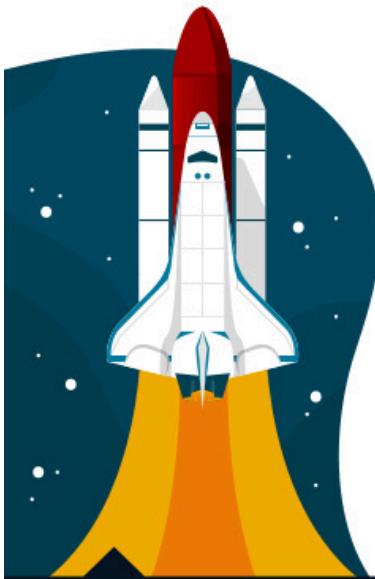
Engage in thousands of active conversations and posts



Join and interact with hundreds of certified training instructors



Unlock badges as you participate and accomplish new goals



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

Access free Red Hat training videos

Discover the latest Red Hat Training and Certification news

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

Join peers as you explore Red Hat products

Join the conversation learn.redhat.com



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

Multicluster Management with Red Hat OpenShift Platform Plus



ACM 2.4 DO480
Multicluster Management with Red Hat OpenShift Platform
Plus
Edition 5 20221130
Publication date 20221130

Authors: Alejandro Coma, Álex Córcoles, Federico Fapitalle,
Michael Jarrett, Rafael Ruiz, Harpal Singh
Course Architect: Fernando Lozano
DevOps Engineer: Benjamín Chardí
Editor: Nicole Muller

Copyright © 2022 Red Hat

The contents of this course and all its modules and related materials, including handouts to audience members, are
Copyright © 2022 Red Hat.

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

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

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

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

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

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

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

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

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

The OpenStack™ Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. Red Hat is not affiliated with, endorsed or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: **Sajith Eyamkuzhy, Adarsh Krishnan, David Sacco**

Document Conventions	ix
Admonitions	ix
Inclusive Language	x
Introduction	xi
Multicloud Management with Red Hat OpenShift Platform Plus	xi
Orientation to the Classroom Environment	xii
Performing Lab Exercises	xx
1. Managing a Multicloud Kubernetes Architecture	1
The Kubernetes Multicloud and Hybrid Cloud Architecture Landscape	2
Quiz: The Kubernetes Multicloud and Hybrid Cloud Architecture Landscape	8
Describing Red Hat OpenShift Platform Plus	12
Quiz: Describing Red Hat OpenShift Platform Plus	16
Installing Red Hat Advanced Cluster Management for Kubernetes	18
Guided Exercise: Installing Red Hat Advanced Cluster Management for Kubernetes	24
Lab: Installing the RHACM Operator and importing clusters	34
Summary	44
2. Inspecting Resources from Multiple Clusters Using the RHACM Web Console	45
Navigating the RHACM Web Console	46
Guided Exercise: Navigating the RHACM Web Console	51
Configuring Access Control for Multicloud Management	56
Guided Exercise: Configuring Access Control for Multicloud Management	59
Lab: Managing Resources from Multiple Clusters Using the RHACM Web Console	67
Summary	73
3. Deploying and Managing Policies for Multiple Clusters with RHACM	75
Deploying and Managing Policies with RHACM	76
Guided Exercise: Deploying and Managing Policies with RHACM	85
Deploying and Configuring the Compliance Operator for Multiple Clusters Using RHACM	90
Guided Exercise: Deploying and Configuring the Compliance Operator for Multiple Clusters Using RHACM	101
Integration of Other Policy Engines with RHACM	110
Guided Exercise: Integration of Other Policy Engines with RHACM	116
Lab: Deploying and Managing Policies with RHACM	129
Summary	136
4. Installing and Customizing the RHACM Observability Stack	137
Installing the RHACM Observability Stack	138
Guided Exercise: Installing the RHACM Observability Stack	143
Customizing the RHACM Observability Stack	151
Guided Exercise: Customizing the RHACM Observability Stack	155
Lab: Installing and Customizing the RHACM Observability Stack	161
Summary	170
5. Deploying Applications Across Multiple Clusters with RHACM	171
Introducing RHACM GitOps and the Application Model	172
Quiz: Introducing RHACM GitOps and the Application Model	176
Managing Multicloud Application Resources with RHACM	180
Guided Exercise: Managing Multicloud Application Resources with RHACM	186
Customizing Resources with Kustomize for RHACM	192
Guided Exercise: Customizing Resources with Kustomize for RHACM	197
Lab: Managing Applications Across Multiple Clusters with RHACM	208
Summary	225
6. Installing and Configuring Red Hat Quay	227

Identifying the Components and Features of Red Hat Quay	228
Quiz: Identifying the Components and Features of Red Hat Quay	236
Deploying and Configuring Red Hat Quay	238
Guided Exercise: Deploying and Configuring Red Hat Quay	245
Describing the Red Hat Quay Repository Model and RBAC	250
Guided Exercise: Describing the Red Hat Quay Repository Model and RBAC	258
Lab: Installing and Configuring Red Hat Quay	265
Summary	274
7. Integrating Red Hat Quay with Red Hat OpenShift and RHACM	275
Describe Red Hat Quay Use Cases in a Multicluster Architecture	276
Quiz: Describe Red Hat Quay Use Cases in a Multicluster Architecture	281
Restricting Image Registry Sources	285
Guided Exercise: Restricting Image Registry Sources	290
Deploying from Red Hat Quay to the Cluster Fleet	296
Guided Exercise: Deploying from Red Hat Quay to the Cluster Fleet	300
Lab: Integrating Red Hat Quay with OpenShift and RHACM	307
Summary	321
8. Installing and Configuring RHACS	323
Security Considerations in Kubernetes Multicluster Environments	324
Quiz: Security Considerations in Kubernetes Multicluster Environments	329
Installing and Configuring RHACS	333
Guided Exercise: Installing and Configuring RHACS	342
Importing Managed Clusters into RHACS	354
Guided Exercise: Importing Managed Clusters into RHACS	360
Quiz: Installing and Configuring RHACS	369
Summary	373
9. Multicluster Operational Security Using RHACS	375
Integrating RHACS with External Registries	376
Guided Exercise: Integrating RHACS with External Registries	380
Perform Multicluster Vulnerability Management with RHACS	386
Guided Exercise: Perform Multicluster Vulnerability Management with RHACS	392
Multicluster Configuration Management with RHACS	400
Guided Exercise: Multicluster Configuration Management with RHACS	405
Lab: Multicluster Operational Security Using RHACS	410
Summary	424
10. Comprehensive Review	425
Comprehensive Review	426
Lab: Configure RHACS, Red Hat Quay, and RHACM Observability Stack	429
Lab: Deploy and Update an Application to Align with the RHACS Policy	443

Document Conventions

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

Admonitions

Red Hat Training courses use the following admonitions:



References

These describe where to find external documentation that is relevant to a subject.



Note

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



Important

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



Warning

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

Inclusive Language

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

Introduction

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 Red Hat OpenShift clusters. You will learn to control applications by using placement rules that are determined by capacity and criticality, ensure that cluster configurations comply with governance and security policies, and automate these capabilities according to DevOps principles.

Course Objectives

- Deploy Red Hat Advanced Cluster Management for Kubernetes (RHACM).
- Add managed clusters to RHACM.
- Define and apply cluster configuration policies.
- Detect and correct non-conformance to cluster configuration policies.
- Deploy applications by using RHACM
- Deploy a private Red Hat Quay registry.
- Deploy Red Hat Advanced Cluster Security for Kubernetes (RHACS).
- Integrate Red Hat Quay and RHACS security with RHACM.

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

The Workstation Machine

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

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

You type the **oc** commands to manage the OpenShift cluster from the **workstation** machine that comes preinstalled as part of your classroom environment.

You run the shell scripts and Ansible Playbooks that are required to complete the exercises for this course from the **workstation** machine.

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



Note

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

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

Log in on OpenShift from the Shell

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

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

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

Accessing the OpenShift Web Console

If you prefer to use the OpenShift web console, open a Firefox web browser on your **workstation** machine and navigate to `https://console-openshift-console.apps.ocp4.example.com`.

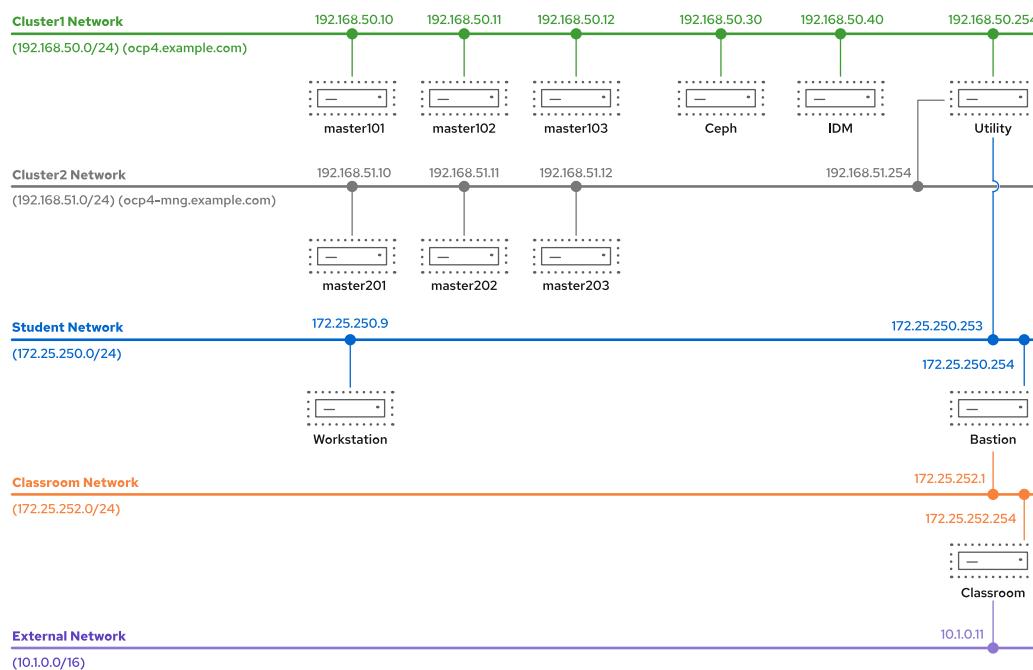
Click **htpasswd_provider** and provide the log in credentials for either the **admin** or **developer** user.

The Classroom Environment

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

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

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



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

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

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

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

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

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

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

Classroom Machines

Machine name	IP addresses	ROLE
workstation.lab.example.com	172.25.250.9	Graphical workstation used for system administration
classroom.example.com	172.25.254.254	Router linking the Classroom Network to the Internet
bastion.lab.example.com	172.25.250.254	Router linking the Student Network to the Classroom Network
utility.lab.example.com	172.25.250.253	Router linking the Student Network to the Cluster Network and also storage server
ceph.ocp4.example.com	172.25.250.252	Red Hat Storage Ceph 4.2
idm.ocp4.example.com	172.25.250.252	Red Hat Identity Management server

Classroom Machines ocp4.example.com

Machine name	IP addresses	ROLE
master101.ocp4.example.com	192.168.50.10	Control plane node
master102.ocp4.example.com	192.168.50.11	Control plane node
master103.ocp4.example.com	192.168.50.12	Control plane node

Classroom Machines ocp4-mng.example.com

Machine name	IP addresses	ROLE
master201.ocp4.example.com	192.168.51.10	Control plane node
master202.ocp4.example.com	192.168.51.11	Control plane node
master203.ocp4.example.com	192.168.51.12	Control plane node

About Screen Resolution

The web consoles used in this course use responsive web design. Responsive web design adapts user interfaces to the size of the screen. On small screen sizes, some user interface elements can change their appearance and behavior. Most screen captures in the course show the web console using a 1920x1080 screen resolution. If you use the web consoles using a different resolution, then you might see differences respect the screen captures in the course. You can increase the screen resolution to prevent responsive web design issues.

Dependencies on Internet Services

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

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

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

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

The Dedicated OpenShift Clusters

There are two Red Hat OpenShift Container Platform 4 clusters pre-installed inside the classroom environment: `ocp4.example.com` and `ocp4-mng.example.com`. All nodes are treated as bare metal servers, even though they are actually virtual machines in an OpenStack cluster.

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

- The classroom provides a Ceph server with Red Hat Storage Ceph 4.2 all-in-one pre-installed. The laboratory infrastructure automatically integrates the external Red Hat Storage Ceph 4.2 cluster with the OpenShift cluster by using the Red Hat OpenShift Container Storage operator (OCS), to provide storage to the cluster.
- The classroom provides a Red Hat Identity Management server on RHEL 8.5 named `idm`. This server provides common authentication and authorization to all OCP4 clusters managed by the Red Hat Advanced Cluster Manager.
- There is an HTPasswd Identity Provider (IdP) preconfigured with two users: `admin` and `developer`.

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

Restoring Access to your OpenShift Cluster

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

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

Troubleshooting Access to Your OpenShift Cluster

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

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

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

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

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

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

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

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

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

Approving Node Certificates on your OpenShift Cluster

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

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

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

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

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

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

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

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

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

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

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

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



Note

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

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

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

Controlling Your Systems

Controlling Your Systems

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

Controlling the Virtual Machines

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

The screenshot shows a web-based interface for managing a lab environment. At the top, there are tabs for 'Table of Contents', 'Course', 'Lab Environment', and icons for a star and help. Below this is a section titled 'Lab Controls' with instructions: 'Click CREATE to build all of the virtual machines needed for the classroom lab environment. This may take several minutes to complete. Once created the environment can then be stopped and restarted to pause your experience.' It also states: 'If you DELETE your lab, you will remove all of the virtual machines in your classroom and lose all of your progress.' A large button bar at the bottom left contains 'DELETE' (red), 'STOP' (teal), and an information icon ('i'). Below this is a table listing five virtual machines:

Machine Name	Status	Action	Open Console
bastion	active	ACTION -	OPEN CONSOLE
classroom	active	ACTION -	OPEN CONSOLE
servera	building	ACTION -	OPEN CONSOLE
serverb	building	ACTION -	OPEN CONSOLE
workstation	active	ACTION -	OPEN CONSOLE

Figure 0.2: An example course Lab Environment management page

Machine States

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

Classroom Actions

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

Machine Actions

Button or Action	Description
OPEN CONSOLE	Connect to the system console of the virtual machine in a new browser tab. You can log in directly to the virtual machine and run commands, when required. Normally, log in to the workstation virtual machine only, and from there, use <code>ssh</code> to connect to the other virtual machines.
ACTION > Start	Start (power on) the virtual machine.
ACTION > Shutdown	Gracefully shut down the virtual machine, preserving disk contents.
ACTION > Power Off	Forcefully shut down the virtual machine, while still preserving disk contents. This is equivalent to removing the power from a physical machine.
ACTION > Reset	Forcefully shut down the virtual machine and reset associated storage to its initial state. All saved work on that system's disks is lost.

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

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

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



Warning

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

The Auto-stop and Auto-destroy Timers

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

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

Performing Lab Exercises

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

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

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

The syntax for running an exercise script is as follows:

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

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

start

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

grade

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

finish

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

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

Chapter 1

Managing a Multicluster Kubernetes Architecture

Goal

Describe multicluster architectures and use Red Hat OpenShift Platform Plus to solve their challenges.

Objectives

- Identify common use cases for multiple Kubernetes clusters and the challenges presented by multicluster architectures.
- Identify the tools provided by OpenShift Platform Plus to facilitate the end-to-end management, security, and compliance of Kubernetes fleets on hybrid clouds.
- Install the RHACM operator from the operator hub in a Red Hat OpenShift hub cluster.

Sections

- The Kubernetes Multicluster and Hybrid Cloud Architecture Landscape (and Quiz)
- Describing Red Hat OpenShift Platform Plus (and Quiz)
- Installing Red Hat Advanced Cluster Management for Kubernetes (and Guided Exercise)

Lab

- Installing the RHACM Operator and importing clusters

The Kubernetes Multicloud and Hybrid Cloud Architecture Landscape

Objectives

- Identify common use cases for multiple Kubernetes clusters and the challenges presented by multicloud architectures.

Interpreting the Background of Kubernetes Multicloud 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 many 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 run on a global scale
- Establishing new methodologies that support the work of modernizing, building, and adapting 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 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 storing their data. 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
- Autoscaling 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 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 infrastructures are not always clear, particularly as the technologies evolve. For example, many public cloud providers install services, and sometimes physical resources, in their customer's private data centers.

Using multiple clouds has the following benefits:

- High availability and failover capabilities: Other clouds can assume the workload if the primary cloud is down.
- 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 close to the user, although this depends on the customer's location.
- Regulatory restrictions: In some cases, regulatory data restrictions can apply.
- Network latency: Network traffic latency is a critical concern for some industries like Telco.

The following diagram shows an example of a hybrid cloud; it 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.

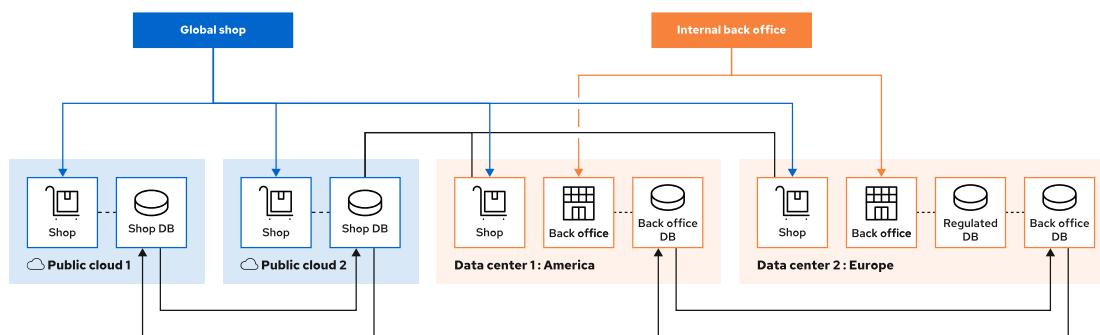


Figure 1.1: A hybrid cloud example

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 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 reusable 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 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 challenging to implement but are the key to rapidly bringing new business value through software development.



Note

To learn more about CI/CD and GitOps, see *Introducing the RHACM Application Model*.

Challenges of Multicloud Architectures

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

The following sections discuss the main challenges to adopting a multicloud 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, distributed business applications, and sometimes inconsistent security controls across 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 complies with standard or custom policies. 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 ensure enforcement.

Another concern of IT departments from a security viewpoint is the origin of the software that runs in the Kubernetes clusters. Container packaging techniques make it 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.

Multicloud architectures need repositories of IT artifacts available from different clouds. This need represents another challenge that affects the applications and the deployment of infrastructure in the multicloud 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 multicloud architecture:

Technology or process	Challenges introduced by multicloud architectures
Cloud computing	<ul style="list-style-type: none"> - 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 for performance and scalability

Technology or process	Challenges introduced by multicluster architectures
Containers	<ul style="list-style-type: none"> - Managing communications between services provided by containers, even when they are in different clusters or clouds - Requires a containers orchestration system - Requires a controlled place to store the artifacts and images that containers use - Requires security scanning of the artifacts and images that containers use
Automation processes	<ul style="list-style-type: none"> - 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

The Kubernetes Multicloud and Hybrid Cloud Architecture 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. Monolithic applications
 - 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 multicloud 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

The Kubernetes Multicloud and Hybrid Cloud Architecture 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. Monolithic applications
 - 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 multicloud 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

- Identify the tools provided by OpenShift Platform Plus to facilitate the end-to-end management, security, and compliance of Kubernetes fleets on hybrid clouds.

Describing Red Hat OpenShift Platform Plus

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

Red Hat OpenShift Platform Plus is a collection of software tools running on Red Hat OpenShift Container Platform. OpenShift Platform Plus allows administrators to manage the software lifecycle across Kubernetes and Red Hat OpenShift clusters. These tools provide end-to-end visibility and control over multiple Kubernetes clusters from a single user interface. Furthermore, OpenShift Platform Plus brings Kubernetes-native security capabilities to protect the software supply chain, infrastructure, and workloads. Finally, OpenShift Platform Plus 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 multicloud 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 OpenShift or Kubernetes clusters across multiple private and public clouds.
- Find and modify any Kubernetes resource across the entire domain by using the built-in search engine.
- Troubleshoot and resolve issues by 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 the lifecycle of applications across data centers and hybrid clouds.
- Visualize cluster metrics from a centralized monitoring stack.
- Create and manage alerts that are 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 gain the following advantages:

Visibility

Centralized viewing 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

Visualizing allowed and active traffic, simulating network policy changes, network policy recommendations, and network enforcement capabilities.

Risk profiling

Deployment ranking based on security risk calculation and security tracking to validate configuration changes.

Configuration management

Prebuilt 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 deployments.

Runtime detection and response

System-level event monitoring, automatically allowing process activity, prebuilt policies to detect crypto mining, privilege escalation, and various other exploits, and system-level data collection that uses the external Berkeley Packet Filter (eBPF) or other Linux kernel modules.

Integration

API and prebuilt plug-ins 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 roll back 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 by using build triggers, git hooks, and robot accounts.
- Auditing of CI and API actions.
- Integrated Prometheus metrics export.

Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation is a collection of software-defined storage for containers that runs on OpenShift. Thus, you can use and install it on public or private clouds, on virtual infrastructure, and on bare-metal environments.

OpenShift Data Foundation provides the following list of storage components for Kubernetes and OpenShift clusters:

- Block storage devices
- Shared file system
- On premises object storage
- Multicloud object storage, which abstracts the access to different cloud object stores by using a S3 API endpoint

Challenges Addressed by Red Hat OpenShift Platform Plus

The challenges of multicloud architectures are discussed in *Challenges of Multicloud Architectures*. The following table relates these challenges to the Red Hat OpenShift Platform Plus software collection.

Challenges of multicloud architectures	OpenShift Platform Plus software solution
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

Challenges of multicloud architectures	OpenShift Platform Plus software solution
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, Red Hat OpenShift Data Foundation
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 OpenShift Data Foundation

https://access.redhat.com/documentation/en-us/red_hat_openshift_data.foundation

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 does Red Hat OpenShift Platform Plus include? (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 Red Hat OpenShift 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 OpenShift 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 does Red Hat OpenShift Platform Plus include? (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 Red Hat OpenShift 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 OpenShift 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

Installing Red Hat Advanced Cluster Management for Kubernetes

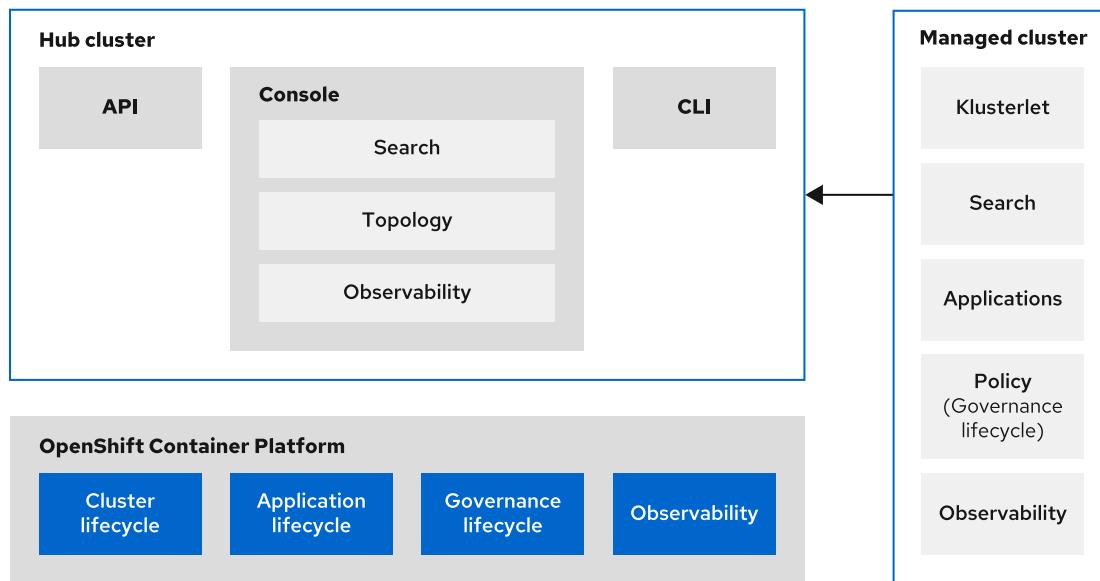
Objectives

- Install the RHACM operator from the operator hub in a Red Hat OpenShift hub cluster.

Introducing Red Hat Advanced Cluster Management for Kubernetes

Red Hat Advanced Cluster Management for Kubernetes (RHACM) gives you visibility into your OpenShift and Kubernetes clusters from a single user interface. RHACM provides built-in governance, cluster lifecycle management, application lifecycle management, and 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 following diagram provides a high-level overview of an RHACM deployment.



The RHACM operator creates more than 20 necessary custom resource definitions (CRDs) for the RHACM components.

RHACM Components on the Hub Cluster

The central controller runs the core components of RHACM running in the hub cluster. The **MultiClusterHub** object, part of the RHACM operator installation, is responsible for managing, upgrading, and installing the components of RHACM. You can interact with RHACM by using the Kubernetes API, the RHACM web console, and the `oc` or `kubectl` commands.

In RHACM, the local cluster is the Red Hat OpenShift cluster hosting the hub cluster. The hub cluster is also a managed cluster unless you specify otherwise when creating the **MultiClusterHub** object.

When you install RHACM, the operator creates the following namespaces in the hub cluster:

- **open-cluster-management**: contains the deployments for the main features of RHACM, such as the web console, the applications for the search engine, the deployments that manage applications and their topologies, and the deployments that manage the lifecycle of the clusters.
- **open-cluster-management-hub**: contains the deployments that receive data from the managed clusters.
- **open-cluster-management-observability**: contains all the Thanos deployments that store, create and visualize the observability data. This namespace is only present if you enable the RHACM observability engine.

If you manage the hub cluster as another managed cluster, the RHACM operator creates also the namespaces present on the managed clusters. Furthermore, RHACM creates a namespace for each managed cluster.

RHACM Components on the Managed Clusters

A managed cluster is an additional Kubernetes or OpenShift cluster that reports data to, and is managed by the hub cluster. As displayed in the previous diagram, the *klusterlet* agent controls the connection between clusters. The hub cluster controls managed clusters by communicating with the klusterlet agent running on them. Apart from the core components running in the hub cluster, every managed cluster runs agents to communicate with the hub cluster. The following list includes these agents, also called add-ons:

- the klusterlet agent
- the search engine add-on
- the add-on for managing application deployments from RHACM
- the add-on for applying policies
- the add-on for the observability components

When you install RHACM, the operator creates the following namespaces in the managed clusters:

- **open-cluster-management-agent**: contains the deployments for the klusterlet agent
- **open-cluster-management-agent-addon**: contains the application's deployments, search engine, and policies engines
- **open-cluster-management-agent-addon-observability**: contains the observability engine's deployments

RHACM uses capabilities from the Red Hat OpenShift local cluster for cluster lifecycle management, application lifecycle management, applying governance, risk and compliance policies, and observability.

Prerequisites and Sizing for Installing RHACM

The following list shows the basic prerequisites to install RHACM.

- An OpenShift or Kubernetes cluster listed in the Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix [<https://access.redhat.com/articles/6218901>].
- A browser listed in the Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix [<https://access.redhat.com/articles/6218901>].
- Network communication between the hub cluster and the managed clusters. For more specific information about network ranges and ports, refer to the *Network configuration*

chapter in the *Install* guide at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/install/installing#network-configuration.

The following list describes some of the factors that determine RHACM resource usage:

- The platform on which RHACM runs
- The number of managed clusters in the fleet
- The speed of storage or networks
- The number of applications in each cluster

Thus, it is not possible to accurately calculate the resources that a RHACM hub cluster needs.

For example sizing of an RHACM installation that creates and manages 1000 single node OpenShift clusters in different infrastructures, refer to the *Sizing your cluster* chapter in the *Install* guide at https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/install/index#sizing-your-cluster.

Installing RHACM

Red Hat recommends that you install the Advanced Cluster Management for Kubernetes operator from the **OperatorHub** menu in the RHOCP 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 **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 choosing between automatic and manual **Update approval** strategies. Choosing the manual update approval approach requires human intervention to update whenever a new version of the operator is available in the update stream. Alternatively, the automatic update approval will trigger an update whenever a new operator version is available.

After the operator is installed, you must create the **MultiClusterHub** custom resource to deploy the RHACM central services. You are required to perform these steps elsewhere in this course.

You can also install the RHACM operator by using the command line. 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. You can find that route in the `Networking > Routes` menu by using the Red Hat OpenShift web console. Alternatively, you 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 Red Hat Advanced Cluster Management

When the `MultiClusterHub` custom resource installation completes, the next step is to aggregate or import more Kubernetes or OpenShift 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 OpenShift cluster where the hub cluster runs) is automatically imported into RHACM, unless otherwise specified during the creation of the `MultiClusterHub` object.

You can import new clusters from the web console or by using the command line.

The following prerequisites are common to both import methods:

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

Importing Existing Clusters by Using the RHACM Web Console

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

1. From the RHACM web console, navigate to the `Infrastructure > Clusters` menu.
2. Click **Add a cluster**.
3. Click **Import an existing cluster**. Give the import a name.
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. Paste and run the copied command.
8. Return to the **Infrastructure > Clusters** menu.

The newly imported cluster is immediately displayed in the **Infrastructure > Clusters** menu. It takes several minutes to see all the cluster details. This is because RHACM is installing the add-on 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 Command Line

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

The following steps are a summary of the actions that you must perform in each cluster, for a better understanding of the process.



Note

For detailed instructions on how to import clusters into RHACM using the command line, 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` custom resource (CR) with the `imported-cluster` name.
4. Create a new `KlusterletAddonConfig` CR with the klusterlet configuration file.

In the hub cluster, you must extract existing secrets and generate YAML files containing the following resources:

1. The klusterlet CRD. For instance in a `klusterlet-crd.yaml` file.
2. The secret for importing new clusters. For instance in a `import.yaml` file.

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

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

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 add-ons running on the target cluster namespace `open-cluster-management-agent-addon`.

Removing Imported Clusters from Red Hat Advanced Cluster Management

To remove an imported cluster from the RHACM console, navigate to the **Clusters** page and click **Actions > Detach cluster** from the options for the cluster to remove.

To remove an imported cluster by using the command line, log in to the hub cluster, locate the correspondent `managedcluster` object, and use the `oc` command to delete it.

Because RHACM removes all the add-ons and other RHACM components from the managed cluster, the detach process takes a few minutes.

To clean up all the elements that RHACM creates in an imported cluster, remove the following elements from the cluster:

1. The `klusterlet` cluster role.
2. The `open-cluster-management:klusterlet-admin-aggregate-clusterrole` cluster role.
3. The `klusterlet` cluster role binding.



References

For more general information about RHACM, 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 about performance and scalability, refer to the *Performance and scalability* chapter in 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/installing#performance-and-scalability

For more information about installation, 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 about managing clusters with RHACM, 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

Installing Red Hat Advanced Cluster Management for Kubernetes

- 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 multicloud-acm
```



Note

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

- 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, navigate to the Red Hat OpenShift web console at <https://console-openshift-console.apps.ocp4.example.com>. When prompted, click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
 - 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** field.

The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar is titled 'Administrator' and includes 'Home', 'Overview', 'Projects', 'Search', 'API Explorer', 'Events', 'Operators' (selected), and 'OperatorHub' (selected). Under 'Operators', there are also 'Installed Operators' and 'Workloads'. The main content area is titled 'OperatorHub' and displays a message about discovering operators from the Kubernetes community and Red Hat partners. It shows two operator catalogs: 'do480 Operator Catalog' for 'Advanced Cluster Management for Kubernetes' and another 'do480 Operator Catalog' for 'Gatekeeper Operator'. A link at the bottom provides the URL: <https://console-openshift-console.apps.ocp4.example.com/operatorhub/all-namespaces?keyword=Advanced+Management&details-item=advanced-cluster-management-do480-catalog-openshift-marketplace>.

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

In the **Update Channel**, ensure that the **release-2.4** radio button is selected. In the **Update approval** section, select the **Manual** radio button to select a manual approval strategy. Then, click **Install**.

The screenshot shows the 'Advanced Cluster Management for Kubernetes' operator details page. The 'Install' button is highlighted. To its left, there is a note: 'Namespace creation Namespace open-cluster-management does not exist and will be created.' Below this, there is a 'Select a Namespace' section. Under 'Update approval', the 'Manual' radio button is selected. To the right of the main content, there are several informational cards: 'ClusterManager', 'App Subscription', 'Channel', and 'Helm Release'. At the bottom of the page, there is a note: 'Represent a helm chart selected by the operator'.

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:

The screenshot shows the Red Hat OpenShift Container Platform web console. On the left, the navigation sidebar includes 'Administrator', 'Home', 'Overview', 'Projects', 'Search', 'API Explorer', 'Events', 'Operators', 'OperatorHub' (which is selected), and 'Installed Operators'. The main content area displays an operator named 'Advanced Cluster Management for Kubernetes' version 2.4.2, provided by Red Hat. It has a green checkmark icon. Below the operator name, it says 'Installed operator - operand required' with the note: 'The Operator has installed successfully. Create the required custom resource to be able to use this Operator.' Underneath, there is a section for 'MultiClusterHub' with a red error icon and the text 'Required Advanced provisioning and management of OpenShift and Kubernetes clusters'. A blue 'Create MultiClusterHub' button is present. At the bottom, a link says 'View installed Operators in Namespace open-cluster-management'.

Click **Create MultiClusterHub**.

On the **Create MultiClusterHub** page, leave the default values and click **Create**. You are redirected to the **MultiClusterHubs** tab.

Initially, the **multicloudhub** object has a **Phase: Installing** status.

After some minutes, the status **Phase: Running** displays in the **Status** column.

► 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 API server address 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. From the workstation machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click **htpasswd_provider** and log in as the **admin** user with the **redhat** password.

- 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**.

The screenshot shows the Red Hat Advanced Cluster Management for Kubernetes web interface. The left sidebar has sections for Home, Infrastructure (selected), and Applications. Under Infrastructure, 'Clusters' is selected. The main content area is titled 'Clusters' and shows a 'Managed clusters' section with one entry: 'local-cluster' (Status: Ready). Below this is a search bar and a table with columns: Name, Status, Infrastructure provider, Distribution version, Labels, and Nodes. At the top right of the main content area, there is a blue button labeled 'Import cluster' which is highlighted with a red box. Other buttons like 'Create cluster' and 'Actions' are also visible.

On the **Import an existing cluster** page, type the name to identify this cluster as follows:

- **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**.

The screenshot shows the Red Hat Advanced Cluster Management for Kubernetes web interface. The left sidebar has a navigation menu with 'Clusters' selected under 'Infrastructure'. The main content area is titled 'Import an existing cluster'. It contains two sections: '1. Copy this command' for OpenShift 4 and Kubernetes 1.16.0 clusters, which includes a 'Copy command' button, and '2. Run this command with kubectl configured for your targeted cluster to start the import', which includes a note about compatibility with OpenShift 3.11 clusters and a 'Copy command for OpenShift 3.11' button.

**Note**

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

- 3.4. From the terminal, log in to the `ocp4-mng.example.com` cluster as the `admin` user. The API server address 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...
```

- 3.5. Paste the import code into the terminal and then press Enter to run it. Paste content from the clipboard by pressing `Ctrl+Shift+V` or use your preferred method. The paste command is quite long and most of it is base64 encoded.

```
[student@workstation ~]$ echo "Ci0tLQph.....G9ydCBhZ2Fpbj4=" | 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.

Name	Status	Infrastructure provider	Distribution version	Labels	Nodes
local-cluster	Ready	Other	OpenShift 4.10.3 Upgrade available	13 labels	3
managed-cluster	Ready	Other	OpenShift 4.10.3 Upgrade available	11 labels	3

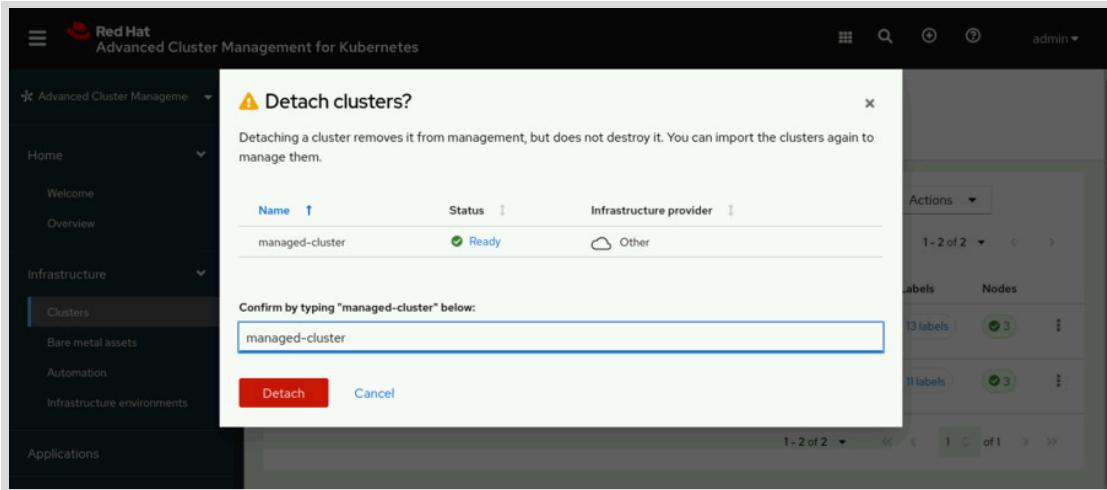
The **managed-cluster** is displayed in the RHACM web console almost immediately after executing the import code. Initially, you will see limited information about the cluster. After a few minutes, all the details of the cluster will become available.

▶ 4. Remove the imported cluster.

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

In the **Infrastructure > Clusters** pane, click the vertical ellipsis (⋮) menu to the right of the **managed-cluster**, and then click **Detach cluster**.

Type the **managed-cluster** cluster name in the confirmation window, and then click **Detach**.



The status changes to **Detaching** and after a few minutes the **managed-cluster** is completely detached.



Warning

Wait until the managed cluster is completely detached before continuing with the next step.

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

5.1. Delete the `multicloudhub` object.

From the Red Hat OpenShift web console, navigate to **Operators > Installed Operators** and click **MultiClusterHub** in the **Provided APIs** column.

Name	Managed Namespaces	Status	Provided APIs
Advanced Cluster Management for Kubernetes	open-cluster-management	Succeeded Up to date	multicloudhubs.operator.open-cluster-management.io

To the right of the `multicloudhub` object, click the vertical ellipsis menu, and then click **Delete MultiClusterHub**.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar is titled 'Administrator' and includes 'Home', 'Overview', 'Projects', 'Search', 'API Explorer', 'Events', 'Operators' (selected), 'OperatorHub', and 'Workloads'. Under 'Operators', 'Installed Operators' is selected. The main content area is titled 'Project: open-cluster-management'. It shows the 'Advanced Cluster Management for Kubernetes' operator details. Below this, there is a table titled 'MultiClusterHubs' with one entry: 'multiclustherhub' (Kind: MultiClusterHub, Status: Phase: Running). There are 'Edit MultiClusterHub' and 'Delete MultiClusterHub' buttons.

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

Warning

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

5.2. Uninstall the Advanced Cluster Management for Kubernetes operator.

From the OpenShift 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**.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar is titled 'Administrator' and includes 'Home', 'Overview', 'Projects', 'Search', 'API Explorer', 'Events', 'Operators' (selected), 'OperatorHub', and 'Workloads'. Under 'Operators', 'Installed Operators' is selected. The main content area is titled 'Project: open-cluster-management'. It shows the 'Advanced Cluster Management for Kubernetes' operator details. The status is listed as 'Succeeded Up to date'. There are 'Edit Subscription' and 'Uninstall Operator' buttons.

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. From the terminal, log in to the `ocp4.example.com` cluster as the `admin` user.

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

5.4. 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 ~]$
```

5.5. Delete the `klusterlet` and `open-cluster-management:klusterlet-admin-aggregate-clusterrole` cluster roles.

```
[student@workstation ~]$ oc delete clusterrole klusterlet
clusterrole.rbac.authorization.k8s.io "klusterlet" deleted
[student@workstation ~]$ oc delete clusterrole \
  open-cluster-management:klusterlet-admin-aggregate-clusterrole
clusterrole.rbac.authorization.k8s.io "open-cluster-management:klusterlet-admin-aggregate-clusterrole" deleted
```

5.6. Delete the `klusterlet` cluster role binding.

```
[student@workstation ~]$ oc delete clusterrolebinding klusterlet
clusterrolebinding.rbac.authorization.k8s.io "klusterlet" deleted
```

5.7. Log in to the `ocp4-mng.example.com` cluster as the `admin` user.

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

5.8. Delete the `klusterlet` and `open-cluster-management:klusterlet-admin-aggregate-clusterrole` cluster roles.

```
[student@workstation ~]$ oc delete clusterrole klusterlet
clusterrole.rbac.authorization.k8s.io "klusterlet" deleted
[student@workstation ~]$ oc delete clusterrole \
  open-cluster-management:klusterlet-admin-aggregate-clusterrole
clusterrole.rbac.authorization.k8s.io "open-cluster-management:klusterlet-admin-aggregate-clusterrole" deleted
```

5.9. Delete the `klusterlet` cluster role binding.

```
[student@workstation ~]$ oc delete clusterrolebinding klusterlet  
clusterrolebinding.rbac.authorization.k8s.io "klusterlet" deleted
```

Finish

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

```
[student@workstation ~]$ lab finish multicloud-acm
```

This concludes the section.

▶ Lab

Installing the RHACM Operator and importing clusters

- Install Red Hat Advanced Cluster Management for Kubernetes (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 multicloud-review
```



Note

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

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

1. Install the Advanced Cluster Management for Kubernetes operator on the `ocp4.example.com` cluster using the command line. Use the `do480-catalog` offline catalog. The lab script includes the `D0480/labs/multicloud-review/operator-group.yaml`, and the `D0480/labs/multicloud-review/subscription.yaml` solution files.
2. Create the RHACM `MultiClusterHub` object. The lab script includes the `D0480/labs/multicloud-review/mch.yaml` solution file.
3. Prepare RHACM to import a cluster using the name `managed-cluster`. The lab script includes the `D0480/labs/multicloud-review/mngcluster.yaml`, and the `D0480/labs/multicloud-review/klusterlet.yaml` solution files.
4. Generate the files to import the `managed-cluster` into RHACM.
5. Import the RHOCP cluster available in the lab environment, `ocp4-mng.example.com`, into RHACM. Use `managed-cluster` as the name of the imported cluster.
6. Log back into the hub cluster and verify the `managed-cluster` status.

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 multicloud-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 multicloud-review
```

This concludes the section.

► Solution

Installing the RHACM Operator and importing clusters

- Install Red Hat Advanced Cluster Management for Kubernetes (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 multicloud-review
```



Note

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

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

- Install the Advanced Cluster Management for Kubernetes operator on the `ocp4.example.com` cluster using the command line. Use the `do480-catalog` offline catalog. The lab script includes the `D0480/labs/multicloud-review/operator-group.yaml`, and the `D0480/labs/multicloud-review/subscription.yaml` solution files.
 - From the `workstation` machine, open a terminal and log in to the `ocp4` cluster as the `admin` user. The API server address is `https://api.ocp4.example.com:6443`.

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

- 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...
```

1.3. Change to the ~/D0480/labs/multicluster-review directory.

```
[student@workstation ~]$ cd D0480/labs/multicluster-review
[student@workstation multicluster-review]$
```

1.4. Create an operator group for the open-cluster-management namespace.

First, review the file named operator-group.yaml with the following contents:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: acm-operator-group
spec:
  targetNamespaces:
    - open-cluster-management
```

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

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

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

First, review the file named subscription.yaml with the following contents:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: acm-operator-subscription
spec:
  sourceNamespace: openshift-marketplace
  source: do480-catalog
  channel: release-2.4
  installPlanApproval: Manual
  name: advanced-cluster-management
```

Then, create the Subscription object using the oc command.

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

1.6. Approve the installation of the operator.

You must approve the installation manually because you created the subscription with the Manual approval strategy.

First, retrieve the installation plans.

```
[student@workstation multicloud-review]$ oc get installplan
NAME          CSV                               APPROVAL   APPROVED
install-4k2q8 advanced-cluster-management.v2.4.2  Manual     false
```

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

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

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

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

```
[student@workstation multicloud-review]$ watch oc get csv
...output omitted...
NAME                      DISPLAY
VERSION ... PHASE
advanced-cluster-management.v2.4.2  Advanced Cluster Management for Kubernetes
2.4.2    ... Succeeded
```

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

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

2. Create the RHACM `MultiClusterHub` object. The lab script includes the `D0480/labs/multicloud-review/mch.yaml` solution file.
 - 2.1. From the terminal, review the file named `mch.yaml` containing the definition of the `MultiClusterHub` object.

```
apiVersion: operator.open-cluster-management.io/v1
kind: MultiClusterHub
metadata:
  name: multicloudhub
  namespace: open-cluster-management
spec: {}
```

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

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



Note

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

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

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

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

The creation of the `MultiClusterHub` object is complete.

3. Prepare RHACM to import a cluster using the name `managed-cluster`. The lab script includes the `D0480/labs/multicluster-review/mngcluster.yaml`, and the `D0480/labs/multicluster-review/klusterlet.yaml` solution files.

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

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

- 3.2. Label the namespace with the cluster name used by RHACM.

```
[student@workstation multicluster-review]$ oc label namespace managed-cluster \
  cluster.open-cluster-management.io/managedCluster=managed-cluster
namespace/managed-cluster labeled
...output omitted...
```

- 3.3. Review the file named `mngcluster.yaml` with the following contents:

```
apiVersion: cluster.open-cluster-management.io/v1
kind: ManagedCluster
metadata:
  name: managed-cluster
spec:
  hubAcceptsClient: true
```

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

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

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

```
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: 2.4.2

```

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

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

Next, the `ManagedCluster-Import-Controller` generates a secret named `managed-cluster-import`. This secret contains the `import.yaml` file that contains the secret used in the imported cluster.

- 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 multicluster-review]$ oc get secret managed-cluster-import \
-n managed-cluster -o jsonpath={.data.crd\\.yaml} | base64 \
--decode > klusterlet-crd.yaml
```

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

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

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

- 5.** Import the RHOCP cluster available in the lab environment, `ocp4-mng.example.com`, into RHACM. Use `managed-cluster` as the name of the imported cluster.

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

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

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

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

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

```
[student@workstation multicloud-review]$ 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 multicloud-review]$ 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-bwkvf         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 multicloud-review]$ 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 54s	3/3	Running	0
klusterlet-addon-search-7b5bb78f98-h2jlg 53s	1/1	Running	0
klusterlet-addon-workmgr-5b84cf6dc-nxwvt 52s	1/1	Running	0

**Note**

It takes around two minutes before the pods in the `open-cluster-management-agent-addon` start.

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

6. Log back into the hub cluster and verify the managed-cluster status.

6.1. Log back into the hub cluster.

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

6.2. Verify the managed-cluster status.

```
[student@workstation multicloud-review]$ 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` must have a `True` status.

6.3. Change to the `/home/student` directory.

```
[student@workstation multicloud-review]$ cd /home/student
[student@workstation ~]$
```

Evaluation

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

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

This concludes the section.

Summary

- Hybrid cloud environments and multicloud architectures bring new challenges.
- Red Hat OpenShift Platform Plus tools help to address the challenges of multicloud architectures.
- Red Hat Advanced Cluster Management for Kubernetes (RHACM) runs in a hub cluster.
- You can import managed clusters into RHACM.

Chapter 2

Inspecting Resources from Multiple Clusters Using the RHACM Web Console

Goal

Describe and navigate the Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console. Configure role-based access control (RBAC) and search for 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 Red Hat Advanced Cluster Management for Kubernetes (RHACM) features through the web console.
- Create different user roles in Red Hat Advanced Cluster Management for Kubernetes (RHACM) and define an authentication model for multicloud management.

Sections

- Navigating the RHACM Web Console (and Guided Exercise)
- Configuring Access Control for Multicloud Management (and Guided Exercise)

Lab

- Managing Resources from Multiple Clusters Using the RHACM Web Console

Navigating the RHACM Web Console

Objectives

- Locate objects across a fleet of managed clusters by using the search engine and enumerate Red Hat Advanced Cluster Management for Kubernetes (RHACM) features through the web console.

RHACM Web Console Overview

The Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console provides a single access point to all the management and observation features of a fleet of Kubernetes or OpenShift clusters.

You can access the different components of the RHACM web console through the navigation panel. The following sections describe the items in the navigation panel.

Home

The **Home** page provides information about RHACM and its use cases, and links to the main product features. The **Home** page 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

The **Infrastructure** menu provides access to cluster lifecycle management, bare-metal asset management, Ansible automation configuration, and infrastructure environment management. This menu provides access to the following pages:

Clusters

The **Clusters** page provides access to the following features:

- Creating or upgrading a cluster in many different public cloud providers.
- Importing an existing cluster to the RHACM hub cluster so that you can manage it.
- Manually scaling the clusters, or enabling autoscaling. The process 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 discovering 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 interact with the bare metal assets by using their Baseboard Management Controller (BMC) microcontrollers. The **Bare metal assets** page integrates bare metal assets into RHACM via their BMC address. RHACM uses the integration of bare metal assets to deploy and manage clusters hosted by the virtual or physical infrastructure.

Automation

Use the **Automation** page to create Ansible job templates and run Ansible jobs automatically during different stages of a cluster lifecycle. To create Ansible templates you need Ansible Automation Platform Resource Operator installed on the RHACM hub cluster. You also need Ansible Tower 3.7.3 or later available.

Infrastructure environments

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

Applications

You can use the **Applications** page to create, deploy, and manage applications across a cluster fleet.

You can find more information in *Chapter 5, Deploying Applications Across Multiple Clusters with RHACM*.

Governance

You can use the **Governance** page to create and manage policies and policy controllers, and apply those to the cluster fleet.

You can find more information in *Chapter 3, Deploying and Managing Policies for Multiple Clusters with RHACM*.

Credentials

In RHACM, a credential stores the access information for a cloud provider. The RHACM credentials use the format of Kubernetes secrets objects. Each credential has two keys: the cloud provider

access information, and a DNS name within that cloud provider. RHACM uses the following credential types:

- Cloud provider credentials: For example, Amazon Web Services, Google Cloud Platform, and 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: A credential type for on-premise environments.

You can use the [Credentials](#) page to create and administer credentials for all cloud providers and systems.

RHACM Search Engine

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

The RHACM search engine is always enabled. Click the **Search** icon in the upper-right of the console to access it.

The search engine indexes and stores Kubernetes objects in the cluster fleet, and determines their relationships with other objects.

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

Collector

A collector is deployed in each of the fleet clusters. In the hub cluster, the search collector is deployed in the `open-cluster-management` namespace. In the other 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 Kubernetes objects information and computes relationships for objects within the managed clusters.

Aggregator

The aggregator is only deployed in the RHACM hub cluster, in the `open-cluster-management` namespace. The aggregator collects data from multiple collectors in different managed clusters, and writes that data to a Redis database 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

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

Search UI

The search UI is deployed as part of the RHACM `MulticlusterHub` object.

Using the Search Engine

You can type free text in the **Search** field of the user interface. The search engine tries to locate that text in the Redis database of indexes, and shows results for any object that contains that string. The UI orders the results in a table by type of Kubernetes or OpenShift object, represented by the `kind` filter.

You can refine each search by adding more filters to the query. As you add new filters, the UI displays the values that are stored for that index.

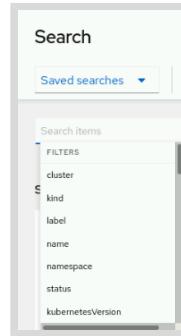
Figure 2.1: Example of search auto-completion

Some filters allow arithmetic comparators for fields of numeric nature, such as the `cpu`, `replicas`, `capacity`, and `memory` filters.

You can also use the search UI to edit objects. If you click an object displayed in a result, you can 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 in the cluster fleet. Those filters are the keys of the indexing process. The **Search** UI provides access to all the available filters.



You can use any of the filters to search, and you can also make free text searches. You cannot make a search refining by all the fields of the Kubernetes object: if the field is not part of a filter, it is not indexed for search.

For example, the `Deployment` Kubernetes object has `desired` and `replicas` fields defined in its specification. The RHACM search engine only indexes the `desired` field, however, and you cannot search with the `strategy` filter, which does not exist in the index.



Note

The RHACM search engine always uses the `label` filter to index every Kubernetes object. It is a good practice to set labels on objects during the different phases of the CI/CD process, to make more accurate and faster 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 RHACM Web Console

- Use the Red Hat Advanced Cluster Management for Kubernetes (RHACM) console to list the failed deployments, diagnose the cause of a failed MySQL deployment, and fix it. You also identify which database server is most used by developers for their relational persistence layer in the applications. You also ensure that there are no running containers in any cluster that are using a specific MySQL image with known vulnerabilities.

Outcomes

- Use saved searches in the RHACM web console to quickly identify the failing deployments across a cluster fleet.
- Use the RHACM web console to locate Kubernetes and OpenShift objects across a cluster fleet.
- Use the RHACM web console to edit Kubernetes objects across a cluster fleet.

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 populates them with nine different applications using three different database servers.



Note

The preparation of this lab changes the screen resolution to 1920x1080 for better viewing of the RHACM console. The resolution is reverted at the end of the lab.

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

- 1. Use the RHACM web console **Saved searches** feature to identify any failed database server deployments across the cluster fleet.
- 1.1. From the **workstation** machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click **htpasswd_provider** and log in as the **admin** user with the **redhat** password.
 - 1.2. Click the **Search** icon in the upper-right corner of the window to open the RHACM search web interface.
 - 1.3. Click the **Unhealthy pods** tile in the list of suggested search templates. Notice that the syntax of the search is based on the status of the pods.



Figure 2.3: Unhealthy pods search syntax

**Note**

The search results can show pods unrelated to MySQL deployments. Ignore those results and use the results related to the MySQL deployments.

Notice the two failing pods with names `mysql-POD_ID`. The two failing pods are in the `company-applications-5` namespace. Each managed cluster contains a `company-applications-5` namespace.

Name	Namespace	Cluster	Status	Restarts	Host IP	Pod IP	Created	Labels
mysql-finance-application-2-5648465868-4tzz64	company-applications-5	managed-cluster	Pending	0			4 hours ago	app=mysql-finance-application-2 pod-template-hash=5648465868
mysql-finance-application-2-5648465868-gmv45	company-applications-5	local-cluster	Pending	0			4 hours ago	app=mysql-finance-application-2 pod-template-hash=5648465868

Figure 2.4: Unhealthy pods search results

- 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. This section is near the end of the YAML file.

Notice that the pods are pending because there is a reference to a Persistent Volume Claim (PVC) that does not exist.

```

197   status:
198     phase: Pending
199     conditions:
200       - type: PodScheduled
201         status: 'False'
202         lastProbeTime: null
203         lastTransitionTime: '2022-03-31T22:16:19Z'
204         reason: Unschedulable
205         message: >-
206           0/3 nodes are available: 3 persistentvolumeclaim "nonexistingdbclaim"
207             not found.
208     qosClass: BestEffort
209

```

Figure 2.5: Message indicating the reason for the pod failure

- Navigate back to **Unhealthy pods** in the list of suggested search templates and then click **Related deployment**. The deployment that contains the pods is `mysql-finance-application-2`. The deployment is present in all the managed clusters.
- Click the **Search** icon again to search for 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`.

The screenshot shows the RHACM Web Console search interface. The search bar at the top contains the query "namespace:company-applications-5 kind:persistentvolumeclaim". Below the search bar, there are two tabs: "Related cluster" (2) and "Related persistentvolume" (2). The "Related persistentvolume" tab is selected. Under the heading "Persistentvolumeclaim (2)", there is a table listing two entries. Both entries have their names highlighted with red boxes: "dbclaim" and "dbclaim". The table columns include Name, Namespace, Cluster, Status, Persistent volume, Requests, Access mode, Created, and Labels. The "Labels" column for both entries lists "app=mysql-persistent-template", "application=finance-application-2", and "template=mysql-persistent-template".

Figure 2.6: Results of the search for the existing persistent volume claims

- 1.7. Click the **Search** icon again to start a new search.
Type `namespace:company-applications-5 kind:deployment` in the search field.
- 1.8. Click the name of the first deployment in the **Deployment** results to display the YAML file. In the upper-right corner of the window, click **Edit**, and set `claimName:` to `dbclaim` as the PVC name.

```

141     labels:
142       app: mysql-finance-application-2
143     spec:
144       volumes:
145         - name: db-volume
146           persistentVolumeClaim:
147             claimName: dbclaim
148       containers:
149         - name: mysql

```

Figure 2.7: Correcting the PVC name in the YAML file for a pod

- Click **Save** to trigger a redeployment automatically using the correct PVC name.
- 1.9. Repeat the operation to fix the failing deployment in the other cluster.
 - 1.10. Navigate to the saved searches page and run the **Unhealthy pods** saved search 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 cluster fleet offered by the search engine. Notice that the company application deployment namespaces always use the `company-applications-` prefix.

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

Finally, add the free text `mysql` to the search.

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

Name	Namespace	Cluster	Desired	Current	Ready	Available	Created	Labels
mysql-finance-application-2	company-applications-5	local-cluster	1	1	1	1	17 hours ago	app=mysql-persistent-template application=finance-application-2 template=mysql-persistent-template
mysql-globalshop-application	company-applications-6	local-cluster	1	1	1	1	17 hours ago	app=mysql-persistent-template application=globalshop-application template=mysql-persistent-template
mysql-globalshop-application	company-applications-15	managed-cluster	1	1	1	1	17 hours ago	app=mysql-persistent-template application=globalshop-application template=mysql-persistent-template
mysql-globalshop-application	company-applications-6	managed-cluster	1	1	1	1	17 hours ago	app=mysql-persistent-template application=globalshop-application template=mysql-persistent-template
mysql-globalshop-application	company-applications-6	managed-cluster	1	1	1	1	17 hours ago	app=mysql-persistent-template application=globalshop-application template=mysql-persistent-template
mysql-globalshop-application	company-applications-6	managed-cluster	1	1	1	1	17 hours ago	app=mysql-persistent-template application=globalshop-application template=mysql-persistent-template

Figure 2.8: The results of search for deployments with the "mysql" free text, with the "application" label

- 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 `application` label to locate the name. Two different applications are using MariaDB containers: `humanresources-application-1` and `marketing-application-2`.

- 2.4. In the search field, remove the free text `mariadb` and replace it with `postgresql`. Do not remove the `kind:deployment, deploymentconfig` filter. Notice that PostgreSQL is deployed as OpenShift DeploymentConfig objects. Use the `application` label to locate the name.

Five different applications are 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, which is used in five of the nine applications.

3. Use the RHACM web console to locate any running containers that are using a MySQL image with known vulnerabilities.

Red Hat Container Catalog states that the image in use for MySQL 8 must be `registry.redhat.io/rhel8/mysql-80:1`, because that tag always point to the latest stable version available. For example, the MySQL image with the `registry.redhat.io/rhel8/mysql-80:1-127` tag contains important vulnerabilities and must not be used. See the Red Hat Container Catalog [<https://catalog.redhat.com/software/containers/>

rhel8/mysql-80/5ba0ad4cd19c70b45cbf48c] website for more information about this image.

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

- 3.1. Type `registry.redhat.io/rhel8/mysql-80` in the search field.
Notice that the results only contain objects of type Pod.
- 3.2. Click one of the pod names 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 the 1-152 tag.
- 3.3. Click **Search** in the upper-left of the pane to return to the initial search page, keeping the results. Do not remove the filter `registry.redhat.io/rhel8/mysql-80`, and add "1-127" in the search field.
- 3.4. Inspect the results; notice two pods running in namespaces with the same name, but in different clusters.
- 3.5. Click one of the pods to see the image in use in its YAML definition.
- 3.6. Click the **Logs** tab and verify the running version of MySQL. The 1-152 tag corresponds to the 8.0.26 version, and the 1-127 tag is the earlier one, 8.0.21. This information is stated in the Red Hat Container Catalog.

```
--> 22:17:41      Starting MySQL server with disabled networking ...
--> 22:17:41      Waiting for MySQL to start ...
2022-03-31T22:17:41.308Z 0 [Warning] [MY-011070] [Server] 'Disabling symbolic links using --skip-s
2022-03-31T22:17:41.313Z 0 [System] [MY-010116] [Server] /usr/libexec/mysqld (mysqld 8.0.21) start
2022-03-31T22:17:41.340Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2022-03-31T22:17:41.979Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
--> 22:17:42      Waiting for MySQL to start ...
```

Figure 2.9: Verifying the version of MySQL

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

Finish

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

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

This concludes the section.

Configuring Access Control for Multicluster Management

Objectives

- Create different user roles in Red Hat Advanced Cluster Management for Kubernetes (RHACM) and define an authentication model for multicluster management.

Role-based Access Control in RHACM

Red Hat Advanced Cluster Management for Kubernetes (RHACM) access control uses the Red Hat OpenShift Container Platform (RHOCP) 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 *cluster sets*, represented by the `ClusterSet` object. A cluster set labels all the clusters that belong to it with the `cluster.open-cluster-management.io/clusterSet=managed_clusterSet_name` label. RHACM uses this label to identify the clusters that belong to the cluster set, enabling you to take action on all the member clusters at once.

Role	Definition
<code>open-cluster-management:cluster-manager-admin</code>	RHACM super user, with full access. Can create a <code>ManagedCluster</code> resource.
<code>open-cluster-management:admin:managed_cluster_name</code>	RHACM administrator access to the <code>ManagedCluster</code> resource named <code>managed_cluster_name</code> .
<code>open-cluster-management:view:managed_cluster_name</code>	RHACM view access to the <code>ManagedCluster</code> resource named <code>managed_cluster_name</code> .
<code>open-cluster-management:managedclusterset:admin:managed_clusterset_name</code>	RHACM administrator access to the <code>ManagedClusterSet</code> resource named <code>managed_clusterset_name</code> .
<code>open-cluster-management:managedclusterset:view:managed_clusterset_name</code>	RHACM view access to the <code>ManagedClusterSet</code> resource named <code>managed_clusterset_name</code> .
<code>open-cluster-management:subscription-admin</code>	Can create Git subscriptions that deploy Kubernetes resources YAML files to multiple namespaces.

RHACM also supports the following default RHOCP roles:

Role	Definition
<code>cluster-admin</code>	RHOCP super user, with full access.

Role	Definition
admin	RHOCP default roles. A user with a namespace-scoped binding to these roles has access to open-cluster-management resources in a specific namespace.
edit	
view	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. 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 belong to more than one cluster set simultaneously. If you need to define overlapping groups of clusters, you can use cluster labels, acting as selectors.

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

Placement Resources

You can use placement resources to define a subset of clusters that belong to different cluster sets for placing Kubernetes resources.

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

For example, the following placement resource 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 managing the authentication and authorization mechanisms in a fully controlled manner across different Kubernetes and OpenShift clusters.

As noted previously, RHACM uses the Red Hat OpenShift Container Platform authentication and authorization layers. The OpenShift authentication layer uses the OAuth open standard as an authorization framework. By default, OpenShift uses an internal OAuth server. 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 users who access 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 Multicloud Management

- Create cluster sets using existing Red Hat OpenShift clusters, and then configure existing users in Red Hat OpenShift Container Platform (RHOC) to manage those cluster sets.

Outcomes

- Create a cluster set.
- Assign a managed cluster to a cluster set.
- Assign predefined RHACM roles to different users.
- Remove a cluster set.

Before You Begin

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

The lab environment includes one server for Identity Management in Red Hat Enterprise Linux (IdM). This command ensures that the necessary users and roles exist in the IdM server. It also configures an additional LDAP identity provider in the hub cluster.

The following table summarizes the users and groups available in the IdM server for this exercise.

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

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

- 1. From the RHACM web console, create a cluster set named `production` and add the cluster named `local-cluster`.
 - 1.1. From the `workstation` machine, navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`. When prompted, click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
 - 1.2. Navigate to `Infrastructure > Clusters` and click `Cluster sets`.

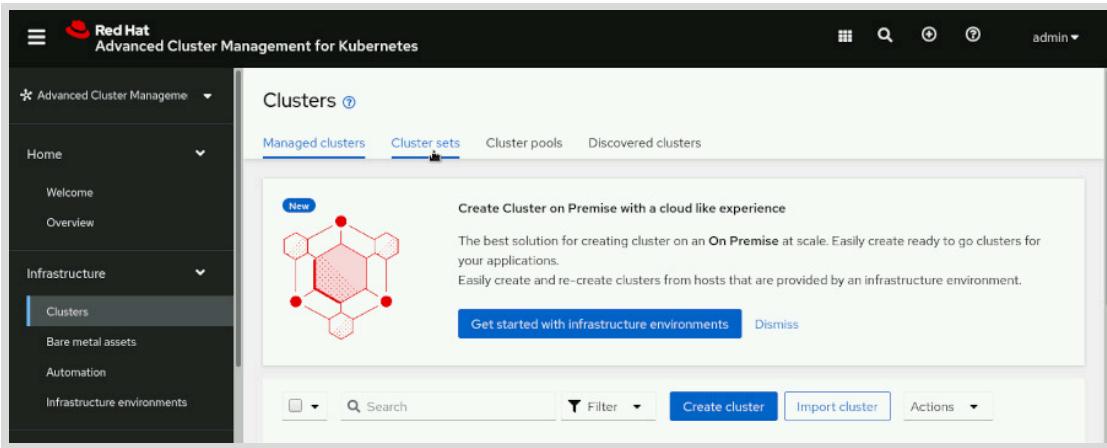


Figure 2.10: Location of the cluster sets button in the RHACM console

Scroll down and click **Create cluster set** to open the **Create cluster set** dialog box.

- 1.3. Type **production** in the **Cluster set name** field and then click **Create**.
- 1.4. In the confirmation dialog box, click **Manage resource assignments**. Select the cluster named **local-cluster** and then click **Review**.

Name	Kind	Current cluster set
<input checked="" type="checkbox"/> local-cluster	ManagedCluster	-
<input type="checkbox"/> managed-cluster	ManagedCluster	-

- 1.5. On the confirmation page, click **Save** to display the **Overview** tab of the **production** cluster set.
- 2. Verify the labels corresponding to the **production** cluster set.
- 2.1. On the **Managed clusters** page of the **production** cluster set, click **Managed clusters**.
 - 2.2. Click **14 labels** to expand the list of labels assigned to this cluster. Notice the **cluster.open-cluster-management.io/clusterset=production** label. RHACM uses this label to identify clusters pertaining to a cluster set.

The screenshot shows the 'Managed clusters' tab selected in the navigation bar. The main table displays a single cluster entry:

Name	Status	Infrastructure provider	Distribution version	Labels	Nodes
local-cluster	Ready	Other	OpenShift 4.10.3	14 labels	3

Figure 2.12: The managed clusters page of the production cluster set and the labels button



Note

The number of labels might vary.

- ▶ 3. From the RHACM web console, create a cluster set named **stage** and add the cluster named **managed-cluster**.
 - 3.1. Navigate to **Infrastructure > Clusters** and click **Cluster sets**. Scroll down and click **Create cluster set** to open the **Create cluster set** dialog box.
 - 3.2. Type **stage** in the **Cluster set name** field and then click **Create**.
 - 3.3. In the confirmation dialog box, click **Manage resource assignments**. Select the cluster named **managed-cluster** and then click **Review**.
 - 3.4. On the confirmation page, click **Save** to display the **Overview** tab of the stage cluster set.
 - 3.5. On 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.

The screenshot shows the Red Hat Advanced Cluster Management for Kubernetes web interface. The left sidebar is collapsed. The main navigation bar includes 'Advanced Cluster Management' (dropdown), 'Cluster sets > stage', 'admin' (dropdown), and a search bar. The 'Managed clusters' tab is selected. The 'stage' cluster set is shown with one entry: 'managed-cluster'. The details pane shows the following information:

- Name: managed-cluster
- Status: Ready (green checkmark)
- Infrastructure provider: OpenShift
- Distribution version: Up-to-date
- Labels: cluster.open-cluster-management.io/clusterset=stage, clusterID=e5ecc7f6-5a2d-4373-ac4d-bb7b47f321bf, feature.open-cluster-management.io/Addon-application-manager-available



Note

The number of labels might vary.

- 4. From the terminal, assign administrator permissions to the **production-administrators** group in the **production** cluster set. Assign view permissions to the **production-administrators** group in the **stage** cluster set.



Note

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

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

- 4.1. Open a terminal and log in to the ocp4 cluster as the **admin** user. The API Server 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 adm** command to assign the **open-cluster-management:managedclusterset:admin:production** cluster role to the **production-administrators** group.



Note

The **production-administrators** group already exists in the Red Hat Identity Management server in the lab environment. The **lab start** command 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:managedclusterset:admin:production \
  production-administrators
clusterrole.rbac.authorization.k8s.io/open-cluster-
management:managedclusterset:admin:production added: "production-administrators"
```

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

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

- 5. From the terminal, assign administrator permissions to the `stage-administrators` group in the `stage` cluster set. Do not assign any permissions to the `stage-administrators` group for the `production` cluster set.
- 5.1. Use the `oc adm` command to assign the `open-cluster-management:managedclusterset:admin:stage` cluster role to the `stage-administrators` group.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-group \
  open-cluster-management:managedclusterset: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

- 6. Log out of RHACM and close Firefox to remove any cached credentials in the browser. Open Firefox again and log in to the RHACM web console as 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 of the cluster named `local-cluster`. Also, verify that the `prod-admin` user only has view permissions on the `stage` cluster set.

**Note**

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

6.1. Log out of RHACM.

From the RHACM web console, click **admin** in the upper-right corner, and click **Logout**.

6.2. Close all windows of Firefox. Open a new Firefox window and log in to the RHACM web console as the prod-admin user from the Red Hat Identity Management provider.

When prompted, click **Red Hat Identity Management** and log in as the **prod-admin** user with the **redhat** password.

6.3. Navigate to **Infrastructure > Clusters**. Note the differences between the two clusters.

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

6.4. Click the vertical ellipsis (:) menu to the right of the **local-cluster** row.

Note that as an administrator, all options are enabled.

Name	Status	Infrastructure provider	Distribution version	Labels	Nodes
local-cluster	Ready	Other	OpenShift 4.10.3 Upgrade available	14 labels	
managed-cluster	Ready	Other	OpenShift 4.10.3 Upgrade available	11 labels	

A cluster set administrator can perform various actions on clusters, such as editing labels, upgrading the version of the cluster set, selecting the channel for updates, searching for cluster resources, and detaching a cluster from RHACM management.

- 6.5. Click the vertical ellipsis menu to the right of the **managed-cluster** row.

Name	Status	Infrastructure provider	Distribution version	Labels	Nodes
local-cluster	Ready	Other	OpenShift 4.10.3 Upgrade available	14 labels	3
managed-cluster	Ready	Other	OpenShift 4.10.3 Upgrade available	11 labels	3

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

- 7. Log out of RHACM and close Firefox to remove any cached credentials in the browser. Open Firefox again and log in to the RHACM web console as 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 of the cluster named **managed-cluster**. Also, verify that the **stage-admin** user does not have any permissions on the **production** cluster set.



Note

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

- 7.1. Log out of RHACM.

In the RHACM web console, click **prod-admin** at the upper-right corner, and click **Logout**.

- 7.2. Close all windows of Firefox. Open a new Firefox window and log in to the RHACM web console as the **stage-admin** user from the Red Hat Identity Management provider.

When prompted, click **Red Hat Identity Management** and log in as the **stage-admin** user with the **redhat** password.

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

The **Upgrade available** link is also active. Click the vertical ellipsis menu and notice that all options 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.

- ▶ **8.** Log out of the RHACM web console and close all Firefox windows to remove any cached credentials.
 - 8.1. From the RHACM web console, click **stage-admin** at the upper-right corner, and then click **Logout**.
Close the Firefox window.

Finish

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

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

This concludes the section.

► Lab

Managing Resources from Multiple Clusters Using the RHACM Web Console

- Assign the cluster-wide `view` role to a group of users so that they can perform searches across the cluster fleet. Then, you log in as a user from that group and search for a deployment with a number of replicas less than three. Finally, you log in to Red Hat Advanced Cluster Management for Kubernetes (RHACM) as a user with administrative permissions in the managed cluster set, and scale the deployment to three replicas.

Outcomes

- Manage RHACM roles and groups to allow searches across a cluster fleet.
- Locate Kubernetes and OpenShift objects across a cluster fleet by using the correct group of users.
- Edit Kubernetes objects across a cluster fleet 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 ensures that the necessary users and roles exist in the Red Hat Identity Management (IdM) server. Finally, it creates all the deployments across the cluster fleet.

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

- Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.
Assign the cluster-wide `view` role to the `fleet-searchers` group to allow searches across all the objects in the cluster fleet.
- Locate all Kubernetes objects from the applications that have the `app=finance-application` label and contain fewer than three replicas.
The RHACM web console is at `https://multicloud-console.apps.ocp4.example.com`. Log in as the `fleet-searcher` user with the `redhat` password.
- Change the number of replicas of the two deployments by logging in as a user belonging to the `emea-operators` group. Log in as the `emea-operator` user with the `redhat` password.

Evaluation

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

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

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

This concludes the section.

► Solution

Managing Resources from Multiple Clusters Using the RHACM Web Console

- Assign the cluster-wide `view` role to a group of users so that they can perform searches across the cluster fleet. Then, you log in as a user from that group and search for a deployment with a number of replicas less than three. Finally, you log in to Red Hat Advanced Cluster Management for Kubernetes (RHACM) as a user with administrative permissions in the managed cluster set, and scale the deployment to three replicas.

Outcomes

- Manage RHACM roles and groups to allow searches across a cluster fleet.
- Locate Kubernetes and OpenShift objects across a cluster fleet by using the correct group of users.
- Edit Kubernetes objects across a cluster fleet 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 ensures that the necessary users and roles exist in the Red Hat Identity Management (IdM) server. Finally, it creates all the deployments across the cluster fleet.

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

1. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.
Assign the cluster-wide `view` role to the `fleet-searchers` group to allow searches across all the objects in the cluster fleet.
 - 1.1. From the `workstation` machine, open a terminal and log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API Server 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"
```

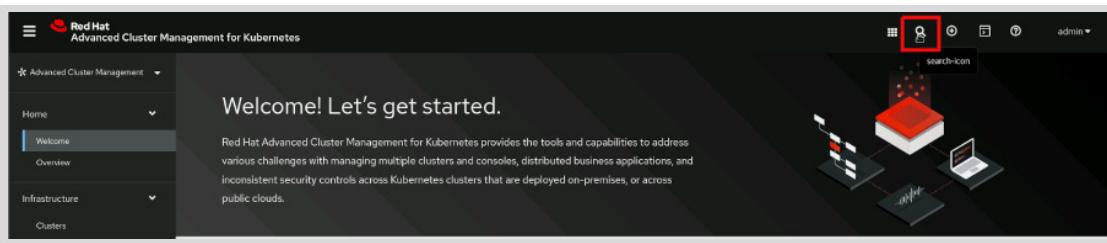
- 1.5. Reverify the **fleet-searchers** group roles to identify the additional roles that have been assigned.

```
[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 three replicas.

The RHACM web console is at <https://multicloud-console.apps.ocp4.example.com>. Log in as the **fleet-searcher** user with the **redhat** password.

- 2.1. From the **workstation** machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click **Red Hat Identity Management** and log in as the **fleet-searcher** user with the **redhat** password.
- 2.2. Click the **Search** icon to navigate to the RHACM search web interface.



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

The `fleet-searcher` user cannot modify these two deployments.

- 2.3. Click **Related cluster** in the related objects of the results. Expand the related cluster to see all the information about the cluster named `managed-cluster`.

Name	Value
Available	True
Hub accepted	True
Joined	True
Nodes	3
Kubernetes version	v1.21.1+05lac4f
CPU	12
Memory	48074Mi
Console URL	Launch
Labels	<ul style="list-style-type: none"> cluster.open-cluster-management.io/clusterset=emea clusterID=e5eec7f6-5a2d-4373-ac4c-b12b12780bf4 name=managed-cluster 2 more

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

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

3. Change the number of replicas of the two deployments by logging in as a user belonging to the `emea-operators` group. Log in as the `emea-operator` user with the `redhat` password.

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

When prompted, click **Red Hat Identity Management** and log in as the `emea-operator` user with the `redhat` password.

- 3.2. Click the **Search** icon to display the **Search** page.
Repeat the previous search by typing `label:app=finance-application current:<3` in the search field. The same two 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, and then click **Edit**.
- 3.4. Find the text `replicas: 1` in the `spec:` section of the deployment. You can use **Ctrl+F** to search inside the YAML file.
- 3.5. Change the value to `replicas: 3` and then click **Save**.
- 3.6. Repeat the preceding operation in the deployment present in the `company-applications-7` namespace.
- 3.7. Ensure that the previous search has no more results by clicking the **Search** link.

```

Cluster: managed-cluster Namespace: company-applications-7
Read only mode Edit
125   operation: Update
126   time: '2021-12-02T11:02:36Z'
127   name: mysql-finance-application
128   namespace: company-applications-7
129   resourceVersion: '162984'
130   uid: 4386f5ec-f96c-40a5-94cb-eef407616943
131 spec:
132   progressDeadlineSeconds: 600
133   replicas: 3
134   revisionHistoryLimit: 10
135   selector:
136     matchLabels:
137       app: mysql-finance-application
138   strategy:
139     rollingUpdate:
140       maxSurge: 25%
141       maxUnavailable: 25%

```

Evaluation

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

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

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

This concludes the section.

Summary

- The Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console provides access to all the product functionalities.
- The RHACM search engine can locate Kubernetes objects across a cluster fleet.
- The Role-based access control (RBAC) model of RHACM and its default roles help to define a multicluster authentication model for a cluster fleet.
- RHACM classifies the managed clusters in the concept of cluster set to apply the RBAC rules.

Chapter 3

Deploying and Managing Policies for Multiple Clusters with RHACM

Goal

Deploy and manage policies in a multicluster environment by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance.

Objectives

- Deploy policies on multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.
- Deploy the compliance operator policy and view compliance reports from multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.
- Deploy the gatekeeper policy and gatekeeper constraints to multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (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)
- Integration of Other Policy Engines with RHACM (and Guided Exercise)

Lab

- Deploying and Managing Policies with RHACM

Deploying and Managing Policies with RHACM

Objectives

- Deploy policies on multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.

Red Hat Advanced Cluster Management for Kubernetes Governance Overview

The increasing complexity of modern business systems demands that companies can quickly adapt company policies to uphold legal requirements and ensure compliance. The multicloud infrastructure must support these legal and regulatory requirements by ensuring that security standards are upheld.

Governance refers specifically to the set of rules, controls, policies, and resolutions put in place to dictate behavior. Red Hat Advanced Cluster Management for Kubernetes governance provides a policy-based approach for companies to monitor these standards automatically.

The Governance Architecture

RHACM governance provides a complete graphical and command-line interface within the policy-based framework. The governance architecture includes the following components:

RHACM governance dashboard

The RHACM governance dashboard displays a list of policies, cluster violations, and policy violations.

Governance framework

The governance framework supports policy creation and deployment to the managed clusters based on placement rules.

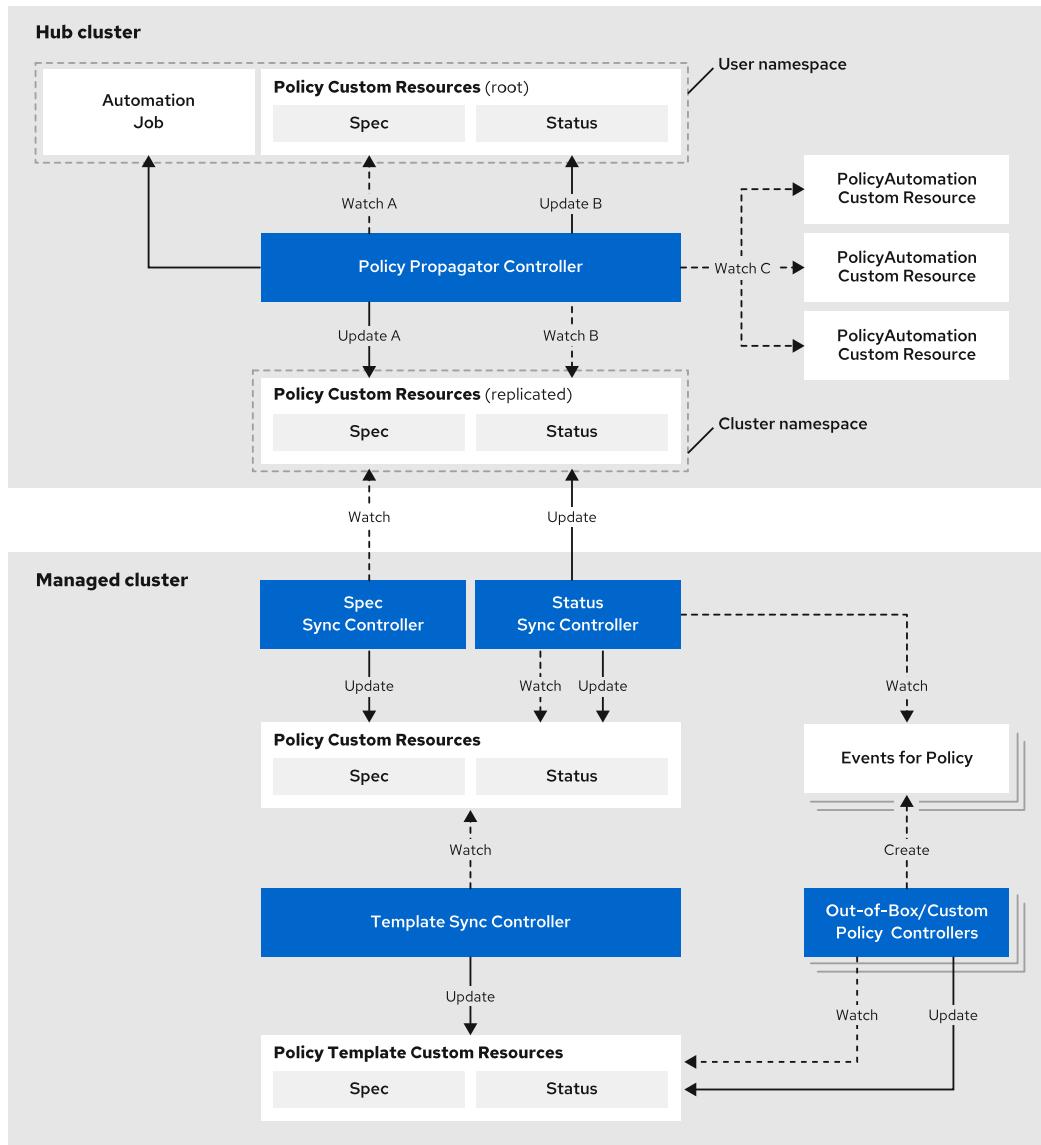
Policy controller

The policy controller monitors policies and generates Kubernetes events for violation.

Open source community

RHACM governance supports community contribution. The `stolostron/policy-collection` repository has stable and community policy examples.

The following diagram shows an overview of the RHACM governance architecture.



The Policy Propagator Controller monitors policies and updates policy status. This controller also watches PlacementRules and PlacementBindings and handles replicated policies in the cluster namespace.

The Spec Sync Controller operates on the managed cluster. This controller synchronizes local policy spec with the cluster namespace policies and updates the local policy spec if required.

The Status Sync Controller also operates on the managed cluster. This controller monitors events and changes in the managed cluster and updates the status policies for both local and hub clusters.

The Template Sync Controller operates on the managed cluster. This controller monitors policies in the cluster namespace and updates defined objects in the policies templates for any policy change.

Governance Dashboard

The governance dashboard provides an option to create, deploy, and edit policies with a graphical user interface and a YAML editor.

After the first policy creation, the governance dashboard displays a list of policies and policy violations. Use the filter list to further narrow the list of policies by Violations, Source, Remediation, and Status.

You can also change the status and remediation of selected policies from the **Actions** button. Every policy has a vertical ellipsis (⋮) menu to perform similar tasks and the policy deletion.

The screenshot shows the RHACM Governance Dashboard. At the top, there's a header with 'Governance' and a 'Create policy' button. Below the header, a section for 'NIST-CSF' shows a shield icon and the message 'No violations found'. Based on the industry standards, there are no cluster or policy violations. The main area contains a table of policies:

Policy name	Namespace	Status	Action
policy-example	default	Enabled	Enable

Below the table, there's a detailed view for the 'policy-example' policy. It shows 'Cluster violations' (0/2), 'Source' (Local), 'Controls' (PRJP-1 Baseline Configuration), 'Automation' (Configure), and 'Created' (a few seconds ago). A vertical ellipsis (⋮) menu is highlighted with a red box.

When you click a policy from the policy list, it redirects you to the following tabs:

Details

This tab shows policy details and placement details.

Clusters

This tab shows the list of all the clusters associated with the policy placement. The **View details** link shows template details, and the **View history** link shows compliance status, message, and the last report time.

Templates

This tab shows the template-wise list of all the clusters associated with the policy placement. The **View details** link shows template details, and the **View history** link shows compliance status, message, and the last report time.

RHACM Policy Overview

RHACM governance uses a policy-based framework. Every policy requires the **policy-template**, **PlacementRule** and **PlacementBinding** templates. A policy can have one or more **policy-templates**, but a minimum of one **policy-template** is needed. The **PlacementRule** template specifies one cluster or set of groups for deploying the policy. As the name implies, the **PlacementBinding** template binds the policy to the **PlacementRules**.

**Note**

There is no restriction for the namespace in the policy creation. However, you cannot create a policy in cluster namespace.

For example, if the managed cluster is named `managed-cluster` in RHACM, the reserved namespace will be `managed-cluster`. If you create a policy in the `managed-cluster` namespace, it is deleted by RHACM.

Describing the 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:11
  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 used for policy creation.
- ⑪** `clusterSelector`: This parameter specifies a cluster or set of clusters for policy placement.

Stable and Community Policies

RHACM can use stable policies, supported by Red Hat, and also policies from the community. You can see most of the built-in stable policies in the RHACM governance dashboard. Examples of all policies are available in the `policy-collection` repository at <https://github.com/stolostron/policy-collection>.

Deploying Policies on a Multicluster Architecture with RHACM

The RHACM governance supports both dashboard and command-line methods to deploy policies.

Deploying policy from the governance dashboard

RHACM ships with some policy templates. You can access these templates from the governance dashboard. You can use the governance dashboard to create policies.

The policy creation page provides a graphical interface to define the policy name, namespace, policy template, cluster selector, and remediation selection space to create the policy. The governance dashboard also provides a YAML editor to customize further.

```

1  apiVersion: policy.open-cluster-management.io/v1
2  kind: Policy
3  metadata:
4    name: policy-grc
5    namespace:
6    annotations:
7      policy.open-cluster-management.io/standards:
8        policy.open-cluster-management.io/categories:
9          policy.open-cluster-management.io/controls:
10   spec:
11     remediationAction: inform
12     disabled: false
13     policyTemplates: []
14   .
15   apiVersion: policy.open-cluster-management.io/v1
16   kind: PlacementBinding
17   metadata:
18     name: binding-policy-grc
19     namespace:
20     placementRef:
21       name: placement-policy-grc
22       kind: PlacementRule
23       apiGroup: apps.open-cluster-management.io
24     subjects:
25       - name: policy-grc
26         kind: Policy
27         .
28         apiGroup: policy.open-cluster-management.io
29   .
30   apiVersion: apps.open-cluster-management.io/v1
31   kind: PlacementRule
32   metadata:
33     name: placement-policy-grc
34     namespace:
35     spec:

```

Deploying policy from the command line

You can also create and edit policies from the command line. First, log in to the hub cluster and deploy the policy from the YAML file.

```

[user@demo ]$ oc login -u user -p password \
https://api.ocp4.example.com:6443
Login successful.

...output omitted...

[user@demo ]$ oc create -f policy-example.yaml
policy.policy.open-cluster-management.io/policy-example created
placementbinding.policy.open-cluster-management.io/binding-policy-example created
placementrule.apps.open-cluster-management.io/placement-policy-example created

```

RHACM Policy Controllers Overview

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

The supported policy controllers are described in the following sections.

Kubernetes Configuration Policy Controller

This policy controller manages, configures, and monitors Kubernetes resources. The configuration policy controller monitors policy violations and remediates them by using the enforce feature. The configuration policy controller can also install, configure, and retrieve reports from other operators, such as a compliance operator.

Certificate Policy Controller

This policy controller monitors certificates for the selected namespaces. The certificate policy detects when a certificate is about to expire in the default namespace. You can customize the certificate policy by adding or changing additional parameters, as described in the following table.

Parameter	Description
minimumDuration	The minimum duration for the certificate expiration. For example, if the parameter value is set to 300h, and there are one or more certificates with an expiration age of less than 300 hours, then policy displays a Not-compliant status.
minimumCADuration	The minimum duration for the signing certificate expiration.
maximumDuration	The maximum duration for the certificate expiration. For example, if the parameter value is set to 300h, and there are one or more certificates with an expiration age of more than 300 hours, then the policy displays a Not-compliant status.
maximumCADuration	The maximum duration for the signing certificate expiration.
allowedSANPattern	You can specify an expression with this parameter. That expression must be present in every SAN entry in the certificates. It compares patterns against DNS names.
disallowedSANPattern	You can specify an expression with this parameter. That expression must not be present in any SAN entry in the certificates. It compares patterns against DNS names.

The certificate policy also has a namespace selector to exclude and include namespaces. The sample certificate policy template shows that the included namespace is `default`, and the excluded namespace is `kube-*`. This policy template checks the certificate expiration age against the `minimumDuration` parameter value.

```
policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: CertificatePolicy
      metadata:
        name: policy-certificate-example
      spec:
        namespaceSelector:
          include:
            - default
          exclude:
            - kube-*
```

```
remediationAction: inform
severity: low
minimumDuration: 300h
```

IAM Policy Controller

This policy controller sends notifications about IAM policy violations. The IAM policy checks for the number of users with cluster role bindings for the `cluster-admin` role. IAM policy also has a namespace selector to further customize IAM policy to exclude or include namespaces.

```
policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: IamPolicy
      metadata:
        name: policy-example-limitclusteradmin
      spec:
        severity: medium
        namespaceSelector:
          include:
          - '*'
          exclude:
          - kube-*
          - openshift-*
        remediationAction: inform
        maxClusterRoleBindingUsers: 5
```

The sample IAM policy template shows that the `kube-*` and `openshift-*` namespaces are excluded, and the maximum number of `ClusterRoleBinding` users is set to five.

Integrating Third-party Policy Controllers

RHACM governance supports third-party policy controllers such as gatekeeper policy controllers. You can install a gatekeeper operator with a Kubernetes configuration policy controller, and then use a gatekeeper policy controller.



Note

A sample of the gatekeeper policy is available at <https://github.com/stolostron/policy-collection/blob/main/community/CM-Configuration-Management/policy-gatekeeper-sample.yaml>

Custom Policy Controller

You can create a custom policy controller with the help of a `governance-policy-framework` repository. The `governance-policy-framework` is available at <https://github.com/stolostron/governance-policy-framework>.



References

Governance Policy Framework

<https://github.com/stolostron/governance-policy-framework>

For more information about the policy YAML structure, refer to the *The Policy YAML Structure* section in 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/governance/index#policy-yaml-structure

► Guided Exercise

Deploying and Managing Policies with RHACM

- 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

- 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
```

- 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 the `policy-governance` project.

```
[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.

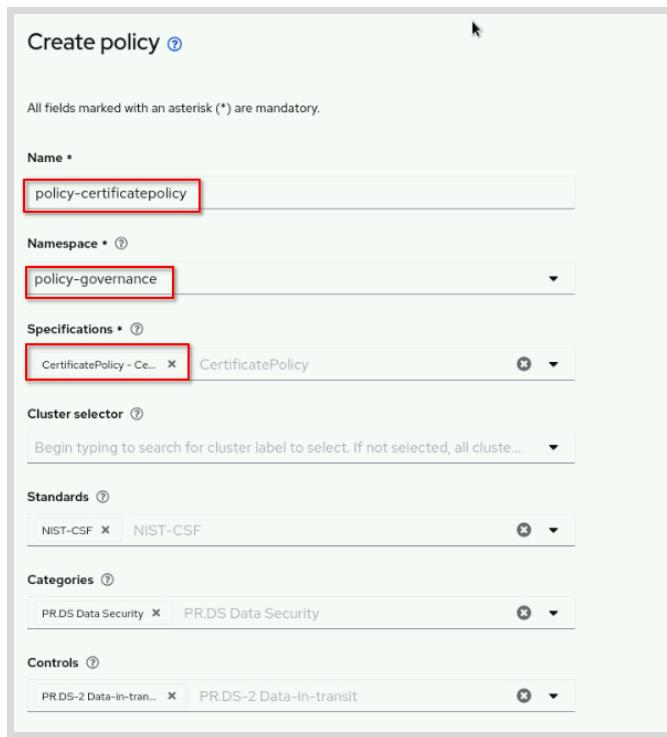
- 3. Create the certificate policy with the following parameters for `openshift-console` and `openshift-ingress` namespace:

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

- 3.1. Click **Governance** on the left pane to navigate to the governance dashboard. Then, click **Create policy**. The **Create policy** page displays.

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

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



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`.

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

The screenshot shows the 'Clusters' tab of the policy-certificatepolicy interface. It lists two clusters: 'managed-cluster' and 'local-cluster'. The 'managed-cluster' row is highlighted with a red box. The 'Compliance' column shows 'Not compliant' for the managed-cluster and 'Compliant' for the local-cluster. The 'Message' column provides details: 'NonCompliant; 1 certificates expire in less than 300h0m0s: openshift-ingress:wildcard-tls' for the managed-cluster, and 'Compliant' for the local-cluster. The 'Last Report' column shows '4 minutes ago' for both. A 'View details' link is present in the message for the managed-cluster.

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



Note

The **Not compliant** policy status indicates that one certificate has already expired or has less than 300 hours 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 `~/D0480/labs/policy-governance/` directory.

```
[student@workstation ~]$ cd ~/D0480/labs/policy-governance/
```

- 4.2. Log in to the **managed-cluster** as the **admin** user. The API server address 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. Change 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 screenshot shows the 'policy-certificatepolicy' details page in the Red Hat Governance interface. The policy is named 'policy-certificatepolicy' and is located in the 'policy-governance' namespace. It is currently enabled and has an 'Inform' remediation level. There are no cluster violations. In the 'Placement' section, there is a 'Cluster selector' with 'matchExpressions: []' and 2 clusters listed. Both clusters are marked as 'Compliant: managed-cluster, local-cluster'. A red box highlights the 'Compliant' status for one of the clusters.

The policy status is now **Compliant** for both clusters.



Note

It takes around a minute before the router pods to restart and use the updated certificate. Also, it takes a few seconds to change the policy status from **Not compliant** to **Compliant**.

Finish

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

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

This concludes the section.

Deploying and Configuring the Compliance Operator for Multiple Clusters Using RHACM

Objectives

- Deploy the compliance operator policy and view compliance reports from multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.

Definition of Compliance

Compliance management is a continuous process to ensure that IT systems are compliant with the organization's policies and procedures. Through the entire lifecycle of IT infrastructure management, policy and standards continuously evolve to best suit the needs of the business. It is crucial to identify any systems that are non-compliant with the latest standards for any reason. Failure to recognize non-compliant systems can result in the following issues:

- Loss of the client's trust in the IT company, leading clients to leave the company
- Legal ramifications such as fines or being blocked from working in specific geographical locations
- Loss of IT systems certifications

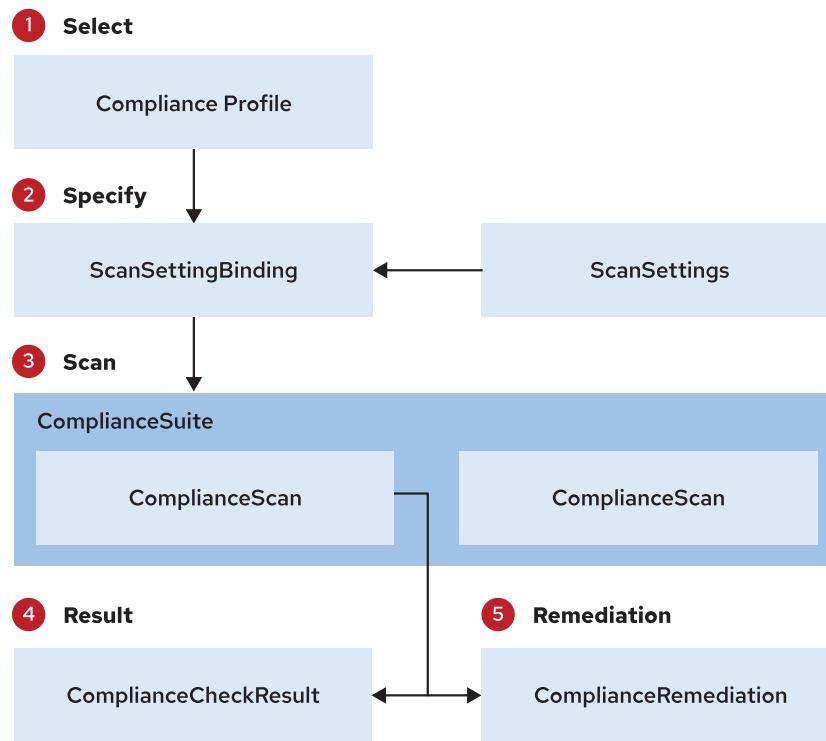
Compliance management must be agile in its approach to managing security, regulatory, and configuration changes. As security threats evolve technically and strategically, compliance management aims to secure or harden cloud environments such as those comprised of multiple Kubernetes clusters.

Describing the RHOCP Compliance Operator

The compliance operator is an OpenShift operator that employs OpenSCAP and enables an administrator to run compliance scans and provide remediation for the issues found. The compliance operator inspects Kubernetes objects and the nodes running the cluster to detect any gaps in compliance. The operator scans specific profiles and then produces a summary report to assist you in discovering non-compliant systems. The recommended remediation can be applied by hand, step by step, or automatically. The following is a summary of steps involved in the process:

1. Select a compliance profile for scanning.
2. Specify the scan settings.
3. Initiate the scan.
4. Generate the reports.
5. Apply remediation.

OpenSCAP performs a compliance assessment based on the policy rules defined by the content.



Describing OpenSCAP

OpenSCAP is a Security Content Automation Protocol (SCAP) compliant scanner that performs compliance and vulnerability scanning. The compliance operator uses OpenSCAP to scan the OpenShift clusters and nodes. OpenSCAP employs community-based compliance content that was developed as part of the [ComplianceAsCode/content](#) project. The content is distributed as a container image and decoupled from the operator for rapid content updates.

The compliance operator consumes the compliance content from the container image and exposes it in a custom resource.



Note

The Security Content Automation Protocol (SCAP) is a set of specific standards supporting automated vulnerability and patch monitoring, security measurement, and technical control compliance activities.

Policies

The Red Hat Advanced Cluster Management for Kubernetes policies use custom resource definitions. Policies contain one or more policy templates.

All policies require the following two custom resources definitions (CRDs):

PlacementRule

Defines the targeted clusters to receive the policy.

PlacementBinding

Binds the policy to the placement rule.

**Note**

You can create a policy in any namespace on the hub cluster except the cluster namespace. If you create a policy in the cluster namespace, it is deleted by RHACM.

Compliance Operator Policy

RHACM leverages the compliance operator policy to deploy the compliance operator across a fleet of clusters. The RHACM console enables administrators to create a policy that will deploy the compliance operator to all the managed clusters.

Creating a compliance operator policy results in the creation of a namespace, OperatorGroup and subscription.

Namespace

The compliance operator `openshift-compliance` namespace is created for the operator installation.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: comp-operator-ns
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: v1
        kind: Namespace
        metadata:
          name: openshift-compliance
```

OperatorGroup

The `compliance-operator` operator group is used to specify the target namespace.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: comp-operator-operator-group
spec:
  remediationAction: inform # will be overridden by remediationAction in parent policy
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: operators.coreos.com/v1
        kind: OperatorGroup
        metadata:
          name: compliance-operator
          namespace: openshift-compliance
```

```
spec:
  targetNamespaces:
    - openshift-compliance
```

Subscription

A subscription `comp-operator-subscription` to reference the name and channel. The subscription pulls the profile, as a container, that it supports.

```
apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: comp-operator-subscription
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave
      objectDefinition:
        apiVersion: operators.coreos.com/v1alpha1
        kind: Subscription
        metadata:
          name: compliance-operator
          namespace: openshift-compliance
        spec:
          installPlanApproval: Automatic
          name: compliance-operator
          source: redhat-operators
          sourceNamespace: openshift-marketplace
```

Deploying the Compliance Operator Policy

RHACM governance provides a compliance policy to deploy across the cluster fleet. To deploy the compliance policy, perform the following steps:

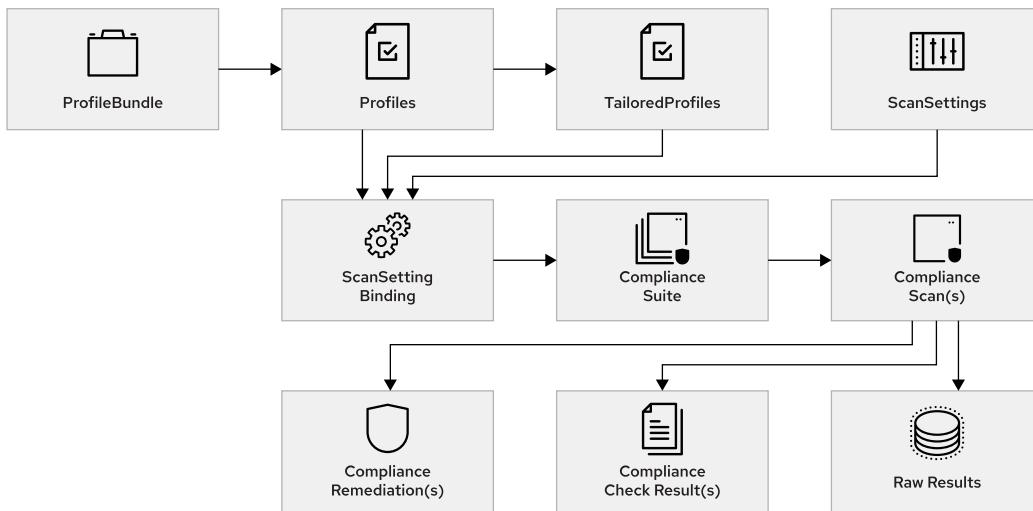
1. Log in to the RHACM console.
2. From the navigation menu in the left pane, select **Governance**.
3. Click **Create policy** to create the policy.
4. Select **ComplianceOperator** from the **Specifications** field and complete the YAML form.

Compliance Operator Policy CRDs

A **Custom Resource Definition** (CRD) defines a unique object **Kind** to create the **Custom Resource** (CR) objects, and the API Server manages the entire lifecycle of CRD.

The compliance operator uses custom resource definitions that represent security policy content, configuration or result, and its scanning processes.

The following diagram provides an overview of the custom resource definitions.



ScanSetting

The `ScanSetting` custom resource defines the scan setting such as:

- The nodes that should be scanned.
- The amount of storage allocated for the results collection.
- What time and frequency should be used for the scan.
- The retention policy.

The compliance operator creates a `ScanSetting` object named `default`.

ScanSettingBinding

The `ScanSettingBinding` object binds the `ScanSetting` object with the profiles.

The following example of a `ScanSettingBinding` object references two profiles: `ocp4-e8` and `rhcose4-e8`. The `settingsRef` value is a reference to the `ScanSetting` object.

The `remediationAction` is defined with `inform` so that RHACM will verify that the object exists but will not create the object if the object does not exist.

```

apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
  name: compliance-e8-scan
spec:
  remediationAction: inform
  severity: high
  object-templates:
    - complianceType: musthave # this template creates
      ScanSettingBinding:e8
        objectDefinition:
          apiVersion: compliance.openshift.io/v1alpha1
          kind: ScanSettingBinding
          metadata:
            name: e8
            namespace: openshift-compliance
          profiles:
            - apiGroup: compliance.openshift.io/v1alpha1
              kind: Profile
  
```

```

name: ocp4-e8
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: rhcos4-e8
  settingsRef:
    apiGroup: compliance.openshift.io/v1alpha1
    kind: ScanSetting
    name: default

```

The ScanSetting and ScanSettingBinding objects generate the following ComplianceSuite object.

ComplianceScan

The ComplianceScan represents a single scan and is represented by the ComplianceSuite object.

ComplianceSuite

The ComplianceSuite object is a bundle of ComplianceScan objects and maintains a record of each scan. Although the ComplianceSuite objects are automatically created when the ScanSetting and ScanSettingBinding objects are created, you can create them directly in specific use cases.

Compliance Profiles

Compliance profiles define and package a list of compliance benchmarks, scans, and remediation steps. The compliance profiles included in the compliance operator can be viewed with the `oc get profile.compliance` command.

```
[user@host ~]$ oc get profile.compliance -n openshift-compliance \
-l compliance.openshift.io/profile-bundle=rhcos4
NAME          AGE
rhcos4-e8     127m
rhcos4-moderate 127m
rhcos4-nerc-cip 127m
```

By default, when the compliance operator is deployed, it creates two `ProfileBundle` objects, one for OCP and one for Red Hat CoreOS (RHCOS). The OCP and RHCOS objects are created for the cluster level scan and node level scan respectively. You can find the profiles from the `ProfileBundle` that contain a reference to the container image. The operator creates profile objects that represent or define the compliance benchmarks. Profiles are a set of rules that are used to confirm or validate that a system is in compliance. Red Hat provides ready to use profiles. To view the `ProfileBundle` objects, use the `oc get profilebundle.compliance` command:

```
[user@host ~]$ oc get profilebundle.compliance -n openshift-compliance
NAME      ...      CONTENTFILE      STATUS
ocp4      ...      ssg-ocp4-ds.xml  VALID
rhcos4   ...      ssg-rhcos4-ds.xml VALID
```

To view the objects in each bundle, use the `oc get profile.compliance` command:

```
[user@host ~]$ oc get profile.compliance -n openshift-compliance \
-l compliance.openshift.io/profile-bundle=rhcos4
NAME          AGE
rhcos4-e8    127m
rhcos4-moderate 127m
rhcos4-nerc-cip 127m
```

To view the profile rules, use the `oc get profile.compliance` command as well:

```
[user@host ~]$ oc get profile.compliance -n openshift-compliance rhcos4-e8 \
-o yaml
apiVersion: compliance.openshift.io/v1alpha1
description: 'This profile contains configuration checks for Red Hat Enterprise
Linux
CoreOS that align to the Australian Cyber Security Centre (ACSC) Essential
Eight.

id: xccdf_org.ssgproject.content_profile_e8
kind: Profile
...output omitted...
rules:
- rhcos4-accounts-no-uid-except-zero
- rhcos4-audit-rules-dac-modification-chmod
- rhcos4-audit-rules-dac-modification-chown
- rhcos4-audit-rules-execution-chcon
- rhcos4-audit-rules-execution-restorecon
- rhcos4-audit-rules-execution-semanage
...output omitted...
```

The compliance operator enables auditing by exposing the Rule object as an API object.



Note

Starting in RHACM 2.4 you need to be the `Subscription-Admin` user before you deploy policies with the script. One option is to execute: `community/CM-Configuration-Management/policy-configure-subscription-admin-hub` from the command line and set it to enforce.

Essential 8 (E8) Policy Scan

This policy assumes that the compliance operator is installed and running on the managed clusters. It deploys a scan that will check the compute and worker nodes, and verifies for compliance with the Essential 8 (E8) security profile.

```
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: inform
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy ②
        metadata:
          name: compliance-e8-scan ③
      spec:
        remediationAction: inform
        severity: high
        object-templates:
          - complianceType: musthave # this template creates
ScanSettingBinding:e8
          objectDefinition:
            apiVersion: compliance.openshift.io/v1alpha1
            kind: ScanSettingBinding
            metadata:
              name: e8
              namespace: openshift-compliance
            profiles: ④
              - apiGroup: compliance.openshift.io/v1alpha1
                kind: Profile
                name: ocp4-e8
              - apiGroup: compliance.openshift.io/v1alpha1
                kind: Profile
                name: rhcos4-e8
            settingsRef:
              apiGroup: compliance.openshift.io/v1alpha1
              kind: ScanSetting
              name: default

```

- ① The Policy object defines the type of policy.
- ② The ConfigurationPolicy object checks the existence of the ScanSettingBinding.
- ③ The remediationAction parameter is defined as inform so if the ScanSettingBinding does not exist, it will be reported but not automatically enforced.
- ④ The profiles spec defines the profiles,ocp4-e8 and rhcos4-e8, that are deployed to comply with the E8 benchmarks.

```

  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: compliance-suite-e8 ①
    spec:
      remediationAction: inform ②
      severity: high
      object-templates:
        - complianceType: musthave # this template checks if scan has
completed by checking the status field
        objectDefinition:

```

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: e8
  namespace: openshift-compliance
status:
  phase: DONE ③

```

- ① The `ComplianceSuite` object tracks scans and outputs reports of the scanning process.
- ② The `remediationAction` parameter is defined as `inform` so if the `ScanSettingBinding` does not exist, it will be reported but not automatically enforced.
- ③ The `ComplianceSuite` object must be created and reach the `DONE` phase.

```

- objectDefinition:
  apiVersion: policy.open-cluster-management.io/v1
  kind: ConfigurationPolicy
  metadata:
    name: compliance-suite-e8-results
  spec:
    remediationAction: inform
    severity: high
    object-templates:
      - complianceType: mustnothave # this template reports the results for
        scan suite: e8 by looking at ComplianceCheckResult CRs
        objectDefinition:
          apiVersion: compliance.openshift.io/v1alpha1
          kind: ComplianceCheckResult ①
          metadata:
            namespace: openshift-compliance
            labels:
              compliance.openshift.io/check-status: FAIL ②
              compliance.openshift.io/suite: e8 ③

```

- ① The compliance operator scan results are defined as `ComplianceCheckResults` objects.
- ② Verifies that no rules have failed.
- ③ The applied rules originate from an E8 scan.

```

apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding ①
metadata:
  name: binding-policy-e8-scan
placementRef:
  name: placement-policy-e8-scan
  kind: PlacementRule
  apiGroup: apps.open-cluster-management.io
subjects:
- name: policy-e8-scan
  kind: Policy
  apiGroup: policy.open-cluster-management.io
---

```

```

apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule ②
metadata:
  name: placement-policy-e8-scan
spec:
  clusterConditions:
    - status: "True"
      type: ManagedClusterConditionAvailable
  clusterSelector: ③
    matchExpressions:
      - {key: vendor, operator: In, values: ["OpenShift"]}

```

- ①** The PlacementBinding object binds the rules to the policy.
- ②** The PlacementRule object defines the schedule conditions and the target cluster for the policy.
- ③** The clusterSelector field specifies a cluster or set of clusters for policy placement.

Deploying the E8 Policy Scan from the Console

RHACM enables administrators to create and deploy the E8 policy across a fleet of clusters. To deploy the compliance policy, perform the following steps:

1. Clone the policy collection repository. The policy-collection repository has policy examples for Open Cluster Management.
2. Edit the `policy-compliance-operator-e8-scan.yaml` YAML file.
3. Deploy the E8-scan policy using the following command:

```
[user@host]$ 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
```

View the Details of an E8 Policy Scan from the Console

1. On the left pane, click **Governance**. The dashboard displays the `policy-e8-scan` policy.
2. Click `policy-e8-scan` and then select the **Clusters** tab to check the status.
3. Click **View details** and review the details.



References

What is compliance management?

<https://www.redhat.com/en/topics/management/what-is-compliance-management>

ComplianceAsCode/content project

<https://github.com/ComplianceAsCode/content>

OpenScap

<https://github.com/OpenSCAP/openscap>

For more information about the compliance operator, refer to the *Understanding the Compliance Operator* section in the *Red Hat OpenShift documentation* at

https://docs.openshift.com/container-platform/4.6/security/compliance_operator/compliance-operator-understanding.html

For more information about Compliance operator policy, refer to the *Compliance operator policy* chapter in the *Red Hat Advanced Cluster Management for Kubernetes Governance Guide* at

https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/governance/index#compliance-operator-policy

For more information about E8 scan policy, refer to the *E8 scan policy* section in the *Red Hat Advanced Cluster Management for Kubernetes Governance Guide* at

https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/governance/governance#e8-scan-policy

► Guided Exercise

Deploying and Configuring the Compliance Operator for Multiple Clusters Using RHACM

- 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

- 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
```

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

```
[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. In the left pane, click **Governance** 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** yet.

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

```

55     severity: high
56     object-templates:
57       - complianceType: musthave
58         objectDefinition:
59           apiVersion: operators.coreos.com/v1alpha1
60           kind: Subscription
61           metadata:
62             name: compliance-operator
63             namespace: openshift-compliance
64             spec:
65               installPlanApproval: Automatic
66               name: compliance-operator
67               source: redhat-operators
68               sourceNamespace: openshift-marketplace
69 ...
70           apiVersion: policy.open-cluster-management.io/v1
71           kind: PlacementBinding
72           metadata:
73             name: binding-policy-complianceoperator
74             namespace: policy-compliance
75             placementRef:
76               name: placement-policy-complianceoperator
77               kind: PlacementRule
78               apiGroup: apps.open-cluster-management.io
79             subjects:
80               - name: policy-complianceoperator
81                 kind: Policy
82                 apiGroup: policy.open-cluster-management.io
83 ...
84           apiVersion: apps.open-cluster-management.io/v1
85           kind: PlacementRule
86           metadata:
87             name: placement-policy-complianceoperator
88             namespace: policy-compliance
89             spec:
90               clusterConditions:
91                 - status: 'True'
92                 type: ManagedClusterConditionAvailable
93               clusterSelector:
94                 matchExpressions:
95                   - key: location
96                     operator: In

```

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

```

object-templates:
  - complianceType: musthave
    objectDefinition:
      apiVersion: operators.coreos.com/v1alpha1
      kind: Subscription
      metadata:
        name: compliance-operator
        namespace: openshift-compliance
      spec:
        installPlanApproval: Automatic
        name: compliance-operator
        source: do480-catalog
        sourceNamespace: openshift-marketplace

```

Click **Create**.



Note

The source change is required only for the DO480 lab environment because the lab environment uses an offline operator catalog named **do480-catalog**.

- 3.4. 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**.

- 3.5. On the left pane, click **Governance**. The dashboard shows the **policy-complianceoperator** policy. Click the vertical ellipsis (⋮) menu 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 the 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 at the upper-right corner of the RHACM web console.

- 4.2. Type the following parameters in the search bar to search for 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

The APAC location has one cluster, named managed-cluster.

- 5. Clone the do480-policy-collection repository. The do480-policy-collection repository has policy examples for Open Cluster Management.

- 5.1. Open a terminal on the workstation machine. Run the following command to clone the do480-policy-collection repository:

```
[student@workstation ~]$ git clone \
https://github.com/RedHatTraining/do480-policy-collection.git
Cloning into 'do480-policy-collection'...
...output omitted...
```

- 5.2. Change to the directory where you cloned the Git repository.

```
[student@workstation ~]$ cd do480-policy-collection
```

► 6. Deploy the Essential 8 (E8) scan policy to 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/stolostron/policy-collection/blob/main/README.md>.

```
[student@workstation do480-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. Change to the home directory.

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

► 7. Check the compliance operator E8 scan results.

- 7.1. In the left pane, click **Governance**. The dashboard shows the `policy-e8-scan` policy. Click `policy-e8-scan` and then click `Clusters` 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` shows `Not compliant`.

Chapter 3 | Deploying and Managing Policies for Multiple Clusters with RHACM

Cluster	Compliance	Template	Message	Last Report	History
managed-cluster	Not compliant	compliance-suite-e8-results	NonCompliant; violation - compliancecheckresults found: [ocp4-e8-api-server-encryption-provider-cipher, ocp4-e8-ocp-a-d-bpf-dnslabel, rhcos4-e8-worker-systcl-kernel-yama-prtrace-scope, rhcos4-e8-worker-systcl-net-core-bpf-jit-harden] in namespace openshift-compliance	5 minutes ago	View history
managed-cluster	Compliant	compliance-suite-e8	Compliant; notification - compliancesuites [e8] in namespace openshift-compliance found as specified, therefore this Object template is compliant	4 minutes ago	View history
managed-cluster	Compliant	compliance-e8-scan	Compliant; notification - scansettingbindings [e8] in namespace openshift-compliance found as specified, therefore this Object template is compliant	6 minutes ago	View history



Note

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

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

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

Template details

Name	compliance-suite-e8-results
Cluster	managed-cluster
Kind	ConfigurationPolicy
API groups	policy.open-cluster-management.io/v1
Compliant	NonCompliant
Details	<pre>[{"Compliant": "NonCompliant", "Validity": {}}, {"conditions": [{"lastTransitionTime": "2021-12-31T10:10:13Z", "message": "compliancecheckresults found: [ocp4-e8-api-server-encryption-provider-cipher, ocp4-e8-ocp-allowed-registries, ocp4-e8-ocp-allowed-registries-for-import, ocp4-e8-ocp-ldp-no-htpasswd, rhcos4-e8-master-audit-rules-dac-modification-chmod, rhcos4-e8-master-audit-rules-dac-modification-chown, rhcos4-e8-master-audit-rules-execution-chroot, rhcos4-e8-master-audit-rules-execution-restarton, rhcos4-e8-master-audit-rules-execution-semage, rhcos4-e8-master-audit-rules-execution-settles, rhcos4-e8-master-audit-rules-execution-setsebool, rhcos4-e8-master-audit-rules-execution-seunshare, rhcos4-e8-master-audit-rules-kernel-module-loading-delete, rhcos4-e8-master-audit-rules-kernel-module-loading-finit, rhcos4-e8-master-audit-rules-kernel-module-loading-init, rhcos4-e8-master-audit-rules-kernel-module-loading-init}], "status": "NonCompliant"}]</pre>

Template yaml

```

1  apiVersion: policy.open-cluster-management.io/v1
2  kind: ConfigurationPolicy
3  metadata:
4    creationTimestamp: '2021-12-31T11:07:15Z'
5    generation: 1
6    labels:
7      cluster-name: managed-cluster
8      cluster-namespace: managed-cluster
9    policy.open-cluster-management.io/cluster-name: managed-cluster
10   policy.open-cluster-management.io/cluster-namespace: managed-cluster
11   managedFields:
12     - apiVersion: policy.open-cluster-management.io/v1
13       fieldType: FieldsV1
14       fieldsV1:
15         f:metadata:
16           f:labels:
17             .: {}
18             f:cluster-name: {}
19             f:cluster-namespace: {}
20             f:policy.open-cluster-management.io/cluster-name: {}
21             f:policy.open-cluster-management.io/cluster-namespace: {}
22         f:ownerReferences:
23           .: []
24           k: "uid"
25           l: "bb8de32-821a-4a6b-bf07-bb6de52b0975b"
26           r: {}
27           t: {}
28         f:controller:
29           f:kind: {}
30           f:name: {}
31           f:uid: {}
32       f:spec:

```

- To list the compliance check results, click the search icon on the upper-right corner of the RHACM web console and type `kind:configurationpolicy`, and then click search.
- Click **compliance-suite-e8-results** and check the status. It shows a list of NonCompliant objects.

Search **compliance-suite-e8-results**

YAML

Cluster: **managed-cluster** Namespace: **managed-cluster** Read only mode **Edit**

```

76   complianceDetails:
77     - Compliant: NonCompliant
78     Validity: {}
79     conditions:
80       - lastTransitionTime: '2021-11-11T06:59:50Z'
81       message: '>
82         compliancecheckresults found:
83         [ocp4-e8-api-server-encryption-provider-cipher,
84         ocpd-e8-ocp-allowed-registries,
85         ocpd-e8-ocp-allowed-registries-for-import,
86         ocpd-e8-ocp-ldp-no-htpasswd,
87         rhcos4-e8-master-audit-rules-dac-modification-chmod,
88         rhcos4-e8-master-audit-rules-dac-modification-chown,
89         rhcos4-e8-master-audit-rules-execution-chcon,
90         rhcos4-e8-master-audit-rules-execution-restorecon,
91         rhcos4-e8-master-audit-rules-execution-semange,
92         rhcos4-e8-master-audit-rules-execution-setfiles,
93         rhcos4-e8-master-audit-rules-execution-setsebool,
94         rhcos4-e8-master-audit-rules-kernel-module-loading-share,
95         rhcos4-e8-master-audit-rules-kernel-module-loading-delete,
96         rhcos4-e8-master-audit-rules-kernel-module-loading-init,
97         rhcos4-e8-master-audit-rules-login-events,
98         rhcos4-e8-master-audit-rules-login-events-faillock,
99         rhcos4-e8-master-audit-rules-login-events-lastlog,
100        rhcos4-e8-master-audit-rules-login-events-tallylog,
101        rhcos4-e8-master-audit-rules-networkconfig-modification,
102        rhcos4-e8-master-audit-rules-sysadmin-actions,
103        rhcos4-e8-master-audit-rules-time-adjtime,
104        rhcos4-e8-master-audit-rules-clock-settime,
105        rhcos4-e8-master-audit-rules-time-settimeofday,
106        rhcos4-e8-master-audit-rules-time-stime,
107        rhcos4-e8-master-audit-rules-time-watch-localtime,
108        rhcos4-e8-master-audit-rules-usergroup-modification,
109        rhcos4-e8-master-audit-name-format,
110        rhcos4-e8-master-configure-crypto-policy,
111        rhcos4-e8-master-no-empty-passwords,
112        rhcos4-e8-master-sshd-disable-gssapi-auth,
113        rhcos4-e8-master-sshd-disable-user-known-hosts,
114        rhcos4-e8-master-sysctl-kernel-dmesg-restrict,
115        rhcos4-e8-master-sysctl-kernel-randomize-va-space,
116      ]

```

Finish

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

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

This concludes the section.

Integration of Other Policy Engines with RHACM

Objectives

- Deploy the gatekeeper policy and gatekeeper constraints to multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.

Open Policy Agent Versus Open Policy Agent Gatekeeper

Implementing policies is a challenging task. It usually requires a policy management software to implement the policies, a policy language for writing the rules, and a policy engine to evaluate the policies.

Open Policy Agent (OPA)

OPA is an open source policy engine that unifies policy enforcement across the stack. OPA uses a high-level declarative language named Rego to write the policies, taking the policy decision-making load away from the application. OPA is a general-purpose policy engine, which can be used to enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more.

OPA gatekeeper

The OPA gatekeeper is a use case of OPA for Kubernetes environments. It is built to work as a Kubernetes admission controller webhook. In Kubernetes, when an API request is authenticated and authorized, the Kubernetes admission controllers process and validate the API request. If the request is not compliant with the defined policies, an admission controller can block it.

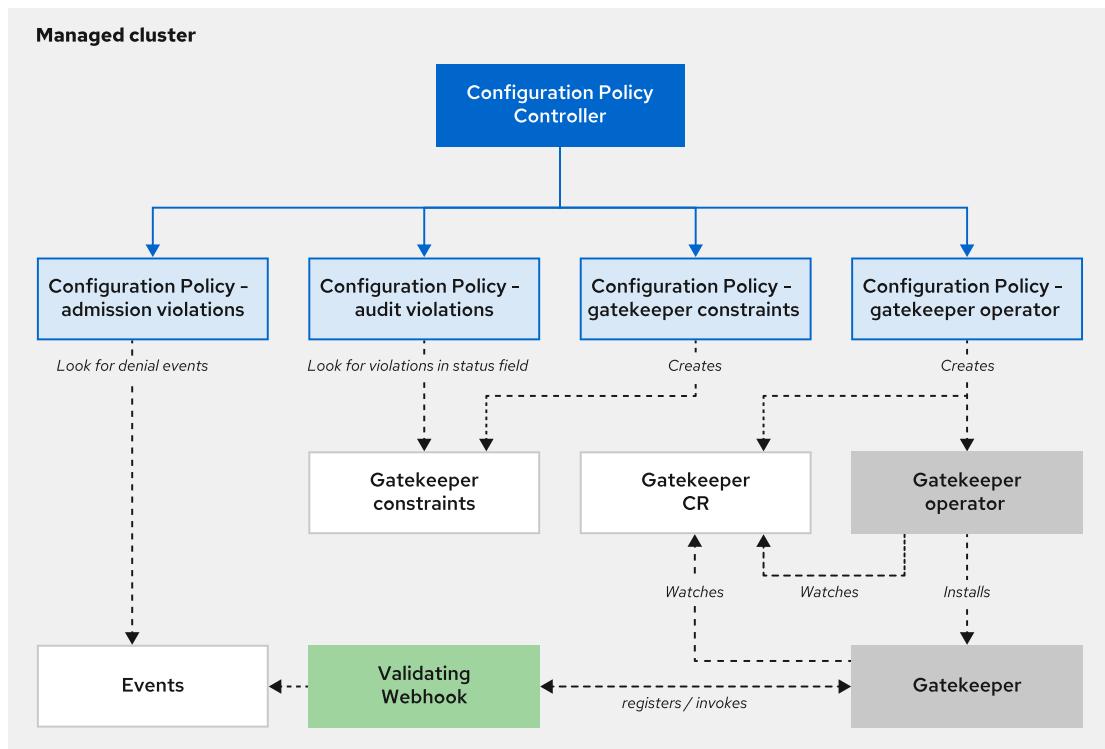
OPA Gatekeeper Integration with RHACM

Red Hat Advanced Cluster Management for Kubernetes (RHACM) provides a built-in policy to deploy the OPA gatekeeper operator. This way, RHACM can use OPA gatekeeper as a policy controller and you can use constraints and constraint templates to define admission and audit templates for multiple clusters.

OPA gatekeeper integration with RHACM is a two-step process.

The first step is to deploy the gatekeeper operator and create the gatekeeper custom resource. After that, you can create constraint, audit, and admission templates. The Kubernetes configuration policy controller creates all the policy templates.

The following diagram provides an overview of how the gatekeeper policy engine integrates with the RHACM configuration policy controller.



Installing the Gatekeeper Operator Using RHACM Governance

RHACM governance provides a policy to install the gatekeeper operator to a single cluster or set of clusters based on a `PlacementRule`.

The following YAML code can be used as a policy to install the gatekeeper operator in the `openshift-operators` namespace. It installs the gatekeeper operator in the targeted cluster or set of clusters based on a `PlacementRule` and creates a `Gatekeeper` custom resource.

```

object-templates:
  - complianceType: musthave
    objectDefinition:
      apiVersion: operators.coreos.com/v1alpha1
      kind: Subscription
      metadata:
        name: gatekeeper-operator-product
        namespace: openshift-operators
      spec:
        channel: stable
        installPlanApproval: Automatic
        name: gatekeeper-operator-product
        source: redhat-operators
        sourceNamespace: openshift-marketplace
    object-templates:
      - complianceType: musthave
        objectDefinition:
          apiVersion: operator.gatekeeper.sh/v1alpha1
          kind: Gatekeeper
          metadata:
            name: gatekeeper

```

```

spec:
  audit:
    logLevel: INFO
    replicas: 1
    validatingWebhook: Enabled
    mutatingWebhook: Disabled
    webhook:
      emitAdmissionEvents: Enabled
      logLevel: INFO
      replicas: 2

```

Deploy Constraints Using Gatekeeper Policies

You can deploy gatekeeper policies when the gatekeeper operator is deployed to the targeted clusters. The components of gatekeeper policies are:

ConstraintTemplate and Constraint

A `ConstraintTemplate` defines the schema for the `Constraint` object. It also contains the Rego code to evaluate the policy violation. A `Constraint` provides the value of the parameters defined in `ConstraintTemplate`.

The following YAML code creates a `ConfigurationPolicy` object named `policy-gatekeeper-k8srequiredlabels` and checks the existence of a `ConstraintTemplate` named `k8srequiredlabels` and a `Constraint` named `ns-must-have-gk` in the targeted cluster. If these resources are available, the policy template shows the `Compliant` status. Otherwise, the status is `Not-compliant`.

However, if the policy is in enforce mode, the policy controller creates these resources in the targeted clusters.

You can define another parameter in the `Constraint` named `enforcementAction`. The value of an `enforcementAction` is either `dryrun` or `deny`. In the following example, this constraint verifies if the `testfail` namespace has the `gatekeeper` label.

```

- objectDefinition:
    apiVersion: policy.open-cluster-management.io/v1
    kind: ConfigurationPolicy
    metadata:
      name: policy-gatekeeper-k8srequiredlabels
    spec:
      remediationAction: enforce # will be overridden by remediationAction in
      parent policy
      severity: low
      object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: templates.gatekeeper.sh/v1beta1
            kind: ConstraintTemplate
            metadata:
              name: k8srequiredlabels
            spec:
              crd:
                spec:
                  names:
                    kind: K8sRequiredLabels

```

```

validation:
  # Schema for the parameters field
openAPIV3Schema:
  properties:
    labels:
      type: array
      items: string
targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      package k8srequiredlabels
      violation[{"msg": msg, "details": {"missing_labels": missing}}] {
        provided := {label |
input.review.object.metadata.labels[label]}
        required := {label | label := input.parameters.labels[_]}
        missing := required - provided
        count(missing) > 0
        msg := sprintf("you must provide labels: %v", [missing])
      }
    - complianceType: musthave
      objectDefinition:
        apiVersion: constraints.gatekeeper.sh/v1beta1
        kind: K8sRequiredLabels
        metadata:
          name: ns-must-have-gk
        spec:
          enforcementAction: dryrun
          match:
            kinds:
              - apiGroups: []
                kinds: ["Namespace"]
            namespaces:
              - testfail
        parameters:
          labels: ["gatekeeper"]

```

```

[user@demo ]$ oc get configurationpolicy -n local-cluster
NAME                      AGE
policy-gatekeeper-k8srequiredlabels   15h
...output omitted...
[user@demo ]$ oc get constrainttemplate
NAME                      AGE
k8srequiredlabels          15h
[user@demo ]$ oc get constraint
NAME                      AGE
k8srequiredlabels          15h

```



Note

The gatekeeper policy YAML code is available at <https://github.com/stolostron/policy-collection/blob/main/community/CM-Configuration-Management/policy-gatekeeper-sample.yaml>

Audit template

An audit template compares the existing resources against deployed constraints, and if the resource fails to satisfy the constraints, then the template returns the Not-compliant status.

The following YAML code creates a ConfigurationPolicy object named policy-gatekeeper-audit. For example, this template returns a Not-compliant status if the testfail namespace is available in the targeted cluster, and it does not contain the gatekeeper label.

```
- objectDefinition:
    apiVersion: policy.open-cluster-management.io/v1
    kind: ConfigurationPolicy
    metadata:
        name: policy-gatekeeper-audit
    spec:
        remediationAction: inform # will be overridden by remediationAction in
parent policy
        severity: low
        object-templates:
            - complianceType: musthave
                objectDefinition:
                    apiVersion: constraints.gatekeeper.sh/v1beta1
                    kind: K8sRequiredLabels
                    metadata:
                        name: ns-must-have-gk
                    status:
                        totalViolations: 0
```

Admission template

An admission template defines parameters such as event_type, constraint_name, constraint_kind, and constraint_action and watches for the specific events in the targeted clusters. This template has the complianceType set to mustnothave. If any event matches the parameters specified in the YAML code, this template returns a Not-compliant status.

The following YAML code creates a ConfigurationPolicy object named policy-gatekeeper-admission. In this example, if you create a testfail namespace without the gatekeeper label, the admission webhook will deny the request and trigger a violation event. In this case, the template would return a Not-compliant status.

```
- objectDefinition:
    apiVersion: policy.open-cluster-management.io/v1
    kind: ConfigurationPolicy
    metadata:
        name: policy-gatekeeper-admission
    spec:
        remediationAction: inform # will be overridden by remediationAction in
parent policy
        severity: low
        object-templates:
            - complianceType: mustnothave
                objectDefinition:
                    apiVersion: v1
```

```

kind: Event
metadata:
  namespace: openshift-gatekeeper-system # set it to the actual
  namespace where gatekeeper is running if different
  annotations:
    constraint_action: deny
    constraint_kind: K8sRequiredLabels
    constraint_name: ns-must-have-gk
    event_type: violation

```

You can create separate policy files for these templates or use a single policy file to deploy the constraint, audit, and admission templates.

```
[user@demo ]$ oc create -f policy-gatekeeper-sample.yaml
policy.policy.open-cluster-management.io/policy-gatekeeper created
placementbinding.policy.open-cluster-management.io/binding-policy-gatekeeper
created
placementrule.apps.open-cluster-management.io/placement-policy-gatekeeper created
```



References

Open Policy Agent (OPA)

<https://www.openpolicyagent.org/docs/latest/>

For more information about the policy YAML structure, refer to the *The Policy YAML Structure* section in 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/governance/index#policy-yaml-structure

For more information, refer to the *Managing Gatekeeper operator policies* section in the *Governance* 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/governance/index#managing-gatekeeper-operator-policies

► Guided Exercise

Integration of Other Policy Engines with RHACM

- Install the gatekeeper operator in all clusters and enforce containers running in the production cluster not to use the `latest` tag.

Outcomes

- Deploy a gatekeeper policy from the Red Hat Advanced Cluster Management for Kubernetes (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
```

- 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 the `policy-gatekeeper` project.

```
[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. In the left pane, click **Governance** 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** yet.

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

Create policy [?](#)
[Cancel](#) Create

All fields marked with an asterisk (*) are mandatory.

Name *

Namespace *

Specifications *

[GatekeeperOperator ...](#) GatekeeperOperator

Cluster selector [?](#)

Standards [?](#)

[NIST-CSF](#) X [NIST-CSF](#)

Categories [?](#)

[PRIP Information Protection Processes and Procedures](#) X [PRIP Information Protection Processes and Procedures](#)

Controls [?](#)

[PRIP-I Baseline Configuration](#) X [PRIP-I Baseline Configuration](#)

Remediation *

Inform - Reports the violation, which requires manual remediation.
 Enforce - Automatically runs remediation action that is defined in the source, if this feature is supported.

```

1  apiVersion: policy.open-cluster-management.io/v1
2  kind: Policy
3  metadata:
4    name: policy-gatekeeperoperator
5    namespace: policy-gatekeeper
6    annotations:
7      policy.open-cluster-management.io/standards: NI
8      policy.open-cluster-management.io/categories: P
9      policy.open-cluster-management.io/controls: PR
10 spec:
11   remediationAction: enforce
12   disabled: false
13   policy-templates:
14     - objectDefinition:
15       apiVersion: policy.open-cluster-management.
16       kind: ConfigurationPolicy
17       metadata:
18         name: gatekeeper-operator-product-sub
19       spec:
20         remediationAction: inform
21         severity: high
22         object-templates:
23           - complianceType: mustHave
24             objectDefinition:
25               apiVersion: operators.coreos.com/v1
26               kind: Subscription
27               metadata:
28                 name: gatekeeper-operator-product
29                 namespace: openshift-operators
30             spec:
31               channel: stable
32               installPlanApproval: Automatic
33               name: gatekeeper-operator-product
34               source: redhat-operators
35               sourceNamespace: openshift-marketplace
36             - objectDefinition:
37               apiVersion: policy.open-cluster-management.
38               kind: ConfigurationPolicy
39               metadata:
40                 name: gatekeeper
41               spec:

```

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

```
object-templates:
- complianceType: musthave
  objectDefinition:
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
      name: gatekeeper-operator-product
      namespace: openshift-operators
    spec:
      channel: stable
      installPlanApproval: Automatic
      name: gatekeeper-operator-product
      source: do480-catalog
      sourceNamespace: openshift-marketplace
```

- 3.4. Click **Create** to create the **policy-gatekeeperoperator** policy.



Note

The source change is required for the DO480 lab environment because the lab environment uses an offline operator catalog named **do480-catalog**.

- 3.5. On the **Governance** page, scroll down and click the **policy-gatekeeperoperator** policy name from the list of policies. Click **Clusters** to review the policy details.

Cluster	Compliance	Template	Message	Last Report	History
managed-cluster	Compliant	gatekeeper	Compliant; notification - gatekeepers [gatekeeper] found as specified, therefore this Object template is compliant	8 minutes ago	View history
managed-cluster	Compliant	gatekeeper-operator-product-sub	Compliant; notification - subscriptions [gatekeeper-operator-product] in namespace openshift-operators found as specified, therefore this Object template is compliant	8 minutes ago	View history
local-cluster	Compliant	gatekeeper	Compliant; notification - gatekeepers [gatekeeper] found as specified, therefore this Object template is compliant	7 minutes ago	View history
local-cluster	Compliant	gatekeeper-operator-product-sub	Compliant; notification - subscriptions [gatekeeper-operator-product] in namespace openshift-operators found as specified, therefore this Object template is compliant	8 minutes ago	View history

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 compliant** to **Compliant** 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 a terminal on the **workstation** machine and change to the `~/D0480/labs/policy-gatekeeper/` directory.

```
[student@workstation ~]$ cd ~/D0480/labs/policy-gatekeeper/
```

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

Field name	Value
excludedNamespaces	app-stage
key	environment
values	stage

```
[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...
      ...
      - 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 screenshot shows the 'Policy details' section of the governance interface. It includes fields for Name (policy-gatekeeper-config-exclude-namespaces), Namespace (policy-gatekeeper), Status (Enabled), Remediation (Enforce), Cluster violations (0/1), Categories (CM Configuration Management), Controls (CM-2 Baseline Configuration), Standards (NIST SP 800-53), Created (a minute ago), and Automation (Configure). The 'Placement' section shows a 'Cluster selector' with the expression 'matchExpressions:[{"key":"environment","operator":"in","values":["stage"]}]'. Below it, there are tabs for 'Clusters' and 'Compliance'. The 'Clusters' tab shows one cluster named 'local-cluster' with a green checkmark. The 'Compliance' tab shows a green checkmark and the text 'Compliant: local-cluster'.

The policy status is **Compliant** for the 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 force 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** 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: []
```

- 5.2. Deploy the gatekeeper policy to force 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.

Cluster selector	Clusters	Compliance
matchExpressions:[]	2	Compliant: local-cluster, managed-cluster

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



Note

It takes 2-3 minutes to for policy to check all object templates. The policy status changes from **Not compliant** to **Compliant** after 2-3 minutes.

- 6. Deploy the `stage-hello` application to the `stage` environment by using the `~/D0480/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 **Clusters** to review the policy details.

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 image (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.

```

You 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 **Clusters** to review the policy details.

Cluster	Compliance	Template	Message	Last Report	History
managed-cluster	Not compliant	policy-gatekeeper-admission-latest	NonCompliant; violation - events found: [prod-hello.16e0d08984595b65] in namespace openshift-gatekeeper-system	View details	a few seconds ago View history
managed-cluster	Compliant	policy-gatekeeper-audit-latest	Compliant; notification - containerimagelatest [containerimagelatest] found as specified, therefore this Object template is compliant	View details	20 minutes ago View history
managed-cluster	Compliant	policy-gatekeeper-containerimage-latest	Compliant; notification - constrainttemplates [containerimagelatest] found as specified, therefore this Object template is compliant; notification - containerimagelatest [containerimagelatest] found as specified, therefore this Object template is compliant	View details	21 minutes ago View history
local-cluster	Compliant	policy-gatekeeper-admission-latest	Compliant; notification - events in namespace openshift-gatekeeper-system missing as expected, therefore this Object template is compliant	View details	21 minutes ago View history
local-cluster	Compliant	policy-gatekeeper-audit-latest	Compliant; notification - containerimagelatest [containerimagelatest] found as specified, therefore this Object template is compliant	View details	20 minutes ago View history
local-cluster	Compliant	policy-gatekeeper-	Compliant; notification - constrainttemplates [containerimagelatest] found as specified, therefore this Object template is compliant; notification - containerimagelatest [containerimagelatest] found as specified, therefore this Object template is compliant	View details	21 minutes ago View history

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

- Click **View details** and review the **policy-gatekeeper-admission-latest** template details. On the details page, scroll down and click **View yaml** for event objects.

Name	Namespace	Kind	API groups	Compliant	Reason
prod-hello.16e0d17ae789c29a	openshift-gatekeeper-system	events	v1	Not compliant	Resource found but should not exist

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

Search
prod-hello.16c03c7851255041
YAML

Error querying for resource: prod-hello.16c03c7851255041

Admission webhook 'validation.gatekeeper.sh' denied request, Resource Namespace: app-prod, Constraint: containerimagelatest, Message: 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.

- ▶ 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 the `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. Change 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 step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

This concludes the section.

▶ Lab

Deploying and Managing Policies with RHACM

- Create a policy to ensure that the `test` namespace does not exist in the production environment. You also create an Identity and Access Management (IAM) policy to limit the number of cluster administrators to two for all clusters.

Outcomes

- Deploy a namespace policy from the Red Hat Advanced Cluster Management for Kubernetes (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
```

1. Log in to the Hub OpenShift cluster as the `admin` user with the `redhat` password and create the `policy-review` project.
The API server address is <https://api.ocp4.example.com:6443>.
2. Log in to the RHACM web console as the `admin` user with the `redhat` password and the `htpasswd_provider` option. Deploy a namespace policy in the `policy-review` namespace to enforce that the `test` namespace does not exist in the production environment.
3. Deploy an IAM policy in the `policy-review` namespace to enforce the number of cluster administrators limits to two 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 step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

This concludes the section.

► Solution

Deploying and Managing Policies with RHACM

- Create a policy to ensure that the `test` namespace does not exist in the production environment. You also create an Identity and Access Management (IAM) policy to limit the number of cluster administrators to two for all clusters.

Outcomes

- Deploy a namespace policy from the Red Hat Advanced Cluster Management for Kubernetes (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
```

1. Log in to the Hub OpenShift cluster as the `admin` user with the `redhat` password and create the `policy-review` project.

The API server address is `https://api.ocp4.example.com:6443`.

- 1.1. Open a terminal 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 the `policy-review` project.

```
[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 as the `admin` user with the `redhat` password and the `htpasswd_provider` option. Deploy a namespace policy in the `policy-review` namespace to enforce 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. In the left pane, click **Governance** 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** yet.

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

```

apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: policy-namespace
  namespace: policy-review
specifications:
  policy.open-cluster-management.io/standards: NIST-CSF
  policy.open-cluster-management.io/categories: PRIP Information Protection Processes and Controls
  policy.open-cluster-management.io/controls: PRIP-1 Baseline Configuration
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: musthave
            objectDefinition:
              kind: Namespace
              apiVersion: v1
              metadata:
                name: prod
...
apiVersion: policy.open-cluster-management.io/v1

```

- 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 **Clusters** to review the policy details.

Cluster	Compliance	Template	Message	Last Report	History
managed-cluster	Compliant	policy-namespace-prod-ns	Compliant; notification - namespaces [test] missing as expected, therefore this Object template is compliant View details	6 minutes ago	View history

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

3. Deploy an IAM policy in the **policy-review** namespace to enforce the number of cluster administrators limits to two for all clusters.
 - 3.1. In the left pane, click **Governance** 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** yet.

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

```

1 apiVersion: policy.open-cluster-management.io/v1
2 kind: Policy
3 metadata:
4   name: policy-iampolicy
5   namespace: policy-review
6   annotations:
7     policy.open-cluster-management.io/standards: NIST-CSF
8     policy.open-cluster-management.io/categories: PR.AC Identity M
9     policy.open-cluster-management.io/controls: PR.AC-4 Access Con
10    spec:
11      remediationAction: inform
12      disabled: false
13      policy-templates:
14        - objectDefinition:
15          apiVersion: policy.open-cluster-management.io/v1
16          kind: IamPolicy
17          metadata:
18            name: policy-iampolicy-limit-clusteradmin
19          spec:
20            severity: medium
21            namespaceSelector:
22              include:
23                - '*'
24              exclude:
25                - kube-*
26                - openshift-
27            remediationAction: inform
28            maxClusterRoleBindingUsers: 5
29 ...
30      apiVersion: policy.open-cluster-management.io/v1
31      kind: PlacementBinding
32      metadata:
33        name: binding-policy-iampolicy
34        namespace: policy-review
35        placementRef:

```

- 3.3. Use the YAML editor on the right of the **Create policy** page to make the following edit 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**.

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

The screenshot shows the 'Clusters' tab of the 'policy-iampolicy' policy in the RHACM interface. It lists two clusters: 'local-cluster' and 'managed-cluster'. The 'local-cluster' row has a red border around the 'Compliance' column, which contains the text 'Not compliant'. A tooltip message next to it says 'NonCompliant; The number of users with the cluster-admin role is at least 2 above the specified limit' with a 'View details' link. The 'managed-cluster' row has a green border around the 'Compliance' column, which contains the text 'Compliant'. A tooltip message next to it says 'Compliant; The number of users with the cluster-admin role is at least 0 above the specified limit' with a 'View details' link. Both rows have a 'Template' column showing 'policy-iampolicy-limit-clusteradmin' and a 'Message' column showing the compliance status. The top right corner of the interface shows 'Refresh every 10s' and 'Last update: 6:30:35 AM'.

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 step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

This concludes the section.

Summary

- The Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance provides a graphical user interface and a YAML editor to create, deploy, and customize policies.
- The RHACM governance uses policies to install the compliance operator and run compliance scans to analyze multiple clusters.
- The RHACM governance uses policies to install the Gatekeeper operator and use the configuration policy controller to deploy gatekeeper `ConstraintTemplate` and `Constraint`.

Chapter 4

Installing and Customizing the RHACM Observability Stack

Goal

Gain insight into the fleet of managed clusters by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability components.

Objectives

- Enable the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability stack and describe the architecture and the benefits of observability in a multicloud environment.
- Customize the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability stack.

Sections

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

Lab

- Installing and Customizing the RHACM Observability Stack

Installing the RHACM Observability Stack

Objectives

- Enable the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability stack and describe the architecture and the benefits of observability in a multicluster environment.

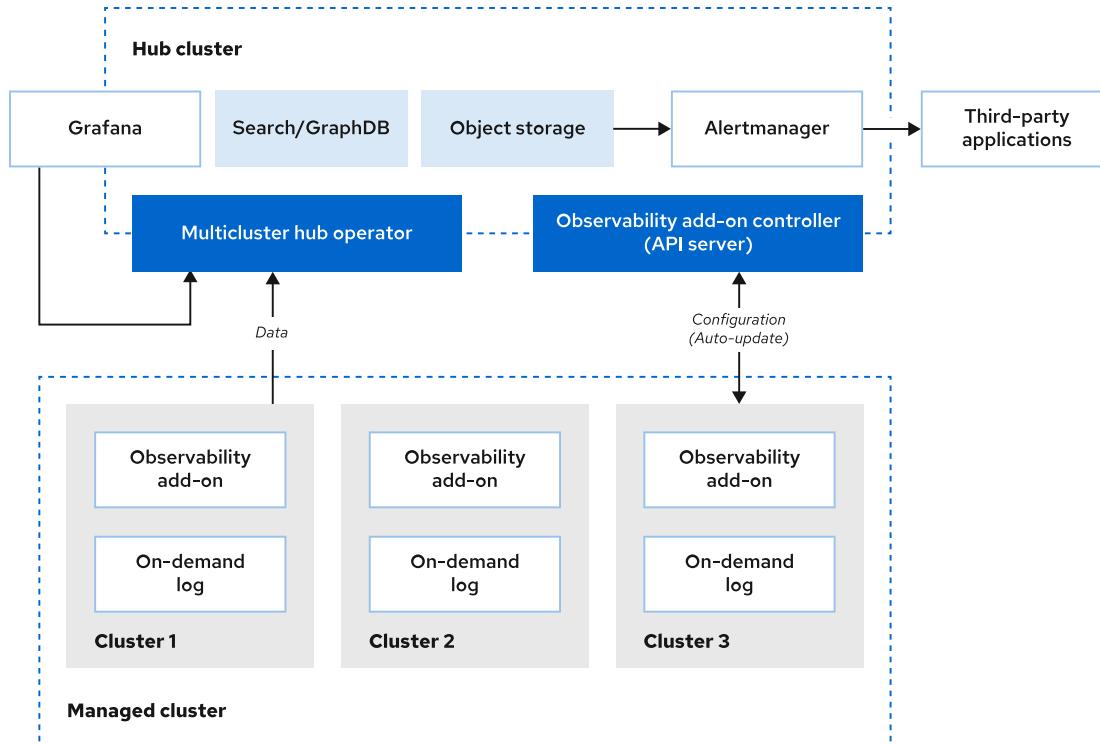
Multicluster Observability in RHACM

The observability service in RHACM provides a unified display of metrics across the fleet of clusters.

This service is based on the Thanos project from the Cloud Native Computing Foundation (CNCF). Thanos can gather data from multiple Prometheus endpoints running on different Red Hat OpenShift clusters. Thanos deploys a sidecar container to the Kubernetes Prometheus pods, which sends metrics and responds to queries, providing high availability to the observability environment. Furthermore, Thanos uses the storage format created by Prometheus to be very efficient at retaining and querying the data. Prometheus was created to inform about live data; Thanos adds long-term storage to multiple Prometheus instances.

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 postmortem analysis.

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



When you enable the observability service, the multicluster observability operator defines and instantiates a `MultiClusterObservability` custom resource. Then, the `MultiClusterObservability` custom resource deploys Grafana and Alertmanager instances in the hub cluster. If not specified otherwise, it also deploys the observability add-on 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 add-on activated.

Grafana ships with many predefined dashboards, and you can also create your own custom dashboards. As explained elsewhere in this chapter, you can also 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 complement the Red Hat OpenShift cluster-level metrics.

The Red Hat OpenShift Container Platform (RHOCP) monitoring stack provides preconfigured 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, which it receives from the fleet of clusters. You can use the Grafana instance of the RHACM observability service to monitor cluster-level metrics across the fleet of clusters.

You can configure the RHACM observability service for extended data retention, as discussed in the *Configuring a Multicluster Observability Custom Resource* section.

Enabling the RHACM Observability Service

The RHACM observability service is always installed in the hub cluster, although due to the requirements for persistent storage, CPU, and memory, it is not enabled by default.

If the RHACM observability service is enabled, the observability service add-on is deployed in each managed cluster, in the `open-cluster-management-addon-observability` namespace. In the hub cluster, the central components of the observability service run in the `open-cluster-management-observability` namespace.

Prerequisites for the Observability Service in RHACM

To enable the RHACM observability service, create the `open-cluster-management-observability` namespace.

Create a secret named `multiclusterhub-operator-pull-secret` in the `open-cluster-management-observability` namespace. You can extract the secret named `multiclusterhub-operator-pull-secret` from the `open-cluster-management` namespace, or the secret named `pull-secret` from the `openshift-config` namespace.

The following commands are an example for storing the secret in a variable and creating the `multiclusterhub-operator-pull-secret` secret in the `open-cluster-management-observability` namespace.

```
[user@demo ~]$ DOCKER_CONFIG_JSON=$(oc extract secret/pull-secret \
-n openshift-config --to=-)
# .dockerconfigjson
[user@demo ~]$ oc create secret generic \
multicloudoperator-pull-secret \
-n open-cluster-management-observability \
--from-literal=.dockerconfigjson="$DOCKER_CONFIG_JSON" \
--type=kubernetes.io/dockerconfigjson
secret/multicloudoperator-pull-secret created
[user@demo ~]$
```

The RHACM observability service needs a persistent object store, and supports the following S3 compatible stores:

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

The persistent volume used by the `thanos-receive` StatefulSet object uses the most storage resources. The Thanos receiver accepts incoming Prometheus remote-write requests and stores the data in a local instance of the Prometheus time-series database. The amount of storage required depends on several factors, including the time period over which data is retained.

The following is an example YAML file to create an object bucket claim named `thanos-bc` using the `ocs-external-storagecluster-ceph-rgw` storage class.

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: thanos-bc
  namespace: open-cluster-management-observability
spec:
  storageClassName: ocs-external-storagecluster-ceph-rgw
  generateBucketName: observability-bucket
```



Note

The `ObjectBucketClaim` custom resource is available because the Red Hat OpenShift Data Foundation operator is installed in the hub cluster.

After creating the object bucket claim, extract the configuration map using the following command:

```
[user@demo ~]$ oc extract configmap/thanos-bc \
-n open-cluster-management-observability --to=-
# BUCKET_REGION
us-east-1
# BUCKET_SUBREGION
# BUCKET_HOST
rook-ceph-rgw-ocs-external-storagecluster-cephobjectstore.openshift-storage.svc
```

```
# BUCKET_NAME
observability-bucket-4702d056-1e1b-435d-8b53-e65bc87d8717
# BUCKET_PORT
8080
```

Extract the credentials to the bucket using the following command:

```
[user@demo ~]$ oc extract secret/thanos-bc \
-n open-cluster-management-observability --to=-
# AWS_ACCESS_KEY_ID
1B7I9PL0C5J2BECSVCP6
# AWS_SECRET_ACCESS_KEY
CPBnRVW8075NoIKqhLvg2rpZ0EdXFpb2ppE0qWbD
```

Use the information from the configuration map and the secret to create the `thanos-object-storage` secret. The following is an example YAML file to create the `thanos-object-storage` secret:

```
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-bucket-4702d056-1e1b-435d-8b53-e65bc87d8717
      endpoint: rook-ceph-rgw-ocs-external-
    storageclustercephobjectstore.openshift-storage.svc:8080
      insecure: true
      access_key: 1B7I9PL0C5J2BECSVCP6
      secret_key: CPBnRVW8075NoIKqhLvg2rpZ0EdXFpb2ppE0qWbD
```

Regarding CPU and memory, the observability service requires a minimum of 2701 mCPU and a minimum of 11.9 Gi. In the managed clusters, the observability service add-on requests a minimum of 12 mCPU and 150 Mi of memory.

Creation of a Multicloud Observability Custom Resource

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.

To deploy the observability components on dedicated infrastructure nodes, you can define a set a label to the `infra` machine set by using the `nodeSelector` field in the `MultiClusterObservability` custom resource YAML file. The following is an example `MultiClusterObservability` custom resource YAML file using the `nodeSelector` field to deploy on the `infra` machine set:

```
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  observabilityAddonSpec: {}
  storageConfig:
    metricObjectStorage:
      name: thanos-object-storage
      key: thanos.yaml
  nodeSelector:
    node-role.kubernetes.io/infra:
```



References

For more general 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/2.4/html-single/observability/index

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/2.4/html-single/observability/index#enable-observability

For more information about the scaling of the observability service, refer to the *Scaling for Observability* section at

https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/install/index#scaling-for-observability

For more information about the community project Prometheus used by the observability service in RHACM and RHOCP, visit the Prometheus web site at

<https://prometheus.io/>

For more information about the community project Thanos used by the observability service in RHACM, visit the Thanos web site at

<https://thanos.io/>

► Guided Exercise

Installing the RHACM Observability Stack

- Deploy the observability stack in all the managed clusters in Red Hat Advanced Cluster Management for Kubernetes (RHACM) and verify the installation in the hub cluster and the managed clusters. Then, you explore the Grafana dashboard provided by the RHACM observability stack. Finally, you disable the metrics collection in a cluster used for the stage environment.

Outcomes

- 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



Note

You can find YAML files to complete this guided exercise in the ~/D0480/labs/observability-install directory.

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 that the cluster called `managed-cluster` exists in RHACM.

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

► 1. Enable the observability service from the terminal.

- Open a terminal and log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address 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 ~]$
```

- Create the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc create namespace open-cluster-management-observability
namespace/open-cluster-management-observability created
```

13. Extract the pull secret from the `openshift-config` namespace and store it in a variable.

```
[student@workstation ~]$ DOCKER_CONFIG_JSON=$(oc extract secret/pull-secret \
-n openshift-config --to=-
# .dockerconfigjson
[student@workstation ~]$
```

14. Create the pull secret in the `open-cluster-management-observability` namespace.

```
[student@workstation ~]$ oc create secret generic \
multicloudoperator-pull-secret \
-n open-cluster-management-observability \
--from-literal=.dockerconfigjson="$DOCKER_CONFIG_JSON" \
--type=kubernetes.io/dockerconfigjson
secret/multicloudoperator-pull-secret created
[student@workstation ~]$
```

15. Change to the `~/D0480/labs/observability-install` folder.

```
[student@workstation ~]$ cd D0480/labs/observability-install
[student@workstation observability-install]$
```

16. Review the file named `obc.yaml` with the definition of a new object bucket claim (OBC):

```
[student@workstation observability-install]$ cat obc.yaml
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: thanos-bc
  namespace: open-cluster-management-observability
spec:
  storageClassName: ocs-external-storagecluster-ceph-rgw
  generateBucketName: observability-bucket
```

17. Use the `oc` command to create the `ObjectBucketClaim` resource.

```
[student@workstation observability-install]$ oc create -f obc.yaml
objectbucketclaim.objectbucket.io/thanos-bc created
```

18. Extract the storage configuration details.

```
[student@workstation observability-install]$ oc extract configmap/thanos-bc \
-n open-cluster-management-observability --to=-
# BUCKET_REGION
us-east-1
# BUCKET_SUBREGION

# BUCKET_HOST
rook-ceph-rgw-ocs-external-storagecluster-cephobjectstore.openshift-storage.svc
```

```
# BUCKET_NAME
observability-bucket-4702d056-1e1b-435d-8b53-e65bc87d8717
# BUCKET_PORT
8080
```

1.9. Extract the secret.

```
[student@workstation observability-install]$ oc extract secret/thanos-bc \
-n open-cluster-management-observability --to=-
# AWS_ACCESS_KEY_ID
1B7I9PL0C5J2BECSVCP6
# AWS_SECRET_ACCESS_KEY
CPBnRVW8075NoIKqhLvg2rpZ0EdXFpb2ppE0qWbD
```

1.10. Create a YAML configuration file for the `thanos-object-storage` secret with the information from the previous steps and the template file named `secret.yaml`. The resulting file should look as follows:

```
[student@workstation observability-install]$ 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-bucket-4702d056-1e1b-435d-8b53-e65bc87d8717
      endpoint: rook-ceph-rgw-ocs-external-storagecluster-
cephobjectstore.openshift-storage.svc:8080
      insecure: true
      access_key: 1B7I9PL0C5J2BECSVCP6
      secret_key: CPBnRVW8075NoIKqhLvg2rpZ0EdXFpb2ppE0qWbD
```



Note

Note that the `endpoint` value is composed by the `BUCKET_HOST` and the `BUCKET_PORT` extracted from the `thanos-bc` configuration map.

1.11. Use the `oc` command to create the secret in RHOCP.

```
[student@workstation observability-install]$ oc create -f secret.yaml
secret/thanos-object-storage created
```

1.12. Review the YAML file named `mcobs.yaml` defining the multicluster observability custom resource.

```
[student@workstation observability-install]$ cat mcobs.yaml
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
```

```

metadata:
  name: observability
spec:
  enableDownsampling: true
  observabilityAddonSpec:
    enableMetrics: true
    interval: 30
  storageConfig:
    alertmanagerStorageSize: 1Gi
    compactStorageSize: 20Gi
    metricObjectStorage:
      key: thanos.yaml
      name: thanos-object-storage
    receiveStorageSize: 20Gi
    ruleStorageSize: 1Gi
    storageClass: ocs-external-storagecluster-ceph-rbd
    storeStorageSize: 10Gi
  
```

**Note**

The provided `mcobs.yaml` file contains custom parameters for the observability persistent storage.

- 1.13. Use the `oc` command to create the multicluster observability instance in RHOCP.

```
[student@workstation observability-install]$ oc create -f mcobs.yaml
multiclusterobservability.observability.open-cluster-management.io/observability
created
```

This initiates the deployment of the observability components.

- 1.14. Monitor the progress of the installation using the following command:

```
[student@workstation observability-install]$ watch 'oc describe mco \
observability | grep -A 6 Status'
...output omitted...

Status:
  Conditions:
    Last Transition Time: 2022-04-08T10:23:37Z
    Message: Installation is in progress
    Reason: Installing
    Status: True
    Type: Installing
    Last Transition Time: 2022-04-08T10:24:58Z
    Message: Observability components are deployed and running
    Reason: Ready
    Status: True
    Type: Ready
  Events: <none>

...output omitted...
```

It takes around one minute to deploy the observability components. When the message **Observability components are deployed and running** is displayed, exit the watch command by pressing **Ctrl+C** and continue with the next step.

1.15. Change to the `/home/student` folder.

```
[student@workstation observability-install]$ cd /home/student
[student@workstation ~]$
```

▶ 2. Verify the deployment of the observability components in the managed cluster.

2.1. Log in to the `ocp4-mng` cluster.

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

2.2. Verify the status of the components of the **observability** add-on in the cluster named `managed-cluster` in RHACM. These components are installed in the `open-cluster-management-addon-observability` namespace.

```
[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-7c5cf96599  1         1         1
  12m
```

**Note**

If the pods are not ready, look for error messages in the pod logs, 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 **Home > Overview** in the RHACM console to display the Grafana dashboard. Analyze the capacity and utilization data provided about CPU and memory of the clusters.

**Note**

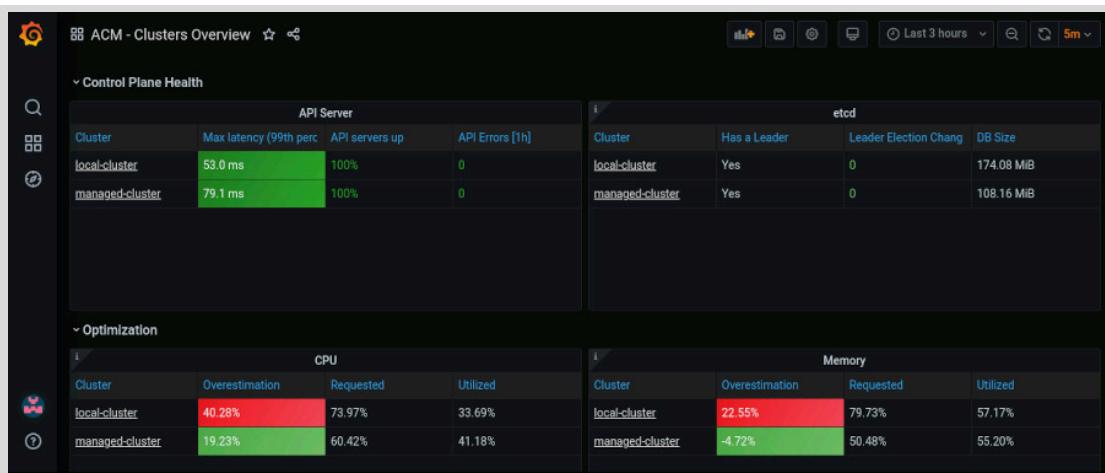
After enabling the observability components, RHACM shows direct access to the Grafana dashboard from within the RHACM console.

- 3.1. From the **workstation** machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click **htpasswd_provider** and log in as the **admin** user with the **redhat** password.
- 3.2. Navigate to **Home > Overview** and then click **Grafana** to display the Grafana dashboard..

The screenshot shows the Red Hat Advanced Cluster Management for Kubernetes web interface. The top navigation bar includes the Red Hat logo, the title 'Advanced Cluster Management for Kubernetes', and a user dropdown for 'admin'. The left sidebar has a 'Home' section with 'Overview' selected, and other sections like 'Welcome', 'Infrastructure', 'Clusters', 'Bare metal assets', 'Automation', and 'Applications'. The main content area is titled 'Overview' and features a card with a cloud icon labeled 'Other' and '2 Cluster'. At the top right of the main area, there are buttons for 'Grafana', '+ Add provider connection', and 'Refresh every 1m'. Below the main content, the URL 'https://multicloud-console.apps.ocp4.example.com/grafana/d/2b679d600f3b9e7676a7c5ac3643d448/acm-clusters-overview' is shown in the address bar.

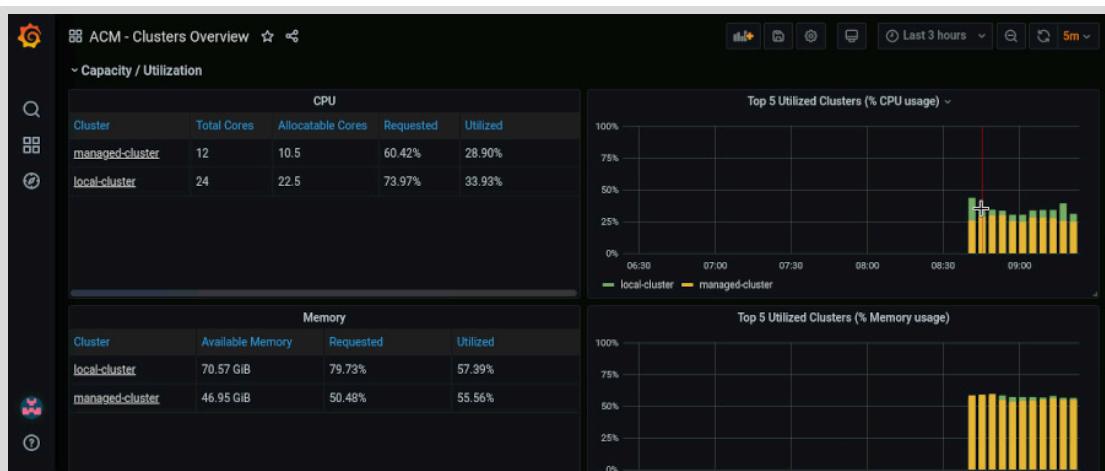
**Note**

Reload the Grafana dashboard if the panels show any error messages.



The RHACM Grafana dashboard contains information about all the clusters in the fleet. The observability add-on, deployed in all the managed clusters, sends information to the central Grafana instance deployed in the hub cluster.

3.3. In the Grafana dashboard, scroll down to the Capacity / Utilization table.



Grafana gathers information about CPU and memory utilization across all the clusters in the fleet. This way, a system administrator can monitor all the clusters from the same dashboard.



Note

In some tables, data can take a few minutes to appear after the observability service is installed.

- ▶ **4.** Disable 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 API server address 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 `observability=disabled` label to the `managedcluster` custom resource 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 API server address 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-addon-observability
No resources found in open-cluster-management-addon-observability namespace.
```

The expected result is that no resources are found in the `open-cluster-management-addon-observability` namespace.



Note

The Grafana dashboard can show cached information about the `managed-cluster` after disabling the observability add-on. All the information about the `managed-cluster` cluster eventually disappears.

Finish

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

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

This concludes the section.

Customizing the RHACM Observability Stack

Objectives

- Customize the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability stack.

Observability Service Customizations in RHACM

In production environments, you might need to adapt the observability service to the workloads and processes of your fleet of clusters.

This customization usually involves creating custom Prometheus metrics, and adding or modifying existing dashboards in Grafana. You can also create Prometheus recording rules to precalculate time-consuming expressions, and configure custom alerting rules to detect any potential issue in their environments.

With the RHACM observability service, you can also configure Prometheus to forward metrics to an external or centralized alerting engine.

Prometheus RHACM Default Metrics

The RHACM observability service receives metrics from Prometheus. Prometheus defines a custom functional query language named Prometheus Query Language (PromQL). The Prometheus metrics are aggregated in time series and defined using expressions written in PromQL.

The RHACM metrics are defined by the `MultiClusterObservability` custom resource. All the current metrics for the `observability-metrics-allowlist` ConfigMap object exist in the `open-cluster-management-observability` namespace in the hub cluster.

You can remove default metrics by deleting them from the `observability-metrics-allowlist` ConfigMap object.

Grafana RHACM Default Dashboards

When you enable the observability service in RHACM, the `MultiClusterObservability` custom resource creates a Grafana console with more than 15 dashboards. Navigate to the `Overview` page in the RHACM web console and then click the `Grafana` link to access these dashboards.

The RHACM Grafana dashboards display the status of the cluster fleet, always classifying the data by managed clusters. You can hover over the `i` icon in the upper-left of each panel for more information about that panel.

Cluster	Has a Leader	Leader Election Chang	DB Size
local-cluster	Yes	0	362.83 MiB
managed-cluster	Yes	0	115.27 MiB

Some of the Grafana dashboards come from the Red Hat OpenShift Container Platform (RHOC) monitoring stack, but include a `cluster` filter, which refers to each managed cluster in the fleet. Other dashboards are new for RHACM, and show data based on longer ranges of time, such as those dashboards with `Service-level Overview` in their name.

Review the ConfigMap objects in the `open-cluster-management-observability` namespace to learn how each dashboard is defined. ConfigMap names use the `grafana-dashboard-` prefix.

RHACM Default Alerts

When you enable the observability service in RHACM, the `MultiClusterObservability` custom resource creates an instance of Alertmanager, which is a component of Prometheus, along with several default alerting rules.

Review the `thanos-ruler-default-rules` ConfigMap object in the `open-cluster-management-observability` namespace, to learn how each alert is defined.

You can customize the default components of the RHACM observability service discussed previously.

Creating Custom Recording and Alerting Rules

To benefit from the RHACM observability stack, you can add customizations to the Grafana and Alertmanager instances.

Prometheus recording rules

You can use recording rules to precalculate expensive expressions. The resulting metric is saved as a new set of time series, which are used to create custom Grafana dashboards.

Remember that you can study the default existing recording rules in the `observability-metrics-allowlist` ConfigMap object in the `open-cluster-management-observability` namespace in the hub cluster. To add custom metrics, you need to create a new `observability-metrics-custom-allowlist` ConfigMap object in the `open-cluster-management-observability` namespace.

Prometheus alerting rules

You can use alerting rules to receive alerts based on determined conditions. Review the `thanos-ruler-default-rules` ConfigMap object in the `open-cluster-management-observability` namespace to learn how each alert is defined. To add custom alerts, create a new `thanos-ruler-custom-rules` ConfigMap object in the `open-cluster-management-observability` namespace.

Configuring a Multicloud Observability Custom Resource

The `MultiClusterObservability` custom resource contains all the parameters required to configure the observability service. Use this custom resource to specify the service behavior in the following areas:

- Persistent storage configuration
- Alertmanager configuration
- Metrics retention policy
- Number of replicas of different components of the observability service
- Configuration of the observability add-on deployed on the managed clusters

The following is a `MultiClusterObservability` custom resource YAML file with customized values for the storage class and storage size:

```
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  enableDownsampling: true
  observabilityAddonSpec:
    enableMetrics: true
    interval: 300
  storageConfig:
    alertmanagerStorageSize: 1Gi
    compactStorageSize: 20Gi
    metricObjectStorage:
      key: thanos.yaml
      name: thanos-object-storage
    receiveStorageSize: 20Gi
    ruleStorageSize: 1Gi
    storageClass: ocs-external-storagecluster-ceph-rbd
    storeStorageSize: 10Gi
```

See the documentation listed in the References section to learn the complete definition of the `MultiClusterObservability` custom resource.

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/2.4/html-single/observability/index

For more information about designing custom Grafana dashboards, refer to the *Designing your Grafana Dashboard* section of the RHACM Observability Guide at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/observability/index#designing-your-grafana-dashboard

For more information about PromQL, visit the Prometheus documentation at
<https://prometheus.io/docs/prometheus/latest/querying/basics/>

For more information about recording rules, visit the Prometheus documentation at
https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/

For more information about alerting rules, visit the Prometheus documentation at
https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

For more information about all the fields of the MultiClusterObservability custom resource, refer to the *MultiClusterObservability* section of the *APIs Guide* at
https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/apis/index#rhacm-docs_apis_multiclusertoobservability_jsonmulticlusertoobservability

► Guided Exercise

Customizing the RHACM Observability Stack

- Configure an existing Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability service to handle a higher load of monitoring data from the managed clusters. Then, you create a custom alerting rule that triggers when the CPU requests of a cluster are above 60% of the total allocatable CPU. Finally, you disable the RHACM observability service.

Outcomes

- Increase the number of 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 triggering
- Remove the observability components from RHACM

Before You Begin



Note

You can find YAML files to complete this guided exercise in the ~/D0480/labs/observability-customize directory.

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, that the **managed-cluster** cluster exists in RHACM, and that the observability service is enabled.

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

- 1. From the terminal, log in to the **ocp4** cluster and increase the the observability metrics receiver pod replicas to six.

- 1.1. Open a terminal and log in to the **ocp4** cluster as the **admin** user with the **redhat** password. The API server address 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 ~]$
```

12. Use the `oc` command to edit the `multicluseterobservability` object in the `openshift-multicluseter-observability` namespace. Update the `spec` section so that it contains the following highlighted information:

```
[student@workstation ~]$ oc edit multicluseterobservability \
-n openshift-multicluseter-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 is displayed:

```
multicluseterobservability.observability.open-cluster-management.io/
observability edited
```

13. Use the `watch` command to verify that the `observability-thanos-receive-default` stateful set now contains six pods.

```
[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 were created recently, and the other three were already running.

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

**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 60% of the available compute capacity.

- 2.1. Change to the `~/DO480/labs/observability-customize` folder.

```
[student@workstation ~]$ cd DO480/labs/observability-customize
[student@workstation observability-customize]$
```

- 2.2. From the terminal, review the file named `custom_rule.yaml` containing the instructions in PromQL.

```
[student@workstation observability-customize]$ cat custom-rules.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: thanos-ruler-custom-rules
data:
  custom_rules.yaml: |
    groups:
      - name: cluster-health
        rules:
          - alert: ClusterCPUREq-60
            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) / sum(kube_node_status_allocatable{resource="cpu"}) by
              (clusterID, cluster) > 0.6
            for: 5s
            labels:
              cluster: "{{ $labels.cluster }}"
              severity: critical
```

- 2.3. Create the `thanos-ruler-custom-rules` ConfigMap in the `open-cluster-management-observability` namespace.

```
[student@workstation observability-customize]$ 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.

- 2.4. 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 observability-customize]$ 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...
```

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

After the `observability-thanos-rule-X` pods restart, the alerting rule is visible from the RHACM Grafana dashboard.

► 3. Verify that the custom alert triggers for the cluster named `managed-cluster`.

- 3.1. From the **workstation** machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
- 3.2. Navigate to **Home > Overview** and click **Grafana** to display a new tab showing the Grafana dashboard.

The screenshot shows the Red Hat Advanced Cluster Management for Kubernetes web interface. The left sidebar has a navigation menu with items like 'Home', 'Welcome', 'Overview', 'Infrastructure', 'Clusters', 'Bare metal assets', 'Automation', and 'Infrastructure environments'. The 'Overview' item is currently selected. The main content area is titled 'Overview' and features a card with a cloud icon, the text 'Other', and the number '2 Clusters'. At the top right, there are links for 'Grafana', '+ Add provider connection', and 'Refresh every 1m'. A status bar at the bottom right indicates 'Last update 5:06:19 AM'.

- 3.3. On the left menu of the Grafana dashboard, click **Explore**.

The screenshot shows the ACM - Clusters Overview dashboard. In the Control Plane Health section, there are two tables: 'Top 50 Max Latency API Server' and 'etc'. The first table shows 'Cluster' (managed-cluster) with 'Max latency (99th perc)' at 49.6 ms, 'API servers up' at 100%, and 'API Errors [1h]' at 0. The second table shows 'Cluster' (local-cluster) and (managed-cluster) both with 'Has a Leader' set to Yes, 'Leader Election Chance' at 0, and 'DB Size' at 141.00 MiB and 92.31 MiB respectively. In the Optimization section, there are two tables: 'Top 50 CPU Overestimation Clusters' and 'Top 50 Memory Overestimation Clusters'. The CPU table shows 'managed-cluster' with 37.66% overestimation, 60.70% requested, and 23.04% utilized. The memory table shows 'local-cluster' with 21.03% overestimation, 65.51% requested, and 44.48% utilized.

- 3.4. In the Metrics text field, type `ALERTS{alertname="ClusterCPUReq-60", alertstate="firing"}`. Click Instant to select the correct query type, and then click Run Query.

Verify that the Table field contains a row with the ClusterCPUReq-60 custom alert. Also, verify that the ClusterCPUReq-60 alert is firing in the managed-cluster cluster.

The screenshot shows the Metrics browser interface. The query entered is `ALERTS{alertname="ClusterCPUReq-60", alertstate="firing"}`. The 'Query type' dropdown is set to 'Instant'. The results table shows one row: Time (2022-02-28 05:50:57), __name__ (ALERTS), alertname (ClusterCPUReq-60), alertstate (firing), cluster (managed-cluster), clusterID (61ba3eb3-bc6d-433...), severity (critical), and Value (1).

- 3.5. Change to the /home/student folder.

```
[student@workstation observability-customize]$ cd /home/student
[student@workstation ~]$
```

▶ 4. Disable the RHACM observability service.

- 4.1. Open a terminal and log in to the ocp4 cluster as the admin user. The API server address 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 typically takes less than five minutes.

Finish

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

```
[student@workstation ~]$ lab finish observability-customize
```

This concludes the section.

▶ Lab

Installing and Customizing the RHACM Observability Stack

- Enable the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability service in all the managed clusters. Then, you create a custom alerting rule that triggers a warning when the memory requested in any cluster of the fleet exceeds 45%. Finally, you verify that the alert triggers for every cluster that meets the condition.

Outcomes

- Enable the observability service in RHACM.
- Create a Prometheus custom alerting rule.
- Verify that the `alertstate` is `firing` in the Grafana dashboard.

Before You Begin



Note

You can find YAML files to complete this guided exercise in the `~/D0480/labs/observability-review` directory.

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, that the cluster named `managed-cluster` exists in RHACM, and that the observability service is not enabled.

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

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 45%.
3. From Grafana, verify that the custom alert triggers for the `local-cluster` and `managed-cluster` 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 observability-review
```

Finish

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

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

This concludes the section.

► Solution

Installing and Customizing the RHACM Observability Stack

- Enable the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability service in all the managed clusters. Then, you create a custom alerting rule that triggers a warning when the memory requested in any cluster of the fleet exceeds 45%. Finally, you verify that the alert triggers for every cluster that meets the condition.

Outcomes

- Enable the observability service in RHACM.
- Create a Prometheus custom alerting rule.
- Verify that the `alertstate` is `firing` in the Grafana dashboard.

Before You Begin



Note

You can find YAML files to complete this guided exercise in the `~/D0480/labs/observability-review` directory.

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, that the cluster named `managed-cluster` exists in RHACM, and that the observability service is not enabled.

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

- Enable the observability service from the terminal.

- Open a terminal and log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address 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 ~]$
```

- 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 and 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 open-cluster-management-observability namespace.

```
[student@workstation ~]$ oc create secret generic \
multicloudoperator-pull-secret \
-n open-cluster-management-observability \
--from-literal=.dockerconfigjson="$DOCKER_CONFIG_JSON" \
--type=kubernetes.io/dockerconfigjson
secret/multicloudoperator-pull-secret created
[student@workstation ~]$
```

- 1.5. Change to the ~/D0480/labs/observability-review folder.

```
[student@workstation ~]$ cd D0480/labs/observability-review
[student@workstation observability-review]$
```

- 1.6. Review the file named `obc.yaml` with the definition of a new object bucket claim (OBC):

```
[student@workstation observability-review]$ cat obc.yaml
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: thanos-bc
  namespace: open-cluster-management-observability
spec:
  storageClassName: ocs-external-storagecluster-ceph-rgw
  generateBucketName: observability-bucket
```

- 1.7. Create the `ObjectBucketClaim` object by using the `oc` command.

```
[student@workstation observability-review]$ oc create -f obc.yaml
objectbucketclaim.objectbucket.io/thanos-bc created
```

- 1.8. Extract the storage configuration information.

```
[student@workstation observability-review]$ oc extract configmap/thanos-bc \
-n open-cluster-management-observability --to=-
# BUCKET_REGION
us-east-1
# BUCKET_SUBREGION

# BUCKET_HOST
rook-ceph-rgw-ocs-external-storagecluster-cephobjectstore.openshift-storage.svc
```

```
# BUCKET_NAME
observability-bucket-4702d056-1e1b-435d-8b53-e65bc87d8717
# BUCKET_PORT
8080
```

- 1.9. Extract the `thanos-bc` secret.

```
[student@workstation observability-review]$ oc extract secret/thanos-bc \
-n open-cluster-management-observability --to=-
# AWS_ACCESS_KEY_ID
1B7I9PL0C5J2BECSVCP6
# AWS_SECRET_ACCESS_KEY
CPBnRVW8075NoIKqhLvg2rpZ0EdXFpb2ppE0qWbD
```

- 1.10. Create a YAML configuration file for the `thanos-object-storage` secret with the information from the previous steps and the template file named `secret.yaml`. The resulting file should look as follows:

```
[student@workstation observability-review]$ 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-bucket-4702d056-1e1b-435d-8b53-e65bc87d8717
      endpoint: rook-ceph-rgw-ocs-external-storagecluster-
cephobjectstore.openshift-storage.svc:8080
      insecure: true
      access_key: 1B7I9PL0C5J2BECSVCP6
      secret_key: CPBnRVW8075NoIKqhLvg2rpZ0EdXFpb2ppE0qWbD
```

- 1.11. Use the `oc` command to create the secret in RHOCP.

```
[student@workstation observability-review]$ oc create -f secret.yaml
secret/thanos-object-storage created
```

- 1.12. Review the YAML file named `mcobs.yaml` defining the multicluster observability custom resource.

```
[student@workstation observability-review]$ cat mcobs.yaml
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  enableDownsampling: true
  observabilityAddonSpec:
    enableMetrics: true
```

```

interval: 30
storageConfig:
  alertmanagerStorageSize: 1Gi
  compactStorageSize: 20Gi
  metricObjectStorage:
    key: thanos.yaml
    name: thanos-object-storage
  receiveStorageSize: 20Gi
  ruleStorageSize: 1Gi
  storageClass: ocs-external-storagecluster-ceph-rbd
  storeStorageSize: 10Gi

```

**Note**

The provided `mcobs.yaml` file contains custom parameters for the observability persistent storage.

- 1.13. Use the `oc` command to create the multicluster observability instance in Red Hat OpenShift Container Platform (RHOCP).

This initiates the deployment of the observability components.

```
[student@workstation observability-review]$ oc create -f mcobs.yaml
multiclusterobservability.observability.open-cluster-management.io/observability
created
```

- 1.14. Monitor the progress of the installation using the following command:

```
[student@workstation observability-install]$ watch 'oc describe mco \
observability | grep -A 6 Status'
...output omitted...

Status:
  Conditions:
    Last Transition Time: 2022-04-08T10:23:37Z
    Message:           Installation is in progress
    Reason:            Installing
    Status:             True
    Type:              Installing
    Last Transition Time: 2022-04-08T10:24:58Z
    Message:           Observability components are deployed and running
    Reason:            Ready
    Status:             True
    Type:              Ready
  Events:             <none>

...output omitted...
```

It takes around one minute to deploy the observability components. When the message **Observability components are deployed and running** is displayed, exit the `watch` command by pressing `Ctrl+C` and continue with the next step.

2. Create a Prometheus alerting rule so that cluster administrators receive alerts when the memory usage of a cluster exceeds 45%.

- 2.1. From the terminal, review the file named `custom_rules.yaml` containing the ConfigMap object with the instructions in Prometheus Querying Language (PromQL).

```
[student@workstation observability-review]$ cat custom-rules.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: thanos-ruler-custom-rules
data:
  custom_rules.yaml: |
    groups:
      - name: cluster-health
        rules:
          - alert: MemoryRequested-45
            annotations:
              summary: Notify when the total memory requested in the clusters is greater than 45%.
              description: "The cluster {{ $labels.cluster }} has more than 45% of the memory requested."
            expr: |
              cluster:memory_requested:ratio > 0.45
            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 observability-review]$ 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. Use the `watch` command to monitor the restart of the pods in the `open-cluster-management-observability` namespace.

```
[student@workstation observability-review]$ 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...
```

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

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 triggers for the `local-cluster` and `managed-cluster` clusters.
 - 3.1. From the `workstation` machine, navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`. When prompted, click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
 - 3.2. Navigate to **Home > Overview** and click **Grafana** to display the Grafana dashboard, and then click **Explore**.
 - 3.3. In the **Metrics** text field, type `ALERTS{alertname="MemoryRequested-45", alertstate="firing"}`.
 - 3.4. Click **Instant** to select the correct query type, and then click **Run Query**.
 - 3.5. Verify that the **Table** field contains a row with the `MemoryRequested-45` custom alert, and that the `alertstate` is `firing` in both clusters.

Time	_name_	alertname	alertstate	cluster	severity	usage	Value
2022-04-08 11:51:51...	ALERTS	MemoryRequested-45	firing	local-cluster	warning	grafana-dashboard	1
2022-04-08 11:51:51...	ALERTS	MemoryRequested-45	firing	managed-cluster	warning	grafana-dashboard	1

- 3.6. Change to the `/home/student` folder.

```
[student@workstation observability-review]$ cd /home/student
[student@workstation ~]$
```

Evaluation

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

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

Finish

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

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

This concludes the section.

Summary

- Administrators can enable the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability service which creates dedicated instances of Grafana and Prometheus.
- You can customize the observability service to add new custom Prometheus rules.
- You can customize the observability service to add new alerting rules.
- The RHACM observability service provide Grafana dashboards with information about the status of the cluster fleet.

Chapter 5

Deploying Applications Across Multiple Clusters with RHACM

Goal

Deploy and manage applications in a multicloud environment with Red Hat Advanced Cluster Management for Kubernetes GitOps.

Objectives

- Describe and define GitOps concepts and the resources of the Red Hat Advanced Cluster Management for Kubernetes (RHACM) application model.
- Deliver applications into multiple clusters by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) GitOps.
- Describe Kustomize concepts and features, and deploy new customizations with the Kustomize command-line tool.

Sections

- Introducing RHACM GitOps and the Application Model (and Quiz)
- Managing Multicloud 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

- Describe and define GitOps concepts and the resources of the Red Hat Advanced Cluster Management for Kubernetes (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 efficiently deploy code to production clusters 24 hours a day.

Implementing GitOps includes introducing practices to automate the process of managing and provisioning infrastructure, such as version control, code review, and CI/CD. GitOps requires storage of resource files in a repository, such as Git. The resource files in the Git repository are referred to as the *source of truth* because they generate the same infrastructure every time you deploy them, much like using source code to build binaries.

GitOps practices enable you to manage the resources required for continuous deployments and integration of rapidly growing infrastructures.

GitOps Principles

Adopting GitOps principles and then implementing and maintaining GitOps practices is essential to your multicloud environment.

Declarative desired state

This state is the final result that the user desires. The declarative description code, or configuration, is centralized and secured within a Git repository as a single source of truth for the entire system.

Systems must use declarative data to export the desired system state. The data must be readable and writable by both machines and humans.

Immutable desired state

This state refers to versioned IaC files that you cannot modify after the deployment. All declarative states are stored in Git. In Git, system infrastructure changes are available chronologically to help with troubleshooting, auditing, and rollbacks.

Infrastructure as code

IaC includes storing all configurations for managing and provisioning infrastructure in a repository. This practice ensures that the same environment is provisioned every time. Similar to the mechanics of software source code, all files under version control can be modified and then deployed. IaC automates infrastructure provisioning of storage and operating systems so that manual provisioning is no longer needed.

Merge requests

In the Git Ops workflow, merge requests are used effectively as a catalyst for infrastructure updates. Merging the updated source code after a review triggers an automatic deployment of the updated resources to the targeted clusters. Merge requests enable project developers to collaborate and review IaC before deploying to production systems.

Continuous integration (CI)

This practice describes 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, ideally several times a day. This speeds up the integration, makes code review easier, and reduces potential problems.

Continuous integration normally includes the validation of integrated changes by using automated integration tests. After the changes are integrated and validated, the team can decide when to deploy a new release. However, continuous integration by itself does not involve automating the release process.

Continuous Integration Tools

At a high level, implementing CI requires two essential tools, a version control system and a CI service. A version control system is a system that assists in managing and tracking your source code in a repository. The repository uses a *main* or *trunk* branch to track the main development line. Feature development occurs in separate branches that are integrated into the main branch after review and validation. Git is one popular version control system. A continuous integration automation service is a server or a daemon that monitors the repository for changes. If the repository changes, the CI automation service checks out the new code and verifies that everything is valid. Red Hat Advanced Cluster Management for Kubernetes (RHACM) deploys the subscription operator to monitor Git for repository changes.

GitOps Workflow

GitOps workflows use Git as the version control system that contains the configurations for managing and provisioning infrastructures. GitOps and IaC use a declarative approach to configuration management, instead of an imperative approach. A declarative model in programming focuses on how the application should accomplish the desired state; an imperative model focus on how it should do it.

A typical GitOps workflow includes the following steps:

- Store all IaC files in Git.
- Create a branch from the main repository to contain the updated IaC file.
- Update the branch, push the change to Git, and create a pull request.
- Request a peer review of the code to ensure proper configuration.
- Approve and merge the updated code.
- The RHACM subscription operator deploys the new code.

Introducing the RHACM Application Model

The ability to subscribe to channel repositories that contain the resources for managing deployments is the foundation of the application model. Application resource definitions are created, managed, and updated in the YAML resource files spec section.

This section enumerates 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 identify 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.

The following example displays a typical subscription YAML file.

```
apiVersion: apps.open-cluster-management.io/v1
kind: Subscription
metadata:
  name: web-app-cluster1
  namespace: web-app
  labels:
    deployment: hello
  annotations:
    apps.open-cluster-management.io/github-branch: main ①
    apps.open-cluster-management.io/github-path: mysql ②
```

- ① The optional `github-branch` annotation specifies that the branch, `main`, contains the deployment IaC files.
- ② The optional `github-path` annotation specifies that the application is deployed from the `mysql` directory from the source repository.

Channels

The application channels (`channel.apps.open-cluster-management.io`) are source definitions of repositories that the cluster can access. The channels can be GitHub repositories, Helm charts, object stores, and deployable resource namespaces on the hub cluster.

Except for GitHub, all channels require individual namespaces. The following example 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 value for `pathname` is required and specifies the URL of the repository that contains the IaC resource files.
- ② The value for `type` is required and specifies the type of repository selected to contain the IaC resource files.

Placement Rules

Placement rules (`placementrule.apps.open-cluster-management.io`) ensure that the application subscription runs on the correct cluster in your infrastructure. Placement rules assist in managing a multicluster deployment and can be shared across multiple subscriptions. The following example illustrates a typical placement rule YAML file:

```
apiVersion: apps.open-cluster-management.io/v1 #API name of the object
kind: PlacementRule
metadata:
  name: cluster1 ①
  namespace: web-app ②
spec:
  clusterSelector:
    matchLabels:
      clusterid: cluster1
```

- ① The value of `name` is required and identifies the placement rule as `cluster1`.
- ② The value of `namespace` is required and specifies the namespace resource `web-app` for the placement rule.

Applications

Applications (`application.app.k8s.io`) in RHACM are used for grouping Kubernetes resources that build an application. The following example 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 value of `namespace` is required and specifies the namespace resource `web-app`.
- ② The value of `matchLabel` is required and specifies that the key and value pair, `app: web-app`, are contained in the subscription resource.



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

Choose the correct answers to the following items:

- ▶ 1. **Which three of the following best practices are implemented through GitOps? (Choose three.)**
 - a. Version control
 - b. Code review
 - c. CI/CD automation
 - d. Multitenancy
 - e. Security
 - f. Database clusters
- ▶ 2. **Which three types of application subscriptions are available through the Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console? (Choose three.)**
 - a. Git
 - b. Helm
 - c. Object storage
 - d. Local storage
 - e. Kustomize
 - f. Templateless
- ▶ 3. **Which of the following operators pulls from the source repository and deploys to the targeted managed clusters?**
 - a. Policy
 - b. Subscription
 - c. Channel
 - d. Placement
 - e. Cluster
 - f. Application

- ▶ **4. Which of the following resources provides source definitions of repositories that the cluster can access?**
 - a. Channel
 - b. Cluster
 - c. Application
 - d. Placement rule
 - e. GitHub
- ▶ **5. Which of the following GitOps principles implements description code, or configuration, that is centralized and contained within a Git repository as a single source of truth for the entire system?**
 - a. Declarative desired state
 - b. Immutable desired state
 - c. Continuous integration
 - d. Merge requests
 - e. Placement rules
- ▶ **6. Which of the following GitOps practices stores all configurations for managing and provisioning infrastructure in a repository?**
 - a. Subscription
 - b. Merge requests
 - c. Continuous integration
 - d. Infrastructure as code
 - e. Channels

► Solution

Introducing RHACM GitOps and the Application Model

Choose the correct answers to the following items:

- ▶ 1. **Which three of the following best practices are implemented through GitOps? (Choose three.)**
 - a. Version control
 - b. Code review
 - c. CI/CD automation
 - d. Multitenancy
 - e. Security
 - f. Database clusters
- ▶ 2. **Which three types of application subscriptions are available through the Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console? (Choose three.)**
 - a. Git
 - b. Helm
 - c. Object storage
 - d. Local storage
 - e. Kustomize
 - f. Templateless
- ▶ 3. **Which of the following operators pulls from the source repository and deploys to the targeted managed clusters?**
 - a. Policy
 - b. Subscription
 - c. Channel
 - d. Placement
 - e. Cluster
 - f. Application

- 4. Which of the following resources provides source definitions of repositories that the cluster can access?
- a. Channel
 - b. Cluster
 - c. Application
 - d. Placement rule
 - e. GitHub
- 5. Which of the following GitOps principles implements description code, or configuration, that is centralized and contained within a Git repository as a single source of truth for the entire system?
- a. Declarative desired state
 - b. Immutable desired state
 - c. Continuous integration
 - d. Merge requests
 - e. Placement rules
- 6. Which of the following GitOps practices stores all configurations for managing and provisioning infrastructure in a repository?
- a. Subscription
 - b. Merge requests
 - c. Continuous integration
 - d. Infrastructure as code
 - e. Channels

Managing Multicloud Application Resources with RHACM

Objectives

- Deliver applications into multiple clusters by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) GitOps.

Introducing Red Hat Advanced Cluster Management GitOps

Red Hat Advanced Cluster Management for Kubernetes (RHACM) provides GitOps capabilities as part of the application deployment lifecycle. RHACM implements GitOps components and adhere to GitOps principles. RHACM combines infrastructure as code (IaC), merge requests, and continuous integration (CI) and continuous deployment (CD) with the GitOps *push* pattern or method.

The GitOps *push* pattern uses a CI/CD pipeline to push changes to multicloud applications. A merge commit in Git triggers the CI/CD pipeline. 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 does not require running agents in each infrastructure component.

This GitOps pattern uses documented CI/CD tooling that is familiar to a large segment of administrators. This familiarity enables a quick start to collaboration on projects with a large pool of contributors. The deployments are standardized and use the same deployment methodology.

Organizing a Git Repository for RHACM GitOps

The RHACM application lifecycle can use multiple types of directory structures. A directory structure provides the logical implementation and deployment of the resource YAML files for one or multiple directories.

For example, the `mysql` application contains a `mysql-app` directory and a `subscriptions` directory containing a `mysql-sub` directory.

In the following `tree` output, the `subscription` directory targets the `mysql-sub` subdirectory containing a single subscription. The subscription operator applies the subscriptions to the hub cluster.

From the `mysql-sub` directory, the subscription operator applies subscriptions and policies to the managed cluster based on the placement rule. Placement rules determine which managed clusters are affected by each subscription.

```

  └── mysql-app
    └── subscriptions
      └── mysql-sub

```

Subscriptions content from the `mysql-sub` directory is applied to all managed clusters, common applications, and common configurations that match the placement rule.

Managing Git Repository Applications

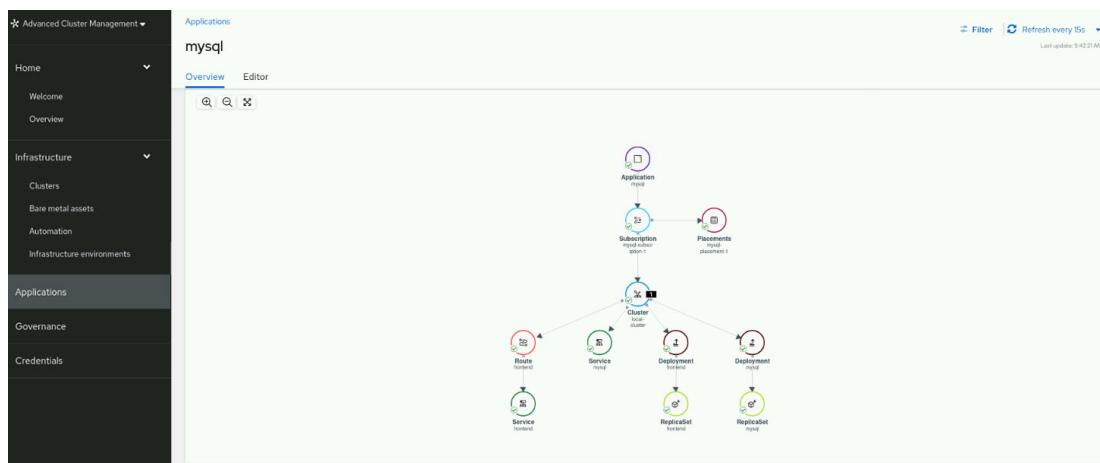
RHACM provides application management functions with options for building and deploying applications and updates. You can manage applications across multicloud environments by implementing subscription and channel automation. In this course, all the application resources are located in the Git repository. RHACM also supports Helm repositories and object storage repositories.

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 visually represents the application and related resources. This visualization of resources assists in troubleshooting and confirming application issues.



Advanced Configuration

The **Advanced configuration** tab provides access to application tables and terminology. You can also access filters for subscriptions, placement rules, and channel resources.

The screenshot shows the RHACM Application console interface with the 'Advanced configuration' tab selected. The left sidebar navigation menu is identical to the previous screenshot. The main area displays a table titled 'Applications' with the 'Advanced configuration' tab selected. The table lists various application subscriptions, each with columns for Name, Namespace, Channel, Applications, Clusters, Time window, and Created. The table includes a search bar at the top and pagination controls at the bottom. The listed subscriptions include: application-chart-sub, assisted-service-sub, cluster-lifecycle-sub, console-chart-sub, discovery-operator-sub, grc-sub, hive-clusterimagesets-subscription-fast-O, management-ingress-sub, policyreport-sub, and search-prod-sub. All subscriptions are in the 'charts-v1' channel and have a 'Local' cluster, with a creation time of '9 hours ago'.

Deploying an Application from RHACM Using a Git Repository

To prepare for application deployment from RHACM by using a Git repository, you must define Kubernetes resources for the application in advance and add them to a Git repository. The RHACM hub uses these resources to create the desired application architecture in the managed clusters.

To deploy and manage an application with RHACM, you must create custom resources. For example, you can create a subscription, namespace, service, channel, placement rule, and a route custom resource to deploy a MySQL application. After creating the resource YAML file, you can use the `kubectl` command to verify that the resource definition is created without an error. The following sections describe the custom resource files.

Namespaces

Each deployed application requires a namespace resource. The following example namespace definition displays the namespace resource created for a MySQL application.

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

Creating Subscriptions

The following example shows a subscription resource created for a MySQL application. You must specify the API version and full group name in the resource YAML file.

You can subscribe to the `main` branch or any branch within the repository by specifying the branch name annotation in the subscription. You can also specify the `mysql` path directory 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
```

Channels

Channels define the source repositories to which a cluster can subscribe by using a subscription.

The following channel definition is 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` path name that 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

```

Placement Rules

Placement rules describe and specify the target clusters where subscriptions are deployed. The following example placement rule definition specifies the `mysql` namespace, and that the application can occupy only the hub or `local-cluster` 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'

```

Deploying an Application from the RHACM Console by Using a Git Repository

After creating the required resource YAML files for the application and pushing them to a Git repository, you can deploy the application by using RHACM.

- Log in to the RHACM web console.
- Navigate to **Applications > Create application**. Use the built-in YAML editor to configure, update, and preview the resources in the YAML file.
- Enter a valid Kubernetes name and namespace for the application. A namespace list shows current namespaces based on your access role. You can choose a namespace from the list or create a new one, if you have the correct access role.
- The **Location for repository resources** menu contains three repository types: Git, Helm, and object storage. Selecting the Git repository displays the following fields:

Field	Value
URL	The location of the Git repository.
Branch	The branch that contains the applicable resource files.
Path	The directory path that contains the applicable resource files.
Commit hash	The commit hash of the specific Git commit.

Field	Value
Tag	The tag annotation of the specific Git branch.
Reconcile	The reconcile option to override the default behavior that reconciles on a commit merge. The other option, replace, replaces the existing resource with the Git resource.
Pre/Post deployment task	The pre- and post-development tasks for Ansible jobs that run before or after the subscription deploys the application resources.

- Use the **Select clusters to deploy** field to define and update the placement rule allocated for the application. You can deploy the placement rule 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 can also choose **Select existing placement configuration** to use a previously defined placement rule for an application in an existing namespace.
- From the **Settings** list, 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 a specified interval, or blocked within a specified interval for the specified **Time window**.

For example, you could define an active **Time window** for web application updates between 3:00 PM and 3:30 PM every Monday. This **Time window** allows application deployments to occur between 3:00 PM and 3:30 PM every Monday. If a deployment begins during a defined time window and is still running when the time window closes, the deployment continues until it completes.

In the previous example, if you were to define the type as blocked instead of active, application deployments could not begin between 3:00 PM and 3:30 PM, but could start at any other time. The subscription operator continues monitoring regardless of the defined deployment windows. The following example shows a **Time window** YAML definition that defines an active time zone window between 10:00 PM and 12:00 AM. Notice the time zone is set to US Eastern time.

```
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 Multicloud Application Resources with RHACM

- Use an RHACM GitOps workflow to deploy a MySQL task list application to the local cluster based on an active time window interval.

Outcomes

- Deliver applications into multiple clusters by using RHACM GitOps.
- Implement application placement rules.
- Define an active time interval for application deployments.

Before You Begin

A GitHub account is required to complete this exercise.

As the student user on the **workstation** machine, use the `lab` command to prepare your system for this exercise. This command ensures that Red Hat Advanced Cluster Management for Kubernetes is installed and that the local cluster is ready and available.

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

- ▶ 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 the **workstation** machine, navigate to the `do480-apps` repository at <https://github.com/redhattraining/do480-apps/>.
 - 1.2. In the upper-right corner of the GitHub repository page, click **Fork** to create a fork.
 - 1.3. Ensure that the **Copy the main branch only** option is disabled. You cannot perform this exercise if you copy only the **main** branch of the repository.



Note

If you have already performed this exercise or made other changes to the fork, then you must delete your fork to avoid problems during this exercise.

To delete the fork, navigate to <https://github.com/your-fork-name/DO480-apps/>. Click **Settings**. Click **Delete this repository** and enter the name of the repository to confirm the deletion.

- ▶ 2. Access the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>.
 - 2.1. From the **workstation** machine, navigate to <https://multicloud-console.apps.ocp4.example.com>.

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

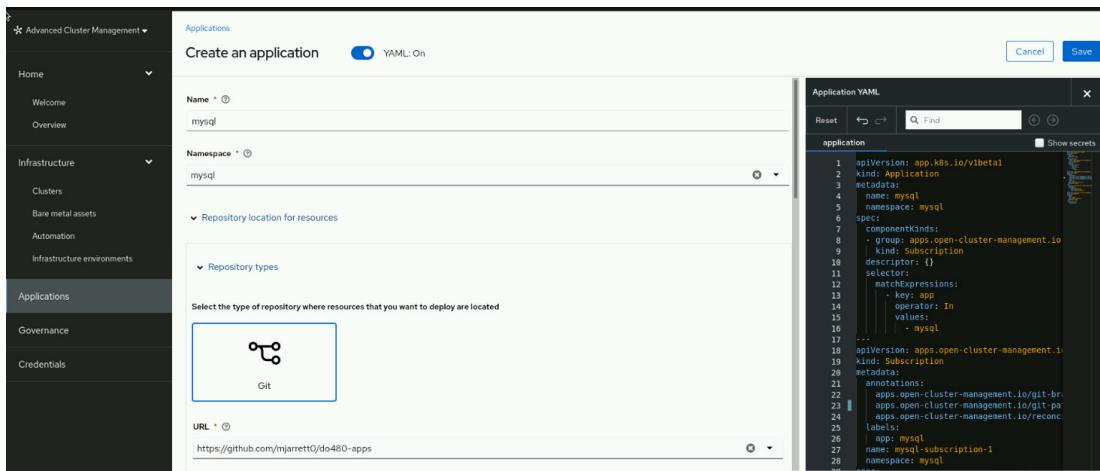
► 3. Use RHACM GitOps to create a new MySQL application based on the following criteria.

MySQL for Local Cluster

Field	Value
Name	mysql
Namespace	mysql
Repository types	Git
URL	https://github.com/your-fork-name/do480-apps/
Branch	main
Path	mysql

- 3.1. On the left navigation bar, navigate to **Applications**, click **Create application** in the middle of the page and then click **Subscription**.
 - 3.2. On the **Applications** page, set **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 application resources on the managed clusters..
 - 3.4. Click **Repository location for resources** and choose **Git** as the repository type for your deployable resources.

For the channel source, enter the URL of your forked `do480-apps` repository in the `URL` field.



- 3.5. Type `main` and `mysql` for the `Branch` and `Path` fields, respectively.

View the subscription and channel configuration in [Application YAML](#) pane.

```

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 to configure a placement rule for the application. Select **Deploy on local cluster** to create a placement rule that deploys the application only on the local cluster.
 - 3.7. Click **Save**.
- ▶ 4. Verify that you can access the MySQL application front end for the local cluster.
- 4.1. On the **Topology** page, click the **Route** pod, and then click the **Location URL** to view the application front end on the local cluster.
 - 4.2. On the subsequent web page, append `/todo/` to the URL to display the application.
- ▶ 5. Define an active time window to deploy updated versions of the application resources.
- 5.1. Click the **Applications** link in the upper-left of the page.
 - 5.2. To update the application resources, click the vertical ellipsis (⋮) menu to the right of the `mysql` application row and click **Edit application**.

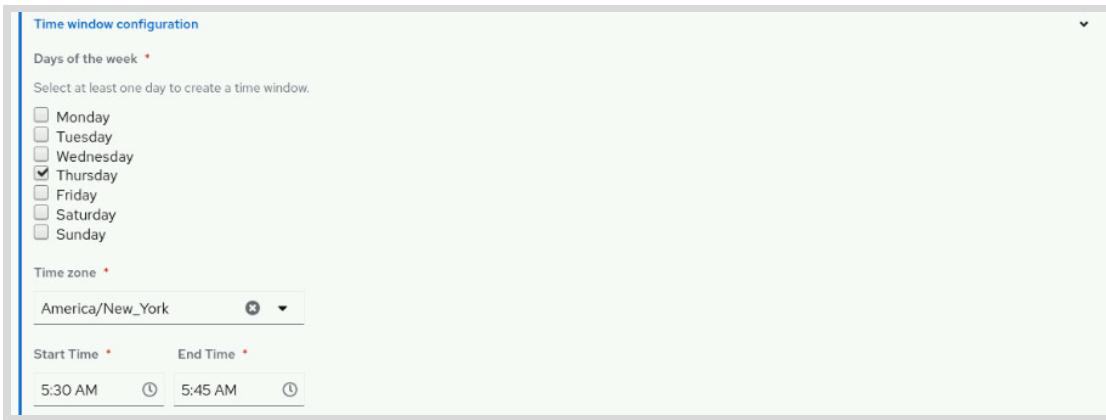
5.3. Scroll down and click **Settings: Specify application behavior**. Click **Active with specified interval** to display the **Time window configuration pane**.

Deployment window

- Always active
- Active within specified interval
- Blocked within specified interval

- 5.4. Specify the day of the week and the start and end times for the time interval. Select the appropriate day of the week, time zone, and start and end times, based on your current day of the week, time zone, and local time. For example, if today is Thursday

and the current time is 05:30 AM, the active time window is defined for Thursday between 05:30 AM and 05:45 AM in the US Eastern time zone.



5.5. Click Update.

Applications

mysql YAML: On Cancel Update

Overview Editor

Name *

Namespace *

Repository location for resources

Repository types

Select the type of repository where resources that you want to deploy are located

 Git

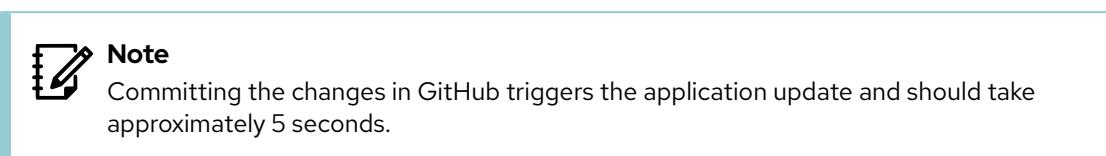
URL *

Application YAML

```
1 apiVersion: v1
2 kind: Application
3 metadata:
4   name: mysql
5   namespace: mysql
6   annotations:
7     app.kubernetes.io/name: mysql
8   spec:
9     component:
10       - group:
11         kind: Deployment
12         selector:
13           matchExpressions:
14             - key: app.kubernetes.io/name
15               operator: In
16               values:
17                 - mysql
18 ...
19 ...
20   apiVersion: apps/v1
21   kind: SubService
22   metadata:
23     name: mysql
24     namespace: mysql
25     annotations:
26       app.kubernetes.io/name: mysql
27     spec:
28       ports:
29         - port: 3306
30       selector:
31         app.kubernetes.io/name: mysql
32       type: ClusterIP
```

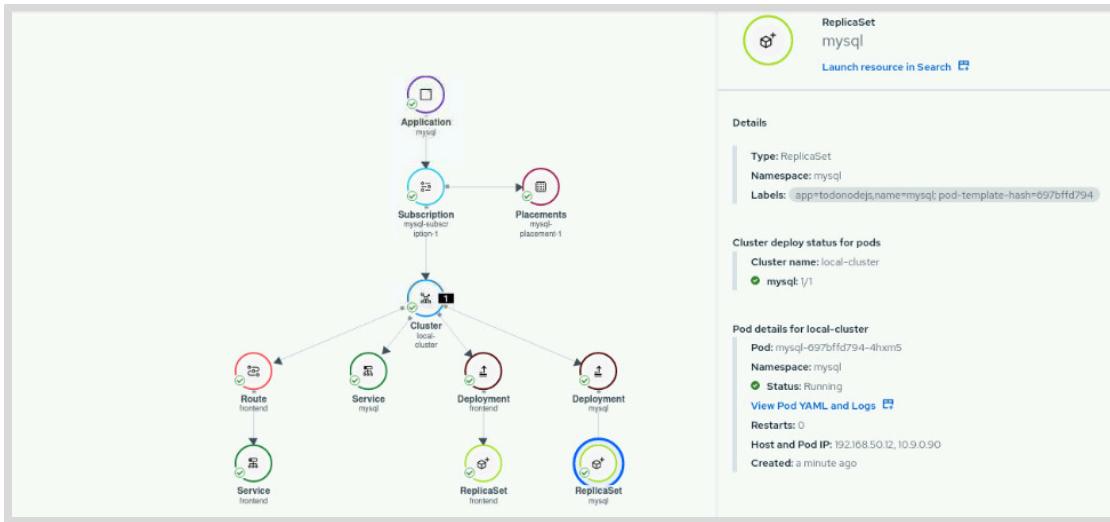
- ▶ 6. Navigate to your forked do480-apps repository in the `main` branch and edit the `do480-apps/mysql/deployment.yaml` file. Update the container image to `mysql-80:156`.

 - 6.1. In your GitHub fork, ensure that you are in the `main` branch. Click `mysql`, and then click the `deployment.yaml` file link to access the file. Click the pencil icon to edit the YAML file.
 - 6.2. In the YAML editor, update the current image tag to `mysql-80:1-156`. Scroll to the end of the page and click `Commit changes` to update the `main` branch.



```
spec:
  containers:
    - image: registry.redhat.io/rhel8/mysql-80:1-156
      name: mysql
      env:
```

- 7. Return to the RHACM web console and view the application **Topology** page to verify that the new pod is created.
- 7.1. On the **Topology** page, click the **ReplicaSet** pod. View the created time for the pod as the resource updates and creates a new pod. After a couple minutes, you see the newly created **mysql** **ReplicaSet** pod.



- 8. Verify that resource updates cannot occur outside the active time window. For example, the current time is 5:30 AM, the time zone is US Eastern time, and the time window interval is defined to end at 5:45 AM.
- 8.1. After the time window interval expires, return to your GitHub fork, ensuring that you are in the **main** branch. Click **mysql** and then click the **deployment.yaml** file link to access the file. 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**. Scroll to the end of the page and click **Commit changes** to update the main branch.

```
spec:
  containers:
    - image: registry.redhat.io/rhel8/mysql-80:latest
      name: mysql
      env:
```

- 8.3. Return to the RHACM web console and view the application **Topology** page to verify that the new pod is created.
- 8.4. On the **Topology** page, click the **ReplicaSet** pod. The pod has not been recreated because the active time window is now closed.

Finish

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

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

This concludes the section.

Customizing Resources with Kustomize for RHACM

Objectives

- Describe Kustomize concepts and features, and deploy new customizations with the Kustomize command-line tool.

Introducing Kustomize

Kustomize is a configuration management tool that you can use to make declarative changes to application configurations and components while preserving the original base YAML files. Kustomize overlays declarative YAML artifacts, or patches, that override the general settings without modifying the original files. For example, you could have a multiphase application for multiple departments. The application configuration is predominately identical across different departments, but some settings are specific to each department and require customization.

Kustomize Benefits

Kustomize implements the `kustomization.yaml` file, which contains specific settings that override those in the original base configuration. The GitOps team can use Kustomize to consume base YAML file updates while maintaining customized settings.

Kustomize implements the concepts of *bases* and *overlays*. Bases are the minimal YAML files that contain the settings shared between the application target environments. Managing Kubernetes configurations with Kustomize provides the following benefits.

Reusability

Kustomize lets you reuse base YAML files across multiple environments. If the deployment is successful with the base settings, but not when using the overlays, you can troubleshoot by comparing the behavior of the overlays to the behavior of the base.

Templateless

Kustomize uses standard YAML syntax that is the same as Kubernetes. It does not have a templating language.

Kustomize Files

Kustomize breaks down configuration deployment manifests into base YAML files and overlay YAML files for environment specific settings. After the customization parameters are defined, Kustomize rebuilds and deploys a new manifest with the `kubectl kustomize` and `kubectl apply` commands, respectively.

Kustomize consists of the following file types:

Base YAML files

The base YAML files contain the application settings that are identical or common to multiple environments.

Overlay YAML files

The overlay YAML files contains the application settings that are unique and 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 are called *variants* and consist of the base YAML files and their overlays. Generally, in a multicloud infrastructure, each environment has a corresponding `kustomization.yaml` file that contains specific settings for different environments, such as production and development.

Kustomize Transformers

Kustomize provides a wide range of default transformers that focus on simplifying application deployments in multicloud environments. Transformers are responsible for modifying the resource and the build process. Kustomize creates new resources by applying a series of transformations to the original set of resources.

For example, you might have production, development, and quality assurance environments running an application that requires different image versions for each environment. Kustomize provides the `images` transformer that can update image names and tags using the `newName` and `newTag` directives. The `images` transformer eliminates the requirement to update all the applicable manifests for each environment individually.

There are many more transformers available depending on your needs. This section discusses several of the most commonly used Kustomize transformers in more detail, and provides example `kustomization.yaml` files to help you implement them.

Adding Prefixes to Resource Names

Use the `namePrefix` transformer to prepend a specific prefix to resource names.

The following `kustomization.yaml` examples apply the prefix `prod-` to all resource names for the production environment and the prefix `dev-` to all resource names in the development environment.

```
#Production
kind: Kustomization

#path to base directory
bases:
- ../../base

#Add name prefix "prod"
namePrefix: prod-
```

```
#Development
kind: Kustomization

#path to base directory
bases:
- ../../base

#Add name prefix "dev"
namePrefix: dev-
```

Replacing Label Values

Use the `commonLabels` transformer to replace label values for a specific keyword.

For example, 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 `prod-todolist.js` and `dev-todolist.js` labels 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 the `commonLabels` transformer does not append or prepend characters but directly replaces the values.

Replacing Images

Use the `images` transformer to replace the image with a customized image name.

For example, the production and development cluster sets require an image that references a specific cluster set environment. The `images` transformer replaces the `mysql-image` placeholder in the `deployment.yaml` file. The images for the production and development cluster sets are defined in their respective `kustomization.yaml` files with the `newName` and `newTag` keys.

The following `kustomization.yaml` files define the `newName` value as `quay.io/example/todo-single` for both the production and development cluster sets. The production cluster set uses the stable `v1.1` tag that has been tested and secured. However, the development cluster set uses the latest open source release of the image, with the `v2.0-beta` tag.

```
#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
```

Merging Patches

Use the `patchesStrategicMerge` transformer to merge specific configuration segments into to the common base configuration.

For example, the development cluster set uses the default `emptyDir` setting with ephemeral storage. However, the production environment requires persistent volumes and persistent volume claims (PVCs) for storage. Define the PVCs in a separate `pvc.yaml` file. Then, edit the `kustomization.yaml` file for the production environment to include `pvc.yaml` in the `resources` section. Finally, use the `patchesStrategicMerge` transformer to apply the volumes defined in the `pvc.yaml` file, as shown in the following example:

```
#Production
patchesStrategicMerge:
  - pvc.yaml
```

Directory Structure

Kustomize does not enforce a strict directory structure. The use case determines the directory structure. A typical use case for Kustomize is an application deployment that contains a base configuration and multiple variants of that configuration.

Consider the following example use case and the proposed directory structure. Your organization has a base set of files to deploy a PostgreSQL database, and an overlay that patches the base manifest to use a specific type of storage. The infrastructure consists of two environments, development and production; both environments run MySQL applications on their respective clusters. Although the configuration settings are mostly the same, there are some differences between the storage requirements that you customize by using a patch overlay.

The following is an example of the directory structure:

```
— base
|   — deployment.yaml
|   — kustomization.yaml
|   — service.yaml
└── overlays
    — development
    |   — kustomization.yaml
    — production
    |   — kustomization.yaml
    |   — pvc.yaml
```

The base directory must contain a `kustomization.yaml` file that specifies the resources required to build a new application manifest.

```
kind: Kustomization

resources:
  - deployment.yaml
  - service.yaml
```

The base directory contains the resource files for the application. You must build this directory by using the `kubectl kustomize .`

```
[student@workstation base]$ kubectl kustomize .
```

This command takes the YAML source (from a path or URL) and creates a new YAML file that can be piped into the `kubectl create` command.



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

- Use Kustomize to maintain a common set of deployment resources and build overlays. The overlays modify the base deployment resources for specific environments. You also build deployment overlays that modify available storage and replicas on the local and managed clusters.

Outcomes

- Build a base deployment for Kustomize.
- Build overlays to modify the base deployment.
- Configure the `kustomization.yaml` file to add resources.
- Deploy an application modified and managed with Kustomize.

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 Red Hat Advanced Cluster Management (RHACM) is installed and that clusters are ready and available.

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

- ▶ 1. Fork the D0480-apps course GitHub repository at <https://github.com/redhattraining/D0480-apps/>. This repository contains the YAML files required to build the application.
 - 1.1. From the `workstation` machine, navigate to the D0480-apps repository at <https://github.com/redhattraining/D0480-apps/>.
 - 1.2. In the upper-right corner of the GitHub repository page, click **Fork** to create a fork for your repository.
 - 1.3. Ensure that the **Copy the main branch only** option is disabled. You cannot perform this exercise if you copy only the `main` branch of the repository.



Note

If you have already performed this exercise or made other changes to the fork, then you must delete your fork to avoid problems during this exercise.

To delete the fork, navigate to <https://github.com/your-fork-name/D0480-apps/>. Click **Settings**. Click **Delete this repository** and enter the name of the repository to confirm the deletion.

Chapter 5 | Deploying Applications Across Multiple Clusters with RHACM

► 2. Access the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>.

2.1. From the workstation machine, navigate to <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 RHACM GitOps to create a new MySQL application based on the following criteria.

MySQL for Development Clusters

Field	Value
Name	mysql
Namespace	mysql
Repository types	Git
URL	https://github.com/your-fork-name/D0480-apps/
Branch	main
Path	mysql

3.1. On the left navigation bar, navigate to **Applications**, click **Create application**, and then click **Subscription**.

3.2. On the **Applications** page, set **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 application resources on the managed clusters.

3.4. Click **Repository location for resources** and select **Git** for the repository type for your deployable resources.

For the channel source, enter the URL of your forked **D0480-apps** repository in the **URL** field.

```

apiVersion: app.k8s.io/v1beta1
kind: Subscription
metadata:
  name: mysql
  namespace: mysql
spec:
  componentKinds:
    - group: apps.open-cluster-management.io
      kind: Subscription
      descriptor: {}
      selector:
        matchExpressions:
          - key: app
            operator: In
            values:
              - mysql
...
  apiVersion: apps.open-cluster-management.io
  kind: Subscription
  metadata:
    annotations:
      apps.open-cluster-management.io/git-br: apps.open-cluster-management.io/git-pa
      apps.open-cluster-management.io/reconc:
      labels:
        name: mysql
        name: mysql-subscription-1
        namespace: mysql

```

- 3.5. Type `main` and `mysql` for the **Branch** and **Path** fields, respectively.
- 3.6. Scroll down and click **Select clusters to deploy to** to configure a placement rule for the application. Select **Deploy on local cluster** to create a placement rule that deploys the application only on the local cluster.
- 3.7. Click **Save**.
- ▶ 4. Verify that you can access the MySQL application front end for the local cluster.
- 4.1. On the **Topology** page, click the **Route** pod, and then click the **Location URL** to view the application front end on the local cluster.
 - 4.2. On the subsequent web page, append `/todo/` to the URL to display the application.
- ▶ 5. Use the RHACM web console to add the `env=development` and `env=production` labels to the `local-cluster` and `managed-cluster` clusters respectively. The web console URL is <https://multicloud-console.apps.ocp4.example.com>.
- 5.1. Navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click **htpasswd_provider** and log in as the **admin** user with the **redhat** password.
 - 5.2. On the left navigation menu, click **Clusters** to add new labels to the `local-cluster` and `managed-cluster` clusters.
 - 5.3. Click the vertical ellipsis (⋮) menu to the right of the `local-cluster` row, and then click **Edit labels**.
 - 5.4. Type `env=development` for the new label and then click **Save**.
 - 5.5. Click the vertical ellipsis menu to the right of the `managed-cluster` row, and then click **Edit labels**.
 - 5.6. Type `env=production` for the new label and then click **Save**.
- ▶ 6. Use the `kubectl kustomize` command to build the `base` directory and corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-kustomize` directory. The `base` directory contents are located at http://github.com/your_fork/D0480-apps in the `main-kustomize` branch.
- 6.1. From the RHACM console, navigate to **Applications**, click the vertical ellipsis menu on the right side of the `mysql` application row, and then click **Delete application**.
 - 6.2. Select **Remove application related resources** and click **Delete** to delete the application.
 - 6.3. From the **workstation** machine, open a terminal and login to the `ocp4` cluster as the **admin** user. The API Server 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...
```

- 6.4. Navigate to the `D0480/labs/applications-kustomize/` directory.

```
[student@workstation ~]$ cd DO480/labs/applications-kustomize
```

- 6.5. Use Git to clone your forked DO480-apps repository to access the common set of YAML resource files.

```
[student@workstation applications-kustomize]$ git clone \
  https://github.com/your-fork-name/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.
```

- 6.6. Navigate to the DO480-apps directory and checkout the main-kustomize branch, which contains the MySQL deployment YAML files.

```
[student@workstation applications-kustomize]$ cd DO480-apps
[student@workstation DO480-apps]$ git checkout main-kustomize
Branch 'kustomize' set up to track remote branch 'main-kustomize' from 'origin'.
Switched to a new branch 'main-kustomize'
```

- 6.7. Create the ~/DO480/labs/applications-kustomize/DO480-apps/base/kustomization.yaml file to contain the resources and associated customizations.

The kustomization.yaml file should consist of the following content:

```
kind: Kustomization

resources:
- deployment-frontend.yaml
- deployment.yaml
- service.yaml
- service-frontend.yaml
```

- 6.8. Save the changes and close the editor.

- 6.9. Verify that the kustomization.yaml file builds the deployment without errors with the kubectl kustomize command.

```
[student@workstation DO480-apps]$ kubectl kustomize base/
...output omitted...
```

You now have a base set of deployment resources that you can share with both the development and production clusters.

- 7. Examine the deployment.yaml file in the base directory and note that it currently configures ephemeral storage. The development clusters require that the MySQL applications use 5 MiB persistent volume claims for storage. Build an overlay for the development clusters that uses persistent storage.

- 7.1. Create the overlays directory and the corresponding development subdirectory.

```
[student@workstation D0480-apps]$ mkdir -p overlays/{development,production}
```

- 7.2. Ensure that the `overlays` directory is at the same directory level as the `base` directory

```
[student@workstation D0480-apps]$ tree -d
base
overlays/
└── development
└── production
```

- 7.3. Create the `dbclaim-pvc.yaml` file resource, which contains the persistent volume claim, in the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/development/` subdirectory. The `dbclaim-pvc.yaml` file should consist of the following content:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Mi
  storageClassName: nfs-storage
```

- 7.4. Save the changes and close the editor.
- 7.5. Create the corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/development/` subdirectory to add the new resource. The `kustomization.yaml` file should consist of the following content:

```
kind: Kustomization
bases:
  - ../../base
resources:
- dbclaim-pvc.yaml
```

- 7.6. Save the changes and close the editor.
- 7.7. Update the storage setting in the `~/D0480/labs/applications-kustomize/D0480-apps/base/deployment.yaml` file to reference the PVC `mysql-pv-claim`. The `deployment.yaml` file should consist of the following content:

```
...output omitted...
ports:
- containerPort: 3306
```

```

    name: mysql
    volumeMounts:
      - mountPath: /var/lib/mysql
        name: db-volume
    volumes:
      - name: db-volume
    persistentVolumeClaim:
      claimName: mysql-pv-claim
    - name: db-init
      emptyDir: {}
  
```

To expand storage in the future, you need only modify the overlay `kustomization.yaml` file.

- 7.8. Use the `kubectl kustomize` command to test the settings for the development clusters with the new storage resource. Verify that the new resource is added to the deployment.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/development/
...output omitted...
```

- ▶ 8. The development clusters require that the MySQL application pods are identifiable by the prefix `dev-`. Build an overlay for the development clusters that uses the `namePrefix` transformer to set `dev-` as the name prefix.
- 8.1. Add the `namePrefix` transformer with the prefix `dev-` to the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/development/kustomization.yaml` file. The `kustomization.yaml` file should consist of the following content:

```

kind: Kustomization

bases:
  - ../../base
resources:
- dbclaim-pvc.yaml
namePrefix: dev-
  
```

- 8.2. Use the `kubectl kustomize` command to test the settings for the development clusters with the `namePrefix` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/development/
...output omitted...
```

- ▶ 9. The development clusters require that a route for the MySQL application.
- 9.1. Create the `route.yaml` file resource in the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/development/` subdirectory. The `route.yaml` file should consist of the following content:

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  
```

```

labels:
  app: todonodejs
  name: frontend
  name: frontend
spec:
  host: todo.apps.ocp4.example.com
  path: "/todo"
  to:
    kind: Service
    name: dev-frontend
    weight: 100
  wildcardPolicy: None

```

- 9.2. Save the changes and close the editor.
- 9.3. Update the corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/development/` subdirectory to add the new resource. The `kustomization.yaml` file should consist of the following content:

```

kind: Kustomization

bases:
  - ../../base

resources:
  - dbclaim-pvc.yaml
  - route.yaml
namePrefix: dev-

```

► 10. Commit the newly created development Kustomize files to your Git repository.

- 10.1. Add the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

- 10.2. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "kustomize storage"
...output omitted...
```

- 10.3. Push the changes to your Git repository.

```
[student@workstation D0480-apps]$ git push --set-upstream origin main-kustomize
...output omitted...
```

► 11. From the RHACM console, deploy the development customized MySQL application.

- 11.1. On the left navigation bar, navigate to **Applications**, click **Create application**, and then click **Subscription**.

- 11.2. On the **Applications** page, set **YAML: On** to view the YAML in the console as you create the application.
 - 11.3. In both the **Name** and **Namespace** fields, type **mysql**.
 - 11.4. Click **Repository location for resources** and choose **Git** for the repository type for your deployable resources.
For the channel source, enter the URL of your forked D0480-apps repository in the **URL** field.
 - 11.5. Type **main-kustomize** in the **Branch** field and **overlays/development** in the **Path** field to reference the customized configuration.
 - 11.6. Scroll down, click **Select clusters to deploy to**, then select **Deploy application resources only on clusters matching specified labels** to configure a placement rule for the application. Type **env** in the **Label** field and **development** in the **Value** field, and then click **Save**.
 - 11.7. On the **Topology** page, navigate to **Subscription > mysql-subscription-1** and click the **Route** pod. Click **todo.apps.ocp4.example.com/**, located under **Location**, to view the application. On the next page, append **/todo/** to the URL to display the application.
- 12. The production clusters require that the MySQL applications use a 50 MiB persistent volume for storage.
Build an overlay for the production clusters that uses persistent storage.
- 12.1. Return to the terminal and create a new **prod-kustomize** branch.

```
[student@workstation D0480-apps]$ git checkout -b prod-kustomize
Branch 'kustomize' set up to track remote branch 'prod-kustomize' from 'origin'.
Switched to a new branch 'prod-kustomize'
```

- 12.2. Create the **dbclaim-pvc.yaml** file resource that contains the 10 MiB persistent volume, in the **~/D0480/labs/applications-kustomize/D0480-apps/overlays/production/** subdirectory. The **dbclaim-pvc.yaml** file should consist of the following content:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 50Mi
  storageClassName: nfs-storage
```

- 12.3. Save the changes and close the editor.
- 12.4. Create the corresponding **kustomization.yaml** file in the **~/D0480/labs/applications-kustomize/D0480-apps/overlays/production/** subdirectory to add the new resource. The **kustomization.yaml** file should consist of the following content:

```

kind: Kustomization

bases:
  - ../../base

resources:
  - dbclaim-pvc.yaml

```

- 12.5. Save the changes and close the editor.

To expand storage in the future, you need only modify the overlay `kustomization.yaml` file.

- 12.6. Use the `kubectl kustomize` command to test the deployment for the production clusters with the new storage resource. Verify that the new resource has been added to the deployment.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/production
...output omitted...
```

- 13. The production clusters require that the MySQL application pods are identifiable by the prefix `prod-`. Build an overlay for the production clusters that uses the `namePrefix` transformer to set `prod-` as the name prefix.

- 13.1. Add the `namePrefix` transformer with the prefix `prod-` to the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/production/kustomization.yaml` file. The `kustomization.yaml` file should consist of the following content:

```

kind: Kustomization

bases:
  - ../../base

resources:
  - dbclaim-pvc.yaml
namePrefix: prod-

```

- 13.2. Use the `kubectl kustomize` command to test the settings for the production clusters with the `namePrefix` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/production
...output omitted...
```

- 14. The production clusters require a route for the MySQL application.

- 14.1. Create the `route-mng.yaml` file resource in the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/production/` subdirectory. The `route-mng.yaml` file should consist of the following content:

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:

```

```

labels:
  app: todonodejs
  name: frontend
  name: frontend
spec:
  host: todo.apps.ocp4-mng.example.com
  path: "/todo"
  to:
    kind: Service
    name: prod-frontend
    weight: 100
  wildcardPolicy: None

```

- 14.2. Save the changes and close the editor.

Update the corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-kustomize/D0480-apps/overlays/production/` subdirectory to add the new resource. The `kustomization.yaml` file should consist of the following content:

```

kind: Kustomization

bases:
- ../../base

resources:
- dbclaim-pvc.yaml
- route-mng.yaml
namePrefix: prod-

```

- 14.3. Save the changes and close the editor.

- 14.4. Use the `kubectl kustomize` command to test the settings for the production clusters with the `namePrefix` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/production/
...output omitted...
```

- 15. Add and push the newly created `production` Kustomize files to the Git repository.

- 15.1. Add the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

- 15.2. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "prod kustomize storage"
...output omitted...
```

- 15.3. Push the newly created Kustomize files to your Git repository.

```
[student@workstation D0480-apps]$ git push --set-upstream origin prod-kustomize
...output omitted...
```

- ▶ 16. From the RHACM web console, update the MySQL application to use the Kustomize storage configuration.
- 16.1. On the left navigation bar, navigate to **Applications** and click **mysql**.
 - 16.2. Click **Editor** to modify the MySQL application.
 - 16.3. Add an additional repository to the MySQL application for the production clusters, based on the following settings.

MySQL for Production Clusters

Field	Value
Repository types	Git
URL	<code>https://github.com/your_fork/D0480-apps/</code>
Branch	prod-kustomize
Path	overlays/production
Label name	env
Label value	production

- 16.4. Click **Update**.
- 16.5. On the **Topology** page, navigate to **Subscription > mysql-subscription-2** and click the **Route** pod. Click `todo.apps.ocp4-mng.example.com/` under **Location** to view the application. On the next page, append `/todo/` to the URL to display the application.

Finish

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

```
[student@workstation D0480-apps]$ cd ~
[student@workstation ~]$ lab finish applications-kustomize
```

This concludes the section.

► Lab

Managing Applications Across Multiple Clusters with RHACM

- Use Red Hat Advanced Cluster Management for Kubernetes GitOps application management and the Kustomize tool to deploy a MySQL application with customized settings for the APAC and EMEA clusters.

Outcomes

- Configure and deploy a MySQL application with RHACM GitOps.
- Assign additional labels to clusters managed with RHACM.
- Deploy a MySQL application customized for APAC and EMEA clusters.

Before You Begin

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

A GitHub account is required to complete this lab.

The following command ensures that RHACM is installed and that the APAC and EMEA clusters are ready and available.

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

1. Fork the `do480-apps` course 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 `env=apac` and `env=emea` labels to the `local-cluster` and `managed-cluster` clusters respectively. The web console URL is <https://multicloud-console.apps.ocp4.example.com>.
3. Use the `kustomize` command to build the base directory and corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-review` directory. The base directory contents are located at http://github.com/your_fork/do480-apps in the `application-lab` branch. Create a new `apac-apps-lab` branch.
4. Create the `~/D0480/labs/applications-review/D0480-apps/base/kustomization.yaml` file to contain the set of resources and associated customizations for the `base` subdirectory.
5. The APAC clusters require that the MySQL application pods are identifiable by the suffix `-apac`. Build an overlay for the APAC clusters that uses the `nameSuffix` transformer to set `-apac` as the name suffix.
6. Commit the newly created `apac` Kustomize files to your Git repository.
7. From the RHACM web console, deploy the APAC customized MySQL application.

- Examine the `deployment.yaml` file in the base directory and note that it currently uses a specific MySQL image. The APAC clusters require that the MySQL applications use the `registry.redhat.io/rhel8/mysql-80:152` image.

Update the `~/D0480/labs/applications-review/do480-apps/base/deployment.yaml` file to use `mysql` for the image and `cluster-mysql` for the name.

Build an overlay for the APAC clusters that uses the `images` transformer with the following settings:

Image Transformer

Field	Value
name:	mysql
newName:	<code>registry.redhat.io/rhel8/mysql-80</code>
newTag:	1-152

- Commit the newly created apac Kustomize files to your Git repository.
- From the RHACM web console, manually synchronize the customized MySQL application.
- The APAC clusters require a route for the MySQL applications.
Create a YAML file containing the following content for routing and update the corresponding `kustomization.yaml` file.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: frontend
    name: frontend
spec:
  host: todo.apps.ocp4.example.com
  path: "/todo"
  to:
    kind: Service
    name: frontend-apac
    weight: 100
  wildcardPolicy: None
```

- Add and push the newly created apac Kustomize files to the Git repository.
- From the RHACM web console, manually synchronize the customized MySQL application.
- The EMEA clusters require that the MySQL application pods are identifiable by the suffix -emea. Create a new branch named `emea-apps-lab`. Build an overlay for the EMEA clusters that uses the `nameSuffix` transformer to set `-emea` as the name suffix.
- Commit the newly created `emea` Kustomize files to your Git repository.

16. From the RHACM web console, deploy the emea customized MySQL application according to the following settings:

MySQL for EMEA Clusters

Field	Value
Repository types	Git
URL	https://github.com/your_fork/D0480-apps/
Branch	emea-apps-lab
Path	overlays/emea
Label name	env
Label value	emea

17. The EMEA clusters require that the MySQL applications use the `registry.redhat.io/rhel8/mysql-80:156` image. Build an overlay for the EMEA clusters that uses the `images` transformer and the following settings:

EMEA Image Transformer

Field	Value
name:	mysql
newName:	<code>registry.redhat.io/rhel8/mysql-80</code>
newTag:	1-156

18. From the RHACM web console, manually synchronize the customized MySQL application.

19. The EMEA clusters require a route for the MySQL applications.

Create a YAML file containing the following content for routing and update the corresponding `kustomization.yaml` file.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: frontend
    name: frontend
spec:
  host: todo.apps.ocp4-mng.example.com
  path: "/todo"
  to:
    kind: Service
    name: frontend-emea
    weight: 100
  wildcardPolicy: None
```

20. Add and push the newly created emea Kustomize files to the Git repository.
21. From the RHACM web console, manually synchronize the EMEA customized MySQL application.
22. Verify that you can access the MySQL application front end for the APAC and EMEA 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 applications-review
```

Finish

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

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

This concludes the section.

► Solution

Managing Applications Across Multiple Clusters with RHACM

- Use Red Hat Advanced Cluster Management for Kubernetes GitOps application management and the Kustomize tool to deploy a MySQL application with customized settings for the APAC and EMEA clusters.

Outcomes

- Configure and deploy a MySQL application with RHACM GitOps.
- Assign additional labels to clusters managed with RHACM.
- Deploy a MySQL application customized for APAC and EMEA clusters.

Before You Begin

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

A GitHub account is required to complete this lab.

The following command ensures that RHACM is installed and that the APAC and EMEA clusters are ready and available.

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

1. Fork the `do480-apps` course repository at <https://github.com/redhattraining/do480-apps/>. This repository contains the YAML files required to build the application.
 - 1.1. From the **workstation** machine, navigate to the `do480-apps` repository at <https://github.com/redhattraining/do480-apps/>.
 - 1.2. In the upper-right corner of the page, click **Fork** to create a fork for your repository.
 - 1.3. Ensure that the **Copy the main branch only** option is disabled. You cannot perform this exercise if you copy only the **main** branch of the repository.



Note

If you have already performed this exercise or made other changes to the fork, then you must delete your fork to avoid problems during this exercise.

To delete the fork, navigate to <https://github.com/your-fork-name/DO480-apps/>. Click **Settings**. Click **Delete this repository** and enter the name of the repository to confirm the deletion.

2. Use the RHACM web console to add the `env=apac` and `env=emea` labels to the `local-cluster` and `managed-cluster` clusters respectively. The web console URL is `https://multicloud-console.apps.ocp4.example.com`.
 - 2.1. Navigate to the RHACM web console at `https://multicloud-console.apps.ocp4.example.com`. When prompted, click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
 - 2.2. On the left navigation menu, click **Clusters** to add new labels to the `local-cluster` and `managed-cluster` clusters.
 - 2.3. Click the vertical ellipsis (⋮) menu at the end of the `local-cluster` row, and then click **Edit labels**.
 - 2.4. Type `env=apac` for the new label and click **Save**.
 - 2.5. Click the vertical ellipsis menu to the right of the `managed-cluster` row, and then click **Edit labels**.
 - 2.6. Type `env=emea` for the new label and then click **Save**.
3. Use the `kustomize` command to build the base directory and corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-review` directory. The base directory contents are located at `http://github.com/your_fork/do480-apps` in the `application-lab` branch. Create a new `apac-apps-lab` branch.
 - 3.1. From the `workstation` machine, open a terminal and login to the `ocp4` cluster as the `admin` user. The API Server 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...
```

- 3.2. Navigate to the `D0480/labs/applications-review/` directory.

```
[student@workstation ~]$ cd D0480/labs/applications-review
```

- 3.3. Use Git to clone your forked `D0480-apps` repository to access the common set of YAML resource files.

```
[student@workstation applications-review]$ git clone \
https://github.com/your-fork-name/D0480-apps
Cloning into 'D0480-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.
```

- 3.4. Navigate to the `D0480-apps` directory and checkout the `application-lab` branch, which contains the MySQL deployment YAML files.

```
[student@workstation applications-review]$ cd D0480-apps
[student@workstation D0480-apps]$ git checkout application-lab
Branch 'application-lab' set up to track remote branch 'application-lab' from
'origin'.
Switched to a new branch 'application-lab'
```

3.5. Create a new apac-apps-lab branch.

```
[student@workstation D0480-apps]$ git checkout -b apac-apps-lab
Branch 'apac-apps-lab' set up to track remote branch 'apac-apps-lab' from
'origin'.
Switched to a new branch 'apac-apps-lab'
```

4. Create the ~/D0480/labs/applications-review/D0480-apps/base/kustomization.yaml file to contain the set of resources and associated customizations for the base subdirectory.
 - 4.1. Create the ~/D0480/labs/applications-review/D0480-apps/base/kustomization.yaml file. The kustomization.yaml file should consist of the following content:

```
kind: Kustomization

resources:
- deployment-frontend.yaml
- deployment.yaml
- service.yaml
- service-frontend.yaml
```

4.2. Save the changes and close the editor.

4.3. Verify that the kustomization.yaml file builds the deployment without errors by using the kubectl kustomize command.

```
[student@workstation D0480-apps]$ kubectl kustomize base/
...output omitted...
```

This command creates a base set of deployment resources that you can share with both the APAC and EMEA clusters.

5. The APAC clusters require that the MySQL application pods are identifiable by the suffix -apac. Build an overlay for the APAC clusters that uses the nameSuffix transformer to set -apac as the name suffix.
 - 5.1. Create the overlays directory and the apac and emea subdirectories.

```
[student@workstation D0480-apps]$ mkdir -p overlays/{apac,emea}
```

5.2. Ensure that the overlays directory is at the same directory level as the base directory.

```
[student@workstation D0480-apps]$ tree -d
base
overlays/
└── apac
└── emea
```

- 5.3. Create the ~/D0480/labs/applications-review/D0480-apps/overlays/apac/kustomization.yaml file and add the nameSuffix transformer with the -apac suffix. The kustomization.yaml file should consist of the following content:

```
kind: Kustomization
bases:
- ../../base
nameSuffix: -apac
```

- 5.4. Save the changes and close the editor.
- 5.5. Use the kubectl kustomize command to test the settings for the APAC clusters with the nameSuffix transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/apac/
```

6. Commit the newly created apac Kustomize files to your Git repository.
 - 6.1. Add the newly created Kustomize files.
 - 6.2. Commit the newly created Kustomize files.
- 6.3. Push the changes to your Kustomize files to your Git repository.
7. From the RHACM web console, deploy the APAC customized MySQL application.
 - 7.1. On the left navigation bar, navigate to **Applications**, click **Create application**, and then click **Subscription**.
 - 7.2. On the **Applications** page, set **YAML: On** to view the YAML in the console as you create the application.
 - 7.3. In both the **Name** and **Namespace** fields, type **mysql**.

- 7.4. Click **Repository location for resources** and choose **Git** for the repository type for your deployable resources.
- Enter the URL of your forked do480-apps repository in the **URL** field for the channel source.
- 7.5. Type **apac-apps-lab** in the **Branch** field and **overlays/apac** in the **Path** field to reference the customized configuration.
- 7.6. Scroll down, click **Select clusters to deploy to**, and then select **Deploy application resources only on clusters matching specified labels** to configure a placement rule for the application. Type **env** in the **Label** field and **apac** in the **Value** field, and then click **Save**.
8. Examine the **deployment.yaml** file in the base directory and note that it currently uses a specific MySQL image. The APAC clusters require that the MySQL applications use the **registry.redhat.io/rhel8/mysql-80:152** image.
- Update the **~/D0480/labs/applications-review/do480-apps/base/deployment.yaml** file to use **mysql** for the image and **cluster-mysql** for the name.
- Build an overlay for the APAC clusters that uses the **images** transformer with the following settings:

Image Transformer

Field	Value
name:	mysql
newName:	registry.redhat.io/rhel8/mysql-80
newTag:	1-152

- 8.1. Update the image setting in the **~/D0480/labs/applications-review/D0480-apps/base/deployment.yaml** file. The **deployment.yaml** file should consist of the following content:

```
...output omitted...
name: mysql
spec:
  containers:
    - image: mysql
      name: cluster-mysql
      env:
```

- 8.2. Save the changes and close the editor.
- 8.3. Update the corresponding **kustomization.yaml** file in the **~/D0480/labs/applications-review/D0480-apps/overlays/apac/** subdirectory to add the resource. The **kustomization.yaml** file should consist of the following content:

```
kind: Kustomization
bases:
  - ../../base
```

```

nameSuffix: -apac

images:
- name: mysql
  newName: registry.redhat.io/rhel8/mysql-80
  newTag: 1-152

```

- 8.4. Save the changes and close the editor.
- 9.** Commit the newly created apac Kustomize files to your Git repository.
- 9.1. Add the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

- 9.2. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "kustomize APAC image"
...output omitted...
```

- 9.3. Push the changes to your Kustomize files to your Git repository.
- 10.** From the RHACM web console, manually synchronize the customized MySQL application.
- 10.1. On the left navigation bar, navigate to **Applications** and click **mysql**.
- 10.2. On the **Topology** page, click **Sync** and then click **Synchronize** to synchronize application resources with the source repository.
- 11.** The APAC clusters require a route for the MySQL applications.
- Create a YAML file containing the following content for routing and update the corresponding `kustomization.yaml` file.

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: frontend
    name: frontend
spec:
  host: todo.apps.ocp4.example.com
  path: "/todo"
  to:
    kind: Service
    name: frontend-apac
    weight: 100
  wildcardPolicy: None

```

- 11.1. Create the `route.yaml` file resource in the `~/D0480/labs/applications-review/D0480-apps/overlays/apac/` subdirectory. The `route.yaml` file should consist of the following content:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: frontend
    name: frontend
spec:
  host: todo.apps.ocp4.example.com
  path: "/todo"
  to:
    kind: Service
    name: frontend-apac
    weight: 100
  wildcardPolicy: None
```

- 11.2. Save the changes and close the editor.

- 11.3. Update the corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-review/D0480-apps/overlays/apac/` subdirectory to add the resource. The `kustomization.yaml` file should consist of the following content:

```
kind: Kustomization

bases:
- ../../base

nameSuffix: -apac

images:
- name: mysql
  newName: registry.redhat.io/rhel8/mysql-80
  newTag: 1-152

resources:
- route.yaml
```

- 11.4. Save the changes and close the editor.

- 11.5. Use the `kubectl kustomize` command to test the `route.yaml` file for the APAC clusters with the `resources` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/apac/
...output omitted...
```

12. Add and push the newly created apac Kustomize files to the Git repository.

- 12.1. Add the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

- 12.2. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "kustomize apac route"
...output omitted...
```

- 12.3. Push the newly created Kustomize files to your Git repository.

```
[student@workstation D0480-apps]$ git push
...output omitted...
```

13. From the RHACM web console, manually synchronize the customized MySQL application.

- 13.1. On the left navigation bar, navigate to **Applications** and click **mysql**.

- 13.2. On the **Topology** page, click **Sync** and then click **Synchronize** to synchronize application resources with the source repository.

14. The EMEA clusters require that the MySQL application pods are identifiable by the suffix `-emea`. Create a new branch named `emea-apps-lab`. Build an overlay for the EMEA clusters that uses the `nameSuffix` transformer to set `-emea` as the name suffix.

- 14.1. Return to the terminal and create a new branch named `emea-apps-lab`.

```
[student@workstation D0480-apps]$ git checkout -b emea-apps-lab
Switched to a new branch 'emea-apps-lab'
```

- 14.2. Create the `~/D0480/labs/applications-review/D0480-apps/overlays/emea/kustomization.yaml` file and add the `nameSuffix` transformer with the `-emea` suffix. The `kustomization.yaml` file should consist of the following content:

```
kind: Kustomization

bases:
- ../../base
resources:
nameSuffix: -emea
```

- 14.3. Use the `kubectl kustomize` command to test the settings for the EMEA clusters with the `nameSuffix` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/emea/
```

15. Commit the newly created `emea` Kustomize files to your Git repository.

- 15.1. Add the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

15.2. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "kustomize EMEA"
...output omitted...
```

15.3. Push the changes to your Kustomize files to your Git repository.

```
[student@workstation D0480-apps]$ git push --set-upstream origin emea-apps-lab
...output omitted...
```

- 16.** From the RHACM web console, deploy the emea customized MySQL application according to the following settings:

MySQL for EMEA Clusters

Field	Value
Repository types	Git
URL	https://github.com/your_fork/D0480-apps/
Branch	emea-apps-lab
Path	overlays/emea
Label name	env
Label value	emea

16.1. On the left navigation bar, navigate to **Applications** and click **mysql**.

16.2. Click **Editor** to modify the MySQL application.

16.3. Add an additional repository to the MySQL application for the EMEA clusters with the following criteria:

MySQL for EMEA Clusters

Field	Value
Repository types	Git
URL	https://github.com/your_fork/D0480-apps/
Branch	emea-apps-lab
Path	overlays/emea
Label name	env
Label value	emea

16.4. Click **Update**.

17. The EMEA clusters require that the MySQL applications use the `registry.redhat.io/rhel8/mysql-80:156` image. Build an overlay for the EMEA clusters that uses the `images` transformer and the following settings:

EMEA Image Transformer

Field	Value
name:	mysql
newName:	registry.redhat.io/rhel8/mysql-80
newTag:	1-156

- 17.1. The overlay for the EMEA clusters uses the `images` transformer and the following settings:

EMEA Image Transformer

Field	Value
name:	mysql
newName:	registry.redhat.io/rhel8/mysql-80
newTag:	1-156

Update the corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-review/D0480-apps/overlays/emea/` subdirectory to add the new image customization for the EMEA clusters. The `kustomization.yaml` file should consist of the following content:

```
kind: Kustomization
bases:
  - ../../base
nameSuffix: -emea

images:
  - name: mysql
    newName: registry.redhat.io/rhel8/mysql-80
    newTag: 1-156
```

- 17.2. Use the `kubectl kustomize` command to test the settings for the EMEA clusters with the `images` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/emea/
...output omitted...
```

- 17.3. Commit the newly created `emea` Kustomize files to your Git repository.

- 17.4. Add the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

17.5. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "kustomize EMEA image"
...output omitted...
```

17.6. Push the changes to your Kustomize files to your Git repository.

```
[student@workstation D0480-apps]$ git push
...output omitted...
```

18. From the RHACM web console, manually synchronize the customized MySQL application.

18.1. On the left navigation bar, navigate to **Applications** and click **mysql**.

18.2. On the **Topology** page, click **Sync** and then click **Synchronize** to synchronize application resources with the source repository.

19. The EMEA clusters require a route for the MySQL applications.

Create a YAML file containing the following content for routing and update the corresponding `kustomization.yaml` file.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: frontend
    name: frontend
spec:
  host: todo.apps.ocp4-mng.example.com
  path: "/todo"
  to:
    kind: Service
    name: frontend-emea
    weight: 100
  wildcardPolicy: None
```

19.1. Create the `route-mng.yaml` file resource in the `~/D0480/labs/applications-review/D0480-apps/overlays/emea/` subdirectory. The `route-mng.yaml` file should consist of the following content:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: todonodejs
    name: frontend
    name: frontend
```

```
spec:
  host: todo.apps.ocp4-mng.example.com
  path: "/todo"
  to:
    kind: Service
    name: frontend-emea
    weight: 100
  wildcardPolicy: None
```

- 19.2. Save the changes and close the editor.
- 19.3. Update the corresponding `kustomization.yaml` file in the `~/D0480/labs/applications-review/D0480-apps/overlays/emea/` subdirectory to add the new resource. The `kustomization.yaml` file should consist of the following content:

```
kind: Kustomization

bases:
  - ../../base

nameSuffix: -emea

images:
  - name: mysql
    newName: registry.redhat.io/rhel8/mysql-80
    newTag: 1-156

resources:
  - route-mng.yaml
```

- 19.4. Save the changes and close the editor.
- 19.5. Use the `kubectl kustomize` command to test the settings for the EMEA clusters with the `resources` transformer.

```
[student@workstation D0480-apps]$ kubectl kustomize overlays/emea/
...output omitted...
```

20. Add and push the newly created `emea` Kustomize files to the Git repository.

- 20.1. Add the newly created Kustomize files to your Git repository.

```
[student@workstation D0480-apps]$ git add .
...output omitted...
```

- 20.2. Commit the newly created Kustomize files.

```
[student@workstation D0480-apps]$ git commit -m "kustomize EMEA route"
...output omitted...
```

- 20.3. Push the newly created Kustomize files to your Git repository.

```
[student@workstation DO480-apps]$ git push  
...output omitted...
```

21. From the RHACM web console, manually synchronize the EMEA customized MySQL application.
 - 21.1. On the left navigation bar, navigate to **Applications** and click **mysql**.
 - 21.2. On the **Topology** page, click **Sync** and then click **Synchronize** to synchronize application resources with the source repository.
22. Verify that you can access the MySQL application front end for the APAC and EMEA clusters.
 - 22.1. On the **Topology** page, navigate to **Subscription > mysql-subscription-1**. Click the **Route** pod, and then click the **URL** under **Location** to view the application front end on the APAC cluster. On the next page, append `/todo/` to the URL to display the application.
 - 22.2. On the **Topology** page, navigate to **Subscription > mysql-subscription-2**. Click the **Route** pod, and then click the **URL** under **Location** to view the application front end on the EMEA cluster. On the next page, append `/todo/` to the URL to display the application.
 - 22.3. Return to the terminal and change to the `/home/student/` directory.

```
[student@workstation DO480-apps]$ cd /home/student/  
[student@workstation ~]$  
...output omitted...
```

Evaluation

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

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

Finish

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

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

This concludes the section.

Summary

- GitOps consists of infrastructure as code (IaC), merge commits, and continuous integration and continuous deployment (CI/CD).
- GitOps applies the principles of declarative desired state and immutable desired state versions for the application lifecycle in Red Hat Advanced Cluster Management for Kubernetes (RHACM).
- You can use RHACM to deploy a subscription based application.
- You can use the Kustomize tool to apply and manage custom settings for multicluster environments.

Chapter 6

Installing and Configuring Red Hat Quay

Goal

Install and configure Red Hat Quay on Red Hat OpenShift Container Platform (RHOCP).

Objectives

- Describe the architecture and identify the main benefits of Red Hat Quay.
- Install the Quay operator and deploy and configure a Quay registry using the command line.
- Describe the Red Hat Quay concepts for image registry organization, and manage role-based access control (RBAC) by using LDAP

Sections

- Identifying the Components and Features of Red Hat Quay (and Quiz)
- Deploying and Configuring Red Hat Quay (and Guided Exercise)
- Describing the Red Hat Quay Repository Model and RBAC (and Guided Exercise)

Lab

- Installing and Configuring Red Hat Quay

Identifying the Components and Features of Red Hat Quay

Objectives

- Describe the architecture and identify the main benefits of Red Hat Quay.

Benefits of an Enterprise Registry

Red Hat Quay is a registry with enterprise-grade support that provides solutions to common challenges of managing a multicluster and multicloud containerized architecture.

The following list describes some of these solutions.

Platform scalability

The registry scales both storage and compute resources to manage large numbers of Kubernetes and OpenShift clusters pulling and pushing container images.

Access control

A well-defined RBAC model helps you manage teams who access different image repositories on different clusters, or who authenticate against different providers.

Content distribution of the images

The registry supports widely-distributed image registry clients by identifying the locations of the stored images and serving each image from the location nearest the client.

Maintenance of the list of images

Because registry clients can store many different versions of the same image, the registry provides a mechanism to delete images that are not used.

Secure source of truth

The registry helps maintain security by providing access control to images, an auditing registry, and an alerting mechanism.

You can use Red Hat Quay.io, which is the Software-as-a-Service version of Red Hat Quay, or run Quay on one of the many supported platforms.



Note

To see the full list of tested Quay integrations and platforms, refer to <https://access.redhat.com/articles/4067991>

Quay Features

Red Hat Quay is an integrated enterprise registry that is both highly-available and highly-scalable. The following list describes the main features of Quay. See the references section for more information about each feature.

Repository Mirroring

You can configure Quay to mirror external registries. You can copy entire registries, or define filters based on tags to decide which images to copy to the internal Quay registry.

When Quay mirrors an external registry, it copies the images to its own back-end storage and stores the metadata information in its database.

Geo-replication

By using the Geo-replication feature, Quay allows you to store registry images in a geographically distributed storage engine. After writing an image to the preferred storage engine, Quay replicates the image to other storage engines across many synchronized local storage instances. This approach provides users a single point of access to the registry and associated database.

Whenever a client pulls a image, Quay serves it directly from the nearest local storage. This feature reduces bandwidth usage and costs and improves performance for users.

The Quay operator supports geo-replication in version 3.7 and later.

You can use the repository mirroring and the geo-replication features simultaneously.

Enterprise Authentication

Quay supports the following identity providers:

- LDAP
- OpenID Connect
- Keystone, the OpenStack identity provider
- GitHub
- Google

Tags Management and Time Machine

Quay identifies images by mapping the tag with the full SHA256 identifier of the stored image. You can add, modify, create, or delete tags for each repository.

For each image tag, you can configure an expiration period. When a tag expires, or is deleted, the Quay time machine feature keeps the tag for an additional configurable period. Note that when an image tag reaches its expiration, or is deleted, the underlying image is deleted only if there are no more tags pointing to its full SHA256 identifier.

Tag management and time machine features address the challenge of maintaining reasonable storage sizes when the number of stored images grows. Furthermore, with these features you can recover deleted tags.

Image Scanning

Quay has an image vulnerability scanner based on the Clair open source project.

You can use the Quay operator to install and configure the Quay vulnerability scanner when Quay is running on OpenShift.

When Quay receives a push, it sends the image to Clair. Clair unpacks the image and indexes its layers. Then, Clair compares the indexed content with the information from the vulnerabilities databases. Finally, Clair generates the results for Quay.

The indexing mechanism helps save compute resources and reduces the number of scans.

The Red Hat OpenShift Platform Plus subscription includes Quay and Red Hat Advanced Cluster Security for Kubernetes (RHACS). RHACS also includes image scanning. However, in addition to scanning images in a registry like Quay, the RHACS scanner can also scan running containers.

The Quay image scanning feature analyzes stored images by consulting the following list of vulnerabilities databases:

- Alpine SecDB

- AWS UpdateInfo
- Debian Oval
- Oracle Oval
- RHEL Oval
- SUSE Oval
- Ubuntu Oval
- Pyup.io (Python libraries)

Builds

You can configure Quay to watch source code repositories and build images when a change is detected.

When Quay is running on OpenShift, Quay builds images using the `buildah` command in a Kubernetes job that uses a containerized virtual machine.

Notifications

Quay can send notifications for repository events. The following list shows the event types.

- An image push
- The start, stop, fail, cancellation, or queue of a build
- A vulnerability detection

The following list shows the notification actions.

- Quay notification in the Quay web console
- Email
- Webhook POST request
- Flowdock chat system
- Hipchat chat system
- Slack chat system

Prometheus and Grafana Metrics

Quay provides an endpoint that exports Prometheus data.

The Quay operator adds a new Grafana dashboard with Quay metrics to the OpenShift console. This dashboard shows the following statistics, among others.

- Number of users, organizations, repositories, and robot accounts
- Compute resources consumption
- Image pulls and pushes
- Authentication requests

Overview of the Red Hat Quay.io Hosted Service

Quay provides a Software-as-a-Service (SaaS) solution, Red Hat Quay.io. Quay.io has the same benefits as running an enterprise-grade registry but without the maintenance and administration costs of the Quay software and its components. The pricing of Quay.io is based on the number of private repositories.

The features of Red Hat Quay.io and Red Hat Quay are the same, but the responsibilities for repository administrators are not. The following table shows differences in responsibilities between Quay.io and Red Hat Quay features, from a system administrator point of view.

Administrator's responsibility	Red Hat Quay	Quay.io
Configure Security Scanning	Yes	No

Administrator's responsibility	Red Hat Quay	Quay.io
Configure Authorization providers	Yes	No
Configure geo-replication	Yes	No
Configure monitoring and alerts	Yes	No
Configure RBAC	Yes	Limited
Configure logging	Yes	Limited
Configure mirroring	Yes	No
Ownership of storage back end	Yes	No

Overview of the Quay Architecture

The following image describes the components that all Quay registries include.

The components in dark blue are stateful components. Thus, if you need high availability, the components in dark blue must support it.

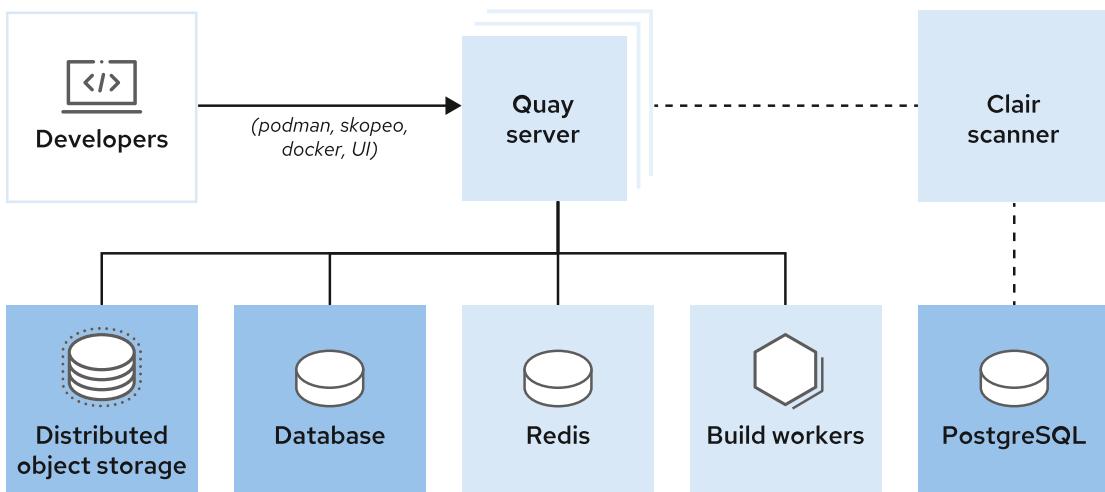


Figure 6.1: Quay Components

Quay server

This application manages the images and their metadata; the server is the main way that users interact with the enterprise registry.

Clair scanner

This application scans the images, searching for known vulnerabilities.

Build workers

These containers can start builds by reacting to source code repositories changes.

Redis

The Redis server stores build logs.

Databases

The Quay database can be PostgreSQL, MariaDB, or MySQL, provided by Red Hat or by a public cloud provider. Red Hat recommends PostgreSQL because all other database engines are scheduled for deprecation in the near future.

The database for Clair can be PostgreSQL only.

Distributed object storage

This storage contains the image's binary content.

For a highly-available storage engine, you can subscribe to Red Hat OpenShift Data Foundation. For more information about the distributed object storage, see *Deploying and Configuring Red Hat Quay*.

The following diagram depicts a deployment of a central Quay registry installed in two different cloud regions. The Quay deployment uses shared databases for the multiple Quay, Clair, and Redis instances. There is an image object storage attached to each region. The Quay instances read and write from their nearest location. The images are replicated in the background to the other storage location.

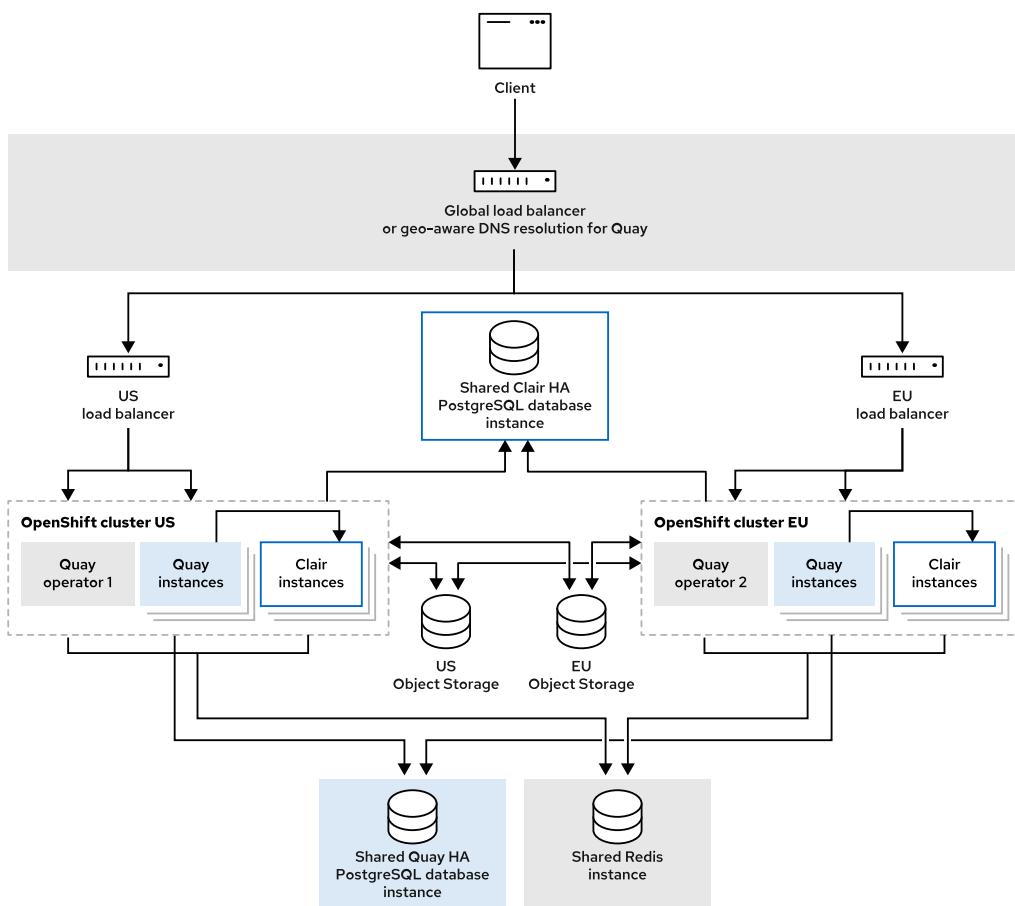


Figure 6.2: Example of Central Quay Running in Two Cloud Regions using Geo-replication

The previous diagram shows the Quay instances running on OpenShift. You can use the same architecture running the Quay instances on other servers in the cloud provider.

Quay on OpenShift

Red Hat provides three different operators to deploy and manage enterprise Quay registries on OpenShift.

The following diagram relates the regular Quay components to the Quay operators:

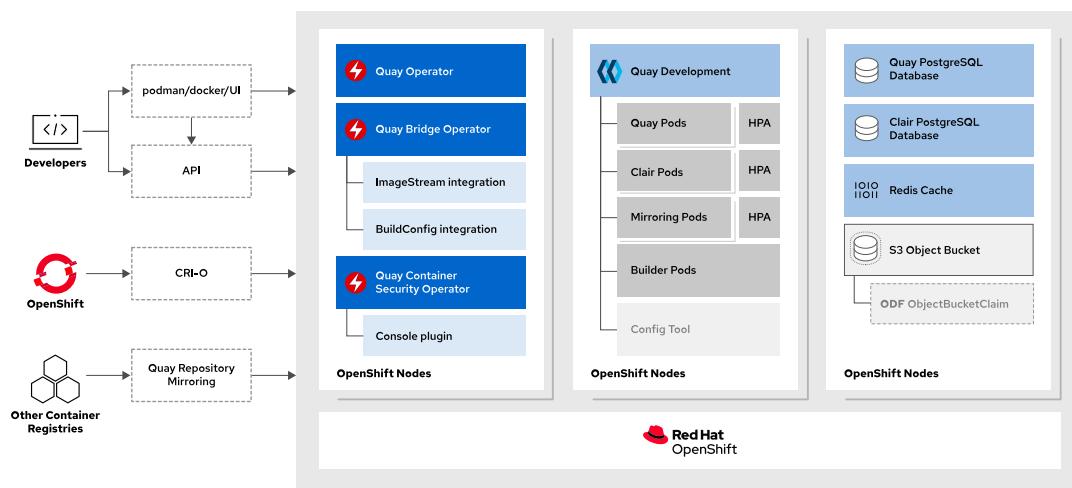


Figure 6.3: Quay Components on OpenShift

The Quay operator

The operator manages the lifecycle of one or more Quay registries running in an OpenShift cluster. This operator provides the `QuayRegistry` custom resource definition (CRD) and allows you to choose which components the operator controls.

The Quay operator can manage the following components.

- `postgres`: If managed, the operator creates the needed pods for the Quay metadata database.
- `clair`: If managed, the operator creates the needed pods for the scanner database and application.
- `redis`: If managed, the operator creates the Redis pods needed for build logs storage.
- `objectstorage`: If managed, the operator uses the `ObjectBucketClaim` API to provide storage for image data.
- `route`: If managed, the operator creates an OpenShift route to access the registry.
- `monitoring`: If managed, the operator configures OpenShift and Quay to gather metrics from the Quay registries.
- `tls`: If managed, the operator uses the default wildcard certificate for the OpenShift route to access Quay.
- `horizontalpodautoscaler`: If managed, the operator defines resources for autoscaling Quay, Clair, and the pods used for mirroring.
- `mirror`: If managed, the operator enables the mirror workers.

More information about the Quay operator is provided elsewhere in this chapter.

The Quay container security operator

The operator integrates image vulnerability information within the OpenShift console.

The Quay bridge operator

This operator provides deeper integration between Quay and OpenShift. More information about the Quay bridge operator is available elsewhere in this course.

Benefits of Using Quay on OpenShift

As stated elsewhere in this lecture, Quay can be used from a standalone installation or from an OpenShift installation.

The following list enumerates the main benefits of using Quay installed on OpenShift.

- The Quay installation on OpenShift is customizable and repeatable.
- The Quay operator can manage one or more Quay registries.
- The operator can manage Quay and all of its dependent components.
- Quay resources scale automatically when usage grows.
- Updating Quay on OpenShift is easier than updating a standalone installation.
- The Quay monitoring, alerting, and logging engines integrate with OpenShift engines.



References

For more information about Quay repository mirroring, refer to the *Repository Mirroring* chapter in the *Manage Red Quay* guide at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/manage_red_hat_quay/index#repo-mirroring-in-red-hat-quay

For more information about Quay geo-replication, refer to the *Geo-Replication* chapter in the *Manage Red Quay* guide at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/manage_red_hat_quay/index#georepl-intro

For more information about Quay tag expiration, refer to the *Tag expiration options* chapter in the *Configure Red Hat Quay* guide at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/configure_red_hat_quay/config-fields-intro#config-fields-tag-expiration

For more information about Quay image scanning, refer to the *Clair Security Scanning* chapter in the *Manage Red Quay* guide at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/manage_red_hat_quay/index#clair-intro2

For more information about Quay builds, refer to the *Automatically building Dockerfiles with Build workers* chapter in the *Use Red Quay* guide at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/use_red_hat_quay/index#build-support

For more information about Quay notifications, refer to the *Repository Notifications* chapter in the *Use Red Quay* guide at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/use_red_hat_quay/index#repository_notifications

For more information about Quay monitoring on OpenShift, refer to the *Console monitoring and alerting* chapter in the *Deploy Red Hat Quay on OpenShift with the Quay Operator* at
https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/deploy_red_hat_quay_on_openshift_with_the_quay_operator/quay_operator_features#operator-console-monitoring-alerting

► Quiz

Identifying the Components and Features of Red Hat Quay

Choose the correct answers to the following questions:

- ▶ 1. **Which two of the following problems can Red Hat Quay address? (Choose two.)**
 - a. Auditing and controlling changes to container images
 - b. Ensuring that workloads do not use images from public registries
 - c. Optimizing container image distribution
 - d. Deploying applications to multiple clusters declaratively
 - e. Detecting insecure configurations
- ▶ 2. **Which three of the following are differences between Quay and Red Hat Quay.io? (Choose three.)**
 - a. Quay.io is a hosted service, but you install Quay on one of the supported platforms.
 - b. Only Quay.io can use authorization providers such as LDAP.
 - c. Quay.io has more flexible logging and RBAC.
 - d. Only Quay can mirror other registries.
 - e. Quay can store images in your preferred object storage system.
- ▶ 3. **Which two of the following are components of Quay? (Choose two.)**
 - a. Redis
 - b. Central
 - c. Clair scanner
 - d. Klusterlet
 - e. The MultiClusterHub object
- ▶ 4. **Which three of the following components can the Quay operator deploy? (Choose three.)**
 - a. PostgreSQL
 - b. Redis
 - c. Object storage
 - d. Clair
 - e. Red Hat OpenShift Streams for Apache Kafka

► Solution

Identifying the Components and Features of Red Hat Quay

Choose the correct answers to the following questions:

- ▶ 1. **Which two of the following problems can Red Hat Quay address? (Choose two.)**
 - a. Auditing and controlling changes to container images
 - b. Ensuring that workloads do not use images from public registries
 - c. Optimizing container image distribution
 - d. Deploying applications to multiple clusters declaratively
 - e. Detecting insecure configurations

- ▶ 2. **Which three of the following are differences between Quay and Red Hat Quay.io? (Choose three.)**
 - a. Quay.io is a hosted service, but you install Quay on one of the supported platforms.
 - b. Only Quay.io can use authorization providers such as LDAP.
 - c. Quay.io has more flexible logging and RBAC.
 - d. Only Quay can mirror other registries.
 - e. Quay can store images in your preferred object storage system.

- ▶ 3. **Which two of the following are components of Quay? (Choose two.)**
 - a. Redis
 - b. Central
 - c. Clair scanner
 - d. Klusterlet
 - e. The MultiClusterHub object

- ▶ 4. **Which three of the following components can the Quay operator deploy? (Choose three.)**
 - a. PostgreSQL
 - b. Redis
 - c. Object storage
 - d. Clair
 - e. Red Hat OpenShift Streams for Apache Kafka

Deploying and Configuring Red Hat Quay

Objectives

- Install the Quay operator and deploy and configure a Quay registry using the command line.

Requirements for Installing Red Hat Quay on OpenShift

Red Hat Quay requires the following resources.

OpenShift Cluster

Quay requires a Red Hat OpenShift cluster, version 4.5 or later, and an account with permissions to install operators and create namespaces at the cluster scope.

Compute Resource Requirements

Each Red Hat Quay application pod has identical resource requirements for requests and limits, which are defined in the `quay-app` deployment object.

By default, the Quay operator scales the Quay application to 2 replicas. Thus, you need at least 4 CPUs and 16 GiB of RAM to install and deploy a single Quay registry. If you deploy more than one registry for your multicluster environment, then multiply these values for each registry.

The Quay operator activates the horizontal pod autoscaler (HPA) by default. The HPA can start more pods when the Quay load grows. Additional resources are required for the pods that the HPA creates.

Object Storage

Quay uses object storage to save container images. The Quay operator uses the `ObjectBucketClaim` Kubernetes API to obtain storage when running on OpenShift. The use of the `ObjectBucketClaim` Kubernetes API allows Quay to access a vendor neutral storage solution.

The following list shows the supported object storage on different clouds:

- Amazon S3
- Azure Blob Storage
- Google Cloud Storage
- Ceph Object Gateway (RADOS)
- OpenStack Swift
- CloudFront + S3

When you choose managed object storage, the Quay operator uses the Red Hat OpenShift Data Foundation operator to provide the `ObjectBucketClaim` Kubernetes API. Thus, if the storage for your clusters is managed by the Quay operator, then you must install the OpenShift Data Foundation operator.

There are two supported methods for the Quay operator to access to the object buckets from OpenShift Data Foundation:

Multicloud object gateway

The multicloud object gateway uses a Kubernetes `PersistentVolume` object to provision the `ObjectBucketClaim` API. The multicloud object gateway is included in the Red Hat OpenShift Plus subscription and with Red Hat Advanced Cluster Management for Kubernetes, Red Hat Advanced Cluster Security for Kubernetes, Red Hat Quay, and Red Hat OpenShift Data Foundation Essentials. The multicloud object gateway is a stand-alone deployment provided by Red Hat OpenShift Data Foundation Essentials, and does not provide high availability of the persistent bucket.

Highly-available storage infrastructure

If you have a full Red Hat OpenShift Data Foundation subscription, then you can use a production deployment of OpenShift Data Foundation. When using this option, image storage is highly available.

When the Quay operator manages the object storage, it creates storage buckets with a default size of 50 GiB. See the references section for a guide to resizing the default bucket claims.

The Red Hat Quay Operator

Red Hat recommends installing Red Hat Quay on OpenShift by using the Quay operator. The Quay operator automates the deployment of Quay registries, including mandatory Quay components, certificate management, and OpenShift resource configuration. If some of the components of Quay are already present in your infrastructure, for example, a highly available and redundant PostgreSQL database, then you can instruct the operator to use the external component instead of the managed one.

By default, the Quay operator creates and manages instances of all components to simplify operation. The following list shows those Quay components that can be excluded from operator management as part of a pre-existing infrastructure:

- Redis
- PostgreSQL database
- Object storage

The following list of components can be disabled to save resources, use alternative implementations, or to manage the component manually.

- Clair scanner
- Horizontal Pod Autoscaler
- OpenShift Monitoring
- Routes for access outside OpenShift
- TLS certificate management

The Quay operator version always matches the Quay version. Quay operator 3.6.4 installs Red Hat Quay 3.6.4.

The Quay operator provides the `QuayRegistry` custom resource definition (CRD).

The `QuayRegistry` object defines which components will be managed by the operator, and the configuration bundle that contains the resources needed for the installation and configuration of a registry. The following YAML file is a complete example of a `QuayRegistry` object:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: central 1
  namespace: registry 2
```

```

spec:
  components: ③
    - kind: clair
      managed: true
    - kind: horizontalpodautoscaler
      managed: true
    - kind: mirror
      managed: true
    - kind: postgres
      managed: true
    - kind: redis
      managed: true
    - kind: objectstorage
      managed: true
    - kind: route
      managed: true
    - kind: monitoring
      managed: true
    - kind: tls
      managed: true
  configBundleSecret: init-config-bundle-secret ④

```

- ① name: Required. The name of the registry.
- ② namespace: Required. The namespace containing the registry. The name and the namespace fields form the default DNS name of the Quay registry.
- ③ components: Required. Defines which components the Quay operator manages. By default, the operator manages all components. Refer to *Identifying the Components and Features of Red Hat Quay* for further details this components.
- ④ configBundleSecret: Optional. Define the Secret object in the same namespace that the registry. This secret can contain a config.yaml file to configure the Quay registry.

Quay Configuration Lifecycle on OpenShift

A Quay registry configuration is stored in a config.yaml file. In OpenShift, the config.yaml file is stored in a Secret object in the same namespace as the Quay registry.

The Quay Registry Configuration File

The config.yaml file contains all the configuration fields needed to run a Quay registry.

There are some configuration fields that are always required, although the Quay operator can start a Quay registry on OpenShift with an initial basic configuration for all the configuration fields. See the references section for the full reference about the config.yaml fields.

The following config.yaml example file shows options that are suggested for automation.

```

FEATURE_USER_INITIALIZE: true ①
BROWSER_API_CALLS_XHR_ONLY: false ②
SUPER_USERS: ③
  - quayadmin
FEATURE_USER_CREATION: false ④

```

- ❶ Enables the /api/v1/user/initialize Quay API endpoint to create the first user
- ❷ Allows access to the Quay registry API without using an X-Requested-With header. Thus, command-line and automation tools can use the API without a browser.
- ❸ List of users that become super users after first log in.
- ❹ Restrict the creation of users to super users. By default, users can register by themselves.

The following config.yaml example file shows a complete configuration with authentication and authorization from an LDAP server:

```
FEATURE_USER_INITIALIZE: true ❶
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: true
AUTHENTICATION_TYPE: LDAP ❷
LDAP_ADMIN_DN: uid=admin, cn=users, cn=accounts, dc=ocp4, dc=example, dc=com ❸
LDAP_ADMIN_PASSWD: Q@&fh3eR5
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
- cn=accounts
- dc=ocp4
- dc=example
- dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://myldap.ocp4.example.com
FEATURE_TEAM_SYNCING: true ❹
```

- ❶ First four options, suggested for automation.
- ❷ Required field to set the authentication type to LDAP. The field can be the default, Database, or LDAP, JWT, Keystone, or OIDC.
- ❸ Seven fields to set the configuration of the LDAP directory containing users and groups. See the references section for more information about the Quay LDAP configuration fields.
- ❹ Field that allows the mapping between Quay team members and groups in the authorization provider, LDAP, or Keystone.

To find the full list of fields, refer to the references.

The Configuration Bundle Secret

The configuration bundle secret is a Kubernetes Secret that must be present in the namespace where the QuayRegistry object resides. The QuayRegistry object references this secret in the configBundleSecret field.

The configuration bundle secret contains the config.yaml file and other resources, such as custom certificates for different secure connections.

Installing a Quay Registry on OpenShift

To run a Quay registry on OpenShift you must install the Quay operator. Then, use the configuration secret to create a `QuayRegistry` object.

Installing the Quay Operator from the Command Line

See the references section for a step-by-step guide to installing the Quay operator from the OpenShift web console.

You must create a `Subscription` object, which subscribes the Quay operator to one or all of the cluster namespaces. The subscription searches and uses the required software in the Red Hat OperatorHub.

The following YAML file defines a subscription to the Quay operator. Because the namespace is `openshift-operators`, this subscription installs the Quay operator cluster-wide.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: quay-operator
  namespace: openshift-operators
spec:
  sourceNamespace: openshift-marketplace
  source: redhat-operators
  channel: stable-3.6
  installPlanApproval: Automatic
  name: quay-operator
```

Red Hat recommends installing the operator cluster-wide, making it available in all namespaces. If you install the Quay operator in a specific namespace, then the OpenShift monitoring integration is disabled by default.

See the references section for a full explanation of the generic fields of a `Subscription` object.

Creating the Quay Registry on OpenShift

You can create the Quay registry on OpenShift by using the `config.yaml` file inside a configuration bundle secret and a `quay-registry.yaml` file like the one shown elsewhere in this lecture.

The following steps describe how to create the Quay registry on OpenShift:

- Create a namespace to deploy the registry

```
[user@demo ~]$ oc create namespace registry
```

- Create the initial configuration bundle secret using a `config.yaml` file.

```
[user@demo ~]$ oc create secret generic \
--from-file config.yaml=./config.yaml \
init-config-bundle-secret \
-n registry
```

- Create the registry

```
[user@demo ~]$ cat <<EOF | oc apply -f -
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: central
  namespace: registry
spec:
  configBundleSecret: init-config-bundle-secret
EOF
```

After creating the `QuayRegistry` object, the Quay operator creates all managed resources and starts a new Quay application to serve a full enterprise container images registry.

Modifying the Configuration of a Quay Registry on OpenShift

These are the two methods for modifying the configuration of a Quay registry deployed on OpenShift.

- Changing the `config.yaml` file referenced by the configuration bundle secret. This is the only method that allows you to change all the configuration fields of a Quay registry.
- Using the `Config Editor` application. Refer to the application's official documentation at https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/deploy_red_hat_quay_on_openshift_with_the_quay_operator/index#operator-config-ui

To change the `config.yaml` file, note that the Quay operator does not watch the configuration bundle secret to roll out a new deployment when it changes. Thus, to make a change you must create a new secret containing the new `config.yaml` file and all other resources. Then, modify the `QuayRegistry` object to point to the new configuration bundle secret. When you modify the `QuayRegistry` object, the Quay operator rolls out the new version of the Quay application with the new configuration.



References

For a step-by-step guide to installing the Quay operator from the OpenShift web console, refer to the *Installing the Quay Operator from OperatorHub* chapter in the *Deploy Red Hat Quay on OpenShift with the Quay Operator* guide at https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/deploy_red_hat_quay_on_openshift_with_the_quay_operator/index#installing_the_quay_operator_from_operatorhub

For more information about the generic fields of a `Subscription` object, refer to the *Adding Operators to a cluster* chapter in the OpenShift official documentation at <https://docs.openshift.com/container-platform/4.10/operators/admin/olm-adding-operators-to-cluster.html>

For more information about configuration of unmanaged storage, refer to the *Image Storage* chapter in the *Configure Red Hat Quay* guide at https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/configure_red_hat_quay/index#config-fields-storage

For more information on how to resize the default bucket claims, refer to the *Resizing Managed Storage* in the *Deploy Red Hat Quay on OpenShift with the Quay Operator* guide at https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/deploy_red_hat_quay_on_openshift_with_the_quay_operator/index#operator-resize-storage

For more information about the LDAP configuration fields for Quay, refer to the *LDAP configuration fields* in the *Configure Red Hat Quay* guide at https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/configure_red_hat_quay/index#config-fields-ldap

For more information about all the configuration fields in the `config.yaml` file, refer to the *Configuration fields* chapter in the *Configure Red Hat Quay* guide at https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/configure_red_hat_quay/index#config-fields-intro

► Guided Exercise

Deploying and Configuring Red Hat Quay

- Install the Red Hat Quay operator in the hub cluster. Then, you deploy and configure a Red Hat Quay registry by using the command line and preconfiguration files.

Outcomes

- Install the Red Hat Quay operator from OperatorHub.
- Configure a Quay registry before the deployment by using a preconfiguration file.
- Create a `QuayRegistry` object.
- Push an image to test the registry.

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 Quay is not installed, creates the `cloudadmin` IdM user, and gives administrator privileges to that user in all consoles.

```
[student@workstation ~]$ lab start quay-deploy
```

- 1. Using OperatorHub, install the Quay operator in the `ocp4.example.com` cluster.
- 1.1. From the `workstation` machine, navigate to the Red Hat OpenShift web console at <https://console-openshift-console.apps.ocp4.example.com>. When prompted, click **Red Hat Identity Management** and log in as the `cloudadmin` user with the `redhat` password.
 - 1.2. Install the Quay operator from OperatorHub.
Navigate to **Operators > OperatorHub** and type `quay` in the **Filter by keyword** field. Click **Red Hat Quay**, and then click **Install**.

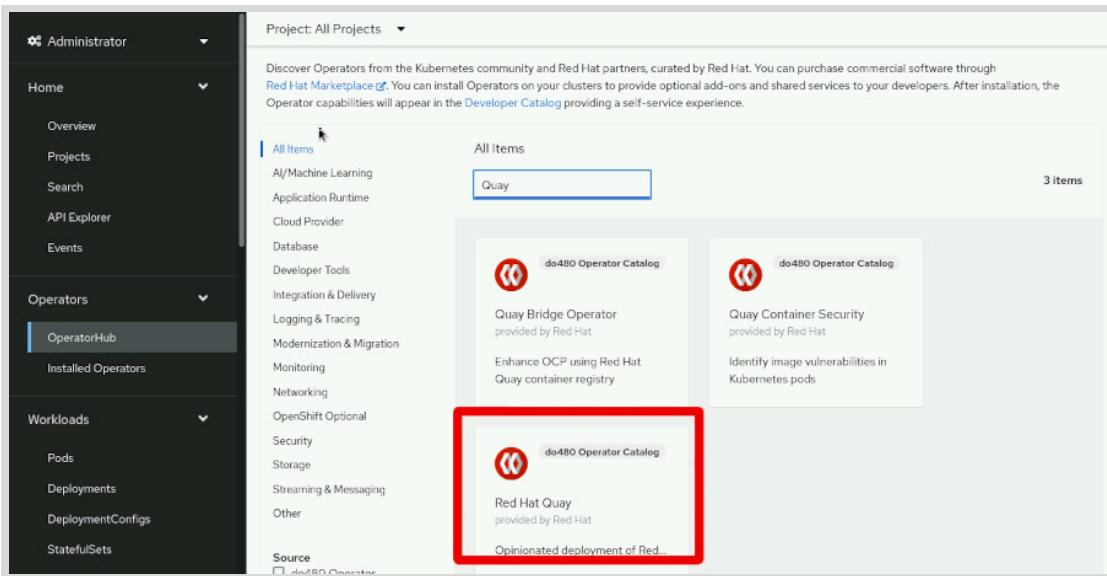
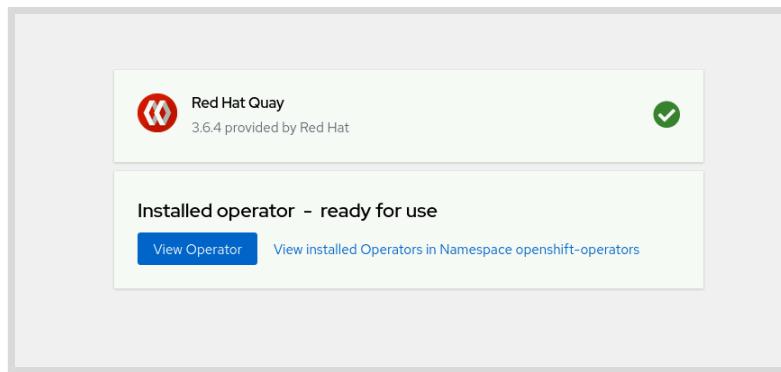


Figure 6.4: The three Red Hat Quay operators in OperatorHub

13. In the **Update Channel** section, ensure that the **stable-3.6** radio button is selected. In the **Update approval** section, ensure that the **Automatic** radio button is selected. Then, click **Install**.

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



- 2. Create the preconfiguration file and the initial secret for the Quay registry configuration.

- 2.1. Open a terminal and log in to the ocp4 hub cluster as the **cloudadmin** user with the **redhat** password. The API server address is <https://api.ocp4.example.com:6443>.

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

- 2.2. From the terminal, change to the **~/D0480/labs/quay-deploy** folder.

```
[student@workstation ~]$ cd D0480/labs/quay-deploy
[student@workstation quay-deploy]$
```

- 2.3. Review the `config.yaml` file, which contains the preconfiguration of the Quay registry. The Quay operator uses this information when creating the `QuayRegistry` object to define the features of the registry.

```
AUTHENTICATION_TYPE: LDAP
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- cloudadmin
FEATURE_USER_CREATION: true
LDAP_ADMIN_DN: uid=admin,cn=users,cn=accounts,dc=ocp4,dc=example,dc=com
LDAP_ADMIN_PASSWD: Redhat123@!
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
- cn=accounts
- dc=ocp4
- dc=example
- dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldaps://idm.ocp4.example.com
LDAP_USER_RDN:
- cn=users
FEATURE_TEAM_SYNCING: true
TEAM_RESYNC_STALE_TIME: 60m
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: true
```

This file configures a Quay registry for the Quay operator that uses the following features:

- Authentication against the LDAP provider in the classroom
- The recommended configuration for automation of Quay on OpenShift
- The `cloudadmin` superuser for Quay

2.4. Create the `registry` namespace.

```
[student@workstation quay-deploy]$ oc create namespace registry
namespace/registry created
```

- 2.5. Create the `init-config-bundle-secret` object using the configuration files that you created in a previous step. You need a secret containing the `config.yaml` Quay configuration file and the `ldap.crt` file to connect to the LDAP server through a secure channel.

```
[student@workstation quay-deploy]$ oc create secret generic \
--from-file config.yaml=./config.yaml \
--from-file ldap.crt=./ldap.crt init-config-bundle-secret \
-n registry
secret/init-config-bundle-secret created
```

- 3. Create the `QuayRegistry` object.

- 3.1. Review the `quay-registry.yaml` file, which contains the configuration of the `QuayRegistry` object.

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: central
  namespace: registry
spec:
  configBundleSecret: init-config-bundle-secret
  components:
    - kind: clair
      managed: false
    - kind: horizontalpodautoscaler
      managed: false
    - kind: mirror
      managed: false
```

This file configures the `QuayRegistry` object to deploy on OpenShift with the `central` name in the `registry` namespace. Thus, the Quay operator generates the `https://central-quay-registry.apps.ocp4.example.com` OpenShift route to access the registry. To save compute resources, the `components` key in the YAML file disables the Clair scanner, horizontal pod autoscaler, and registry mirror.

- 3.2. Create the Quay registry using the `quay-registry.yaml` file.

```
[student@workstation quay-deploy]$ oc apply -f quay-registry.yaml
quayregistry.quay.redhat.com/central created
```

- 3.3. Wait until the Quay registry application pods are in the Running state.

```
[student@workstation quay-deploy]$ oc get pods -n registry| grep central-quay-app
| grep Running
central-quay-app-555844bbbd-77gxr           1/1     Running   0        44m
central-quay-app-555844bbbd-g7chz           1/1     Running   0        44m
```

► 4. Test the new registry by pushing an image to a private user repository.

- 4.1. Create a `Containerfile` file containing a minimal image definition.

```
[student@workstation quay-deploy]$ vi Containerfile
FROM quay.io/podman/hello
```

- 4.2. Build the image using the `podman build` command.

```
[student@workstation quay-deploy]$ podman build . -t hello-to-quay:dev
...output omitted...
Successfully tagged localhost/hello-to-quay:dev
...output omitted...
```

- 4.3. Log in as the `cloudadmin` user with the `redhat` password.

**Warning**

This exercise uses a plain text password for brevity. Commands such as podman store passwords in the file system unencrypted. If a malicious actor obtains access to the file system, then the actor can impersonate your account.

In real-world situations, always use encrypted passwords as described elsewhere in this course.

```
[student@workstation quay-deploy]$ podman login -u=cloudadmin -p=redhat \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
...output omitted...
```

- 4.4. Push the image to the `cloudadmin` user's private repository.

```
[student@workstation quay-deploy]$ podman push localhost/hello-to-quay:dev \
central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-to-quay:dev
...output omitted...
Storing signatures
```

- 4.5. From the `workstation` machine, navigate to the Red Hat Quay console at `https://central-quay-registry.apps.ocp4.example.com`.
Log in as the `cloudadmin` user with the `redhat` password.
If prompted, click **Confirm Username**.
The Quay console shows the `cloudadmin/hello-to-quay` private repository, which contains the image that you pushed in the previous step.
The Quay registry is fully operative in the OpenShift hub cluster.

Finish

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

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

This concludes the section.

Describing the Red Hat Quay Repository Model and RBAC

Objectives

- Describe the Red Hat Quay concepts for image registry organization, and manage role-based access control (RBAC) by using LDAP

The Red Hat Quay Tenancy Model

Companies often need to control the origin of software running their workloads. If malicious actors can modify the code running a workload, then they can cause significant damage.

Using an image registry can help control the code when running containerized workloads. If deployments only use container images from a registry, then registry access permissions control who can deploy workloads.

Red Hat Quay has a tenancy model for managing access to container images in the registry. The Quay model uses the following concepts:

Image repository

An image repository contains tagged container images.

For example, the quay repository in the Red Hat Quay.io public registry contains container images for each release of the upstream Project Quay. Each image has a tag to identify its release.

Image repositories usually contain tags for different versions of the same software.

By default, new image repositories are private.

Organization

An organization contains image repositories and teams.

For example, the projectquay organization in the Quay.io public registry contains image repositories related to Quay.

Organizations are closely related to users. Many organization features are available to users. For example, both organizations and users can own image repositories. Organizations and users share the same namespace, so you cannot create an organization and a user with the same name.

Team

Organization administrators can assign roles to groups of users by using teams. Quay teams are similar to Kubernetes groups, but by default, there is no correspondence between Quay teams and Kubernetes groups.

Administrators can grant permissions over image repositories and general roles to the team.

Teams with the member role only have the permissions granted by administrators over repositories. Teams with the creator role can create repositories. Teams with the administrator role have full access to the organization.

For example, an organization has the following teams and permissions:

- The development team has write permissions to the development image repositories.
- The administrators team has administrator permissions to the production image repositories.

With these permissions, developers can deploy changes to development environments, but only administrators can deploy changes to production environments.

User

Quay users are similar to Kubernetes users, but Quay has its own authentication back end.

Quay users also have a home organization. Users can create image repositories in their home organizations.

The Quay configuration can designate some users as super users. Super users can access some administrative features by using a separate part of the web console and some protected API calls. Administrative features include viewing usage logs, managing ownership of organizations, and managing service keys to authenticate external services.

The Quay tenancy model is a form of role-based access control (RBAC). A user's group membership determines their roles in an organization and roles control permissions.

Managing Organizations Using the Web Console

Quay users can create and manage organizations by using the web console.

Create an organization by clicking **Create New Organization** on the **Repositories** page.

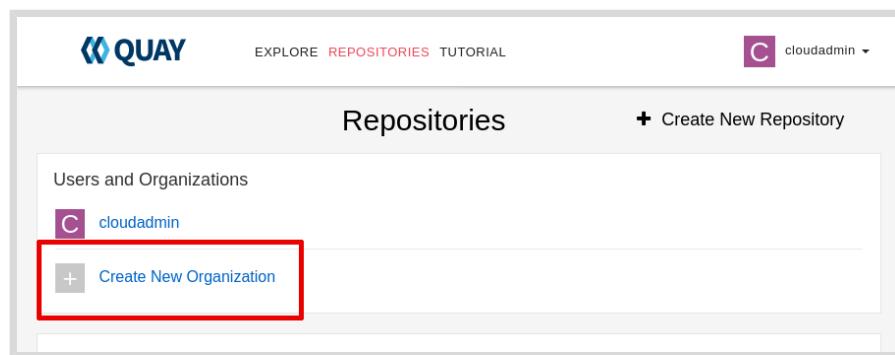
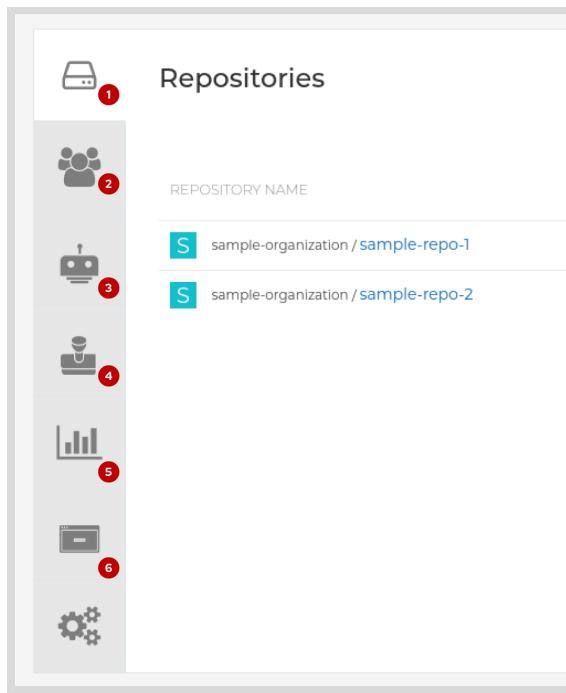


Figure 6.6: Creating organizations using the web console

Organizations automatically contain an owner's group. This group contains the organization creator and has administrator permissions over the organization.

On the organization page, you can find tabs to access different features.



Accessing different organization features.

- ① Click **Repositories** to create repositories and manage repository permissions.
- ② Click **Teams and Membership** to manage team membership and permissions, and to create teams.
- ③ Click **Robot Accounts** to manage robot accounts.
- ④ Click **Default Permissions** to manage default permissions.
- ⑤ Click **Usage Logs** to review actions performed by users, tokens, and applications.
- ⑥ Click **Applications** to create applications and generate API tokens.

After clicking **Repositories**, click **Settings** to manage repository permissions.

User and Robot Permissions

TEAM PERMISSIONS
team Write

USER PERMISSIONS
alice Admin

Select a team or user... Read Add Permission

Events and Notifications
No notifications have been setup for this repository.
Click the "Create Notification" button above to add a new notification for a repository eve

Repository Visibility
This Repository is currently **private**. Only users on the permissions list may view and interact with it.
Make Public

Managing repository settings.

- ① Use the **User and Robot Permissions** section to view and change permissions for teams.
- ② Use the **Repository Visibility** section to toggle visibility of the repository.

From the organization page, click **Teams and Membership** to manage teams.

TEAM NAME	MEMBERS	REPOSITORIES	TEAM ROLE
owners	1 member	No repositories	Admin
team1	1 member	2 repositories	Member
team2	0 members	1 repository	Member

Managing teams.

- ① Click **Create New Team** to create a new team.
- ② Click the team name under the **TEAM NAME** column, the member number under the **MEMBERS** column, or the **Manage Team Members** option in the **Options** gear box icon to manage team membership.

- ③ Select a role from the list in the TEAM ROLE column to assign an organization role to the team.
- ④ Click Set Repository Permissions from the Options gear box icon to manage repository permissions. Click Delete Team from the Options gear box icon to delete a team.

Managing Organizations via the API

Organization administrators can also use the Quay API to manage organizations.

Administrators can create OAuth applications and generate tokens to access the API. HTTP requests to the Quay API that use a token act on behalf of the token creator.

To create a token, click **Applications** on the organization page, then click **Create New Application** and type a name for the application.

On the application page, click **Generate Token** to generate a token.

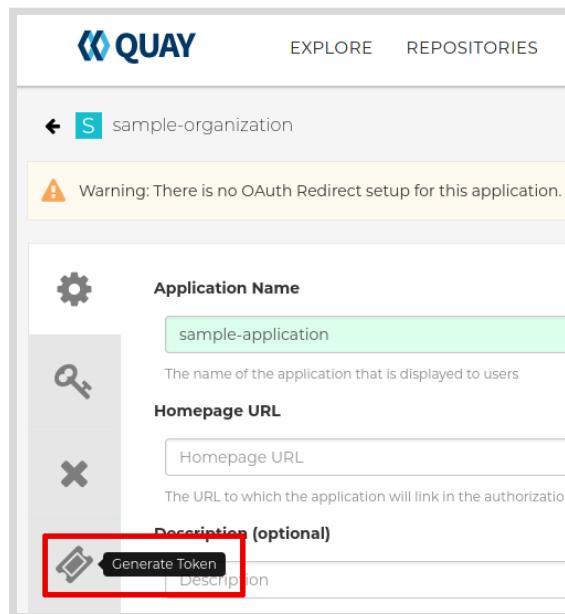


Figure 6.7: Creating an application token.

After clicking **Generate Token**, a page listing available permissions is displayed. Choose the permissions you need and click **Generate Access Token**.

If you chose permissions that are potentially dangerous, a confirmation page shows additional warnings. Click **Authorize Application**. A page displays the token. After this page is closed, you cannot access the token. Save the token to use with the API.



Warning

If a malicious actor obtains access to the token, they can act on your behalf.

Only grant the minimum permissions you require. Store the token as confidential information. Review usage logs to detect unusual activity.

To revoke a token, remove the application. You can also recreate applications periodically to reduce the period a leaked token is usable.

Note that by default, the Quay API is public. Many API calls work without providing a token or when you use a token incorrectly, but the data returned is incomplete public data. Ensure that you log in correctly to see protected data.

To authenticate a call to the Quay API, make an HTTP request with an `Authentication` header. The value of the authentication header is `Bearer token`.

The following command shows how to execute an API call by using the `curl` command.

```
[user@demo ]$ $ curl -k -H 'Authorization: Bearer token' \
  https://quay.example.com/api/v1/users/user
{"anonymous": false, "username": "alice", "avatar": {"name": "alice", "hash": "9d...2e", "color": "#ad494a", "kind": "user"}}
```

Only use the `-k` option if the Quay certificate is not trusted by your workstation.

Creating Organizations Using the API

To create an organization by using the API, you need a token with administer user permissions.

Use the `POST` method and provide a payload in JSON format for API calls that require data, such as creating an organization. Specify the `Content-Type` header with the `application/json` value.

Use the following command to create an organization.

```
[user@demo ]$ curl -k -X POST -H 'Authorization: Bearer token' \
  -H 'Content-Type: application/json' \
  -d @- <<EOF \
  https://quay.example.com/api/v1/organization/
{
  "name": "name",
  "email": "email"
}
EOF
"Created"
```

When creating organizations by using the API, you must provide a unique organization email.

The previous command uses the `-d @-` option to read the payload from standard input and a here document to provide the payload in standard input. This change allows the JSON payload to contain new lines for readability. You can modify the command to provide the payload without using a here document.

```
[user@demo ]$ curl ... -d {"name":...} \
  https://quay.example.com/api/v1/organization/
"Created"
```

Creating an Image Repository Using the API

To create an image repository by using the API, you need a token with the create repositories permission.

To create repositories, post a payload with the following structure to the `/api/v1/repository` endpoint.

```
{
  "namespace": "namespace", ①
  "repository": "repository_name",
  "description": "description",
  "visibility": "visibility" ②
}
```

- ①** The `namespace` value is the name of the organization or user that contains the repository.
- ②** The `visibility` value is either `public` or `private`.

Managing Teams Using the API

To manage teams by using the API, you need a token with the administer organization permission.

To create teams, make an HTTP PUT request with a payload with the following structure to the `/api/v1/organization/organization/team/team_name` endpoint.

```
{
  "name": "name",
  "role": "role", ①
  "description": "description"
}
```

- ①** The `role` has a `member`, `creator`, or `admin` value.

To grant teams permissions over repositories, make an HTTP PUT request with a payload with the following structure to the `https://quay.example.com/api/v1/repository/organization/repository/permissions/team/` endpoint.

```
{"role": "role"}
```

The `role` has a `read`, `write`, or `admin` value.

Creating Users Using the API

To manage users by using the API, you need a token with the super user access permission.

To create users, make an HTTP POST request with a payload with the following structure to the `/api/v1/user/initialize` endpoint.

```
{
  "username": "username",
  "password": "password", ①
  "email": "email",
  "access_token": true ②
}
```

- ①** The password must be at least 8 characters and contain no white space.

- ② If true, then the response contains a token for the user.

Synchronizing Team Membership Using LDAP

If you configure Quay to authenticate by using LDAP, then Quay can also synchronize team membership with LDAP groups. The FEATURE_TEAM_SYNCING configuration field is true by default. With this configuration, superusers can configure any team to synchronize with an LDAP group.

The FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP is false by default. If you change the value to true, then other users can configure team synchronization.

For example, if a user moves to a different department, then team membership in Quay might need to be updated. With team synchronization, administrators update the LDAP server and Quay team membership replicates the changes automatically.

To configure synchronization for a team, go to the team page and click **Enable Directory Synchronization**.

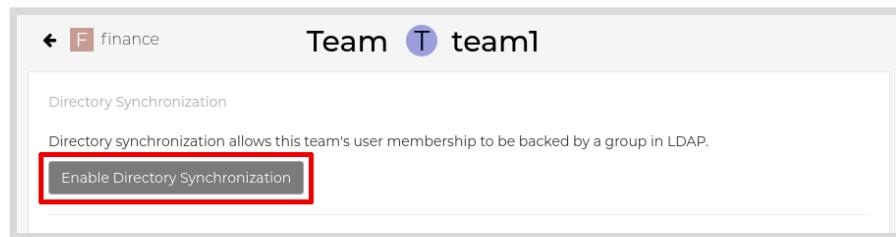


Figure 6.8: Configuring team synchronization.

Type the distinguished name of the group, relative to the base DN for LDAP authentication. For example, cn=deployers, cn=groups.

To configure synchronization by using the API, post a payload with the following structure to the /api/v1/organization/organization/team/team syncing endpoint.

```
{"group_dn": "group_dn"}
```

The group_dn value is the distinguished name of the group relative to the base DN for LDAP authentication.



References

For more information, refer to the *Users and organizations in Red Hat Quay* chapter in the Red Hat Quay 3.6 Use Red Hat Quay documentation at

https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/use_red_hat_quay/index#user-org-intro

For more information, refer to the *LDAP Authentication Setup for Red Hat Quay* chapter in the Red Hat Quay 3.6 Use Red Hat Quay documentation at

https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/manage_red_hat_quay/index#ldap-authentication-setup-for-quay-enterprise

► Guided Exercise

Describing the Red Hat Quay Repository Model and RBAC

- Create an organization in Red Hat Quay. Then, you create teams and users synchronized from Identity Management (IdM) in Red Hat Enterprise Linux. Finally, you push images and move images in the registry as different users with limited permissions.

Outcomes

- Grant users controlled privileges.
- Interact with Quay by using command-line tools.
- Use encrypted passwords.

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 Quay is in a valid state and creates IdM users.

```
[student@workstation ~]$ lab start quay-rbac
```

► 1. Log in to the Quay web console.

- 1.1. From the `workstation` machine, navigate to the Quay web console at `https://central-quay-registry.apps.ocp4.example.com`. Log in as the `cloudadmin` user with the `redhat` password.

If prompted, then click **Confirm Username**.

► 2. Create the `finance` organization.

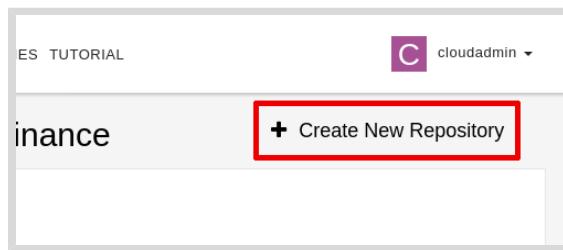
- 2.1. Click **Create New Organization**.

The screenshot shows the Red Hat Quay web interface. At the top, there is a navigation bar with the Quay logo, links for 'EXPLORE', 'REPOSITORIES', and 'TUTORIAL', and a user dropdown for 'cloudadmin'. Below the navigation is a header titled 'Repositories' with a 'Create New Repository' button. A sidebar on the left is titled 'Users and Organizations' and shows a list with one item: 'cloudadmin'. At the bottom of this sidebar is a button labeled '+ Create New Organization' with a plus sign icon, which is highlighted with a red rectangular box.

2.2. Type **finance** as the organization name, then click **Create Organization**.

► 3. Create the **budget-app-dev** repository in the **finance** organization.

3.1. Click **Create New Repository**.



3.2. Type **budget-app-dev** as the repository name, then click **Create Private Repository**.

► 4. Create the **budget-app** repository in the **finance** organization.

4.1. Click **REPOSITORIES**, then click **finance** to navigate to the **finance** organization.

4.2. Click **Create New Repository**, type **budget-app** as the repository name, and then click **Create Private Repository**.

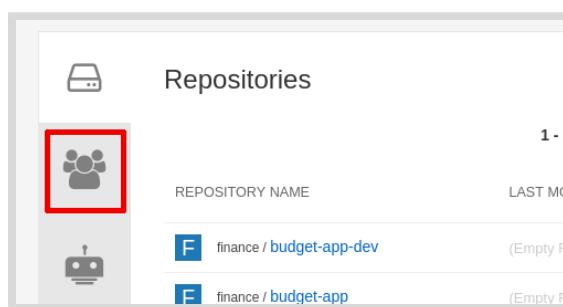
► 5. Create the **deployers** team in the **finance** organization.

The **deployers** team has read permissions on the **budget-app-dev** repository and administrator permissions on the **budget-app** repository.

The **deployers** team replicates the IdM group with the distinguished name **cn=deployers, cn=groups** relative to **cn=accounts, dc=ocp4, dc=example, dc=com**.

5.1. Click **REPOSITORIES**, then click **finance** to navigate to the **finance** organization.

5.2. Click **Teams and Membership**.



5.3. Click **Create New Team**, type **deployers** as the team name, and then click **Create team**.

5.4. Select the **Read** permission for the **budget-app-dev** repository. Select the **Admin** permission for the **budget-app** repository.

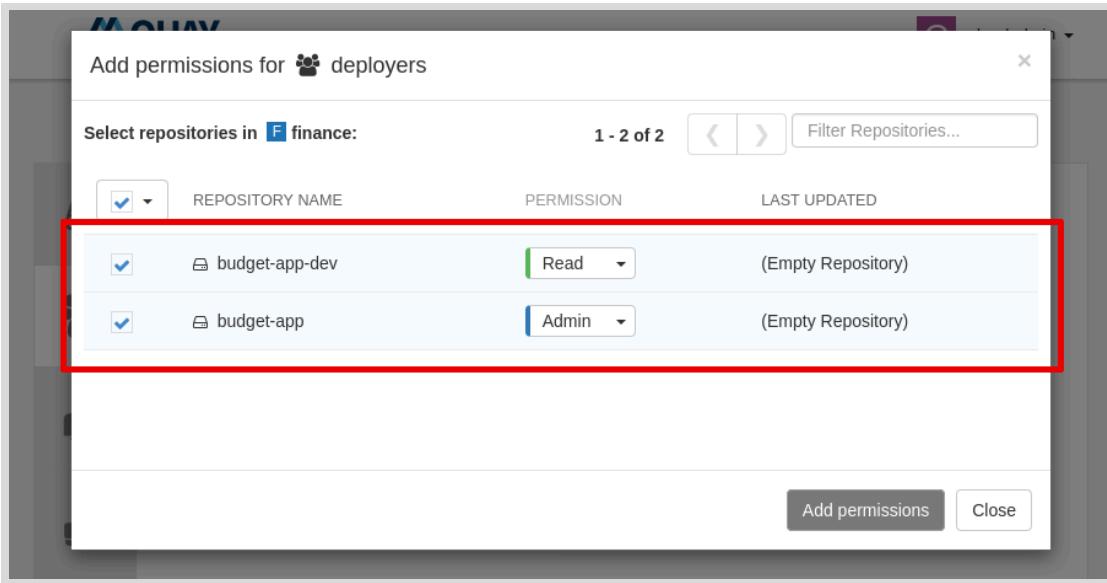


Figure 6.12: The permissions for the deployers team

Click Add permissions.

- 5.5. Click **deployers** in the team list to navigate to the team configuration page. Click **Enable Directory Synchronization**, then type `cn=deployers,cn=groups` as the distinguished name, then click **Enable Group Sync**.
- ▶ 6. Create the **developers** team in the **finance** organization.
The **developers** team has admin permissions on the **budget - app - dev** repository.
The **developers** team replicates the IdM group with the distinguished name `cn=developers,cn=groups` relative to `cn=accounts,dc=ocp4,dc=example,dc=com`.
 - 6.1. Click **REPOSITORIES**, then click **finance** to navigate to the **finance** organization.
 - 6.2. Click **Teams and Membership**.
 - 6.3. Click **Create New Team**, then type **developers** as the team name, then click **Create team**.
 - 6.4. Select the **Admin** permission for the **budget - app - dev** repository.

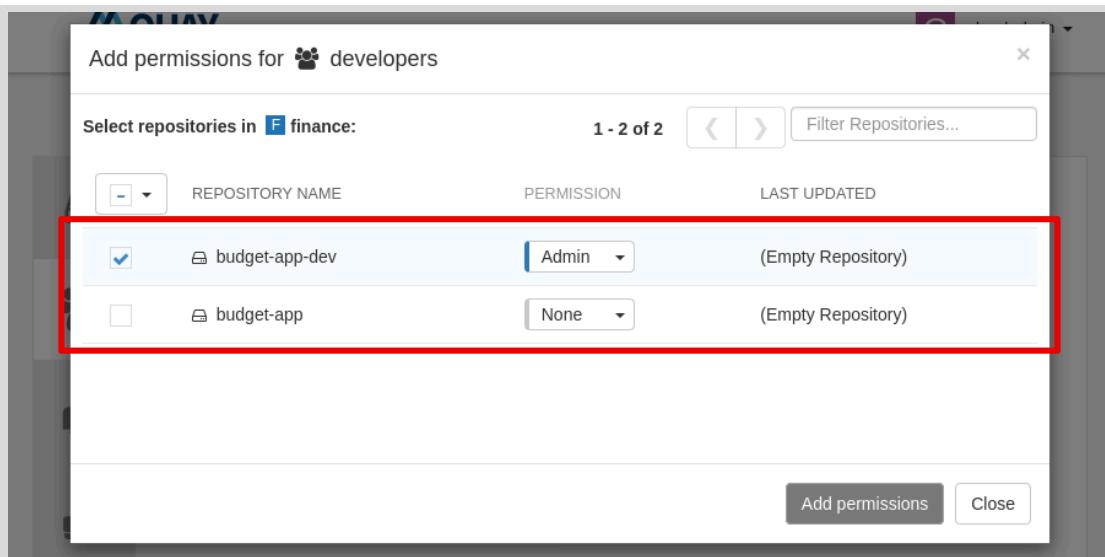
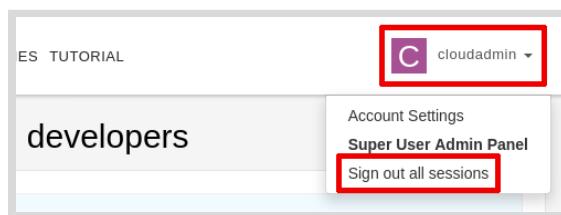


Figure 6.13: The permissions for the developers team

Click Add permissions.

- 6.5. Click **developers** in the team list to navigate to the team configuration page. Click **Enable Directory Synchronization**, type `cn=developers, cn=groups` as the distinguished name, and then click **Enable Group Sync**.
- ▶ 7. As the **bob** user, a member of the **developers** team, generate an encrypted password to use command-line tools.
 - 7.1. Sign out the **clouadmin** user. Click **clouadmin**, then click **Sign out all sessions**.



- 7.2. Log in as the **bob** user with the **redhat** password.
If prompted, click **Confirm Username**.
- 7.3. Click **bob**, then **Account Settings**.



- 7.4. Click **Generate Encrypted Password**.

B bob

Docker CLI Password

The Docker CLI stores passwords entered on the command line in **plaintext**. It is therefore highly recommended to generate an encrypted version of your password to use for `docker login`.

CLI Password: [Generate Encrypted Password](#)

- 7.5. Type the `redhat` password and click **Verify**.
- 7.6. Copy the password and store it in a temporary location to use in a later step. The password is a string of letters, numbers, and symbols.

Credentials for bob

Username & Encrypted Password:

bob	
0pLdLSuvV3hlWJfq8WuCA3BgC8wduu5o7XajF0LD8FDS4l6zWNvCPB Vc	

- 8. Create an image by using Podman.
- 8.1. Create a `Containerfile` file containing a minimal image definition. The result should look like the following file.

```
[student@workstation ~]$ vi Containerfile
FROM quay.io/podman/hello
```

- 8.2. Build the image by using the `podman build` command.
- ```
[student@workstation ~]$ podman build . -t budget-app:development
...output omitted...
Successfully tagged localhost/budget-app:development
...output omitted...
```
- 9. Push the image to Quay.

- 9.1. Log in as the **bob** user with the password from a previous step.

```
[student@workstation ~]$ podman login -u=bob -p=password \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
...output omitted...
```

9.2. Push the image to the budget-app-dev repository.

```
[student@workstation ~]$ podman push localhost/budget-app:development \
central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev
...output omitted...
Storing signatures
```

9.3. Try pushing the image to the budget-app repository.

```
[student@workstation ~]$ podman push localhost/budget-app:development \
central-quay-registry.apps.ocp4.example.com/finance/budget-app
...output omitted...
Error: trying to reuse blob sha256:... at destination: checking whether a blob
sha256:... exists in central-quay-registry.apps.ocp4.example.com/finance/budget-
app: unauthorized: authentication required
```

Quay rejects the image because the bob user is not allowed to push to the budget-app repository.

► 10. As the alice user, promote the image to the budget-app repository by using the skopeo command.

10.1. Log in as the alice user with the redhat password.



### Warning

This exercise uses a plain text password for brevity. Commands such as skopeo store passwords in the file system unencrypted. If a malicious actor obtains access to the file system, then the actor can impersonate your account.

In real-world situations, always use encrypted passwords as described in a previous step.

```
[student@workstation ~]$ skopeo login -u=alice -p=redhat \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
...output omitted...
```

10.2. Promote the image to the budget-app repository by using the skopeo command.

```
[student@workstation ~]$ skopeo copy \
docker://central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev \
docker://central-quay-registry.apps.ocp4.example.com/finance/budget-app
...output omitted...
Storing signatures
```

► 11. Review the repositories.

- 11.1. Navigate to the Quay web console at <https://central-quay-registry.apps.ocp4.example.com>. If you are not logged in as the **bob** user, then log in as the **bob** user with the **redhat** password.

Note that you can only see the **budget-app-dev** repository.

| REPOSITORY NAME            | LAST MODIFIED    |
|----------------------------|------------------|
| F finance / budget-app-dev | Today at 6:11 AM |

Figure 6.18: Repositories visible to the bob user

- 11.2. Click **bob**, then click **Sign out all sessions**.
- 11.3. Log in as the **alice** user with the **redhat** password.  
If prompted, click **Confirm Username**.
- 11.4. Note that you can see both the **budget-app-dev** and **budget-app** repositories.

| REPOSITORY NAME            | LAST MODIFIED    |
|----------------------------|------------------|
| F finance / budget-app-dev | Today at 6:11 AM |
| F finance / budget-app     | Today at 6:11 AM |

Figure 6.19: Repositories visible to the alice user

## Finish

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

```
[student@workstation ~]$ lab finish quay-rbac
```

This concludes the section.

## ▶ Lab

# Installing and Configuring Red Hat Quay

- Use the command line to install Red Quay and push a container image by using a robot account.

## Outcomes

- Install Quay by using the command line.
- Use the Quay API.
- Use robot accounts.

## 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 Quay is not present in the hub cluster.

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

1. Log in to the ocp4 cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.

2. Install the Quay operator cluster-wide. Use the `do480-catalog` offline catalog. The `lab` script includes a `D0480/solutions/quay-review/subscription.yaml` solution file.

3. Create a registry namespace.

4. Create an `init-config-bundle-secret` object in the `registry` namespace with a `config.yaml` key containing the Quay configuration.

Use the suggested options for automation.

- Enable the `/api/v1/user/initialize` Quay API endpoint to create the first user.
- Allow access to the Quay registry API without the use of an `X-Requested-With` header.
- Define a `quayadmin` super user.

The `lab` script includes a `D0480/solutions/quay-review/config.yaml` solution file.

5. Create a `QuayRegistry` object named `central` in the `registry` namespace. The `QuayRegistry` object references the `init-config-bundle-secret` object containing the Quay configuration.

Do not include Clair, the horizontal pod autoscaler, or the mirror feature.

The `lab` script includes a `D0480/solutions/quay-review/quay-registry.yaml` solution file.

6. Create the `quayadmin` user and obtain an access token. The password must be at least 8 characters and contain no white space. The `lab` script includes a `D0480/solutions/quay-review/quayadmin-user.json` solution file.

Store the token in a `/home/student/quayadmin_token` file. This step is necessary so that the lab script can use the API to grade your work.

7. Create the `finance` organization by using the API. The lab script includes a `D0480/solutions/quay-review/finance-organization.json` solution file.
8. Create the `budget-app-dev` image repository by using the API. The lab script includes a `D0480/solutions/quay-review/budget-app-dev-repository.json` solution file.
9. Create the `deployer` robot account by using the API. The lab script includes a `D0480/solutions/quay-review/deployer-robot.json` solution file.
10. Grant permissions to the `deployer` robot account over the `budget-app-dev` image repository.
11. Create a `budget-app:development` image by using the `podman` command. You can create an image based on the `quay.io/podman/hello` image.
12. As the `deployer` robot account, push the image to the `budget-app-dev` image repository in the `finance` organization.

## 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 quay-review
```

## Finish

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

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

This concludes the section.

## ► Solution

# Installing and Configuring Red Hat Quay

- Use the command line to install Red Quay and push a container image by using a robot account.

## Outcomes

- Install Quay by using the command line.
- Use the Quay API.
- Use robot accounts.

## 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 Quay is not present in the hub cluster.

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

1. Log in to the ocp4 cluster as the `admin` user with the `redhat` password. The API server address is <https://api.ocp4.example.com:6443>.

- 1.1. Log in to the ocp4 cluster.

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

2. Install the Quay operator cluster-wide. Use the `do480-catalog` offline catalog. The `lab` script includes a `D0480/solutions/quay-review/subscription.yaml` solution file.

- 2.1. Create a `subscription.yaml` file with the following content.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
 name: quay-operator
 namespace: openshift-operators
spec:
 sourceNamespace: openshift-marketplace
 source: do480-catalog
 channel: stable-3.6
 installPlanApproval: Automatic
 name: quay-operator
```

- 2.2. Create the subscription by using the `oc create` command.

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

- 2.3. Use the `watch` command to verify that the `quay-operator.v3.6.4` deployment is available.

```
[student@workstation ~]$ watch oc get deployment -n openshift-operators
NAME READY UP-TO-DATE AVAILABLE AGE
quay-operator.v3.6.4 1/1 1 1 22s
```

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

- 3.** Create a registry namespace.

- 3.1. Create the namespace by using the `oc create` command.

```
[student@workstation ~]$ oc create namespace registry
namespace/registry created
```

- 4.** Create an `init-config-bundle-secret` object in the `registry` namespace with a `config.yaml` key containing the Quay configuration.

Use the suggested options for automation.

- Enable the `/api/v1/user/initialize` Quay API endpoint to create the first user.
- Allow access to the Quay registry API without the use of an `X-Requested-With` header.
- Define a `quayadmin` super user.

The lab script includes a `D0480/solutions/quay-review/config.yaml` solution file.

- 4.1. Create a `config.yaml` file with the following content.

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
```

- 4.2. Create the secret by using the `oc create` command.

```
[student@workstation ~]$ oc create secret generic \
--from-file config.yaml=./config.yaml init-config-bundle-secret -n registry
secret/init-config-bundle-secret created
```

- 5.** Create a `QuayRegistry` object named `central` in the `registry` namespace. The `QuayRegistry` object references the `init-config-bundle-secret` object containing the Quay configuration.

Do not include Clair, the horizontal pod autoscaler, or the mirror feature.

The lab script includes a `D0480/solutions/quay-review/quay-registry.yaml` solution file.

- 5.1. Create a `quay-registry.yaml` file with the following content.

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
 name: central
 namespace: registry
spec:
 configBundleSecret: init-config-bundle-secret
 components:
 - kind: clair
 managed: false
 - kind: horizontalpodautoscaler
 managed: false
 - kind: mirror
 managed: false
```

- 5.2. Create the registry by using the `oc create` command.

```
[student@workstation ~]$ oc create -f quay-registry.yaml
quayregistry.quay.redhat.com/central created
```

- 5.3. Use the `watch` command to verify that the registry deployments are available. The `central-quay-app` deployment can take a few minutes to become available.

```
[student@workstation ~]$ watch oc get deployment -n registry
NAME READY UP-TO-DATE AVAILABLE AGE
central-quay-app 2/2 2 2 4m26s
central-quay-config-editor 1/1 1 1 4m26s
central-quay-database 1/1 1 1 4m26s
central-quay-redis 1/1 1 1 4m26s
```

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

6. Create the `quayadmin` user and obtain an access token. The password must be at least 8 characters and contain no white space. The lab script includes a `D0480/solutions/quay-review/quayadmin-user.json` solution file.

Store the token in a `/home/student/quayadmin_token` file. This step is necessary so that the lab script can use the API to grade your work.

- 6.1. Create a `quayadmin-user.json` file with the `quayadmin` user definition.

```
{
 "username": "quayadmin",
 "password": "redhat123",
 "email": "quayadmin@example.com",
 "access_token": true
}
```

- 6.2. Use the `curl` command to post the user definition in JSON format to the initialize user endpoint. Note the returned access token for a later step.

```
[student@workstation ~]$ curl -X POST -k \
 https://central-quay-registry.apps.ocp4.example.com/api/v1/user/initialize \
 --header "Content-Type: application/json" \
 --data @quayadmin-user.json
{"access_token":"0ITUIV9KMVJ0RPMWV5PT3D8ETWE82GBIA0U8NA6X",
 "email":"quayadmin@example.com",
 "encrypted_password":"Yv0cxZBtrRkmkEhqcu3DhUTkXoTwWmA/
 Cdp4lr9W7ySiE99nQVI8VnECRmwoYItz",
 "username":"quayadmin"}
```

- 6.3. Store the new token in a /home/student/quayadmin\_token file. This step is necessary so that the lab script can use the API to grade your work.

```
[student@workstation ~]$ echo 0ITUIV9KMVJ0RPMWV5PT3D8ETWE82GBIA0U8NA6X \
>quayadmin_token
```

- 6.4. (Optional) Store the token in an environment variable. If you store the token, then you can use the QUAYADMIN\_TOKEN variable in the following commands instead of copying and pasting the token.

```
[student@workstation ~]$ QUAYADMIN_TOKEN=0ITUIV9KMVJ0RPMWV5PT3D8ETWE82GBIA0U8NA6X
```

7. Create the finance organization by using the API. The lab script includes a DO480/solutions/quay-review/finance-organization.json solution file.

- 7.1. Create a finance-organization.json file with the finance organization definition.

```
{
 "name": "finance",
 "email": "finance@example.com"
}
```

- 7.2. Use the curl command to post the organization definition in JSON format. Replace the token for the quayadmin user with the token that you obtained in a previous step.

```
[student@workstation ~]$ curl -X POST -k \
 https://central-quay-registry.apps.ocp4.example.com/api/v1/organization/ \
 --header "Content-Type: application/json" \
 --header "Authorization: Bearer 0ITUIV9KMVJ0RPMWV5PT3D8ETWE82GBIA0U8NA6X" \
 --data @finance-organization.json
"Created"
```

8. Create the budget-app-dev image repository by using the API. The lab script includes a DO480/solutions/quay-review/budget-app-dev-repository.json solution file.

- 8.1. Create a budget-app-dev-repository.json file with the finance organization definition.

```
{
 "namespace": "finance",
 "repository": "budget-app-dev",
 "description": "development",
 "visibility": "private"
}
```

- 8.2. Use the `curl` command to post the image repository definition in JSON format. Replace the token for the `quayadmin` user with the token that you obtained in a previous step.

```
[student@workstation ~]$ curl -X POST -k \
https://central-quay-registry.apps.ocp4.example.com/api/v1/repository \
--header "Content-Type: application/json" \
--header "Authorization: Bearer 0ITUIV9KMVJ0RPMWV5PT3D8ETWE82GBIA0U8NA6X" \
--data @budget-app-dev-repository.json
{"namespace": "finance", "name": "budget-app-dev", "kind": "image"}
```

9. Create the deployer robot account by using the API. The lab script includes a `D0480/solutions/quay-review/deployer-robot.json` solution file.

- 9.1. Create a `deployer-robot.json` file with the `finance` organization definition.

```
{
 "description": "deployer"
}
```

- 9.2. Use the `curl` command to post the robot account definition in JSON format. Replace the token for the `quayadmin` user with the token that you obtained in a previous step.

```
[student@workstation ~]$ curl -X PUT -k \
https://central-quay-registry.apps.ocp4.example.com/api/v1/organization/finance/
robots/deployer \
--header "Content-Type: application/json" \
--header "Authorization: Bearer 0ITUIV9KMVJ0RPMWV5PT3D8ETWE82GBIA0U8NA6X" \
--data @deployer-robot.json
{"name": "finance+deployer", "created": "Fri, 27 May 2022 17:21:48 -0000", "last_accessed": null, "description": "deployer", "token": "0X829E05TH48W8UX08GUBPALKE3CXMUZY51GNTOYPFUSJRZMK3X5DSGFUYYD1B0R", "unstructured_metadata": null}
```



### Note

- The `https://central.../robots/deployer \` fragment must be typed as a single line.

Note the new token. The previous token identified the `quayadmin` user. This token identifies the `deployer` robot account.

10. Grant permissions to the `deployer` robot account over the `budget-app-dev` image repository.

- 10.1. Use the curl command to grant the deployer robot account the write role on the budget-app-dev repository in the finance organization. Replace the token for the quayadmin user with the token that you obtained in a previous step.

```
[student@workstation ~]$ curl -X PUT -k \
 https://central-quay-registry.apps.ocp4.example.com/api/v1/repository/finance/
budget-app-dev/permissions/user/finance+deployer \
 --header "Content-Type: application/json" \
 --header "Authorization: Bearer 0ITUIV9KMVJ0RPMW5PT3D8ETWE82GBIA0U8NA6X" \
 --data '{"role": "write"}'
{"role": "write", "name": "finance+deployer", "is_robot": true, "avatar": {"name": "finance+deployer", "hash": "d55a13210f2c19148de2691ad3d9652140531e9b267d39ca468459a3c2e6961d", "color": "#5254a3", "kind": "robot"}, "is_org_member": true}
```

**Note**

- The https://central.../user/finance+deployer \ fragment must be typed as a single line.

11. Create a budget-app:development image by using the podman command. You can create an image based on the quay.io/podman/hello image.

- 11.1. Create a Containerfile file containing a minimal image definition. Your file should match the following example.

```
FROM quay.io/podman/hello
```

- 11.2. Build the image by using the podman build command.

```
[student@workstation ~]$ podman build . -t budget-app:development
...output omitted...
Successfully tagged localhost/budget-app:development
...output omitted...
```

12. As the deployer robot account, push the image to the budget-app-dev image repository in the finance organization.

- 12.1. Log in as the deployer robot account by using the token that you obtained in a previous step."

**Note**

- You must use the deployer robot account token, not the quayadmin user token.

```
[student@workstation ~]$ podman login -u=finance+deployer \
-p=0X829E05TH48W8UX08GUBPALKE3CXMUZY51GNTOYPFUSJRZMK3X5DSGFUYYD1B0R \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
...output omitted...
```

12.2. Push the image to the budget-app-dev repository.

```
[student@workstation ~]$ podman push localhost/budget-app:development \
 central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev
...output omitted...
Storing signatures
```

## 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 quay-review
```

## Finish

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

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

This concludes the section.

# Summary

---

- Red Hat Quay runs on public clouds and private data centers.
- Red Hat Quay has a highly-available architecture.
- Red Hat Quay uses vendor-neutral object storage protocols.
- Red Hat Quay uses a tenancy model to organize image repositories by using an RBAC model.
- Red Hat recommends installing Quay by using the Quay operator.
- You can manage Quay by using the web console or the Quay API.

## Chapter 7

# Integrating Red Hat Quay with Red Hat OpenShift and RHACM

### Goal

Describe Red Hat Quay use cases in a multicluster environment, and use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to deploy applications and control the image sources allowed in the cluster fleet.

### Objectives

- Identify the different Red Hat Quay use cases and architectures in a multicluster environment, and the benefits of using a central registry.
- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to restrict container image source registries in your cluster fleet.
- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to deploy an application using a central Red Hat Quay registry.

### Sections

- Describe Red Hat Quay Use Cases in a Multicluster Architecture (and Quiz)
- Restricting Image Registry Sources (and Guided Exercise)
- Deploying from Red Hat Quay to the Cluster Fleet (and Guided Exercise)

### Lab

- Integrating Red Hat Quay with OpenShift and RHACM

# Describe Red Hat Quay Use Cases in a Multicloud Architecture

---

## Objectives

- Identify the different Red Hat Quay use cases and architectures in a multicloud environment, and the benefits of using a central registry.

## Quay Deployment Patterns on Multicloud Environments

When the number of users and clusters interacting with the registry grows, you might face scaling challenges. Red Hat Quay has many deployment options that can help you scale your environment.

Red Hat Quay.io is a managed service that offers geo-replication and high availability without the operating costs of managing your own Quay instances. Quay.io does not cover all usage scenarios, such as disconnected operation. However, Quay.io can also be used in combination with other registries.

If Quay.io is not suitable for your environment, then you can deploy Quay on standalone hosts or on Red Hat OpenShift. Red Hat recommends installing Quay on an OpenShift cluster by using the Quay operator. There are some elements that you need to evaluate to choose the Quay deployment pattern and features to use in your environment.

- The geo-replication feature supports Quay users that are geographically distributed.
- High availability and autoscaling of front-end components is provided. To determine the requirements of back-end components, you must calculate the expected service level for each registry.
- The mirroring feature can mirror parts of some public registries to support disconnected Kubernetes or OpenShift clusters.
- The Quay bridge operator can adapt the Quay tenancy model to the OpenShift permissions model so that you can replace the OpenShift cluster's internal registry.

The Quay bridge operator is described elsewhere in this section.

## Calculating the Number of Registries

One important deployment choice is deciding if your cluster fleet needs one central registry, multiple registries, or a mix of both.

In Quay, one registry is represented by the database, the shared storage, the collections of users, and the registry DNS.

A central Quay registry provides the following benefits in a multicloud and multicloud environment.

### Saves compute resources

The more registries you have, the more instances of the Quay application you need.

### Saves operational costs

The fewer Quay components that you must maintain, the less effort is needed.

### **Centralizes access and security**

A primary Quay registry can centralize the security scanning and set control over the images used in your cluster fleet.

### **Saves storage space**

A high number of container images can share common layers that compound the image, and a central registry stores these layers just once.

### **Supports disconnected environments**

A registry simplifies the configuration of Kubernetes and OpenShift clusters to run in disconnected environments.

The following list reflects concerns that might lead you to create multiple registries and how a central Quay registry can address them.

### **Separation between environments**

You can define clear separation of environments by using the Quay tenancy model with organizations and role-based access control.

### **Stability in registry upgrades**

If you use Quay on OpenShift, the Quay operator upgrades the Quay base software.

### **Need for multiple data center deployments and high availability**

You can deploy a central Quay registry across data centers by using the Quay geo-replication and repository mirroring features.

### **Scalability of the registry platform**

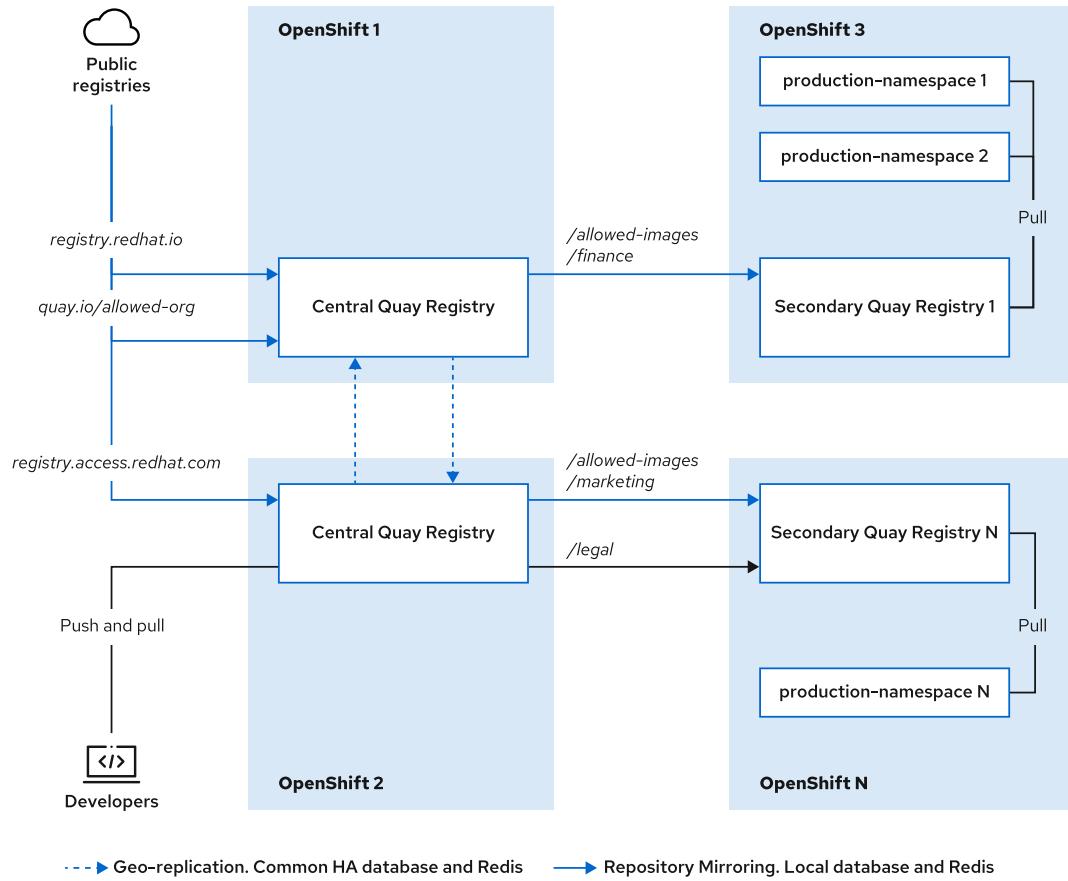
Red Hat Quay code base is the same as the base in Quay.io. Thus, Quay can scale to serve big registries without losing quality of service. When Quay is installed on OpenShift, the Quay operator uses horizontal pod autoscaler components to automatically performs this scaling.

Although a centralized registry has many advantages, there are times when multiple registries might be necessary. The main reason to have more than one registry is to provide more than one configuration. For example, if you must modify the `config.yaml` file for an environment to change the authentication type, or activate the repository mirroring feature, then you need more than one registry.

The Quay architecture gives supports mixed configurations. The Quay operator can deploy and manage multiple registries by creating additional instances of the `QuayRegistry` custom resource definition.

For example, you can install a central primary registry across many data centers or clusters by using the geo-replication feature. Afterwards, you can add secondary registries that mirror only a subset of the images of the primary registry.

The following diagram shows an example of a central Quay primary registry with one or many secondary Quay registries.



In the previous diagram, the **Central Quay Registry** is the primary registry, and becomes the single source of truth for your cluster fleet.

In this scenario, the clusters can pull images only from the secondary registries. The secondary registries are accessible only within the internal network and do not need to access to the Internet because they pull all the images from the primary registry. You can have different configurations on each of the different secondary registries because they are independent Quay instances, with their own database. However, you can also still control which images are allowed by controlling the central registry.

## Quay as a Registry for OpenShift

A common deploy pattern for Quay is to replace the internal OpenShift registry for builds and workloads. However, the Quay tenancy model does not follow the OpenShift cluster permissions model. The Kubernetes and OpenShift image registry access is configured per namespace. You must add the robot account credentials for access to Quay organizations to each namespace.



### Note

To learn more about the Quay robot accounts, see *Deploying from Red Hat Quay to the Cluster Fleet*.

You can also add the Quay robot accounts tokens to the global pull secret for OpenShift clusters. For more information about how to modify the OpenShift global pull secret, refer to the

*Updating the global cluster pull secret section of the official OpenShift documentation at [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/images/managing-images#images-update-global-pull-secret\\_using-image-pull-secrets](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/images/managing-images#images-update-global-pull-secret_using-image-pull-secrets).*

Quay provides an operator that helps to integrate Quay with the namespaces and permission model of OpenShift, the Quay bridge operator. The Quay bridge operator must run on every OpenShift cluster where you need automatic integration with a Quay registry.

The operator automates the following operations.

- Creates an organization in Quay for every OpenShift namespace.
- Creates three robot accounts in the Quay organization corresponding to the three OpenShift service accounts in each namespace: `builder`, `default`, and `deployer`.
- Adds the secrets of the previous robot accounts to the service accounts in the namespaces to access to the repositories in the Quay organization.
- Creates a repository in Quay for every OpenShift `ImageStream` object.
- Deletes the repository, the organization, and the robot accounts in Quay when you delete a namespace on OpenShift.
- Triggers redeployments of applications that use `ImageStreams` when there is a push to the Quay repository.

The main operator custom resource definition is the `QuayIntegration` object. This object allows you to set an identifier on the cluster for the integration. Thus, you can use the Quay bridge operator with multiple clusters, avoiding names collision.

The following screen capture shows the Quay console with the organizations created by the Quay bridge operator. The organizations come from 2 different clusters.

| NAME                                      |  |
|-------------------------------------------|--|
| managed-cluster-02_nfs-client-provisioner |  |
| managed-cluster-02_marketing              |  |
| managed-cluster-02_legal                  |  |
| managed-cluster-01_registry               |  |
| managed-cluster-01_nfs-client-provisioner |  |
| managed-cluster-01_marketing              |  |
| managed-cluster-01_legal                  |  |
| managed-cluster-01_finance                |  |

Figure 7.2: Organizations created by the Quay bridge operator

You must install the Quay bridge operator on every OpenShift cluster that requires integration with the Quay registries.



**Note**

The Quay bridge operator cannot integrate with Red Hat Quay.io.



**References**

For more information about the Quay operator, refer to the *Deploy Red Hat Quay on OpenShift with the Quay Operator Guide* at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_quay/3.6/html-single/deploy\\_red\\_hat\\_quay\\_on\\_openshift\\_with\\_the\\_quay\\_operator/index](https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/deploy_red_hat_quay_on_openshift_with_the_quay_operator/index)

For more information about the Quay bridge operator, refer to the *Integrate Red Hat Quay into OpenShift with the Bridge Operator* chapter in the *Manage Red Hat Quay Guide* at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_quay/3.6/html-single/manage\\_red\\_hat\\_quay/index#quay-bridge-operator](https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/manage_red_hat_quay/index#quay-bridge-operator)

## ► Quiz

# Describe Red Hat Quay Use Cases in a Multicloud Architecture

Choose the correct answers to the following questions:

- ▶ 1. **Which two of the following elements must you consider when choosing the deployment pattern for Red Hat Quay? (Choose two.)**
  - a. The programming language used by the builds
  - b. The use of the SaaS Quay.io instead of Quay to save operational costs
  - c. The need for a security scanner
  - d. The need to use the tag expiration and time-machine features
  - e. The need for a highly available registry for disconnected Kubernetes and OpenShift clusters
- ▶ 2. **Which two of the following are benefits of using a central Quay registry? (Choose two.)**
  - a. Centralizes access and security to the container images
  - b. Authenticates with an LDAP server
  - c. Provides high availability
  - d. Saves memory and CPU resources
  - e. Stores images in your preferred object storage system
- ▶ 3. **Which of the following reasons can force you to use more than one registry?**
  - a. You need a highly-available registry.
  - b. You need a vulnerability scanner.
  - c. You need to use the geo-replication feature.
  - d. You need to use the registry mirror feature.
  - e. You need to use two different authentication providers.
- ▶ 4. **Which three of the following operations are automatic if you use and configure the Quay bridge operator? (Choose three.)**
  - a. Creating users in Quay
  - b. Creating organizations in Quay for each namespace in OpenShift
  - c. Distributing the Quay credentials to the cluster namespaces
  - d. Triggering redeployments of applications that use image streams if there is a new push in the Quay repository
  - e. Installing the Clair scanner

► 5. Which custom resource definition uses the Quay bridge operator?

- a. QuayRegistry
- b. RegistryIntegration
- c. ConfigurationPolicy
- d. QuayIntegration

## ► Solution

# Describe Red Hat Quay Use Cases in a Multicloud Architecture

Choose the correct answers to the following questions:

- ▶ 1. **Which two of the following elements must you consider when choosing the deployment pattern for Red Hat Quay? (Choose two.)**
  - a. The programming language used by the builds
  - b. The use of the SaaS Quay.io instead of Quay to save operational costs
  - c. The need for a security scanner
  - d. The need to use the tag expiration and time-machine features
  - e. The need for a highly available registry for disconnected Kubernetes and OpenShift clusters
- ▶ 2. **Which two of the following are benefits of using a central Quay registry? (Choose two.)**
  - a. Centralizes access and security to the container images
  - b. Authenticates with an LDAP server
  - c. Provides high availability
  - d. Saves memory and CPU resources
  - e. Stores images in your preferred object storage system
- ▶ 3. **Which of the following reasons can force you to use more than one registry?**
  - a. You need a highly-available registry.
  - b. You need a vulnerability scanner.
  - c. You need to use the geo-replication feature.
  - d. You need to use the registry mirror feature.
  - e. You need to use two different authentication providers.
- ▶ 4. **Which three of the following operations are automatic if you use and configure the Quay bridge operator? (Choose three.)**
  - a. Creating users in Quay
  - b. Creating organizations in Quay for each namespace in OpenShift
  - c. Distributing the Quay credentials to the cluster namespaces
  - d. Triggering redeployments of applications that use image streams if there is a new push in the Quay repository
  - e. Installing the Clair scanner

► **5. Which custom resource definition uses the Quay bridge operator?**

- a. QuayRegistry
- b. RegistryIntegration
- c. ConfigurationPolicy
- d. QuayIntegration

# Restricting Image Registry Sources

---

## Objectives

- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to restrict container image source registries in your cluster fleet.

## Restricting Image Sources on OpenShift Clusters

Companies often need to control the origin of the software running their workloads. By restricting access to image container registries, companies can control the software that is used in a cluster.

Red Hat OpenShift Container Platform (RHOCP) has a mechanism to control the origin of images in use. Kubernetes does not provide any direct mechanism to control image sources. You could implement a custom a Kubernetes admission controller, which consults a service that allows or denies access to the registry when a workload pulls an image, but RHOCP offers an image controller that performs this function.

## The OpenShift Image Controller Object

OpenShift 4 provides the `config.openshift.io/v1/Image` resource type. This resource type contains configuration fields related to image control, such as a list of allowed registries. Each cluster has a single `config.openshift.io/v1/Image` object. The name of this object is always `cluster`. This object describes the global image configuration of the OpenShift cluster.

The following YAML file shows an example of the `cluster` image controller object.

```
apiVersion: config.openshift.io/v1
kind: Image ①
metadata:
 annotations:
 release.openshift.io/create-only: "true"
 creationTimestamp: "2022-05-17T13:44:26Z"
 generation: 1
 name: cluster ②
 resourceVersion: "8302"
 selfLink: /apis/config.openshift.io/v1/images/cluster
 uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
 additionalTrustedCA: ③
 name: myconfigmap
 registrySources: ④
 allowedRegistries: ⑤
 - example.com
 - quay.io
 - registry.redhat.io
 - image-registry.openshift-image-registry.svc:5000
 - reg1.io/myrepo/myapp:latest
 insecureRegistries: ⑥
```

```

- insecure.com
status:
internalRegistryHostname: image-registry.openshift-image-registry.svc:5000

```

- ➊ Object type and version.
- ➋ The name of the object. It is a fixed value.
- ➌ The config map that contains additional certificate authorities (CA) to trust on external registries.
- ➍ Field that determines how the cluster manages individual registries for builds and workloads.
- ➎ List of allowed registries. You can specify registries by domain and use the \* wildcard for subdomains. Furthermore, you can allow specific repositories and tags. If you use the `allowedRegistries` field, then you cannot use its opposite, the `blockedRegistries` field.
- ➏ List of registries that do not have valid TLS certificates.

The machine config operator (MCO) searches for changes in the OpenShift image controller object. Whenever the object changes, the MCO drains all the workloads from each OpenShift node. Then, the MCO applies the changes by modifying the `/host/etc/containers/policy.json` file and returns each node to the Ready state. Changing the image controller object can be time-consuming on large clusters.

You can use the `oc explain` command to read the documentation of the OpenShift image controller resource definition and all its fields. For example, you can read which fields contain the `registrySources` field by using the following command.

```

[user@host ~]$ oc explain images.config.openshift.io.spec.registrySources
KIND: Image
VERSION: config.openshift.io/v1

RESOURCE: registrySources <Object>

DESCRIPTION:
 registrySources contains configuration that determines how the container
 runtime should treat individual registries when accessing images for
 builds+pods. (e.g. whether or not to allow insecure access). It does not
 contain configuration for the internal cluster registry.

FIELDS:
 allowedRegistries <[]string>
 allowedRegistries are the only registries permitted for image pull and push
 actions. All other registries are denied. Only one of BlockedRegistries or
 AllowedRegistries may be set.

 blockedRegistries <[]string>
 blockedRegistries cannot be used for image pull and push actions. All other
 registries are permitted. Only one of BlockedRegistries or
 AllowedRegistries may be set.

 containerRuntimeSearchRegistries <[]string>
 containerRuntimeSearchRegistries are registries that will be searched when

```

pulling images that do not have fully qualified domains in their pull specs. Registries will be searched in the order provided in the list. Note: this search list only works with the container runtime, i.e CRI-O. Will NOT work with builds or imagestream imports.

```
insecureRegistries <[]string>
insecureRegistries are registries which do not have a valid TLS
certificates or only support HTTP connections.
```

Furthermore, you can view the status of the cluster OpenShift image controller object by using the following command.

```
[user@host ~]$ oc get image.config.openshift.io/cluster -o yaml
```

See the references section for more information about all the fields in the `image.config.openshift.io/cluster` object.

## Creating a Governance Policy to Restrict Source Registries on OpenShift Clusters

You can enforce the status of a Kubernetes object by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance policies.

As stated elsewhere in the course, every RHACM policy requires one or more `Policy`, one `PlacementRule`, and one `PlacementBinding` object definitions. You can modify the status of an object across the cluster fleet by using the `Policy` object.

You can create an RHACM policy to manage the OpenShift image controller object across the cluster fleet. The following example shows a policy that restricts clusters to using a small set of registries.

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
 name: image-policy
 ...omitted output...
spec:
 remediationAction: enforce
 disabled: false
 policy-templates:
 - objectDefinition:
 apiVersion: policy.open-cluster-management.io/v1
 kind: ConfigurationPolicy
 metadata:
 name: image-policy
 spec:
 remediationAction: enforce
 severity: low
 namespaceSelector:
 exclude:
 - kube-*
 include:
 - default
 object-templates:
```

```

- complianceType: musthave
 objectDefinition:
 apiVersion: config.openshift.io/v1
 kind: Image
 metadata:
 name: cluster
 spec:
 registrySources:
 allowedRegistries:
 - image-registry.openshift-image-registry.svc:5000
 - registry.redhat.io
 - registry.access.redhat.com
 - quay.io

```

If your workloads try to pull an image from a different registry, then they fail with the following message.

```

Failed to pull image "external.unknownregistry.io/mysql:8.0": rpc error:
code = Unknown desc = Source image rejected: Running image docker://
external.unknownregistry.io/mysql:8.0 is rejected by policy.

```



### Warning

OpenShift blocks all registries that are not included in the `allowedRegistries` list. Except for disconnected clusters, all clusters require access to the following list of registries for correct operation.

- The OpenShift internal registry
- The `registry.redhat.io` registry
- The `quay.io` registry

When using the `allowedRegistries` field, ensure those registries are always included in the list.

Use a placement rule to define the clusters that receive the policy. The following example defines the target clusters for the policy by using the `location` label.

```

apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
 name: placement-image-policy
spec:
 clusterConditions:
 - status: "True"
 type: ManagedClusterConditionAvailable
 clusterSelector:
 matchExpressions:
 - {key: location, operator: In, values: ["APAC"]}

```



## References

For more information about all the fields of the `image.config.openshift.io/cluster` object, refer to the *Image configuration resources* chapter of the *Images Guide* of the official OpenShift documentation at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/images/index#image-configuration](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/images/index#image-configuration)

For more information about the policy YAML structure, refer to the *The Policy YAML Structure* section in 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/governance/index#policy-yaml-structure](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_management_for_kubernetes/2.4/html-single/governance/index#policy-yaml-structure)

## ► Guided Exercise

# Restricting Image Registry Sources

- Locate deployments that use container images from insecure registries across the cluster fleet. Then, you configure the cluster fleet to avoid untrusted image sources by using a Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance policy.

## Outcomes

- Locate specific images across the cluster fleet.
- Deploy a RHACM governance policy.
- Use the RHACM policy to manage the `image.config.openshift.io/cluster` OpenShift image object.

## Before You Begin

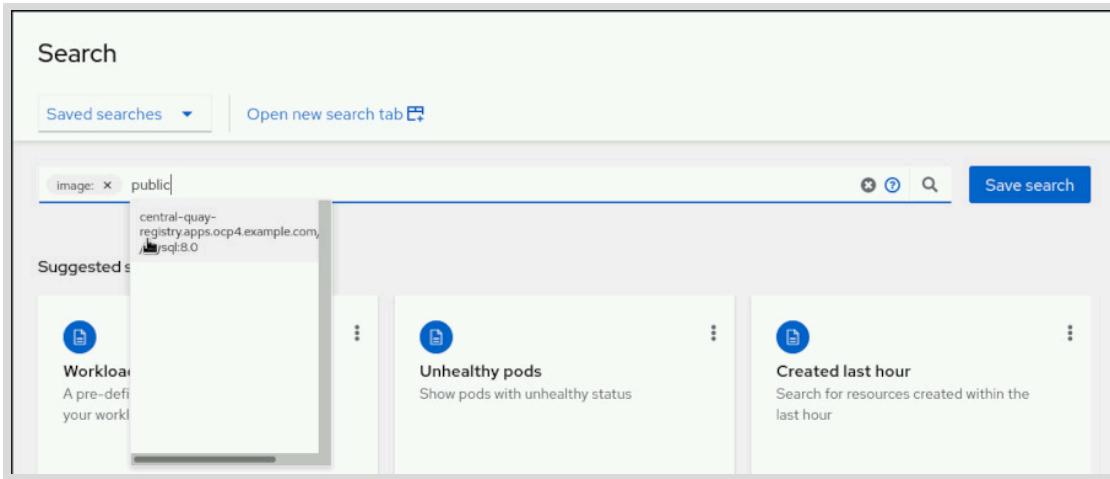
As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that all the deployments are ready on the cluster fleet. The command also creates the following Quay resources:

- Two public image organizations in the internal Quay registry: the `central-quay-registry.apps.ocp4.example.com/public` organization and the `central-quay-registry.apps.ocp4.example.com/finance` organization.
- Two Quay repositories with an image: the `central-quay-registry.apps.ocp4.example.com/public/mysql:8.0` image and the `central-quay-registry.apps.ocp4.example.com/finance/approved-mysql:8.0` image.

```
[student@workstation ~]$ lab start quayacm-restrict
```

1. Use the RHACM web console search feature to identify any images in the internal Quay registry that are pulled from the `public` untrusted organization.
  - 1.1. From the `workstation` machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
  - 1.2. Click the **Search** icon in the upper-right corner of the window to open the RHACM search page.
  - 1.3. In the search field, type `image:` and review the different image registries in use across the cluster fleet. Continue by typing `public` without pressing `Enter` to restrict the search.



Click the name of the MySQL image showed in the searches suggestions.

- 1.4. Click **1 Related deployment** in the results page to find which deployments and clusters contain that image. There is one deployment that uses the `central-quay-registry.apps.ocp4.example.com/public/mysql:8.0` image. The deployment is in the `budget-app` namespace in the `local-cluster` cluster.

The finance department only accepts images that are already tested and stored under the `finance` Quay organization.

- ▶ 2. Use a RHACM governance policy to allow pulling images only from trusted registries and repositories.
  - 2.1. From the terminal, log in to the `ocp4` hub cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 2.2. Create the `policies` namespace to store the new policy.

```
[student@workstation ~]$ oc create namespace policies
namespace/policies created
```

- 2.3. Change to the `~/D0480/labs/quayacm-restrict` directory, and review the `image-policy.yaml` file.

```
[student@workstation ~]$ cd D0480/labs/quayacm-restrict
```

Edit the `image-policy.yaml` file to add the `finance` Quay URL organization to the list of allowed container registries.

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
 name: image-policy
```

```

...output omitted...
spec:
 remediationAction: enforce
 disabled: false
 policy-templates:
 ...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: config.openshift.io/v1
 kind: Image
 metadata:
 name: cluster
 spec:
 registrySources:
 allowedRegistries:
 - central-quay-registry.apps.ocp4.example.com/finance
 - registry.redhat.io
 - registry.access.redhat.com
 - quay.io

apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
...output omitted...
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
...output omitted...
clusterSelector:
 matchExpressions: []

```

This policy uses the `image.config.openshift.io/cluster` image object to specify which container registries the OpenShift cluster can use.

This policy also uses an empty cluster selector to select all the clusters present in the fleet.

#### 2.4. Create the policy in the `policies` namespace.

```
[student@workstation quayacm-restrict]$ oc apply -f image-policy.yaml -n policies
policy.policy.open-cluster-management.io/image-policy created
placementbinding.policy.open-cluster-management.io/binding-image-policy created
placementrule.apps.open-cluster-management.io/placement-image-policy created
```

This RHACM policy is in enforcing mode. Thus, it modifies the `image.config.openshift.io/cluster` image object in each cluster immediately. Then, the `OpenShift Machine Config` operator in each cluster updates all the nodes in the clusters to apply the list of allowed registries.

#### 2.5. Navigate to the RHACM web console and click **Governance** on the left pane. Then, click **image-policy** to review that the `image-policy` policy is compliant in all the matched clusters.

► 3. Deploy the `invoices-app` application to test that only trusted repositories are allowed.

- 3.1. From the terminal, change to the `~/D0480/labs/quayacm-restrict` directory and review the `template-app-deployment.yaml` OpenShift template file. Note the parameter with the image that the application uses.

```
...output omitted...
parameters:
- description: MySQL image
 displayName: MySQL image
 name: MYSQL_IMAGE
 required: true
 value: central-quay-registry.apps.ocp4.example.com/public/mysql:8.0
...output omitted...
```

- 3.2. Log in to a managed cluster.

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

- 3.3. Create the `invoices-app` namespace.

```
[student@workstation quayacm-restrict]$ oc create namespace invoices-app
namespace/invoices-app created
```

- 3.4. Deploy the application by processing the template.

```
[student@workstation quayacm-restrict]$ oc process \
-f template-app-deployment.yaml \
-p "APPLICATION=invoices-app" \
| oc create -n invoices-app -f -
deployment.apps/mysql-invoices-app created
service/mysql-invoices-app created
```

- 3.5. Inspect the status of the `invoices-app` application pods.

```
[student@workstation quayacm-restrict]$ oc get pods -n invoices-app
NAME READY STATUS RESTARTS AGE
mysql-invoices-app-5459f489d9-6gttv 0/1 ImagePullBackOff 0 2m20s
```

The pod cannot pull the image. To review the reason, get the events of the `invoices-app` namespace.

```
[student@workstation quayacm-restrict]$ oc get events -n invoices-app | \
grep Failed
10m Warning Failed pod/mysql-invoices-app-5459f489d9-6gttv
Failed to pull image "central-quay-registry.apps.ocp4.example.com/public/
mysql:8.0": rpc error: code = Unknown desc = Source image rejected: Running image
docker://central-quay-registry.apps.ocp4.example.com/public/mysql:8.0 is rejected
by policy.
10m Warning Failed pod/mysql-invoices-app-5459f489d9-6gttv
Error: ErrImagePull
10m Warning Failed pod/mysql-invoices-app-5459f489d9-6gttv
Error: ImagePullBackOff
```

#### ▶ 4. Correct the `invoices-app` application deployment by using an approved image.

##### 4.1. Delete the `invoices-app` namespace in the managed cluster.

```
[student@workstation quayacm-restrict]$ oc delete namespace invoices-app
namespace "invoices-app" deleted
```

##### 4.2. Recreate the namespace.

```
[student@workstation quayacm-restrict]$ oc create namespace invoices-app
namespace/invoices-app created
```

##### 4.3. Process the deployment template with the `MYSQL_IMAGE` parameter pointing to the approved `central-quay-registry.apps.ocp4.example.com/finance/approved-mysql:8.0` MySQL image.

```
[student@workstation quayacm-restrict]$ oc process \
-f template-app-deployment.yaml \
-p "APPLICATION=invoices-app" \
-p
"MYSQL_IMAGE=central-quay-registry.apps.ocp4.example.com/finance/approved-mysql:8.0"
\
| oc create -n invoices-app -f -
deployment.apps/mysql-invoices-app created
service/mysql-invoices-app created
```



#### Note

- The `-p "MYSQL_IMAGE=...approved-mysql:8.0"` \ fragment must be typed as a single line.

##### 4.4. Inspect the status of the `invoices-app` application pods.

```
[student@workstation quayacm-restrict]$ oc get pods -n invoices-app
NAME READY STATUS RESTARTS AGE
mysql-invoices-app-67b47884c8-7gtp6 1/1 Running 0 71s
```

## Finish

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

```
[student@workstation quayacm-restrict]$ cd ~
[student@workstation ~]$ lab finish quayacm-restrict
```

This concludes the section.

# Deploying from Red Hat Quay to the Cluster Fleet

---

## Objectives

- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to deploy an application using a central Red Hat Quay registry.

## Introducing Robot Accounts

Quay provides robot accounts for automated operations. Robot accounts are similar to user accounts, but the following key differences make them more suitable for automated processes.

- Robot accounts belong to a single organization.
- Robot accounts cannot use the web interface.
- Robot accounts access Quay by using tokens only instead of passwords.

Limiting the capabilities of robot accounts reduces security risks.

Robot accounts are also similar to Kubernetes service accounts. Kubernetes service accounts and robot accounts act as user accounts for automated processes.

## Managing Robot Accounts

Manage team membership of robot accounts from the team page.

From the organization page, click **Robot Accounts** to manage robot accounts by using the web console.

| ROBOT ACCOUNT NAME        | DESCRIPTION                                     | TEAMS    | REPOSITORIES    | CREATED        | LAST ACCESSED |
|---------------------------|-------------------------------------------------|----------|-----------------|----------------|---------------|
| finance+verifier          | Robot account used by the verification service. | No teams | No repositories | 25 minutes ago | Never         |
| finance+automation_server | Robot account used by the automation server     | No teams | 1 repository    | 26 minutes ago | Never         |

Managing robot accounts.

- ➊ Click **Create Robot Account** to create robot accounts.
- ➋ Click **View Credentials** to get the robot account token and display usage instructions.
- ➌ Click **Set Repository Permissions** to manage repository permissions.

You can also manage robots by using the API. You need a token with the administer organization permission to manage robots.

To create a robot account, make an HTTP PUT request with a payload with the following structure to the `https://quay.example.com/api/v1/organization/organization/robots/name/` endpoint.

```
{"description": "description"}
```

The response contains the robot account token.

```
{
 "name": "finance+verifier",
 "created": "Wed, 27 Apr 2022 10:34:33 -0000",
 "last_accessed": null,
 "description": "Robot account used by the verification service.",
 "token": "B2...Z9",
 "unstructured_metadata": null
}
```

To grant robot accounts permissions over repositories, make an HTTP PUT request with a payload with the following structure to the `https://quay.example.com/api/v1/repository/organization/repository/permissions/user/organization+robot_account_name/` endpoint.

```
{"role": "role"}
```

The `role` has a `read`, `write`, or `admin` value.



### Warning

If a malicious actor obtains access to the robot account token, then the actor can perform all the actions that the robot account can perform.

Only grant the minimum permissions that you require. Store the token as confidential information. Review usage logs to detect unusual activity.

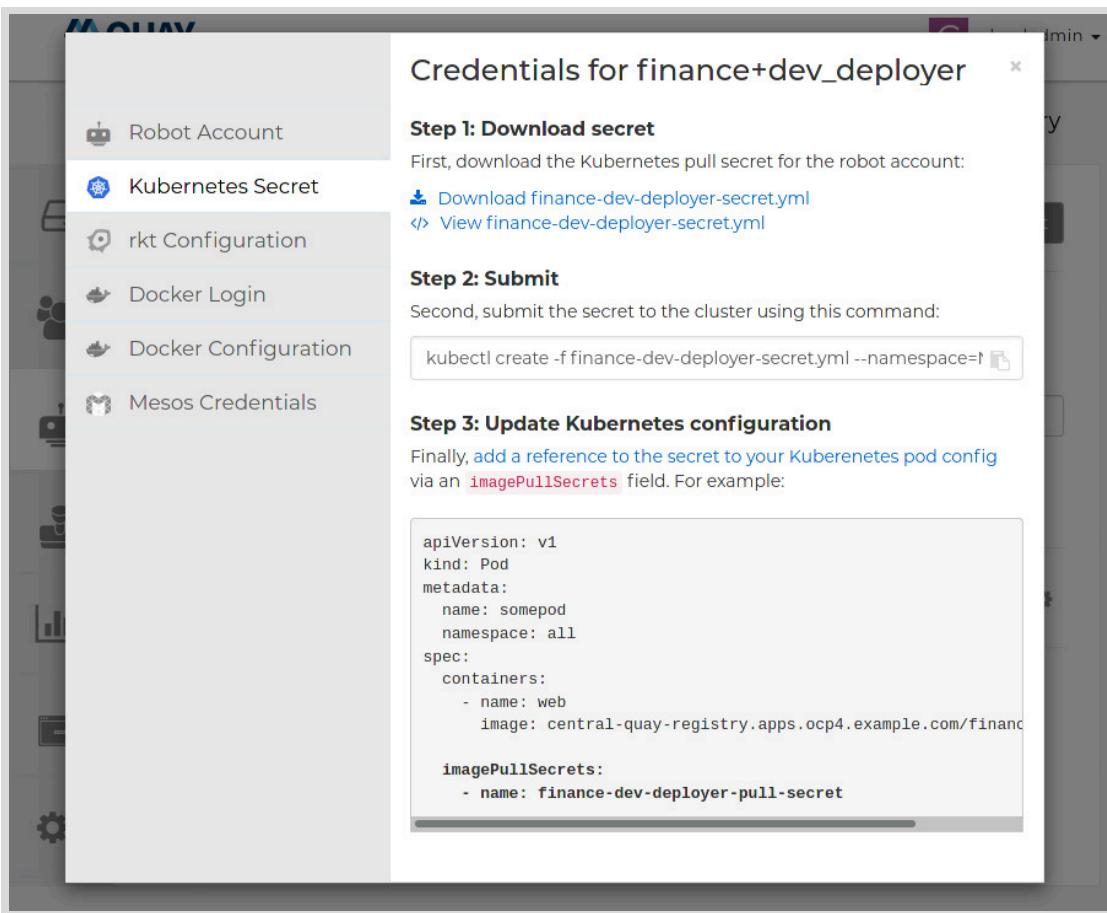
To revoke a token, recreate the robot account. You can also recreate robot accounts periodically to reduce the period a leaked token is usable.

## Using Robot Accounts for Deployments

After creating a robot account, you can use the robot account credentials in deployments.

The Quay console displays robot account credentials and instructions to use robot accounts. Navigate to the **Robot Accounts** page for an organization to display existing robot accounts. Click a robot account name or the **Options** gear icon, then click **View Credentials**.

A window opens displaying instructions about using the robot account.



**Figure 7.5: Robot account usage instructions**

This window contains a secret definition that you can view or download. You can use the `kubectl create` command to create a secret in a cluster containing the robot account credentials. Finally, the window contains an example about how to use the secret in a pod definition.

After you create the secret, you can reference it as an image pull secret in workload definitions.

```

apiVersion: v1
kind: Pod
metadata:
 name: somepod
 namespace: all
spec:
 containers:
 - name: web
 image: central-quay-registry.apps.ocp4.example.com/finance/somerepo

 imagePullSecrets:
 - name: finance-deployer-pull-secret

```

## Deploying Workloads That Use Private Images Across a Cluster Fleet

Using the `imagePullSecrets` key in workload definitions can be effective when only a low number of workloads require an image pull secret. If you are using Red Hat Advanced Cluster Management for Kubernetes (RHACM) applications to deploy workloads, then you can add the image pull secret to the RHACM application definitions. However, when many deployments require a secret, editing all workloads can be inconvenient and error prone.

The Quay bridge operator can synchronize OpenShift namespaces and Quay organizations, and configure the necessary image pull secrets. For more information about the Quay bridge operator, refer to *Describe Red Hat Quay Use Cases in a Multicluster Architecture*. When using the Quay bridge operator, if a workload requires a private image, then you push the image to the Quay organization that matches the workload namespace. This way, workloads can access private images transparently.

When working on a cluster fleet, you can use RHACM policies to install the Quay bridge operator in all managed clusters automatically.

Alternatively, you can add pull secrets to the service accounts that interact with Quay or update the global pull secret.

By default, Kubernetes clusters have a `default` service account in each namespace. Kubernetes uses these service accounts to interact with registries. In addition to the `default` service account, OpenShift clusters also use `builder` and `deployer` service accounts. To use private registries, you can use the `oc secrets link` command to add secrets to service accounts.

You can use Red Hat Advanced Cluster Management for Kubernetes (RHACM) or other automation systems to perform those operations across a cluster fleet.



### References

For more information, refer to the *Allowing robot access to a user repository* section in the *Managing access to repositories* chapter in the Red Hat Quay 3.6 Use Red Hat Quay documentation at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_quay/3.6/html-single/use\\_red\\_hat\\_quay/index#allow-robot-access-user-repo](https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html-single/use_red_hat_quay/index#allow-robot-access-user-repo)

For more information about image pull secrets, refer to the *Using image pull secrets* section in the *Managing images* chapter in the *Images* guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/images/index#using-image-pull-secrets](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/images/index#using-image-pull-secrets)

For more information about default OpenShift service accounts, refer to the *Default service accounts* section in the *Using service accounts in applications* chapter in the *Authentication and authorization* guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/authentication\\_and\\_authorization/index#service-accounts-default\\_using-service-accounts](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/authentication_and_authorization/index#service-accounts-default_using-service-accounts)

## ► Guided Exercise

# Deploying from Red Hat Quay to the Cluster Fleet

- Create an application container image in Red Hat Quay and deploy the application to the cluster fleet by using Red Hat Advanced Cluster Management (RHACM).

## Outcomes

- Configure deployments to use a private container registry.

## 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 and Quay are available.

```
[student@workstation ~]$ lab start quayacm-deploy
```

### ► 1. Create a minimal application image.

- 1.1. Create a file named `Containerfile` with the following content.

```
FROM registry.access.redhat.com/ubi8/httpd-24:1
RUN echo hello >/var/www/html/index.html
```

- 1.2. Use the `podman build` command to build the image.

```
[student@workstation ~]$ podman build . -t hello:1
...output omitted...
Successfully tagged localhost/hello:latest
...output omitted...
```

### ► 2. Create a budget-app-dev namespace.

- 2.1. Log in to the ocp4 hub cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 2.2. Create the `budget-app-dev` namespace.

```
[student@workstation ~]$ oc create namespace budget-app-dev
namespace/budget-app-dev created
```

- 3. Create a finance organization and a budget-app-dev image repository in Quay.
- 3.1. From the workstation machine, navigate to the Quay web console at <https://central-quay-registry.apps.ocp4.example.com>. Log in as the **cloudadmin** user with the **redhat** password.  
If prompted, click **Confirm Username**.
  - 3.2. Click **Create New Organization**.

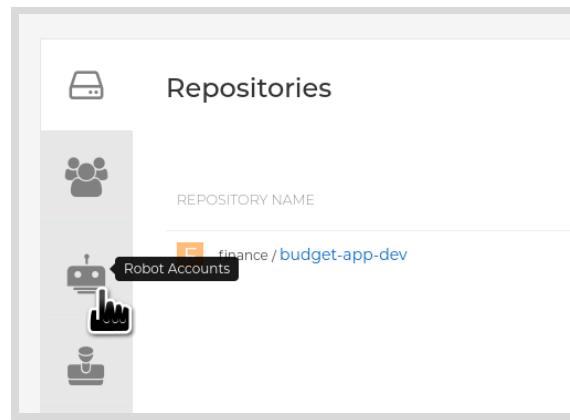


- 3.3. Type **finance** as the organization name, then click **Create Organization**.
  - 3.4. Click **Create New Repository**.
- 
- The screenshot shows the Quay web interface with 'finance' selected as the organization. It displays a list of repositories under this organization. A 'Create New Repository' button is visible on the right side of the screen, highlighted with a red box.

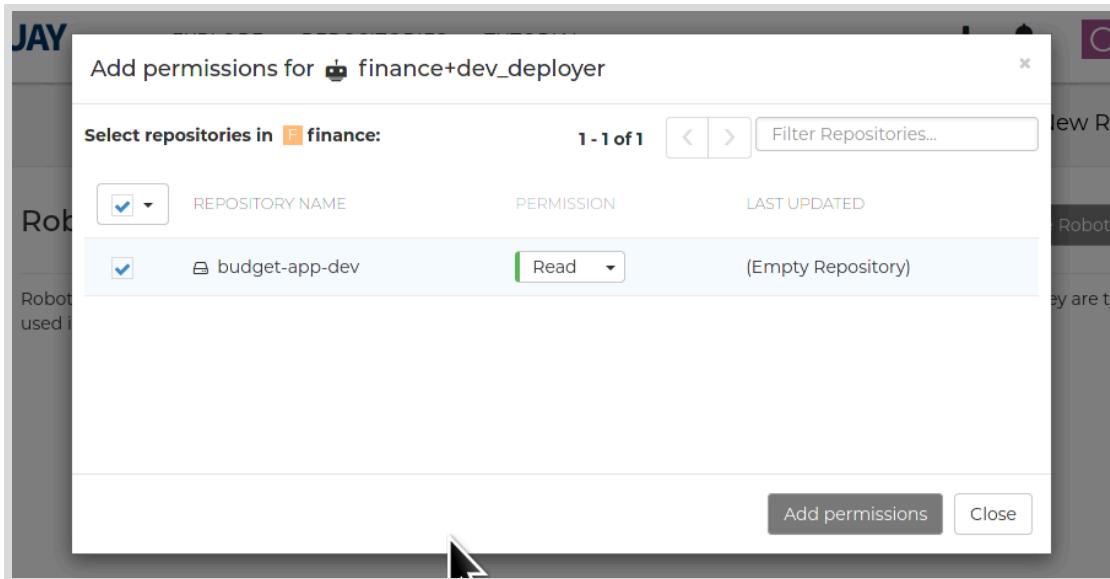
- 3.5. Type **budget-app-dev** as the repository name, then click **Create Private Repository**.

► 4. Create a **dev\_deployer** robot account. Grant read permissions to the robot account over the budget-app-dev repository.

  - 4.1. Click **REPOSITORIES**, then click **finance** to go to the organization page.
  - 4.2. Click **Robot Accounts**, then click **Create Robot Account**.



- 4.3. Type `dev_deployer` as the robot account name, leave the description blank, then click **Create robot account**.
- 4.4. On the page that opens, select the **Read** permission over the `budget-app-dev` repository, then click **Add permissions**.



▶ 5. Obtain the robot account secret.

- 5.1. Click **finance+dev\_deployer** to view the credentials for the robot account, then click **Kubernetes Secret** to view instructions about using a robot account in Kubernetes.

The page describes how to create a Kubernetes secret for the robot account, and how to use the `imagePullSecrets` directive to refer to the secret for pulling images.

**Credentials for finance+dev\_deployer**

**Step 1: Download secret**  
First, download the Kubernetes pull secret for the robot account:  
[Download finance-dev-deployer-secret.yaml](#) [View finance-dev-deployer-secret.yaml](#)

**Step 2: Submit**  
Second, submit the secret to the cluster using this command:  
`kubectl create -f finance-dev-deployer-secret.yaml --namespace=` [Edit](#)

**Step 3: Update Kubernetes configuration**  
Finally, add a reference to the secret to your Kubernetes pod config via an `imagePullSecrets` field. For example:

```
apiVersion: v1
kind: Pod
metadata:
 name: somepod
 namespace: all
spec:
 containers:
 - name: web
 image: central-quay-registry.apps.ocp4.example.com/finance
 imagePullSecrets:
 - name: finance-dev-deployer-pull-secret
```

5.2. Click the download icon to download `finance-dev-deployer-secret.yaml`. In the confirmation window, click **Save File** and then click **OK**. The browser downloads the file to the `/home/students/Downloads` directory.

▶ 6. Use the `oc create` command to create the secret in the `budget-app-dev` namespace.

```
[student@workstation ~]$ oc create -f Downloads/finance-dev-deployer-secret.yaml \
--namespace=budget-app-dev
secret/finance-dev-deployer-pull-secret created
```

▶ 7. Push the image to Quay.

7.1. Log in to Quay as the `clouadmin` user with the `redhat` password.



### Warning

This exercise uses a plain text password for brevity. Commands such as `podman store` passwords in the file system unencrypted. If a malicious actor obtains access to the file system, then the actor can impersonate your account.

In real-world situations, always use encrypted passwords as described in a previous step.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
 central-quay-registry.apps.ocp4.example.com
Login Succeeded!
...output omitted...
```

7.2. Push the image to Quay.

```
[student@workstation ~]$ podman push localhost/hello:1 \
 central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev:1
...output omitted...
Writing manifest to image destination
Storing signatures
```

- ▶ 8. Fork the do480-apps course GitHub repository at <https://github.com/redhattraining/do480-apps/>. This repository contains starter YAML files to build the application.
  - 8.1. From the workstation machine, navigate to the do480-apps repository at <https://github.com/redhattraining/do480-apps/>.
  - 8.2. In the upper-right corner of the GitHub repository page, click **Fork**, then click **Create fork** to create a fork for your repository.



### Note

If you have already performed this exercise or made other changes to the fork, then you must delete your fork to avoid problems during this exercise.

To delete the fork, navigate to <https://github.com/your-fork-name/DO480-apps/>. Click **Settings**. Click **Delete this repository** and enter the name of the repository to confirm the deletion.

- 8.3. In your GitHub fork, navigate to the deployment.. Click **quayacm-deployment** and then click **deployment.yaml** to access the file. Click the pencil icon to edit the YAML file.
- 8.4. Edit the deployment that matches the following content.

```
...output omitted...
spec:
 containers:
 -
 image: central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev:1
 name: budget-app
 ports:
 - containerPort: 8080
 name: budget-app
 imagePullSecrets:
 - name: finance-dev-deployer-pull-secret
```

**Note**

The `- image: central...dev:1` statement must be a single line.

Scroll to the bottom of the page and click **Commit changes**.

- ▶ **9.** Deploy the application by using RHACM.
  - 9.1. From the **workstation** machine, navigate to <https://multicloud-console.apps.ocp4.example.com>.
  - 9.2. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password.
  - 9.3. On the left navigation bar, navigate to **Applications**, click **Create application**, and then click **Subscription**.
  - 9.4. Fill the form with the following values.

| Field            | Value                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------|
| Name             | <code>budget-app-dev</code>                                                                           |
| Namespace        | <code>budget-app-dev</code>                                                                           |
| Repository types | Git                                                                                                   |
| URL              | <a href="https://github.com/your_fork/D0480-apps.git">https://github.com/your_fork/D0480-apps.git</a> |
| Branch           | <code>main</code>                                                                                     |
| Path             | <code>quayacm-deployment</code>                                                                       |

- 9.5. Scroll down and expand **Select clusters to deploy to** to configure a placement rule for the application. Select the **Deploy on local cluster** option to create a placement rule that deploys the application only on the local cluster.
  - 9.6. Click **Save**.
- ▶ **10.** Access the application.
    - 10.1. On the **Topology** page, wait until all the application components have a green checkmark.
    - 10.2. Click the **Route** component. Click the link to <http://budget.apps.ocp4.example.com/> that is displayed. A new tab opens and displays the **hello** text.

## Finish

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

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

This concludes the section.

## ▶ Lab

# Integrating Red Hat Quay with OpenShift and RHACM

- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to restrict image sources in the production clusters. Then you configure a Quay robot account to manage images for production clusters. Finally, you deploy the `budget-app` application in the production clusters by adding a specific image to the list of allowed images.

## Outcomes

- Use a RHACM policy to allow specific images by managing the `image.config.openshift.io/cluster` OpenShift object.
- Use robot accounts to manage Quay organizations.
- Deliver applications into multiple clusters by using RHACM GitOps.
- Configure deployments to use a Quay private container registry.

## Before You Begin

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

This command ensures that all Quay resources are present. The command also tags the production managed clusters for RHACM by using the `env=production` label.

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

1. Log in to the RHACM OpenShift hub cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.
2. Create an RHACM governance policy in the `policies` namespace, to allow only the following list of images and container registries:
  - `registry.redhat.io`
  - `registry.access.redhat.com`
  - `quay.io`
  - `central-quay-registry.apps.ocp4.example.com/finance/budget-app:production`

The policy uses the `image.config.openshift.io/cluster` OpenShift object. Apply the policy to the managed clusters with the `env=production` label. The `lab` script includes a `D0480/labs/quayacm-review/image-policy.yaml` solution file.

3. Create a Quay robot account for the `finance` Quay organization. Grant the account the `read` permission on the `budget-app-dev` repository, and the `write` permission on the `budget-app` repository. Log in to the Quay central registry at `https://`

- central-quay-registry.apps.ocp4.example.com as `cloudadmin`, using `redhat123` as the password.
4. Prepare the managed clusters for access to the central Quay registry. Create the `budget-app` namespace, and the `finance-prod-deployer-pull-secret` pull secret from the robot account on the production and non-production clusters. The API server address of the non-production cluster is `https://api.ocp4.example.com:6443`. The API server address of the production cluster is `https://api.ocp4-mng.example.com:6443`.
  5. Fork the `do480-apps` course GitHub repository at `https://github.com/redhattraining/do480-apps/`. This repository contains starter YAML files to build the application. Modify the deployment to use the `central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev:1.0` container image and the pull secret.
  6. Deploy the `budget-app` application to all the OpenShift managed clusters using RHACM. The application fails on production clusters because it is using an image from a forbidden registry.
  7. Promote the `budget-app-dev:1.0` image to the allowed `budget-app` repository with the production robot account by using the `skopeo` command. Use the password token you saved in the `/home/student/prod_deployer_token` file.
  8. Modify the `budget-app` application in your GitHub account to use the `central-quay-registry.apps.ocp4.example.com/finance/budget-app:production` image.

## 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 quayacm-review
```

## Finish

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

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

This concludes the section.

## ► Solution

# Integrating Red Hat Quay with OpenShift and RHACM

- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to restrict image sources in the production clusters. Then you configure a Quay robot account to manage images for production clusters. Finally, you deploy the `budget-app` application in the production clusters by adding a specific image to the list of allowed images.

## Outcomes

- Use a RHACM policy to allow specific images by managing the `image.config.openshift.io/cluster` OpenShift object.
- Use robot accounts to manage Quay organizations.
- Deliver applications into multiple clusters by using RHACM GitOps.
- Configure deployments to use a Quay private container registry.

## Before You Begin

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

This command ensures that all Quay resources are present. The command also tags the production managed clusters for RHACM by using the `env=production` label.

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

1. Log in to the RHACM OpenShift hub cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.
  - 1.1. Log in to the RHACM hub cluster.

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

2. Create an RHACM governance policy in the `policies` namespace, to allow only the following list of images and container registries:
  - `registry.redhat.io`
  - `registry.access.redhat.com`
  - `quay.io`
  - `central-quay-registry.apps.ocp4.example.com/finance/budget-app:production`

The policy uses the `image.config.openshift.io/cluster` OpenShift object. Apply the policy to the managed clusters with the `env=production` label. The lab script includes a `D0480/labs/quayacm-review/image-policy.yaml` solution file.

- 2.1. Create the `policies` namespace to store the new policy.

```
[student@workstation ~]$ oc create namespace policies
namespace/policies created
```

- 2.2. Change to the `~/D0480/labs/quayacm-review` directory.

```
[student@workstation ~]$ cd D0480/labs/quayacm-review
```

- 2.3. Review the `image-policy.yaml` file.

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
 name: image-policy
...output omitted...
spec:
 remediationAction: enforce
 disabled: false
 policy-templates:
...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: config.openshift.io/v1
 kind: Image
 metadata:
 name: cluster
 spec:
 registrySources:
 allowedRegistries:
 - registry.redhat.io
 - registry.access.redhat.com
 - quay.io
 - central-quay-registry...budget-app:production
...
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
...output omitted...
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
...output omitted...
 clusterSelector:
 matchExpressions:
 - {key: env, operator: In, values: ["production"]}
```

**Note**

As stated in this chapter, the `image.config.openshift.io/cluster` object filters by using domain names, repository names, or specific image names and tags.

- 2.4. Create the policy in the `policies` namespace.

```
[student@workstation quayacm-review]$ oc apply -f image-policy.yaml -n policies
policy.policy.open-cluster-management.io/image-policy created
placementbinding.policy.open-cluster-management.io/binding-image-policy created
placementrule.apps.open-cluster-management.io/placement-image-policy created
```

- 2.5. Read the `image.config.openshift.io/cluster` OpenShift object in the non-production cluster to verify that it does not contain the three allowed registries.

```
[student@workstation quayacm-review]$ oc get image.config.openshift.io/cluster \
-o yaml
apiVersion: config.openshift.io/v1
kind: Image
metadata:
 annotations:
 include.release.openshift.io/ibm-cloud-managed: "true"
 include.release.openshift.io/self-managed-high-availability: "true"
 include.release.openshift.io/single-node-developer: "true"
 release.openshift.io/create-only: "true"
 creationTimestamp: "2022-05-13T15:58:17Z"
 generation: 1
 name: cluster
 ownerReferences:
 - apiVersion: config.openshift.io/v1
 kind: ClusterVersion
 name: version
 uid: 73cada64-5434-48af-8208-ec144ee8555d
 resourceVersion: "25816"
 uid: 214ff292-782c-4b92-b31b-0cc1b4af75a1
spec: {}
status:
 internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- 2.6. Log in to the `managed-cluster` production cluster.

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

- 2.7. Review the `image.config.openshift.io/cluster` OpenShift object in the production cluster to verify that it contains the three allowed registries.

```
[student@workstation quayacm-review]$ oc get image.config.openshift.io/cluster \
-o yaml
apiVersion: config.openshift.io/v1
kind: Image
metadata:
 annotations:
 include.release.openshift.io/ibm-cloud-managed: "true"
 include.release.openshift.io/self-managed-high-availability: "true"
 include.release.openshift.io/single-node-developer: "true"
 release.openshift.io/create-only: "true"
 creationTimestamp: "2022-05-13T16:42:21Z"
 generation: 3
 name: cluster
 ownerReferences:
 - apiVersion: config.openshift.io/v1
 kind: ClusterVersion
 name: version
 uid: 13a25437-a584-434b-a32f-2f3c6d08f87c
 resourceVersion: "175103"
 uid: b4070508-948f-4d43-a7eb-f876e7a72c9e
spec:
 registrySources:
 allowedRegistries:
 - quay.io
 - registry.access.redhat.com
 - registry.redhat.io
 - central-quay-registry.apps.ocp4.example.com/finance/budget-app:production
status:
 internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

**3.** Create a Quay robot account for the `finance` Quay organization.

Grant the account the read permission on the `budget-app-dev` repository, and the write permission on the `budget-app` repository. Log in to the Quay central registry at `https://central-quay-registry.apps.ocp4.example.com` as `cloudadmin`, using `redhat123` as the password.

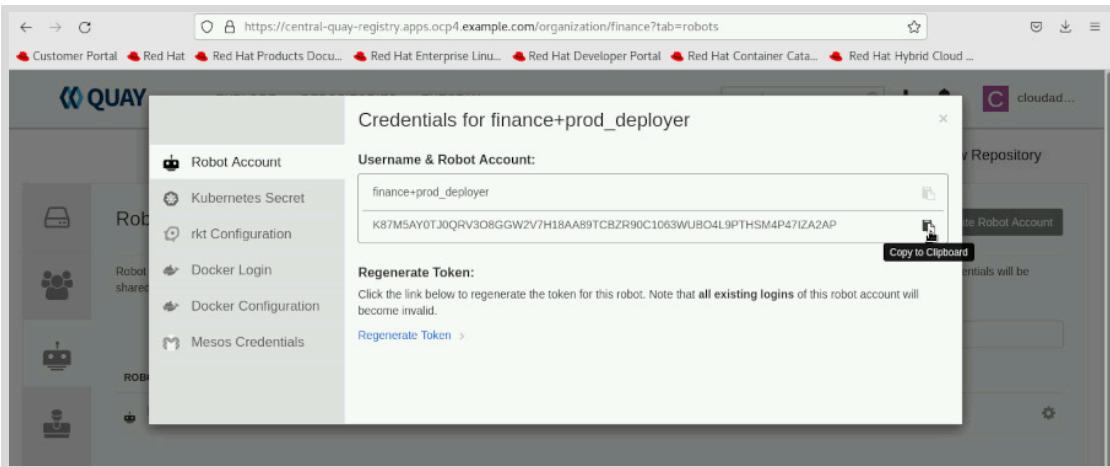
- 3.1. From the `workstation` machine, navigate to the Quay web console at `https://central-quay-registry.apps.ocp4.example.com`. Log in as `cloudadmin` using `redhat123` as the password.

If prompted, click **Confirm Username**.

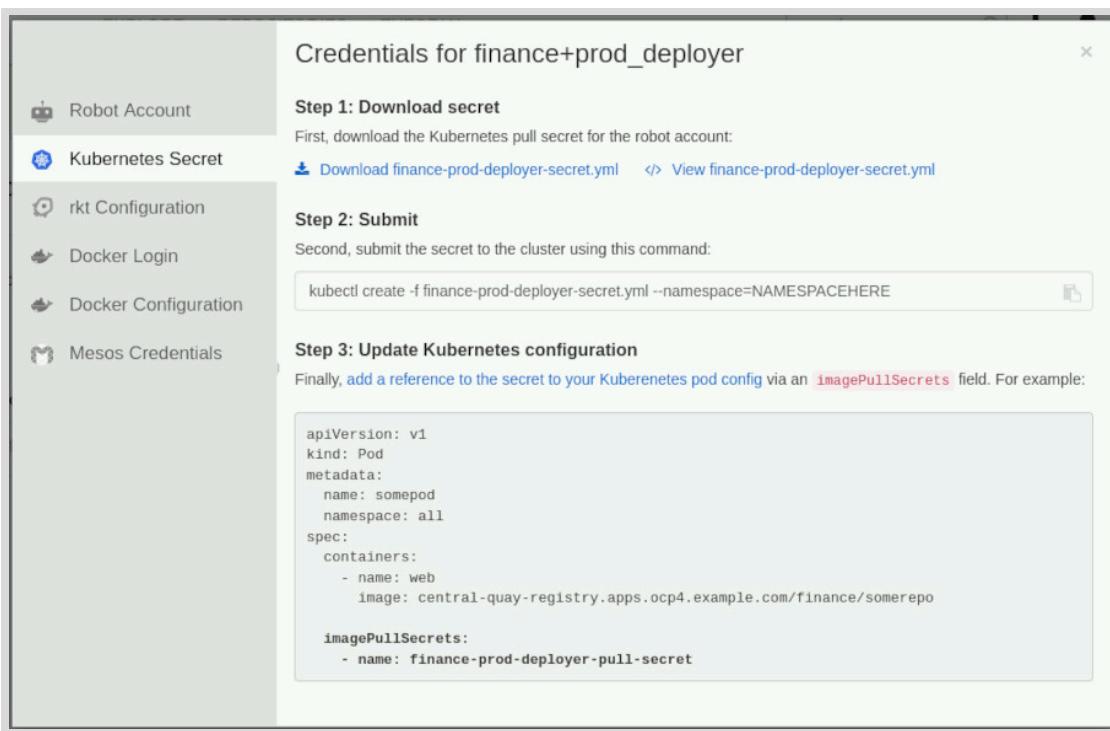
- 3.2. Click **REPOSITORIES**, then click **finance** to go to the organization page.

- 3.3. Click **Robot Accounts**, then click **Create Robot Account**.
- 3.4. Type **prod\_deployer** as the robot account name, leave the description blank, then click **Create robot account**.
- 3.5. On the page that opens, select the **Read** permission over the **budget-app-dev** repository and the **Write** permission over the **budget-app** repository, and then click **Add permissions**.

- 3.6. Click **finance+prod\_deployer** to view the credentials for the robot account. Store the token in a `/home/student/prod_deployer_token` file. This step is necessary so the lab script can use the API to grade your work



3.7. Click **Kubernetes Secret** to view instructions to use the robot account in Kubernetes.



3.8. Click the download icon to download the `finance-prod-deployer-secret.yml` file.

In the confirmation window, click **Save File** and then click **OK**. The browser downloads the file to the `/home/students/Downloads` directory.

4. Prepare the managed clusters for access to the central Quay registry. Create the `budget-app` namespace, and the `finance-prod-deployer-pull-secret` pull secret from the robot account on the production and non-production clusters. The API server address of the non-production cluster is `https://api.ocp4.example.com:6443`. The API server address of the production cluster is `https://api.ocp4-mng.example.com:6443`.

4.1. Log in to the `local-cluster` non-production cluster.

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

4.2. Create the budget-app namespace.

```
[student@workstation quayacm-review]$ oc create namespace budget-app
namespace/budget-app-dev created
```

4.3. Use the oc create command to create the secret in the budget-app namespace.

```
[student@workstation quayacm-review]$ oc create -f \
~/Downloads/finance-prod-deployer-secret.yml \
-n budget-app
secret/finance-prod-deployer-pull-secret created
```

4.4. Create the same namespace and pull secret on the managed-cluster production cluster.

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

```
[student@workstation quayacm-review]$ oc create namespace budget-app
namespace/budget-app created
```

```
[student@workstation quayacm-review]$ oc create -f \
~/Downloads/finance-prod-deployer-secret.yml \
-n budget-app
secret/finance-prod-deployer-pull-secret created
```

5. Fork the do480-apps course GitHub repository at <https://github.com/redhattraining/do480-apps/>. This repository contains starter YAML files to build the application. Modify the deployment to use the central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev:1.0 container image and the pull secret.

- 5.1. From the workstation machine, navigate to the do480-apps repository at <https://github.com/redhattraining/do480-apps/>.
- 5.2. In the upper-right corner of the GitHub repository page, click **Fork**, and then click **Create fork** to create a fork.

**Note**

If you have already performed this exercise or made other changes to the fork, then you must delete your fork to avoid problems during this exercise.

To delete the fork, navigate to <https://github.com/your-fork-name/DO480-apps/>. Click **Settings**. Click **Delete this repository** and enter the name of the repository to confirm the deletion.

- 5.3. In your GitHub fork, navigate to the deployment. Click **quayacm-review** and then click **deployment.yaml** to access the file. Click the pencil icon to edit the YAML file.
- 5.4. Edit the deployment to match the following content.

```
...output omitted...
spec:
 containers:
 -
 image: central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev:1.0
 name: budget-app
 ports:
 - containerPort: 8080
 name: budget-app
 imagePullSecrets:
 - name: finance-prod-deployer-pull-secret
```

**Note**

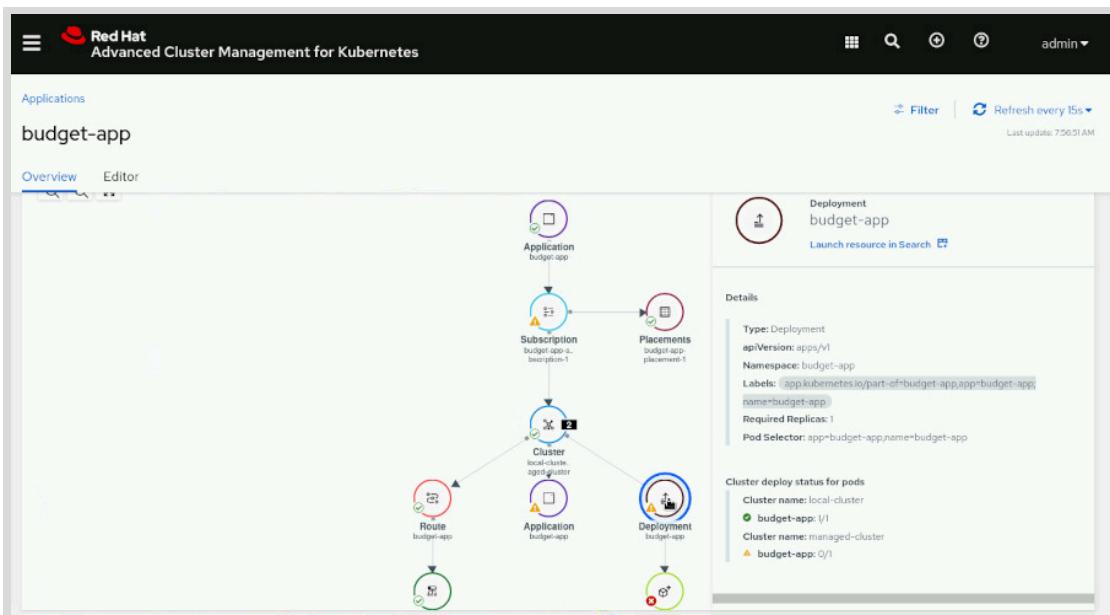
The `- image: central...dev:1.0` statement must be a single line.

Scroll to the bottom of the page and click **Commit changes**.

6. Deploy the **budget-app** application to all the OpenShift managed clusters using RHACM. The application fails on production clusters because it is using an image from a forbidden registry.
  - 6.1. From the **workstation** machine, navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>.
  - 6.2. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password.
  - 6.3. On the left navigation bar, navigate to **Applications**, click **Create application**, and then click **Subscription**.
  - 6.4. Fill the form with the following values.

| Field            | Value                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------|
| Name             | budget-app                                                                                            |
| Namespace        | budget-app                                                                                            |
| Repository types | Git                                                                                                   |
| URL              | <a href="https://github.com/your_fork/D0480-apps.git">https://github.com/your_fork/D0480-apps.git</a> |
| Branch           | main                                                                                                  |
| Path             | quayacm-review                                                                                        |

- 6.5. Scroll down and expand **Select clusters to deploy to** to configure a placement rule for the application. Select the **Deploy to all online clusters and local cluster** option to create a placement rule that deploys the application to all clusters.
- 6.6. Click **Save**. The application starts to deploy in the **local-cluster** and **managed-cluster** clusters.
- 6.7. The console redirects you to the details of the new application. On the **Topology** page, wait about three or four minutes for the deployments to finish. Then, click the **Deployment** node in the application topology view.



The **Cluster deploy status for pods** table indicates that there is one correct pod belonging to the **budget-app** deployment in the **local-cluster** cluster. There is another failing pod in the same deployment, but in the **managed-cluster** cluster.

- 6.8. Log in again to the **managed-cluster** cluster, which is tagged with the **env=production** label.

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

- 6.9. Locate the reason of the failing pod by searching the namespace events.

```
[student@workstation quayacm-review]$ oc get events -n budget-app | grep Failed
10m Warning Failed pod/budget-app-85bbf4585c-skls
Failed to pull image "central-quay-registry.apps.ocp4.example.com/finance/budget-
app-dev:1.0": rpc error: code = Unknown desc = Source image rejected: Running
image docker://central-quay-registry.apps.ocp4.example.com/finance/budget-app-
dev:1.0 is rejected by policy.
...output omitted...
```

The previous messages indicate that there is a policy that does not allow to use images from the requested source. Note that the message is referring to a policy from the `image.config.openshift.io/cluster` object, not to the RHACM policy.

7. Promote the `budget-app-dev:1.0` image to the allowed `budget-app` repository with the production robot account by using the `skopeo` command. Use the password token you saved in the `/home/student/prod_deployer_token` file.

- 7.1. Read the `prod_deployer` token from the `/home/student/prod_deployer_token` file.

```
[student@workstation quayacm-review]$ cat /home/student/prod_deployer_token
8H711MZKS6LWU0467RXG5P8ZMCLBX1RODR4R1U78UQC662T9X2NOK5WFMR2AKCQ
```

- 7.2. Log in to Quay by using the `skopeo` command. Use the token from the previous step.

```
[student@workstation quayacm-review]$ skopeo login \
-u=finance+prod_deployer \
-p=8H711MZKS6LWU0467RXG5P8ZMCLBX1RODR4R1U78UQC662T9X2NOK5WFMR2AKCQ \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
...output omitted...
```

- 7.3. Promote the image to the `budget-app` repository by using the `skopeo` command.

```
[student@workstation quayacm-review]$ skopeo copy \
docker://central-quay-registry.apps.ocp4.example.com/finance/budget-app-dev:1.0 \
docker://central-quay-registry.apps.ocp4.example.com/finance/budget-app:production
...output omitted...
Storing signatures
```



### Note

The `docker://...dev:1.0` and `docker://...app:production` statements must each be on a single line.

8. Modify the budget-app application in your GitHub account to use the central-quay-registry.apps.ocp4.example.com/finance/budget-app:production image.
  - 8.1. In your GitHub fork, navigate to the deployment. Click quayacm-review and then click deployment.yaml to access the file. Click the pencil icon to edit the YAML file.
  - 8.2. Edit the deployment to match the following content.

```
...output omitted...
spec:
 containers:
 -
 image: central-quay-registry.apps.ocp4.example.com/finance/budget-app:production
 name: budget-app
 ports:
 - containerPort: 8080
 name: budget-app
 imagePullSecrets:
 - name: finance-prod-deployer-pull-secret
```



### Note

The - image: central...app:production statement must be a single line.

Scroll to the bottom of the page and then click **Commit changes**.

- 8.3. In the RHACM console, click **Applications**, and then click **budget-app**. Click **Sync** to update the application with the last commit in the repository.

- 8.4. After three or four minutes, test that the deployment in the production cluster is ready.

```
[student@workstation quayacm-review]$ oc get pods -n budget-app
NAME READY STATUS RESTARTS AGE
budget-app-64b4f9d44c-lrgfj 1/1 Running 0 2m
```

8.5. Change to the /home/student directory.

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

## Evaluation

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

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

## Finish

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

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

This concludes the section.

# Summary

---

- Red Hat Quay.io and Red Hat Quay address common problems associated with using images registries in a multicluster and multicloud environment.
- The Quay deployment flexibility allows you to choose different architectures and dependencies between registries, but it is a good practice to have a central Quay registry.
- Red Hat OpenShift Container Platform provides a custom resource definition object to control the registries that the cluster can use.
- The Quay bridge operator automates the Quay integration with OpenShift clusters, but you can perform the integration from others automation platforms.
- Red Hat Advanced Cluster Management for Kubernetes (RHACM) simplifies the configuration of a cluster fleet by using its governance features.



## Chapter 8

# Installing and Configuring RHACS

### Goal

Install and configure Red Hat Advanced Cluster Security for Kubernetes (RHACS) and learn how it can help organizations with security in multicluster environments.

### Objectives

- Identify security concerns in Kubernetes multicluster environments and potential fixes offered by Red Hat Advanced Cluster Security for Kubernetes (RHACS).
- Install and configure Red Hat Advanced Cluster Security for Kubernetes (RHACS) and become familiar with the RHACS architecture, installation methods, and available customizations.
- Import managed clusters into Red Hat Advanced Cluster Security for Kubernetes (RHACS) and retrieve basic security information from the cluster fleet using the RHACS console.

### Sections

- Security Considerations in Kubernetes Multicluster Environments (and Quiz)
- Installing and Configuring RHACS (and Guided Exercise)
- Importing Managed Clusters into RHACS (and Guided Exercise) (and Quiz)

# Security Considerations in Kubernetes Multicloud Environments

---

## Objectives

- Identify security concerns in Kubernetes multicloud environments and potential fixes offered by Red Hat Advanced Cluster Security for Kubernetes (RHACS).

## Security Considerations in Kubernetes

As companies continue adopting cloud-native application models, more critical workloads run in Kubernetes.

Additionally, teams developing cloud-native applications can use a DevOps approach, where the development and operations teams work together following agile principles. By adopting this DevOps model, companies can release new versions of their applications more often.

The combination of using a DevOps approach and the use of containerized applications has security advantages due to the isolation of the processes, but requires a different approach. Development and operations teams who are not familiar with cloud-native application models might not be familiar with Kubernetes security and the security of the platform where the applications are deployed.

To improve aspects of security in a DevOps scenario, the DevSecOps model focuses on integrating the security from the early stages of the software development lifecycle.

A DevSecOps model must include software development security, such as the following considerations:

- Security in the design phase
- Static secure code analysis and best practices
- Container image vulnerability scanning before deploying to production
- Secure access to version control systems (VCS)
- Secure access to the continuous integration system (CI)
- Automatic build security testing
- Defensive security, or implementing a Blue Team approach

Additionally, the model must include operational security, such as the following considerations:

- Continuous container image vulnerability scanning for deployments in production
- Secure access to the continuous deployment system (CD)
- Automatic security testing of production deployments
- Offensive security by implementing a Red Team
- Dynamic security code analysis during run time
- Incident response planning

The previous security considerations apply to every software development company. However, there are some extra security challenges in Kubernetes environments:

- You need extra knowledge to set up users, groups, service accounts, and the role-based access control (RBAC) configuration.
- You must secure the VCS repositories containing the platform configuration.

- You must ensure that container image sources are reliable.
- You need expertise in software defined networks (SDN).
- You must secure many new attack vectors, such as container image vulnerabilities, application vulnerabilities, Kubernetes secrets management, and more.

Although IT professionals are increasingly familiar with Kubernetes, security remains one of the main challenges for adopting and running containerized applications in production Kubernetes clusters.

## Addressing Security in Kubernetes with RHACS

Red Hat Advanced Cluster Security for Kubernetes (RHACS) simplifies most of the challenges discussed in the preceding paragraphs.

RHACS features integrate with DevSecOps models to protect the application workloads and the Kubernetes infrastructure. RHACS has many available features, such as the following considerations:

- Vulnerability management
- Configuration management
- Risk profiling
- Threat detection
- Incident response
- Compliance analysis and reporting
- Network traffic segmentation
- Multicloud status visibility

## Vulnerability Management with RHACS

The RHACS vulnerability management scanner decomposes container images by layers and identifies the software components, whether installed by a package manager or as a programming software dependency.

The RHACS vulnerability scanner provides considerable vulnerability information, but it can be integrated with third-party scanners to take advantage of potential new features.

RHACS can use the vulnerability scanner results to control which images are used by enforcing policies based on the common vulnerability scoring system (CVSS) calculated score.

You can access the vulnerability management dashboard from the **Vulnerability Management > Dashboard** menu. The dashboard provides the following vulnerability information:

- Top risky deployments by CVE count and CVSS score
- Top riskiest images by CVEs
- Frequently violated policies
- Recently detected vulnerabilities
- Most common vulnerabilities
- Deployments with most severe policy violations
- Vulnerabilities by cluster, namespace, deployment, or component
- Vulnerabilities in the OS of the nodes
- Fixable and non-fixable CVEs
- Base OS image layer
- Inactive images vulnerabilities

The RHACS **Vulnerability Management** menu also includes the **Risk Acceptance and Reporting** pages.

You can review the alerts on the **Risk Acceptance** page. If the alert is a false positive, the security risk is low, or the vulnerability is not exploitable, then you can dismiss the alert.

On the **Reporting** page, you can create automated custom reports and send them to a list of recipients.

## Configuration Management with RHACS

To build a secure Kubernetes deployment and avoid security breaches, administrators must know Kubernetes concepts such as secrets, service accounts, and RBAC.

RHACS automatically analyzes RBAC configurations to identify users, groups, and service accounts with elevated privileges. RHACS also monitors the RBAC settings for overly permissive access to secrets, access from deployments to the Kubernetes API server, and secrets passed in as environment variables.

RHACS protects against potentially insecure misconfigurations by creating policies that determine whether privileged containers are allowed to run. These policies limit the resources that can mount a path on the host and their file system permissions and monitor RBAC settings.

The RHACS **Configuration Management** page provides access to the following dashboards:

- Policy violations by severity
- Center for Internet Security (CIS) Docker and Kubernetes scan results
- Those users with the most cluster administrator roles
- Secrets most used across deployments

The RHACS dashboard provides a dedicated via to the following features:

- RHACS policies
- CIS controls
- Application and infrastructure elements such as clusters, namespaces, nodes, deployments, images, and secrets
- RBAC visibility and configuration, including users and groups, service accounts, and roles

## Risk Profiling with RHACS

Risk profiling means determining if a risk is acceptable or not. To make this determination, you must correlate vulnerabilities, their severity, how easy it might be to exploit the vulnerability, whether the service is exposed, and others.

On the **Risk** page, RHACS scores and sorts all deployments across all clusters considering many factors. This risk profile helps you decide which deployments require immediate attention.

## Threat Detection with RHACS

RHACS implements predictive and threat-based protection for running containers. With predictive protection, RHACS identifies abnormal process executions or network connections. With threat-based protection, RHACS detects suspicious processes such as `netcat` or `nmap` processes.

With RHACS, you can create policies to detect suspicious runtime behaviors. You can configure policies to trigger violations and alerts or perform enforcing actions automatically.

RHACS analyzes all the processes running in containers and creates process baselines for every container type in a deployment. New processes that differ from these baselines are considered a risk but do not produce violations automatically.

You can manage container process baselines from the **Risk** page. Click on a deployment, and then click the **Process Discovery** tab. Violations are displayed on the **Violations** page.

After confirming that the process baselines follow expected behavior, you can lock the process baseline so that RHACS triggers a violation event if it detects an abnormal process.

## Incident Response with RHACS

Incident response is how administrators react to security events. In a containerized environment, incidents are typically resolved by removing the container with anomalous behavior as soon as the incident is detected.

RHACS provides details about malicious processes and network activity for post-mortem analysis. You can export these details to centralized Security Information and Event Management (SIEM) systems, such as Splunk or Sumo Logic, and notifier systems, such as Slack, email, Pagerduty, Jira, and others.

In addition to collecting and exporting this data, you can create policies to prevent risky deployments automatically and respond to runtime security incidents.

RHACS logs security incidents and displays them on the **Violations** page.

## Compliance Analysis and Report with RHACS

Compliance standards are security guidelines for configuring systems. At the time of publication, RHACS offers compliance analysis using six different standards:

- CIS Docker [<https://www.cisecurity.org/benchmark/docker>]
- CIS Kubernetes [<https://www.cisecurity.org/benchmark/kubernetes>]
- HIPAA [<https://www.hhs.gov/hipaa/for-professionals/security/guidance/cybersecurity/index.html>]
- PCI DSS [<https://www.pcisecuritystandards.org/>]
- NIST 800-190 [<https://csrc.nist.gov/publications/detail/sp/800-190/final>]
- NIST 800-53 [<https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final>]

From the **Compliance** page, you can run compliance scans and export detailed compliance reports.

RHACS provides detailed information about each compliance report control and guidance about how to make the system compliant.

## Network Traffic Segmentation with RHACS

In Kubernetes, network policies implement network traffic segmentation. Network traffic segmentation allows isolating applications from each other and limiting network traffic to known sources and destinations. This segmentation prevents incoming attacks, lateral movement between applications, and data breaches.

From the **Network Graph** page, access the **Network Policy Simulator**. In the network policy simulator, RHACS generates suggested policies based on the observed network flows. You can apply network policies and analyze the behavior of the network traffic before applying the changes to the network policies in a production environment. The network policy simulator also provides diagrams with live traffic flows to evaluate the implementation of network policies.

## Multicloud Status Visibility with RHACS

In a multicloud environment, security operators need visibility into the applications and configurations in the cluster fleet. RHACS provides visibility for security operators with the following centralized views:

- Risk prioritization
- Runtime activity and forensics
- Vulnerability threats
- Configuration risk
- Deployment and Kubernetes metadata
- Global search capabilities
- SIEM notifications
- SSO and scoped access controls



### References

For more information, refer to the *Red Hat Advanced Cluster Security for Kubernetes Documentation* at

<https://docs.openshift.com/acs/3.69/welcome/index.html>

#### **Operationalize Red Hat Advanced Cluster Security for Kubernetes**

<https://redhat.hightspot.com/items/6180359dc4121d1a49ae2e5f#1>

#### **Most common Kubernetes security issues and concerns to address**

<https://cloud.redhat.com/blog/most-common-kubernetes-security-issues-and-concerns-to-address>

#### **Kubernetes adoption, security, and market trends report 2021**

<https://www.redhat.com/en/resources/kubernetes-adoption-security-market-trends-2021-overview>

## ► Quiz

# Security Considerations in Kubernetes Multicloud Environments

Choose the correct answers to the following questions:

- ▶ 1. **Which of the following modern development models ensures that security is present in all the stages of the software development lifecycle?**
  - a. DevOps
  - b. DevSecOps
  - c. Scrum
  - d. Waterfall
  - e. V-Model
  
- ▶ 2. **Which three of the following challenges are a result of using Kubernetes? (Choose three.)**
  - a. Container image sources are reliable.
  - b. Security vulnerabilities contain current patches.
  - c. The operating system of the platform supporting the applications contains the most current security patches.
  - d. Software defined networks (SDN) require extra knowledge.
  - e. Deploying Kubernetes secrets securely.
  
- ▶ 3. **Which four of the following features is provided by Red Hat Advanced Cluster Security for Kubernetes (RHACS)? (Choose four.)**
  - a. Vulnerability management
  - b. Time capsule
  - c. Configuration management
  - d. Risk profiling
  - e. Decommissioning insecure Kubernetes clusters
  - f. Threat detection
  
- ▶ 4. **Which three of the following information dashboards are provided by the RHACS vulnerability management page? (Choose three.)**
  - a. Frequently violated policies
  - b. Deployments with the most severe policy violations
  - c. Most used operating systems across the cluster fleet
  - d. Top risky deployments by CVE count and CVSS score
  - e. A list of IP addresses blocked due to security attack attempts

► 5. Which three of the following information dashboards are provided by the RHACS configuration management page? (Choose three.)

- a. Inactive images vulnerabilities
- b. Policy violations by severity
- c. Center for Internet Security (CIS) Docker and Kubernetes scans results
- d. Secrets most used across deployments
- e. Users with most deployments across the cluster fleet

## ► Solution

# Security Considerations in Kubernetes Multicloud Environments

Choose the correct answers to the following questions:

- ▶ 1. **Which of the following modern development models ensures that security is present in all the stages of the software development lifecycle?**
  - a. DevOps
  - b. DevSecOps
  - c. Scrum
  - d. Waterfall
  - e. V-Model
  
- ▶ 2. **Which three of the following challenges are a result of using Kubernetes? (Choose three.)**
  - a. Container image sources are reliable.
  - b. Security vulnerabilities contain current patches.
  - c. The operating system of the platform supporting the applications contains the most current security patches.
  - d. Software defined networks (SDN) require extra knowledge.
  - e. Deploying Kubernetes secrets securely.
  
- ▶ 3. **Which four of the following features is provided by Red Hat Advanced Cluster Security for Kubernetes (RHACS)? (Choose four.)**
  - a. Vulnerability management
  - b. Time capsule
  - c. Configuration management
  - d. Risk profiling
  - e. Decommissioning insecure Kubernetes clusters
  - f. Threat detection
  
- ▶ 4. **Which three of the following information dashboards are provided by the RHACS vulnerability management page? (Choose three.)**
  - a. Frequently violated policies
  - b. Deployments with the most severe policy violations
  - c. Most used operating systems across the cluster fleet
  - d. Top risky deployments by CVE count and CVSS score
  - e. A list of IP addresses blocked due to security attack attempts

► 5. Which three of the following information dashboards are provided by the RHACS configuration management page? (Choose three.)

- a. Inactive images vulnerabilities
- b. Policy violations by severity
- c. Center for Internet Security (CIS) Docker and Kubernetes scans results
- d. Secrets most used across deployments
- e. Users with most deployments across the cluster fleet

# Installing and Configuring RHACS

## Objectives

- Install and configure Red Hat Advanced Cluster Security for Kubernetes (RHACS) and become familiar with the RHACS architecture, installation methods, and available customizations.

## RHACS Components

Red Hat Advanced Cluster Security for Kubernetes (RHACS) is composed of centralized services and secured cluster services. Centralized services run on a specific cluster and receive data from multiple secured cluster services. Secured cluster services require a small amount of resources compared to centralized services. Because RHACS uses this distributed architecture model, it supports large scale deployments that minimize impacts to the underlying infrastructure.

### RHACS Centralized Services

The RHACS centralized services run on a single cluster and include two main components, Central and Scanner.

#### Central

Central is the RHACS component that provides centralized services and the application management interface.

Central handles data persistence, API interactions, and user interface access. You can secure multiple Red Hat OpenShift or Kubernetes clusters with a single Central instance.

#### Scanner

Scanner is a Red Hat developed vulnerability scanner. This service analyzes all the layers of the container images to find known Common Vulnerabilities and Exposures (CVEs). Scanner also identifies vulnerabilities in packages installed by package managers, language-level dependencies, and application-level dependencies.

To create all the elements of the RHACS centralized services, you must create a `Central` Kubernetes custom resource (CR). The RHACS operator defines this resource.

By default, RHACS installs a lightweight version of Scanner on each secured cluster to analyze the images in the integrated container registry.

If you do not want to install the lightweight Scanner version on the secured clusters, then you can change this setting when you create the `Central` CR.

## RHACS Secured Cluster Services

The RHACS secured cluster services run on each cluster secured by RHACS and include Sensor, Collector, Scanner, and Admission Controller.

#### Sensor

Sensor analyzes and monitors the cluster. This service interacts with the OpenShift or Kubernetes API server for policy detection and enforcement. Sensor coordinates with Collector.

**Collector**

Collector analyzes and monitors container activity on the nodes. This service collects information about network activity and the container runtime and sends the data to Sensor.

**Scanner**

Scanner, if installed, analyzes container images from the integrated container registry.

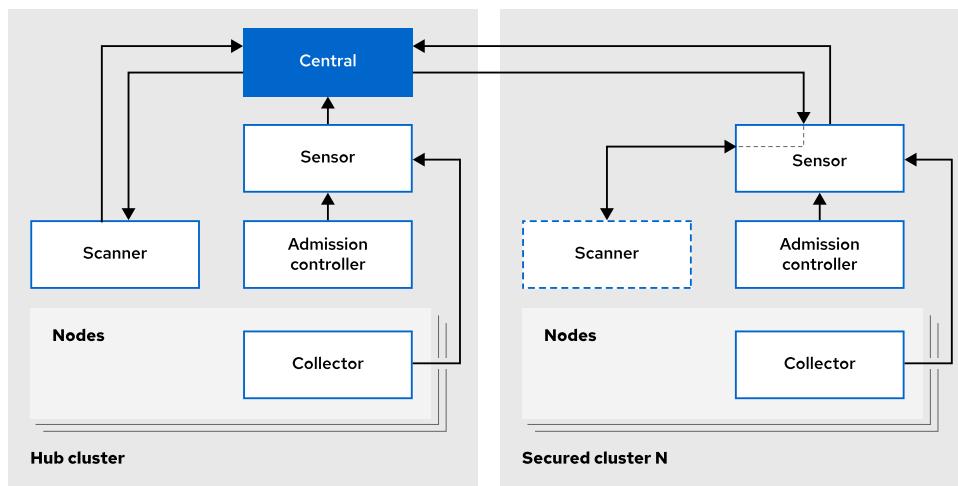
**Admission Controller**

This service prevents the creation of workloads that violate RHACS security policies.

To create all of the elements of the RHACS secured cluster services, you must create a `SecuredCluster` Kubernetes CR. The RHACS operator defines this resource.

## RHACS Architecture

The following example diagram shows a multicluster RHACS deployment and its components:



The Central service only runs in the hub cluster.

The Scanner service runs in the hub cluster; optionally, Scanner can run in the secured clusters to enable scanning of images in the integrated container registry.

The Sensor, Collector, and Admission Controller services run on each cluster secured by RHACS.

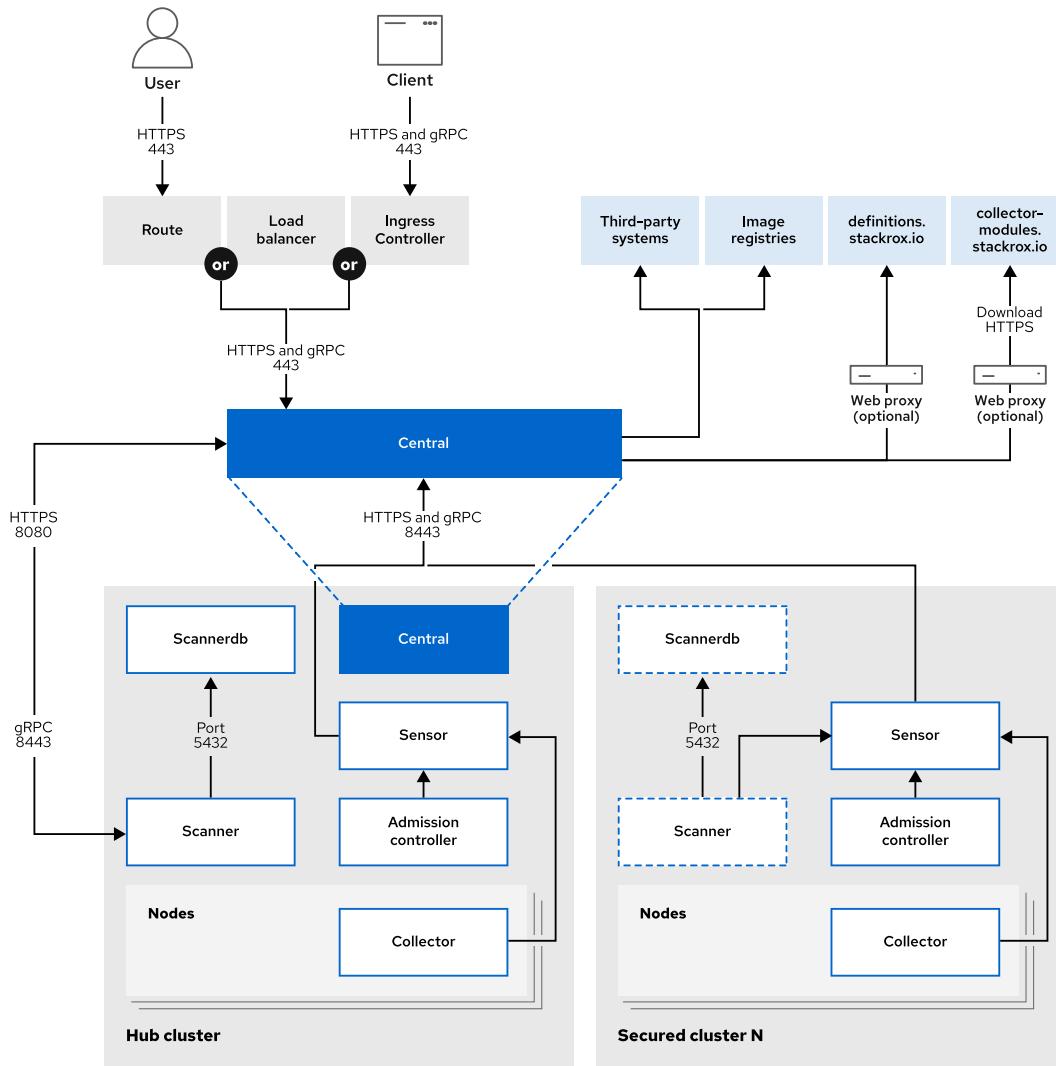
In the preceding diagram, the hub cluster is also a secured cluster.

## Installing RHACS

To install RHACS, you must satisfy the following general prerequisites:

- Red Hat OpenShift version 4.5 or later
- Cluster nodes with a supported operating system, such as Amazon Linux, CentOS, Container-Optimized OS from Google, Red Hat Enterprise Linux CoreOS, Debian, Red Hat Enterprise Linux, or Ubuntu
- At least 4 CPU cores and at least 3 GiB of RAM for the node running the centralized services
- At least 2 CPU cores and at least 3 GiB of RAM for nodes running the secured cluster services
- Persistent block storage; Rados Block Device (RBD) storage is recommended
- Permissions to configure deployments in the OpenShift or Kubernetes clusters
- Access to the Red Hat Container Registry at `registry.redhat.io`

RHACS requires communication with external entities such as image registries, CI/CD pipelines, SIEM systems, and the `stackrox.io` servers. The following diagram represents the prerequisites for network communication between RHACS internal and external services:



To run Central, you must satisfy the following specific prerequisites:

- Allow outbound network traffic to `definitions.stackrox.io` and `collector-modules.stackrox.io`
- At least 1.5 CPU cores
- At least 4 GiB of RAM
- At least 100 GiB of storage

To run Scanner, your cluster must have the following available resources:

- At least 1 CPU core
- At least 1.5 GiB of RAM
- At least 30 GiB of storage

**Note**

The resource requirements above are per replica. Red Hat recommends running two replicas of Scanner.

To run the Scanner database, your cluster must have the following available resources:

- At least 0.2 CPU cores
- At least 200 MiB of RAM

**Note**

The Scanner database requires variable amounts of available persistent storage.

To run Sensor, your cluster must have the following available resources:

- At least 1 CPU core
- At least 1 GiB of RAM

To run Admission Controller, your cluster must have the following available resources:

- At least 0.05 CPU cores
- At least 100 MiB of RAM

**Note**

The resource requirements above are per replica. The Admission Controller runs three replicas.

You can install RHACS using the Advanced Cluster Security for Kubernetes operator, Helm charts, or the `roxctl` command. Red Hat recommends using the Advanced Cluster Security for Kubernetes operator if available in the supporting infrastructure.

Also, you can install RHACS using Red Hat Advanced Cluster Management for Kubernetes (RHACM) policies. The RHACM policy for installing RHACS uses the Advanced Cluster Security for Kubernetes operator.

See the references section for a list of installation methods for different platforms.

## Installing RHACS with the Advanced Cluster Security for Kubernetes Operator

You can install the Advanced Cluster Security for Kubernetes operator from OperatorHub or by using the `oc` command. See the references section for instructions on how to install an operator using the `oc` command.

After you install the operator, install the centralized services in the hub cluster by creating the `Central` CR. You can create a `Central` CR from the **Installed Operators** page in the OpenShift web console, or using a YAML file.

Red Hat recommends installing the `Central` CR in the `stackrox` namespace for compatibility with pre-existing RHACS policies.

The following example shows a `Central` CR definition YAML file.

```

apiVersion: platform.stackrox.io/v1alpha1
kind: Central
metadata:
 name: stackrox-central-services
 namespace: stackrox
spec:
 central:
 exposure:
 loadBalancer:
 enabled: false
 port: 443
 nodePort:
 enabled: false
 route:
 enabled: true
 persistence:
 persistentVolumeClaim:
 claimName: stackrox-db
 egress:
 connectivityPolicy: Online
 scanner:
 analyzer:
 scaling:
 autoScaling: Enabled
 maxReplicas: 5
 minReplicas: 2
 replicas: 3
 scannerComponent: Enabled

```

You can configure settings for Central, Scanner, and other services when you create the Central CR. For a complete list of Central configuration options, refer to the following link [[https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#central-configuration-options-operator\\_install-ocp-operator](https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#central-configuration-options-operator_install-ocp-operator)].

## Installing RHACS Using Helm Charts

In platforms where the operator installation model is not available, Red Hat recommends installing RHACS by using Helm charts.

To install RHACS by using Helm charts, use the Helm CLI v3.2 or later. The Helm charts for installing RHACS are hosted at the following URL: <https://mirror.openshift.com/pub/rhacs/charts/>.

The RHACS Helm repository contains two charts. Use the `central-services` chart for installing the centralized components. Use the `secured-cluster-services` chart for installing the per-cluster components such as Sensor and Admission Controller, and the per-node components, such as Collector.

Before installing the secured cluster services, you must generate the `init bundle` and import the Central certificates in all the secured clusters.

You can generate the `init bundle` from the RHACS web console.

Alternatively, you can generate an API token from the RHACS web console and generate the bundle by using the `roxctl` command. This approach is useful for automating the addition of secured clusters from the command line.

For more information about generating API tokens from the RHACS web console, visit the following link [[https://docs.openshift.com/acs/3.69/cli/getting-started-cli.html#cli-authentication\\_cli-getting-started](https://docs.openshift.com/acs/3.69/cli/getting-started-cli.html#cli-authentication_cli-getting-started)].

For more information about installing RHACS by using Helm charts, visit the following link [[https://docs.openshift.com/acs/3.69/installing/installing\\_helm/install-helm-quick.html](https://docs.openshift.com/acs/3.69/installing/installing_helm/install-helm-quick.html)]

## Installing RHACS Using the Command Line

Red Hat recommends installing RHACS by using the `roxctl` command only if no other method is compatible with your environment.

For detailed steps to install RHACS by using the `roxctl` command, visit the following link [<https://docs.openshift.com/acs/3.69/installing/install-quick-roxctl.html>].

## Installing RHACS Using RHACM Policies

When RHACM is available, you can create RHACM policies to deploy RHACS in the cluster fleet. This approach ensures that every cluster in the fleet is secured by RHACS.

To deploy RHACS, you must create two policies in RHACM, one for the centralized services and another for the secured cluster services. The policy for installing the centralized services should apply to the hub cluster. The policy for installing the secured cluster services should apply to the clusters that you want RHACS to secure. You can achieve this separation by using a the `clusterSelector` parameter of the `PlacementRule` object. You can find more information about how to create policies with placement rules in *RHACM Policy Overview*.

You can find example YAML files to create an RHACM policy to install the centralized services at the following link [<https://github.com/stolostron/policy-collection/blob/main/community/CM-Configuration-Management/policy-acss-operator-central.yaml>].

You can find example YAML files to create an RHACM policy to install the secured cluster services at the following link [<https://github.com/stolostron/policy-collection/blob/main/community/CM-Configuration-Management/policy-acss-operator-secured-clusters.yaml>].

Before installing the secured cluster services, you must generate the `init` bundle and import the Central certificates in all the secured clusters.

## Uninstalling RHACS

If you installed RHACS on Kubernetes, use the `kubectl` command instead of the `oc` command.

Security context constraints (SCCs) are an OpenShift feature, so the `oc delete scc` command does not apply to Kubernetes clusters.

The following steps uninstall RHACS:

1. Delete the `rhacs-operator` namespace
2. Delete the namespace where you installed the `Central` and `SecuredCluster` resources, if different from `stackrox`

If you installed RHACS on Red Hat OpenShift, run the following commands:

```
[user@demo ~]$ oc get clusterrole,clusterrolebinding,role,rolebinding,psp \
-o name | grep stackrox | xargs oc delete --wait
```

```
[user@demo ~]$ oc delete scc -l "app.kubernetes.io/name=stackrox"
```

```
[user@demo ~]$ oc delete ValidatingWebhookConfiguration stackrox
```

```
[user@demo ~]$ for namespace in \
$(oc get ns | tail -n +2 | awk '{print $1}');\
do oc label namespace $namespace metadata.stackrox.io/id-; \
oc label namespace $namespace metadata.stackrox.io/name-; \
oc annotate namespace $namespace modified-by.stackrox.io/namespace-lab
```

If you installed RHACS on Kubernetes, run the following commands:

```
[user@demo ~]$ kubectl get clusterrole,clusterrolebinding,role,rolebinding,psp \
-o name | grep stackrox | xargs kubectl delete --wait
```

```
[user@demo ~]$ kubectl delete ValidatingWebhookConfiguration stackrox
```

```
[user@demo ~]$ for namespace in $(kubectl get ns | tail -n +2 | awk '{print $1}');
\
do kubectl label namespace $namespace metadata.stackrox.io/id-; \
kubectl label namespace $namespace metadata.stackrox.io/name-; \
kubectl annotate namespace $namespace modified-by.stackrox.io/namespace-lab
```

If you installed RHACS by using RHACM policies, remember that deleting the policy does not remove the Advanced Cluster Security for Kubernetes operator. You must follow the preceding steps in each cluster to uninstall RHACS.

## RHACS Settings

After installing RHACS, you can apply a number of configurations depending on your needs. These configurations are described in the following list.

### Add trusted Certificate Authorities (CAs)

If you are using an enterprise CA or self-signed certificates, then you can add the root certificate of your trusted CA to RHACS. You can also add the root certificate of the CA of an image registry.

### Add custom TLS certificates

If the TLS certificate is trusted by any of the CAs configured in RHACS, users and API clients connecting to Central do not have to bypass the untrusted certificate warnings. You can add a custom TLS certificate during the installation or on an existing RHACS deployment.

### Add security notices

Security notices are messages that users see when they log in to RHACS. You can set up organization-wide security notices that appear on the RHACS web console.

### Enable offline mode

You can enable the RHACS offline mode in disconnected clusters.

To install the Advanced Cluster Security for Kubernetes operator in disconnected environments, mirror the OperatorHub content into a local registry as explained in the documentation [[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/operators/index#olm-restricted-networks](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/operators/index#olm-restricted-networks)].

If you are using the Helm installation method, then import the images into your corporate registry by downloading the individual images from the Red{nsbp}Hat registry.

In offline mode, you can download Scanner definitions and kernel support packages from the StackRox Community site and upload them to Central and Scanner by using the `roxtcl` command.

### **Configure alert data retention**

To save storage costs, you can configure RHACS to automatically delete past alerts. When configured, RHACS deletes the alerts after a specified period of time.

### **Expose the RHACS portal over HTTP**

You can configure the RHACS portal to be exposed through an unencrypted HTTP server. This is useful for compatibility with ingress controllers, layer 7 load balancers, Istio, and others.

### **Configure automatic upgrades for secured clusters**

You can configure RHACS to automatically upgrade the secured clusters after updating the central services. This approach is useful for saving time upgrading RHACS in all the secured clusters and avoiding version mismatches. You can only use automatic upgrades if you installed RHACS by using the Advanced Cluster Security for Kubernetes operator.

### **Configure a proxy for external access**

You can configure RHACS to use a proxy if your network configuration restricts outbound traffic through proxies. Traffic between Central and Scanner, or other internal components, does not go through the proxy.

### **Configure endpoints**

You can create a configuration map to configure the RHACS exposed endpoints. With this configuration map, you can specify the protocols to use, change the default ports, specify the server certificates, configure a trusted CA, or specify if mutual TLS authentication is required.

### **Enable Prometheus monitoring**

You can configure the Sensor and Central services to expose metrics to Prometheus. You must create a network policy to allow ingress traffic to the exposed services.

Additionally, you can perform the following administrative tasks:

### **Reissue internal certificates**

Each component of RHACS uses an X.509 certificate to authenticate to other components. When the certificates are close to the expiration date, you must reissue them.

### **Generate a diagnostic bundle**

Red Hat support might request a diagnostic bundle for investigating issues in your RHACS installation. The diagnostic bundle contains the Central heap memory profile, RHACS system logs, YAML definitions for your RHACS components, the events from the namespace where RHACS runs, API endpoint statistics, information about the nodes, and general information about the product versions in your environment.

For more information about the available RHACS configurations, visit the following link [<https://docs.openshift.com/acs/3.69/welcome/index.html#configuring-red-hat-advanced-cluster-security-for-kubernetes>].



## References

For more information, refer to the *Red Hat Advanced Cluster Security for Kubernetes Product Architecture Documentation* at  
<https://docs.openshift.com/acs/3.69/architecture/acs-architecture.html>

For more information about the installation platforms and methods, visit  
<https://docs.openshift.com/acs/3.69/installing/acs-installation-platforms.html>

For more information about RHACS installation prerequisites, visit  
<https://docs.openshift.com/acs/3.69/installing/prerequisites.html>

For more information about installing operators using the command line, refer to the *Installing from OperatorHub using the CLI* section in the *Administrator tasks* chapter in the *Operators* guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/operators/index#olm-installing-operator-from-operatorhub-using-cli\\_olm-adding-operators-to-a-cluster](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/operators/index#olm-installing-operator-from-operatorhub-using-cli_olm-adding-operators-to-a-cluster)

For more information about installation methods for different platforms, refer to the *Installation platforms and methods* chapter in the *Installing* guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_advanced\\_cluster\\_security\\_for\\_kubernetes/3.69/html-single/installing/index#installation-platforms](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_security_for_kubernetes/3.69/html-single/installing/index#installation-platforms)

## ► Guided Exercise

# Installing and Configuring RHACS

- Install Red Hat Advanced Cluster Security for Kubernetes (RHACS) using an Red Hat Advanced Cluster Management for Kubernetes (RHACM) policy. You also configure RHACS to use Red Hat OpenShift Container Platform (RHOCP) authentication, enable audit logging, and verify that the configuration is correct.

## Outcomes

- Install the RHACS Operator using an RHACM policy.
- Install the RHACS Central using an RHACM policy.
- Configure the Rsyslog server using the RHACS Syslog integration.
- Configure the RHOCP Oauth server as an identity provider for RHACS.
- Verify that you can log in to RHACS using credentials from the Red Hat OpenShift ocp4 cluster.
- Verify that the audit logging configuration works as expected.

## 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 rhacs-install
```

- 1. Log in to the ocp4 cluster as the `admin` user and create a `rhacs-install` project.

- 1.1. Open a terminal and log in to the ocp4 cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 1.2. Create the `rhacs-install` project.

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

- 2. Deploy an RHACM policy named `policy-advanced-cluster-security-install` in the `rhacs-install` namespace. This policy installs the RHACS operator in the `rhacs-`

operator namespace in the ocp4 cluster. The ocp4 cluster is named local-cluster in RHACM.

2.1. Change to the ~/D0480/labs/rhacs-install/ directory.

```
[student@workstation ~]$ cd ~/D0480/labs/rhacs-install/
[student@workstation rhacs-install]$
```

2.2. Review the file named policy-rhacs-operator-install.yaml that contains an RHACM policy definition to install RHACS.

```
[student@workstation rhacs-install]$ cat policy-rhacs-operator-install.yaml
...output omitted...
spec:
 remediationAction: enforce
 ...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: v1
 kind: Namespace
 metadata:
 name: rhacs-operator
 - complianceType: musthave
 objectDefinition:
 apiVersion: v1
 kind: Namespace
 metadata:
 name: stackrox
 ...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: operators.coreos.com/v1
 kind: OperatorGroup
 metadata:
 name: rhacs-operator
 namespace: rhacs-operator
 ...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: operators.coreos.com/v1alpha1
 kind: Subscription
 metadata:
 name: rhacs-operator
 namespace: rhacs-operator
 spec:
 channel: latest
 installPlanApproval: Automatic
 name: rhacs-operator
 source: do480-catalog
 sourceNamespace: openshift-marketplace
 ...output omitted...
```

```
clusterSelector:
 matchExpressions:
 - {key: local-cluster, operator: In, values: ["true"]}
```

The policy creates the `rhacs-operator` namespace to install the RHACS operator, the `stackrox` namespace to install the Central resource, and installs the RHACS operator in the `local-cluster` cluster.

- 2.3. Use the `oc` command to create the `policy-advanced-cluster-security-install` policy in the `rhacs-install` namespace.

```
[student@workstation rhacs-install]$ oc create -f \
policy-rhacs-operator-install.yaml -n rhacs-install
policy.policy.open-cluster-management.io/policy-advanced-cluster-security-install
created
placementbinding.policy.open-cluster-management.io/binding-policy-advanced-
cluster-security-install created
placementrule.apps.open-cluster-management.io/placement-policy-advanced-cluster-
security-install created
```

- 2.4. From the `workstation` machine, open Firefox and navigate to the RHACM web console. The URL is `https://multicloud-console.apps.ocp4.example.com`.
- 2.5. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
- 2.6. In the left pane, click **Governance**. The dashboard shows the `policy-advanced-cluster-security-install` policy. Click `policy-advanced-cluster-security-install` and then click **Clusters** to check the status.
- 2.7. Verify that the **Clusters** tab shows three templates: `advanced-cluster-security-namespace`, `rhacs-operator-operator-group`, and `advanced-cluster-security-operator-subscription`.

| Cluster       | Compliance | Template                                        | Message                                                                                                                                                                                                                                                  | Last Report   | History                      |
|---------------|------------|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------------------------|
| local-cluster | Compliant  | advanced-cluster-security-operator-subscription | Compliant; notification - subscriptions [rhacs-operator] in namespace rhacs-operator found as specified, therefore this Object template is compliant <a href="#">View details</a>                                                                        | 2 minutes ago | <a href="#">View history</a> |
| local-cluster | Compliant  | rhacs-operator-operator-group                   | Compliant; notification - operatorgroups [rhacs-operator] in namespace rhacs-operator found as specified, therefore this Object template is compliant <a href="#">View details</a>                                                                       | 2 minutes ago | <a href="#">View history</a> |
| local-cluster | Compliant  | advanced-cluster-security-namespace             | Compliant; notification - namespaces [rhacs-operator] found as specified, therefore this Object template is compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant <a href="#">View details</a> | 2 minutes ago | <a href="#">View history</a> |



### Note

It takes 2-3 minutes for all three templates to show the **Compliant** status.

- 3. As the **admin** user, verify the RHACS operator installation by using the RHACM web console.
- 3.1. Click the search icon at the upper-right corner of the RHACM web console.
  - 3.2. Type the following parameters in the search bar to search for the **rhacs-operator** subscription.

| Field name | Value          |
|------------|----------------|
| kind       | subscription   |
| name       | rhacs-operator |

The search result shows the **rhacs-operator** subscription present in the **local-cluster**.

- 4. Deploy an RHACM policy named **policy-rhacs-operator-central** in the **rhacs-install** namespace. This policy installs the RHACS Central instance in the **stackrox** namespace in the **ocp4**. The **ocp4** cluster is named **local-cluster** in RHACM.
- 4.1. Review the file named **policy-rhacs-operator-central.yaml** with the definition of a RHACM policy to install RHACS Central.

```
[student@workstation rhacs-install]$ cat policy-rhacs-operator-central.yaml
...output omitted...
spec:
 remediationAction: enforce
 ...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: v1
 kind: Namespace
 metadata:
 name: stackrox
 ...output omitted...
 object-templates:
 - complianceType: musthave
```

```

objectDefinition:
 apiVersion: platform.stackrox.io/v1alpha1
 kind: Central
 metadata:
 namespace: stackrox
 name: stackrox-central-services
 spec:
...output omitted...
 clusterSelector:
 matchExpressions:
 - {key: local-cluster, operator: In, values: ["true"]}

```

The policy ensures that the `stackrox` namespace exists in the `local-cluster`. It also creates the RHACS `Central` resource in the `stackrox` namespace.

- 4.2. Use the `oc` command to create the `policy-advanced-cluster-security-central` policy in the `rhacs-install` namespace.

```

[student@workstation rhacs-install]$ oc create -f \
policy-rhacs-operator-central.yaml -n rhacs-install
policy.policy.open-cluster-management.io/policy-advanced-cluster-security-central
created
placementbinding.policy.open-cluster-management.io/binding-policy-advanced-
cluster-security-central created
placementrule.apps.open-cluster-management.io/placement-policy-advanced-cluster-
security-central created

```

- 4.3. In the left pane of the RHACM web console, click **Governance**. The dashboard shows the `policy-advanced-cluster-security-central` policy. Click `policy-advanced-cluster-security-central` and then click **Clusters** to check the status.
- 4.4. Verify that the clusters tab shows the `rhacs-central-namespace` and `advanced-cluster-security-central` templates.

| Cluster       | Compliance | Template                          | Message                                                                                                                                                                          | Last Report       | History                      |
|---------------|------------|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------------|
| local-cluster | Compliant  | advanced-cluster-security-central | Compliant; notification - central [stackrox-central-services] in namespace stackrox found as specified, therefore this Object template is compliant <a href="#">View details</a> | a few seconds ago | <a href="#">View history</a> |
| local-cluster | Compliant  | rhacs-central-namespace           | Compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant <a href="#">View details</a>                                     | a few seconds ago | <a href="#">View history</a> |

**Note**

It takes 2-3 minutes for templates to show the **Compliant** status.

- ▶ 5. As the **admin** user, log in to the RHACS central dashboard.
  - 5.1. From the **workstation** machine, open Firefox and access <https://console-openshift-console.apps.ocp4.example.com>.
  - 5.2. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password.
  - 5.3. Navigate to **Home > Projects** to display the **Projects** page, and then click **stackrox** to select the **stackrox** namespace.
  - 5.4. Navigate to **Operators > Installed Operators** to display installed operators, and then click **Central** to display the **Central** page.

| Name                                                                                   | Managed Namespaces | Status                                                        | Provided APIs                                                                                        |
|----------------------------------------------------------------------------------------|--------------------|---------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <a href="#">Red Hat Quay</a><br>3.6.4 provided by Red Hat                              | All Namespaces     | <span style="color: green;">✔ Succeeded<br/>Up to date</span> | <a href="#">Quay Registry</a>                                                                        |
| <a href="#">Advanced Cluster Security for Kubernetes</a><br>3.69.0 provided by Red Hat | All Namespaces     | <span style="color: green;">✔ Succeeded<br/>Up to date</span> | <span style="border: 1px solid red; padding: 2px;">Central</span><br><a href="#">Secured Cluster</a> |

- 5.5. Click **stackrox-central-services** to display **stackrox-central-services** page. Copy the command from the **Admin Credentials Info** section to retrieve the default password for the **admin** user.

The screenshot shows the Stackrox operator details page for the 'stackrox-central-services' operator. The 'Central' tab is selected. In the 'Admin Credentials Info' section, a note states: 'A password for the 'admin' user has been automatically generated and stored in the "password" entry of the central-htpasswd secret. To view the password, run `oc -n stackrox get secret central-htpasswd -o go-template='{{index .data "password" | base64decode}}'`'. The 'Product Version' is listed as 3.69.0.

5.6. Retrieve the default password for the `admin` user from the command line.

```
[student@workstation rhacs-install]$ oc -n stackrox get secret central-htpasswd \
-o go-template='{{index .data "password" | base64decode}}'
WcRPChXqaP6KAoLbyCSTr9v9d
```

5.7. Run the `oc get route` command to retrieve the URL of the RHACS web console.

```
[student@workstation rhacs-install]$ oc get route -n stackrox
NAME HOST/PORT PATH SERVICES PORT
TERMINATION WILDCARD
central central-stackrox.apps.ocp4.example.com central https
passthrough None
...output omitted...
```

5.8. Open Firefox and access <https://central-stackrox.apps.ocp4.example.com>.



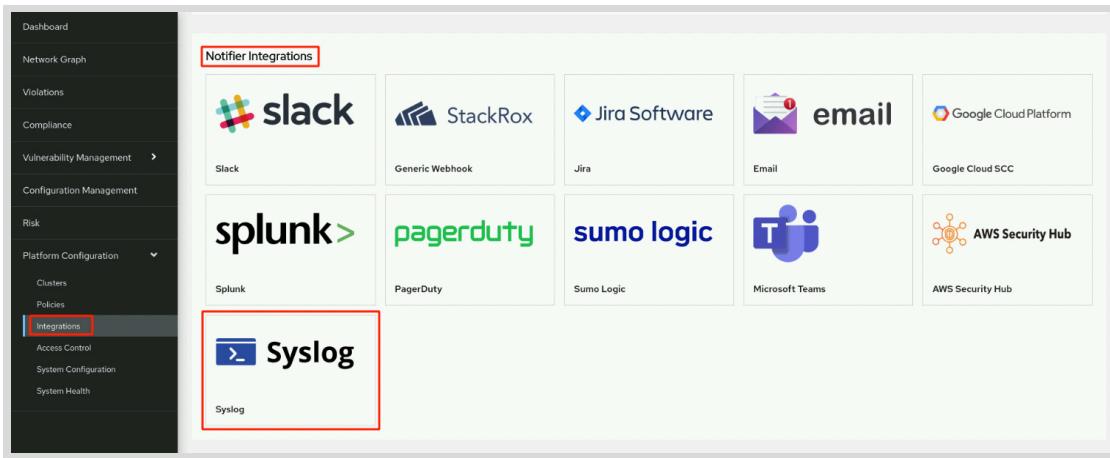
#### Note

If Firefox displays a warning message, then add exceptions to allow using the `CENTRAL_SERVICE` certificate.

5.9. From the **Select an auth provider** list, click **Login with username/password**. Then, log in as the `admin` user with the password retrieved in the preceding step.

#### ► 6. Create and test an integration for RHACS Syslog.

6.1. On the RHACS web console, navigate to **Platform Configuration > Integrations** to display the **Integration** page. Click **Syslog** in the **Notifier Integrations** section.



- 6.2. On the **Integrations** page, click **New integration**.
- 6.3. Complete the **Configure Syslog Integration** form with the following parameters.

| Parameter        | Value                       |
|------------------|-----------------------------|
| Integration name | RemoteLog                   |
| Logging facility | local0                      |
| Receiver host    | workstation.lab.example.com |
| Receiver port    | 514                         |

**Note**

The Rsyslog server is configured to use `local0` facility for remote logging. The destination file of the audit logs is `/var/log/remote/central.log`.

- 6.4. Click **Test** to verify that the integration works. If the integration works, the page displays a **Success** message.
- 6.5. Click **Save**.
7. Configure the OpenShift Oauth authentication provider for RHACS with the **None** minimum access role. Assign the **Admin** role to the **admin** user and the **Analyst** role to the **developer** user.
  - 7.1. In the left pane of the RHACS web console, navigate to **Platform Configuration > Access Control** to display the **Access Control** page.
  - 7.2. Click **Add auth provider** and select **OpenShift Auth** to add a new auth provider.

The screenshot shows the RHACS interface with the 'Access Control' section selected. The left sidebar has a 'Platform Configuration' dropdown open, with 'Access Control' highlighted. The main area shows '0 results found' and a message 'No auth providers' with a note 'Please add one.' A sidebar on the right lists various auth providers, with 'OpenShift Auth' selected and highlighted with a red box.

**Note**

The order of the authentication providers might be different in your environment.  
Make sure to select **OpenShift Auth**.

- 7.3. Add a new authentication provider named **cluster** with the **None** minimum access role. Fill the **Configuration** section with the following details. Do **not** click **Save** yet.

| Field               | Value   |
|---------------------|---------|
| Name                | cluster |
| Minimum access role | None    |

The screenshot shows the 'Add new OpenShift Auth auth provider' dialog. It includes fields for 'Name' (set to 'cluster') and 'Auth provider type' (set to 'OpenShift Auth'). Below these, there's a 'Minimum access role' dropdown set to 'None'. A note at the bottom provides information about the minimum access role.

**Assign roles to your openshift users**

**Minimum access role \***

None

The minimum access role is granted to all users who sign in with this authentication provider.  
To give users different roles, add rules. Users are granted all matching roles.  
Set the minimum access role to **None** if you want to define permissions completely using specific rules below.

- 7.4. Click **Add new rule** and complete the **Add new OpenShift Auth auth provider** form with the following rules. Click **Save**.

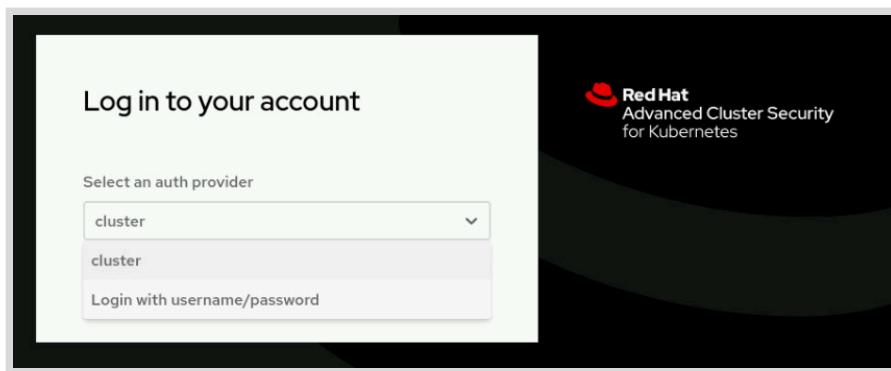
| Key  | Value     | Role    |
|------|-----------|---------|
| name | admin     | Admin   |
| name | developer | Analyst |

Rules

| Key  | Value     | Role    |
|------|-----------|---------|
| name | admin     | Admin   |
| name | developer | Analyst |

[+ Add new rule](#)

- 8. Verify the new authentication configuration for the developer user.
- 8.1. From the RHACS web console, click **ad** in the upper-right corner, and click **Logout**.
  - 8.2. Select the **cluster** auth provider. Click **htpasswd\_provider** and log in as the **developer** user with the **redhat** password, and then click **Allow selected permissions**.



### Note

If the auth provider selection is not available, then reload the web page.

- 8.3. In the RHACS web console, navigate to **Platform Configuration > Integrations** to display the **Integration** page. Click **Syslog** in the **Notifier Integrations** section. On the **Integrations** page, the **New integration** button is unavailable for the **developer** user. The **developer** user can only review the details of the existing **Remotelog** integration.

| Name      | Receiver Host               | Skip TLS Certificate Verification |
|-----------|-----------------------------|-----------------------------------|
| Remotelog | workstation.lab.example.com | No (Secure)                       |

- 8.4. From the RHACS web console, click **de** in the upper-right corner to see the user profile. The user profile displays the **developer** user name with the **Analyst** role. Click **Logout**.
- 8.5. Select the **cluster** auth provider. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password. Click **Allow selected permissions**.
- 8.6. In the RHACS web console, navigate to **Platform Configuration > Integrations** to display the **Integration** page. Click **Syslog** in the **Notifier Integrations** section.

On the **Integrations** page, the **New integration** is available for the **admin** user.

| Name      | Receiver Host               | Skip TLS Certificate Verification |
|-----------|-----------------------------|-----------------------------------|
| Remotelog | workstation.lab.example.com | No (Secure)                       |

- 8.7. From the RHACS web console, click **ad** in the upper-right corner to see the user profile. The user profile displays the **admin** user name and the **Admin** role.
  9. Review the audit logs from the Rsyslog server running on the **workstation** machine. The audit logs recorded the authentication provider change.
- 9.1. Verify that the test operation when configuring the **OpenShift Oauth** authentication provider is logged to the **/var/log/remote/central.log** file.

```
[student@workstation rhacs-install]$ sudo grep Test /var/log/remote/central.log
...output omitted...
Apr 26 09:57:12 central stackRoxKubernetesSecurityPlatform[1]
CEF:0|StackRox|Kubernetes Security Platform|3.69.0|Test|0|Test|
stackroxKubernetesSecurityPlatformTestMessage=test
...output omitted...
```

- 9.2. Verify that the action of configuring the OpenShift OAuth authentication provider is logged to the /var/log/remote/central log file. Search by using the authProviders keyword.

```
[student@workstation rhacs-install]$ sudo grep
authProviders /var/log/remote/central.log
...output omitted...
Apr 26 09:59:06 central stackRoxKubernetesSecurityPlatform[1] CEF:0|
StackRox|Kubernetes Security Platform|3.69.0|AuditLog|3|AuditLog|
rt=1650967146973 spriv=Admin suser= request=/v1/authProviders requestMethod=POST
outcome=REQUEST_SUCCEEDED
...output omitted...
```

- 9.3. Change to the home directory.

```
[student@workstation rhacs-install]$ cd ~
[student@workstation ~]$
```

## Finish

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

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



### Important

Do not delete any resources created in this exercise. They are used in upcoming sections.

This concludes the section.

# Importing Managed Clusters into RHACS

## Objectives

- Import managed clusters into Red Hat Advanced Cluster Security for Kubernetes (RHACS) and retrieve basic security information from the cluster fleet using the RHACS console.

## Importing Secured Clusters to RHACS

When the Red Hat Advanced Cluster Security for Kubernetes (RHACS) Central instance is ready, you can import secured clusters into RHACS.

There are three methods for importing clusters into RHACS:

- Create a `SecuredCluster` resource that is provided by the Advanced Cluster Security for Kubernetes operator.
- Use the `secured-cluster-services` Helm chart, available at <https://mirror.openshift.com/pub/rhacs/charts/>.
- Use the `roxctl` command.

The method that you used to install the RHACS centralized services determines the correct method to import clusters. You can find more information about the different installation methods in *Installing RHACS*.

When Red Hat Advanced Cluster Management for Kubernetes (RHACM) is available, you can create an RHACM policy to import RHACM managed clusters into RHACS automatically. This method is equivalent to creating a `SecuredCluster` resource provided by the Advanced Cluster Security for Kubernetes operator. You can review *Installing RHACS Using RHACM Policies* for more information about installing RHACS using RHACM policies.

Secured clusters require that the Central certificates communicate with the RHACS centralized services. A bundle named `init bundle` contains these certificates.

You can generate the bundle from the RHACS web console, as explained in *Generating an Init Bundle*. Alternatively, you can generate an API token from the RHACS web console and generate the bundle using the `roxctl` command. This approach is useful if you want to automate adding secured clusters from the command line.

## Generating an Init Bundle

The `init bundle` contains the Central server certificates. You must import these certificates to every secured cluster before creating the `SecuredCluster` resource. RHACS provides the Central certificates in the `init bundle` as a list of Kubernetes secrets in a YAML file.

You can find the route to the RHACS web console from the OpenShift web console or with the following command:

```
[user@demo ~]$ oc -n stackrox get route central \
 -o jsonpath=".status.ingress[0].host"
central-stackrox.apps.ocp4.example.com
```

To log in to RHACS, use the `admin` user and the password obtained with the following command:

```
[user@demo ~]$ oc -n stackrox get secret central-htpasswd \
-o go-template='{{index .data "password" | base64decode}}'
...output omitted...
```

The OpenShift web console suggests using this command to obtain the default password from the Central custom resource page.

From the RHACS web console, follow the steps below to generate the `init bundle` YAML files:

1. Navigate to **Platform Configuration > Integrations**.
2. Click **Generate Bundle**.
3. Enter a name for the `init bundle` and click **Generate**.
4. Click **Download Kubernetes Secret File**.

The preceding steps generate and download a YAML file. Use that YAML file to create the secrets in the namespace where you plan to create the `SecuredCluster` resource. Import the same `init bundle` to every secured cluster in the namespace where the `SecuredCluster` resource resides.

## Creating Secured Clusters

After importing the Central certificates by using the `init bundle`, you can import a cluster into RHACS.

You import secured clusters into RHACS by creating the `SecuredCluster` custom resource. Before creating the `SecuredCluster` custom resource, you must install the Advanced Cluster Security for Kubernetes operator in every secured cluster to create the `SecuredCluster` custom resource definition. When you install the operator, you can create a Secured Cluster from the **Installed Operators** page in the OpenShift web console or use a YAML file.

Red Hat recommends installing the `SecuredCluster` custom resource in the `stackrox` namespace for compatibility with pre-existing RHACS policies.

The following default YAML code creates a `SecuredCluster` custom resource provided by the operator.

```
apiVersion: platform.stackrox.io/v1alpha1
kind: SecuredCluster
metadata:
 name: stackrox-secured-cluster-services
 namespace: stackrox
spec:
 auditLogs:
 collection: Auto
 admissionControl:
 listenOnUpdates: true
 bypass: BreakGlassAnnotation
 contactImageScanners: DoNotScanInline
 listenOnCreates: true
 timeoutSeconds: 3
 listenOnEvents: true
```

```

scanner:
analyzer:
scaling:
 autoScaling: Enabled
 maxReplicas: 5
 minReplicas: 2
 replicas: 3
scannerComponent: AutoSense
perNode:
 collector:
 collection: KernelModule
 imageFlavor: Regular
 taintToleration: TolerateTaints
clusterName: managed-cluster
centralEndpoint: 'central-stackrox.apps.ocp4.example.com:443'

```

Ensure that no other secure clusters in RHACS already use the value of the `clusterName` key.

The secured cluster communicates with Central using the URL defined in the `centralEndpoint` key.

You can find the Central endpoint address on the **Routes** page, displayed from the **Networking** menu of the cluster that is running Central.

Alternatively, run the following command in the cluster that is running Central to retrieve the Central endpoint:

```
[user@demo ~]$ oc get route central -n stackrox
NAME HOST/PORT
central central-stackrox.apps.ocp4.example.com ...
```

You can configure Admission Controller, Scanner, Sensor, and other service settings when creating the secured cluster instance. Visit the following link [[https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#secured-cluster-configuration-options-operator\\_install-ocp-operator](https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#secured-cluster-configuration-options-operator_install-ocp-operator)] for a complete list of Secured Cluster configuration options.

## Exploring the RHACS Web Console

You can access the RHACS web console from your preferred browser. If the Central instance runs in the `stackrox` namespace, the URL will be similar to `https://central-stackrox.apps.ocp4.example.com`.

Currently, the RHACS authentication does not integrate by default with the Red Hat OpenShift cluster where it runs. To log in, you must provide a username and password. The default credential is the `admin` user; retrieve the password by using the following command:

```
[user@demo ~]$ oc -n stackrox get secret central-htpasswd \
-o go-template='{{index .data "password" | base64decode}}'
```

The left panel of the RHACS web console contains the following menus:

## Dashboard

On the Dashboard page, you can find aggregated information about policy violations, compliance reports, violations by cluster, top risky deployments, and active violations by time. The Dashboard page also provides a filter to look for resources across the secured clusters.

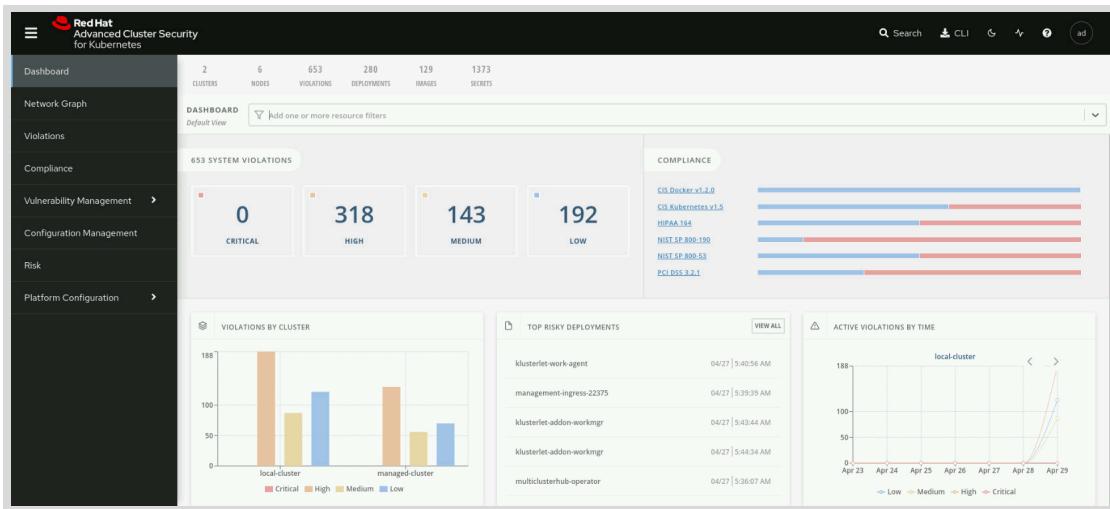


Figure 8.15: The Dashboard page

## Network Graph

The Network Graph page provides a network graph view and a button to access the network policy simulator. From this page, you can visualize network flows on different clusters. You can also filter to locate resources across the cluster fleet.

## Violations

RHACS reports policy violations on the Violations page. You can use this page to investigate the cause of violations and take corrective actions. The reported violations can be related to container images affected by CVEs, violations of DevOps recommended practices, high-risk deployments due to excessive permissions of the service accounts, and more.

## Compliance

The Compliance page contains dashboards with information about the compliance status of the cluster fleet. From the Compliance page, you can click SCAN ENVIRONMENT to visualize compliance graphs.

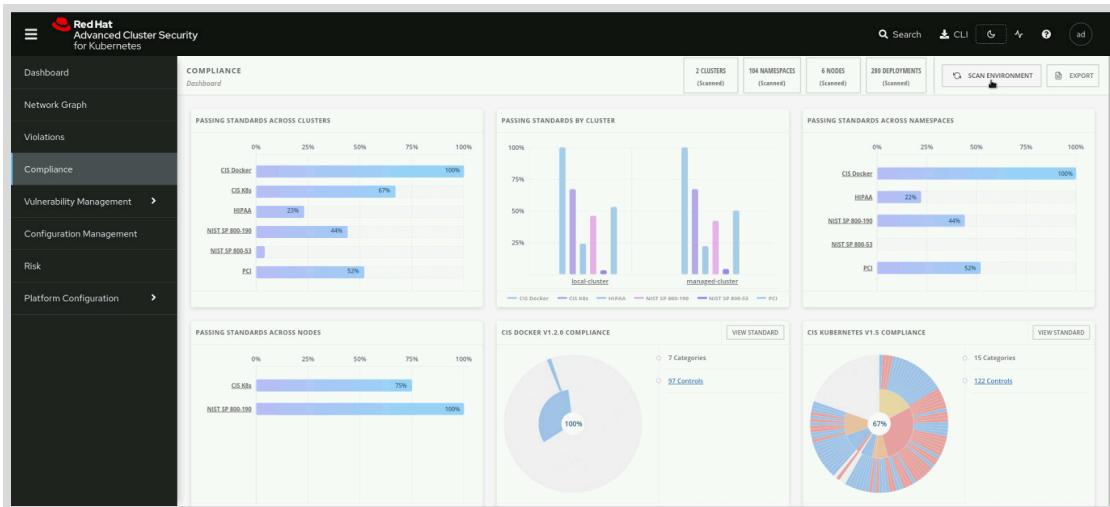


Figure 8.16: The Compliance dashboard

You can obtain more detailed information by clicking on the elements of each dashboard.

## Vulnerability Management

The Vulnerability Management menu contains the **Dashboard**, **Risk Acceptance**, and **Reporting** pages.

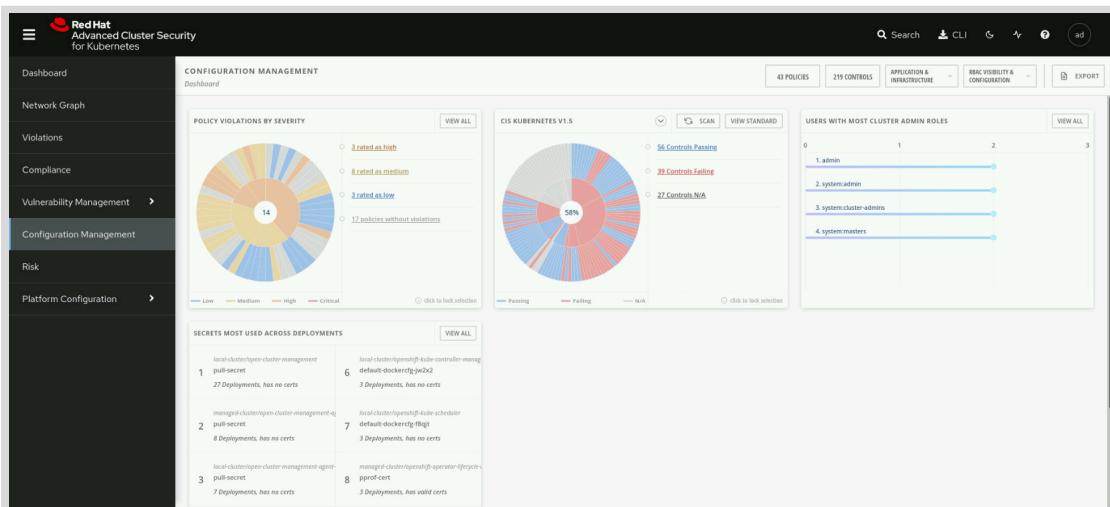
The **Dashboard** page provides a centralized view of the most vulnerable images, frequently violated policies, and vulnerabilities affecting the cluster fleet.

From the **Risk Acceptance** page you can review the alerts generated by RHACS and decide which actions to take.

From the **Reporting** page, you can schedule vulnerability reports to run and send updates to a distribution list.

## Configuration Management

The Configuration Management page provides reports about configuration-related policy violations, CIS Docker and CIS Kubernetes standards, users with elevated privileges, and Kubernetes secrets.



**Figure 8.17: The Configuration Management dashboard**

You can access more detailed information by clicking on each dashboard.

## Risk

On the **Risk** page, RHACS creates a classification of issues using a calculated risk level. This view helps security professionals decide which elements require immediate attention.

## Platform Configuration

The Platform Configuration menu contains the **Clusters**, **Policies**, **Integrations**, **Access Control**, **System Configuration**, and **System Health** pages.

The **Clusters** page shows the status of the secured clusters and allows you to configure automatic upgrades of the secured cluster services.

On the **Policies** page, you can review, create, and import RHACS policies using a JSON policy file.

On the **Integrations** page, you can configure integration with external services, such as image registries, notifiers, and backup systems. You can also manage authentication tokens, such as API tokens and init bundles.

On the **Access Control** page, you can add an authentication provider, manage roles and permission sets, and access scopes.

On the **System Configuration** page, you can manage the data retention configuration, create a header, a footer, or a login notice, and enable or disable the online telemetry data collection.

The **System Health** page provides an overview of the RHACS components running on the cluster fleet. You can view the status of the vulnerability definitions, image integrations, notifier integrations, and backup integrations. You can also generate a diagnostic bundle from this page.



## References

For more information about generating an init bundle, refer to the *Generating an init bundle* section of the *Red Hat Advanced Cluster Security for Kubernetes Documentation* at

<https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#generate-init-bundle-operator>

For more information about generating the API token from the RHACS web console, refer to the *Authenticating using the roxctl CLI* section of the *Red Hat Advanced Cluster Security for Kubernetes Documentation* at

[https://docs.openshift.com/acs/3.69/cli/getting-started-cli.html#cli-authentication\\_cli-getting-started](https://docs.openshift.com/acs/3.69/cli/getting-started-cli.html#cli-authentication_cli-getting-started)

For more information about importing clusters into RHACS, refer to the *Installing secured cluster services* section of the *Red Hat Advanced Cluster Security for Kubernetes Documentation* at

[https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#install-secured-cluster-operator\\_install-ocp-operator](https://docs.openshift.com/acs/3.69/installing/install-ocp-operator.html#install-secured-cluster-operator_install-ocp-operator)

## ► Guided Exercise

# Importing Managed Clusters into RHACS

- Import all the clusters managed by Red Hat Advanced Cluster Management for Kubernetes (RHACM) into Red Hat Advanced Cluster Security for Kubernetes (RHACS) by using an RHACM policy. Then, you explore the RHACS console dashboard, run a compliance scan, and analyze the results.

## Outcomes

- Install the RHACS operator in all the clusters using an RHACM policy.
- Generate a cluster `init bundle` using the RHACS dashboard.
- Create cluster init secrets in all the managed clusters.
- Create the RHACS `SecuredCluster` custom resource in all the clusters that use an RHACM policy.
- Verify the import of all the clusters from the RHACS web console.
- Run a compliance scan on all the clusters and analyze the result.

## Before You Begin

To perform this exercise, ensure you have completed *Guided Exercise: Installing and Configuring RHACS*.

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

```
[student@workstation ~]$ lab start rhacs-import
```

- 1. Edit the RHACM policy named `policy-advanced-cluster-security-install` to install RHACS in all the clusters. This policy installs the RHACS operator in the `rhacs-operator` namespace.
  - From the `workstation` machine, open Firefox and navigate to the RHACM web console. The URL is `https://multicloud-console.apps.ocp4.example.com`.
  - Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
  - In the left pane, click `Governance`. The dashboard shows the `policy-advanced-cluster-security-install` policy. Click `policy-advanced-cluster-security-install` and then click `Edit policy` to edit the policy.
  - Remove the `local-cluster:"true"` value from the `Cluster selector` field to apply the policy to all the clusters.
  - Click `Save`.

- Click **policy-advanced-cluster-security-install** and then click **Clusters** to verify the status. You should see that the **local-cluster** and the **managed-cluster** clusters show **Compliant** in the **Compliance** column.

| Cluster         | Compliance | Template                                        | Message                                                                                                                                                                                                                     | Last Report   | History                      |
|-----------------|------------|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------------------------|
| managed-cluster | Compliant  | advanced-cluster-security-operator-subscription | Compliant; notification - subscriptions [rhacs-operator] in namespace rhacs-operator found as specified, therefore this Object template is compliant                                                                        | 4 minutes ago | <a href="#">View history</a> |
| managed-cluster | Compliant  | rhacs-operator-operator-group                   | Compliant; notification - operatorgroups [rhacs-operator] in namespace rhacs-operator found as specified, therefore this Object template is compliant                                                                       | 4 minutes ago | <a href="#">View history</a> |
| managed-cluster | Compliant  | advanced-cluster-security-namespace-space       | Compliant; notification - namespaces [rhacs-operator] found as specified, therefore this Object template is compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant | 4 minutes ago | <a href="#">View history</a> |
| local-cluster   | Compliant  | advanced-cluster-security-operator-subscription | Compliant; notification - subscriptions [rhacs-operator] in namespace rhacs-operator found as specified, therefore this Object template is compliant                                                                        | 2 days ago    | <a href="#">View history</a> |
| local-cluster   | Compliant  | rhacs-operator-operator-group                   | Compliant; notification - operatorgroups [rhacs-operator] in namespace rhacs-operator found as specified, therefore this Object template is compliant                                                                       | 2 days ago    | <a href="#">View history</a> |



**Note**  
It takes 2-3 minutes for templates to show the **Compliant** status.

## ▶ 2. Create an integration for the cluster `init` bundle.

- From the **workstation** machine, open Firefox and access <https://central-stackrox.apps.ocp4.example.com>.
- Select the **cluster** auth provider. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password.
- On the RHACS web console, navigate to **Platform Configuration > Integrations** to display the **Integration** page. Click **Cluster Init Bundle** in the **Authentication Tokens** section.

- 2.4. On the **Integrations** page, click **Generate bundle**.
- 2.5. Enter **rhacs-import** in the **Cluster init bundle name** field and click **Generate**.
- 2.6. On the **Configure Cluster Init Bundle Integration** page, click **Download Kubernetes secrets file** to download the secret file. In the confirmation window, select **Save File** and click **OK**.

The screenshot shows the 'Configure Cluster Init Bundle Integration' page. At the top, there is a success message: 'Integration was saved successfully'. Below it, a note says: 'Please copy the generated cluster init bundle YAML file and store it safely. You will not be able to access it again after you close this window.' A blue button labeled 'Download Helm values file' is visible. Below this, another note says: 'Use the following file if you do not want your secrets to be managed by Helm. Most users should use the Helm values file above instead.' A blue button labeled 'Download Kubernetes secrets file' is highlighted with a red box. The page lists several details:

- Name: rhacs-import
- Issued: 05/04/2022 | 6:34:27AM
- Expiration: 05/04/2023 | 6:34:00AM
- Created By: sso:4422fe6d-2793-47e4-94d3-ecb8001c56c2:admin

- name: admin
- userid: 55306165-1559-43d3-921c-bcd21ff211
- groups: system:authenticated
- groups: system:authenticated.oauth

At the bottom left is a 'Back' link.

► 3. Create the **rhacs-import** cluster secrets on both **ocp4-mng** and **ocp4** clusters.

- 3.1. Open a terminal and log in to the **ocp4-mng** cluster as the **admin** user with the **redhat** password. The API server address 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...
```

- 3.2. Review the **rhacs-import-cluster-init-secrets.yaml** cluster init secrets file.

```
[student@workstation ~]$ cat ~/Downloads/rhacs-import-cluster-init-secrets.yaml
...output omitted...
apiVersion: v1
kind: Secret
metadata:
 annotations:
 init-bundle.stackrox.io/created-at: "2022-05-04T10:34:27.020155453Z"
 init-bundle.stackrox.io/expires-at: "2023-05-04T10:34:00Z"
 init-bundle.stackrox.io/id: e456829c-e586-4538-8732-cdceb7de2f40
```

```

init-bundle.stackrox.io/name: rhacs-import
creationTimestamp: null
name: collector-tls
stringData:
 ca.pem: |
...output omitted...
apiVersion: v1
kind: Secret
metadata:
 annotations:
 init-bundle.stackrox.io/created-at: "2022-05-04T10:34:27.020155453Z"
 init-bundle.stackrox.io/expires-at: "2023-05-04T10:34:00Z"
 init-bundle.stackrox.io/id: e456829c-e586-4538-8732-cdceb7de2f40
 init-bundle.stackrox.io/name: rhacs-import
 creationTimestamp: null
 name: sensor-tls
 stringData:
 ca.pem: |
...output omitted...
apiVersion: v1
kind: Secret
metadata:
 annotations:
 init-bundle.stackrox.io/created-at: "2022-05-04T10:34:27.020155453Z"
 init-bundle.stackrox.io/expires-at: "2023-05-04T10:34:00Z"
 init-bundle.stackrox.io/id: e456829c-e586-4538-8732-cdceb7de2f40
 init-bundle.stackrox.io/name: rhacs-import
 creationTimestamp: null
 name: admission-control-tls
 stringData:
 admission-control-cert.pem: |
...output omitted...

```

The file contains three secrets: **collector-tls**, **sensor-tls**, and **admission-control-tls**.

- 3.3. Create the secrets in the `stackrox` namespace by using the `rhacs-import-cluster-init-secrets.yaml` file.

```
[student@workstation ~]$ oc create -f \
~/Downloads/rhacs-import-cluster-init-secrets.yaml -n stackrox
secret/collector-tls created
secret/sensor-tls created
secret/admission-control-tls created
```

- 3.4. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 3.5. Create the secrets in the stackrox namespace by using the `rhacs-import-cluster-init-secrets.yaml` file.

```
[student@workstation ~]$ oc create -f \
~/Downloads/rhacs-import-cluster-init-secrets.yaml -n stackrox
secret/collector-tls created
secret/sensor-tls created
secret/admission-control-tls created
```

- ▶ 4. Create an RHACM policy named `policy-advanced-managed-cluster-security` in the `rhacs-install` namespace. This policy installs the RHACS SecuredCluster instance in the `stackrox` namespace on the `ocp4` and `ocp4-mng` clusters. The `ocp4` cluster is named `local-cluster` and the `ocp4-mng` cluster is named `managed-cluster` in RHACM.

- 4.1. Change to the `~/D0480/labs/rhacs-import/` directory.

```
[student@workstation ~]$ cd ~/D0480/labs/rhacs-import/
[student@workstation rhacs-import]$
```

- 4.2. Review the `policy-rhacs-operator-secured-clusters.yaml` file with the definition of a RHACM policy to create the RHACS SecuredCluster instance.

```
[student@workstation rhacs-import]$ cat \
policy-rhacs-operator-secured-clusters.yaml
...output omitted...
object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: v1
 kind: Namespace
 metadata:
 name: stackrox
...output omitted...
object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: platform.stackrox.io/v1alpha1
 kind: SecuredCluster
 metadata:
 namespace: stackrox
 name: stackrox-secured-cluster-services
 spec:
 clusterName: |
 {{ fromSecret "open-cluster-management-agent" "hub-kubeconfig-secret" "cluster-name" | base64dec }}
 auditLogs:
 collection: Auto
 centralEndpoint: |
 central-stackrox.apps.ocp4.example.com:443
 admissionControl:
 listenOnCreates: false
 listenOnEvents: true
```

```

 listenOnUpdates: false
perNode:
 collector:
 collection: KernelModule
 imageFlavor: Regular
 taintTolerations: TolerateTaints
...output omitted...
clusterSelector:
 matchExpressions: []

```

The policy also ensures that the `stackrox` namespace exists in all the clusters.  
The policy also creates the RHACS `SecuredCluster` instance in the `stackrox` namespace.

- 4.3. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.

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

- 4.4. Use the `oc` command to create the `policy-advanced-managed-cluster-security` policy in the `rhacs-install` namespace.

```
[student@workstation rhacs-import]$ oc create -f \
policy-rhacs-operator-secured-clusters.yaml -n rhacs-install
policy.policy.open-cluster-management.io/policy-advanced-managed-cluster-security
created
placementbinding.policy.open-cluster-management.io/binding-policy-advanced-
managed-cluster-security created
placementrule.apps.open-cluster-management.io/placement-policy-advanced-managed-
cluster-security created
```

- 4.5. From the `workstation` machine, open Firefox and navigate to the RHACM web console. The URL is `https://multicloud-console.apps.ocp4.example.com`.
- 4.6. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
- 4.7. In the left pane, click `Governance`. The dashboard shows the `policy-advanced-managed-cluster-security` policy. Click `policy-advanced-managed-cluster-security` and then click `Clusters` to inspect the status.
- 4.8. Verify that the `Clusters` tab shows two templates for each cluster, the `managed-cluster-security-ns` and `managed-cluster-security-endpoints` templates.

| Cluster Name    | Status    | Cloud Provider                     | Description                                                                                                                                                         | Last Update       | Action                       |
|-----------------|-----------|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------------|
| managed-cluster | Compliant | managed-cluster-security-endpoints | Compliant; notification - securedclusters [stackrox-secured-cluster-services] in namespace stackrox found as specified, therefore this Object template is compliant | a minute ago      | <a href="#">View history</a> |
| managed-cluster | Compliant | managed-cluster-security-ns        | Compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant                                                     | a minute ago      | <a href="#">View history</a> |
| local-cluster   | Compliant | managed-cluster-security-endpoints | Compliant; notification - securedclusters [stackrox-secured-cluster-services] in namespace stackrox found as specified, therefore this Object template is compliant | a few seconds ago | <a href="#">View history</a> |
| local-cluster   | Compliant | managed-cluster-security-ns        | Compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant                                                     | a minute ago      | <a href="#">View history</a> |

**Note**

It takes 2-3 minutes for the templates to show the Compliant status.

- 5. Verify that the clusters `local-cluster` and `managed-cluster` are imported successfully in RHACS.

- 5.1. On the RHACS web console, navigate to **Platform Configuration > Clusters** to display the CLUSTERS page. The CLUSTERS page displays both the `local-cluster` and `managed-cluster` clusters.

| Name            | Cloud Provider | Cluster Status                                   | Sensor Upgrade          | Credential Expiration |
|-----------------|----------------|--------------------------------------------------|-------------------------|-----------------------|
| local-cluster   | Not applicable | Healthy<br>Collector   Sensor   AdmissionControl | Up to date with Central | in 12 months          |
| managed-cluster | Not applicable | Healthy<br>Collector   Sensor   AdmissionControl | Up to date with Central | in 12 months          |

**Note**

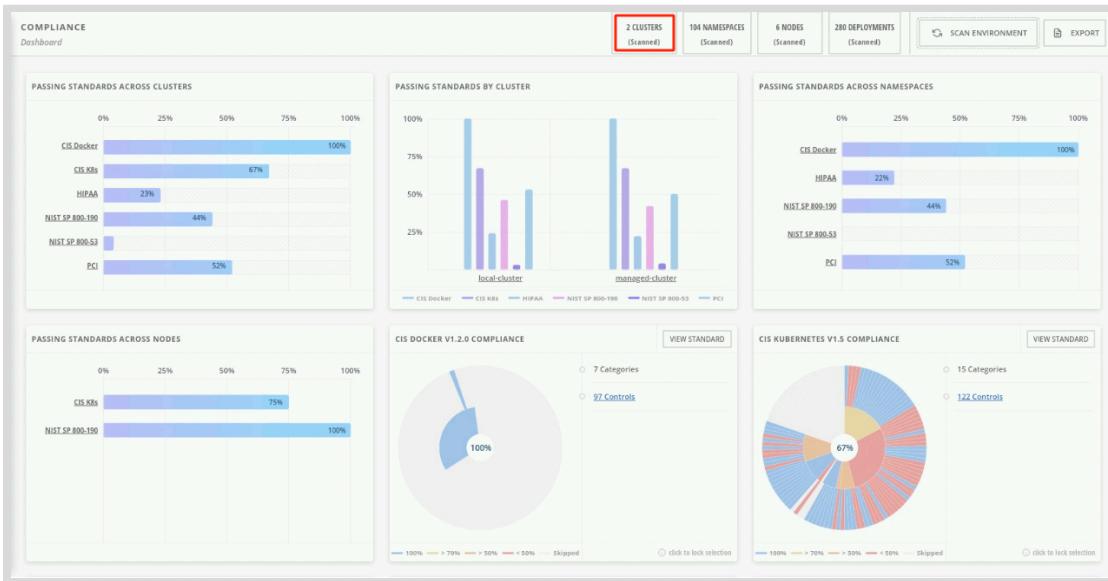
It takes 5-10 minutes for clusters to import to RHACS successfully with Healthy status.

- 6. Run a compliance scan and review the result from RHACS dashboard.

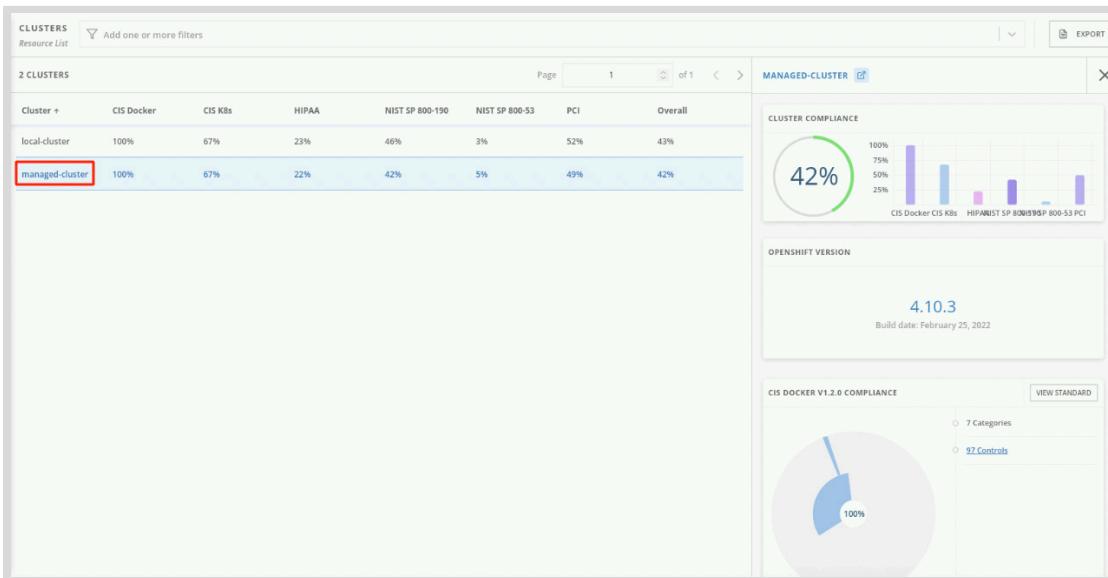
- 6.1. On the RHACS web console, navigate to **Compliance** to display the COMPLIANCE page.

- 6.2. Click **SCAN ENVIRONMENT** to scan all the secured clusters. The **COMPLIANCE** dashboard shows results from the compliance report for both clusters.

- 6.3. Click **2 CLUSTERS (Scanned)** to see the Resource List.



- 6.4. Click **managed-cluster** to view more details for the managed-cluster compliance scan.



You can click the different dashboards to obtain more details about the compliance scan results.

- 6.5. Change to the home directory.

```
[student@workstation rhacs-import]$ cd ~
[student@workstation ~]$
```

## Finish

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

```
[student@workstation ~]$ lab finish rhacs-import
```



### Important

Do not delete any resources created in this exercise. They are used in upcoming sections.

This concludes the section.

## ► Quiz

# Installing and Configuring RHACS

Choose the correct answers to the following questions:

- ▶ 1. **Which four of the following features is provided by Red Hat Advanced Cluster Security for Kubernetes (RHACS)? (Choose four.)**
  - a. Time capsule
  - b. Vulnerability management
  - c. Configuration management
  - d. Decommissioning insecure Kubernetes clusters
  - e. Risk profiling
  - f. Threat detection
  
- ▶ 2. **Which four of the following installation methods are supported ways to install RHACS? (Choose four.)**
  - a. The Advanced Cluster Security for Kubernetes operator
  - b. Helm
  - c. Kustomization files
  - d. The `roxctl` command
  - e. Ansible Playbook
  - f. Red Hat Advanced Cluster Management for Kubernetes (RHACM) policies
  - g. A URL containing a YAML file to use with the `kubectl apply` command
  
- ▶ 3. **Which two of the following components are included in RHACS? (Choose two.)**
  - a. Admission Controller
  - b. Clair
  - c. Scanner
  - d. Klusterlet
  
- ▶ 4. **Which three of the following methods are supported ways to import a secured cluster? (Choose three.)**
  - a. Using the `SecuredCluster` custom resource definition
  - b. A URL containing a YAML file to use with the `kubectl apply` command
  - c. Using a Helm Chart
  - d. Using RHACM applications
  - e. Using RHACM policies

► 5. Which three of the following information dashboards are provided by the RHACS vulnerability management page? (Choose three.)

- a. Most used operating systems across the cluster fleet
- b. Frequently violated policies
- c. A list of blocked IP addresses due to security attack attempts
- d. Deployments with the most severe policy violations
- e. Top risky deployments by CVE count and CVSS score

## ► Solution

# Installing and Configuring RHACS

Choose the correct answers to the following questions:

- ▶ 1. **Which four of the following features is provided by Red Hat Advanced Cluster Security for Kubernetes (RHACS)? (Choose four.)**
  - a. Time capsule
  - b. Vulnerability management
  - c. Configuration management
  - d. Decommissioning insecure Kubernetes clusters
  - e. Risk profiling
  - f. Threat detection
  
- ▶ 2. **Which four of the following installation methods are supported ways to install RHACS? (Choose four.)**
  - a. The Advanced Cluster Security for Kubernetes operator
  - b. Helm
  - c. Kustomization files
  - d. The `roxctl` command
  - e. Ansible Playbook
  - f. Red Hat Advanced Cluster Management for Kubernetes (RHACM) policies
  - g. A URL containing a YAML file to use with the `kubectl apply` command
  
- ▶ 3. **Which two of the following components are included in RHACS? (Choose two.)**
  - a. Admission Controller
  - b. Clair
  - c. Scanner
  - d. Klusterlet
  
- ▶ 4. **Which three of the following methods are supported ways to import a secured cluster? (Choose three.)**
  - a. Using the `SecuredCluster` custom resource definition
  - b. A URL containing a YAML file to use with the `kubectl apply` command
  - c. Using a Helm Chart
  - d. Using RHACM applications
  - e. Using RHACM policies

► 5. Which three of the following information dashboards are provided by the RHACS vulnerability management page? (Choose three.)

- a. Most used operating systems across the cluster fleet
- b. Frequently violated policies
- c. A list of blocked IP addresses due to security attack attempts
- d. Deployments with the most severe policy violations
- e. Top risky deployments by CVE count and CVSS score

# Summary

---

- Red Hat Advanced Cluster Security for Kubernetes (RHACS) helps organizations to address security in multicluster environments.
- The recommended RHACS installation method is using the operator.
- You can customize the RHACS installation to adapt to your organization needs.
- You can integrate RHACS with third-party tools to expand the capabilities of RHACS.
- You can import Kubernetes and Red Hat OpenShift clusters into RHACS to become secured clusters.



## Chapter 9

# Multicluster Operational Security Using RHACS

### Goal

Manage the operational security of a Kubernetes cluster fleet using Red Hat Advanced Cluster Security for Kubernetes (RHACS), and integrate RHACS with external services.

### Objectives

- Integrate Red Hat Advanced Cluster Security for Kubernetes (RHACS) with Red Hat Quay and list the benefits of integrating RHACS with other tools.
- Analyze the vulnerabilities affecting the cluster fleet using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) vulnerability dashboard and reporting, and create a policy to prevent the deployment of vulnerable images.
- Detect security policy violations and application or infrastructure misconfigurations by using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) Configuration Management view.

### Sections

- Integrating RHACS with External Registries (and Guided Exercise)
- Perform Multicluster Vulnerability Management with RHACS (and Guided Exercise)
- Multicluster Configuration Management with RHACS (and Guided Exercise)

### Lab

- Multicluster Operational Security Using RHACS

# Integrating RHACS with External Registries

---

## Objectives

- Integrate Red Hat Advanced Cluster Security for Kubernetes (RHACS) with Red Hat Quay and list the benefits of integrating RHACS with other tools.

## Introducing Red Hat Advanced Cluster Security for Kubernetes Integrations

Red Hat Advanced Cluster Security for Kubernetes (RHACS) integrates with many external systems. The following list describes some of the integrations supported by RHACS.

- Sending notifications by using logging systems, such as Syslog, or communication systems like email
- Making backups and storing them in object storage systems
- Configuring continuous integration systems to scan images built during continuous integration processes
- Obtaining additional vulnerability information from other image scanners

This module describes how RHACS integrates with container image registries.

## Integrating RHACS with Container Image Registries

RHACS uses the Docker Registry HTTP API to obtain vulnerability information about images, such as the container definition, the content of each image layer, and the image creation date.

RHACS integrates by default with many public registries. You can view the default integrations by navigating to **Platform Configuration > Integrations** in the RHACS web console. The **Image Integrations** section shows a list of container image registries with which RHACS can integrate.

The screenshot shows the RHACS interface. On the left, a sidebar menu includes 'Dashboard', 'Network Graph', 'Violations', 'Compliance', 'Vulnerability Management', 'Configuration Management', 'Risk', 'Platform Configuration' (with 'Clusters', 'Policies', 'Integrations' selected, and 'Access Control', 'System Configuration', 'System Health' also listed), and 'Integrations'. The main area is titled 'Integrations' and contains a section for 'Image Integrations'. It shows four cards: 'StackRox Scanner' (1 item), 'Generic Docker Registry' (15 items), 'Amazon ECR Registry', and 'Google Container Registry'.

**Figure 9.1: The list of image integrations**

Click **Generic Docker Registry** to display a list of integrations. By default, RHACS lists Public DockerHub with the `registry-1.docker.io` endpoint. This means that RHACS can obtain information for the images stored in the public Docker Hub registry.

Other image integrations contain further preconfigured registries. Most of those registries are public registries that require no authentication. However, RHACS automatically integrates private registries by searching the cluster for image pull secrets.

Some image registries use other authentication methods. For example, Amazon Elastic Kubernetes Service (EKS) can use node Identity and Access Management (IAM) to allow nodes to authenticate to Amazon Elastic Container Registry (ECR). Because the cluster does not contain image pull secrets for the ECR registry, automatic integration does not work. You must configure integrations for this type of cluster manually.

## Analyzing Images

RHACS scans all clusters periodically and analyzes the images used in workloads.

RHACS can also scan specific images. For example, you can scan images used as a base to build other images, even if they do not run in any workload.

To configure scanning of specific images, navigate to **Vulnerability Management > Dashboard**, then click **Images**.

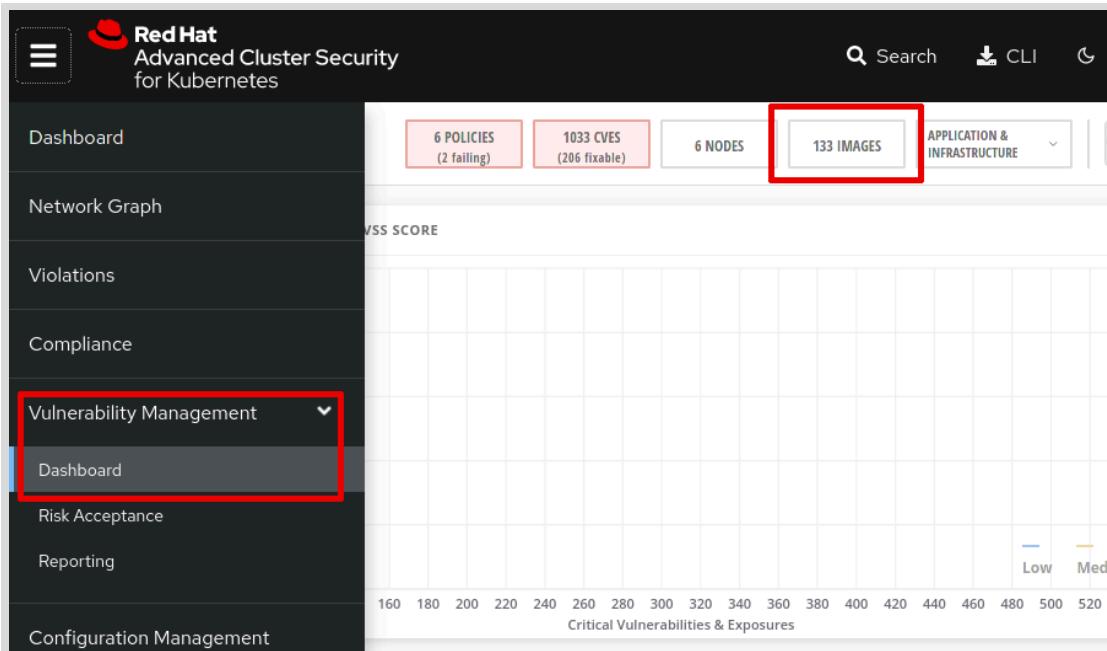


Figure 9.2: Accessing the images list

Click **MANAGE WATCHES** to manage container image watches.

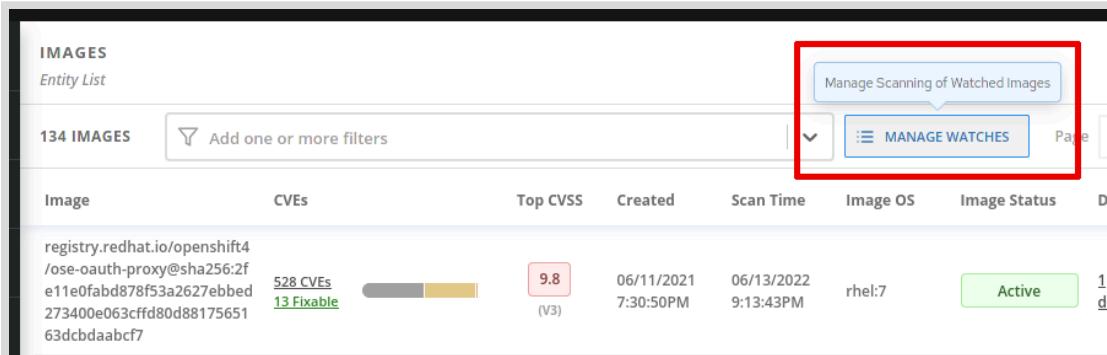
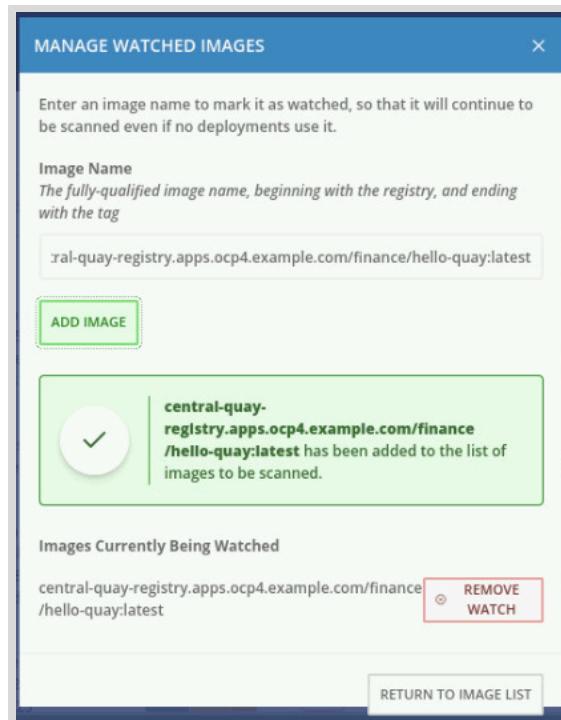


Figure 9.3: Accessing the watched images dialog

In the dialog that opens, you can add or remove watches.



**Figure 9.4: Adding or removing image watches**

Additionally, the `roxctl image check` command scans images on demand. For example, you can configure a continuous integration pipeline to run the `roxctl` command to scan an image as part of a build process.

Note that RHACS does not scan entire repositories, only specific images.



## References

For more information, refer to the *Integrating* guide in the *Red Hat Advanced Cluster Management for Kubernetes* documentation at [https://access.redhat.com/documentation/en-us/red\\_hat\\_advanced\\_cluster\\_security\\_for\\_kubernetes/3.69/html-single/integrating/index](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_security_for_kubernetes/3.69/html-single/integrating/index)

## ► Guided Exercise

# Integrating RHACS with External Registries

- Deploy an application by using an image hosted in the private Red Hat Quay registry and verify that Red Hat Advanced Cluster Security for Kubernetes (RHACS) automatically detects the deployed image. Then, you add a watch to an inactive image hosted on the private Quay registry and verify that it is listed.

## Outcomes

- Add secret to service account to pull images from the private Red Hat Quay registry.
- Verify that RHACS detects a deployed image.
- Review images hosted in the private Red Hat Quay registry by using RHACS.
- Add an inactive image to the RHACS watched images.

## Before You Begin

To perform this exercise, ensure you have completed *Guided Exercise: Installing and Configuring RHACS* and *Guided Exercise: Importing Managed Clusters into RHACS*.

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

```
[student@workstation ~]$ lab start operate-integrate
```

### ► 1. Log in to the Quay web console.

- From the workstation machine, open Firefox and navigate to the Quay web console at <https://central-quay-registry.apps.ocp4.example.com>. Log in as the `cloudadmin` user with the `redhat` password.  
If prompted, click **Confirm Username**.

### ► 2. Create the `finance` organization and push the `quay.io/redhattraining/do480-hello-app` image to the `central` registry.

- Click **Create New Organization**.
- Type `finance` as the organization name, then click **Create Organization**.
- Open a terminal and log in as the `cloudadmin` user with the `redhat` password.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
 central-quay-registry.apps.ocp4.example.com
Login Succeeded!
```

- 2.4. Push the `quay.io/redhattraining/do480-hello-app` image to the central registry by using the `skopeo` command.

```
[student@workstation ~]$ skopeo copy \
 docker://quay.io/redhattraining/do480-hello-app \
 docker://central-quay-registry.apps.ocp4.example.com/finance/do480-hello-app
Getting image source signatures
...output omitted...
```

- 3. Push a `hello-quay` image to the central registry.

- 3.1. Create a `Containerfile` file containing a minimal image definition. The resulting file should look as follows.

```
FROM quay.io/podman/hello
```

- 3.2. Build the image by using the `podman build` command.

```
[student@workstation ~]$ podman build . -t hello-quay
...output omitted...
Successfully tagged localhost/hello-quay
...output omitted...
```

- 3.3. Push the image to the central registry.

```
[student@workstation ~]$ podman push localhost/hello-quay \
 central-quay-registry.apps.ocp4.example.com/finance/hello-quay
...output omitted...
Storing signatures
```

- 4. Create a secret for central Quay registry authentication. Add the secret to the default service account to pull the image from the Quay registry.

- 4.1. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 4.2. Create the `operate-integrate` project.

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

- 4.3. Review the `/run/user/1000/containers/auth.json` file. The JSON file contains authentication details for `central-quay-registry.apps.ocp4.example.com` registry.

```
{
 "auths": {
 "central-quay-registry.apps.ocp4.example.com": {
 "auth": "Y2xvdWRhZG1pbjpyZWRoYXQ="
 }
 }
}
```

**Note**

If authentication details are unavailable in the /run/user/1000/containers/auth.json file, then run podman login.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
 central-quay-registry.apps.ocp4.example.com
```

4.4. Create the quaysecret secret for central Quay registry authentication.

```
[student@workstation ~]$ oc create secret generic quaysecret \
 --from-file=dockerconfigjson=/run/user/1000/containers/auth.json \
 --type=kubernetes.io/dockerconfigjson -n operate-integrate
secret/quaysecret created
```

4.5. Add the quaysecret secret to the default service account named default.

```
[student@workstation ~]$ oc secrets link default quaysecret \
 --for=pull -n operate-integrate
```

▶ 5. Create the hello-app application in the ocp4 cluster.

5.1. Change to the ~/D0480/labs/operate-integrate/ directory.

```
[student@workstation ~]$ cd ~/D0480/labs/operate-integrate/
[student@workstation operate-integrate]$
```

5.2. Review the hello-app.yaml file. The hello-app file is using the central-quay-registry.apps.ocp4.example.com/finance/do480-hello-app image.

```
...output omitted...
spec:
 containers:
 - name: hello-app
 image: central-quay-registry.apps.ocp4.example.com/finance/do480-hello-app
...output omitted...
```

5.3. Create the hello-app application.

```
[student@workstation operate-integrate]$ oc create -f hello-app.yaml
deployment.apps/hello-app created
service/hello-app created
route.route.openshift.io/hello-app created
```

- ▶ 6. Verify that RHACS detects images from the central Quay registry.
- 6.1. From the **workstation** machine, navigate to the RHACS web console at <https://central-stackrox.apps.ocp4.example.com>.
  - 6.2. Select the **cluster** authentication provider and click **Log in**.
  - 6.3. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password. If prompted to authorize access, click **Allow selected permissions**.
  - 6.4. In the RHACS web console, navigate to **Vulnerability Management > Dashboard**, then click **IMAGES**.

The screenshot shows the Red Hat Advanced Cluster Security (RHACS) web interface. At the top, there's a navigation bar with the Red Hat logo, the text "Advanced Cluster Security for Kubernetes", a search bar, and a "CLI" link. Below the navigation bar, there are several status indicators: "6 POLICIES (2 failing)", "1033 CVES (206 fixable)", "6 NODES", and "133 IMAGES". A red box highlights the "133 IMAGES" button. On the left side, there's a sidebar with links like "Dashboard", "Network Graph", "Violations", "Compliance", "Vulnerability Management" (which is expanded to show "Dashboard" as the selected item), "Risk Acceptance", "Reporting", and "Configuration Management". A red box highlights the "Vulnerability Management" link. At the bottom of the screen, there's a chart titled "CVSS SCORE" with a legend for "Low" (blue) and "Medi" (orange).

- 6.5. On the **Image** page, type **Image Registry:central-quay-registry.apps.ocp4.example.com** in the search field to search images in the central Quay registry.
- 6.6. The search result shows the **do480-hello-app:latest** image with the **Active** image status. You can click the **do480-hello-app:latest** image to obtain more details about the image.

1 IMAGE

| Image                                                                              | CVEs                   | Top CVSS     | Created                 | Scan Time               | Image OS  | Image Status | Deployments  | Components     | Risk Pri |
|------------------------------------------------------------------------------------|------------------------|--------------|-------------------------|-------------------------|-----------|--------------|--------------|----------------|----------|
| central-quay-registry.apps.o<br>cp4.example.com/finance/d<br>o480-hello-app:latest | 110 CVEs<br>34 Fixable | 10.0<br>(V2) | 12/09/2021<br>5:26:16AM | 05/16/2022<br>1:14:44PM | debian:11 | Active       | 1 deployment | 108 components | 2        |

- 7. Add the hello-quay image to the watched images in RHACS.

7.1. Click Manage Scanning of Watched Images to add the image to the watched images.

1 IMAGE

| Image                                                                              | CVEs                   | Top CVSS     | Created                 | Scan Time               | Image OS  | Image Status | Deployments  | Components     | Risk Pri |
|------------------------------------------------------------------------------------|------------------------|--------------|-------------------------|-------------------------|-----------|--------------|--------------|----------------|----------|
| central-quay-registry.apps.o<br>cp4.example.com/finance/d<br>o480-hello-app:latest | 110 CVEs<br>34 Fixable | 10.0<br>(V2) | 12/09/2021<br>5:26:16AM | 05/16/2022<br>1:14:44PM | debian:11 | Active       | 1 deployment | 108 components | 2        |

- 7.2. Enter central-quay-registry.apps.ocp4.example.com/finance/hello-quay:latest in the Image Name field. Click ADD IMAGE. The page displays a confirmation message that the image is successfully added to the watched images.

- 7.3. Click RETURN TO IMAGE LIST.

- 7.4. On the **Image** page, type `Image Registry:central-quay-registry.apps.ocp4.example.com` in the search field to search images in the central Quay registry.
- 7.5. The **Image Status** for the `hello-quay` image shows **Inactive** and **Scanning via watch tag** status.

| Image                                                                                   | CVEs                   | Top CVSS     | Created                 | Scan Time               | Image OS  | Image Status                       | Deployments    | Components     | Risk Pri |
|-----------------------------------------------------------------------------------------|------------------------|--------------|-------------------------|-------------------------|-----------|------------------------------------|----------------|----------------|----------|
| <code>central-quay-registry.apps.ocp4.example.com/finance/do480-hello-app:latest</code> | 110 CVEs<br>34 Fixable | 10.0<br>(V2) | 12/09/2021<br>5:26:16AM | 05/16/2022<br>1:14:44PM | debian:11 | Active                             | 1 deployment   | 108 components | 2        |
| <code>central-quay-registry.apps.ocp4.example.com/finance/hello-quay:latest</code>      | No CVEs                | -            | 05/09/2022<br>2:21:30PM | 05/16/2022<br>1:51:50PM | unknown   | Inactive<br>Scanning via watch tag | No deployments | No components  | 14       |

- 7.6. Change to the home directory.

```
[student@workstation operate-integrate]$ cd ~
[student@workstation ~]$
```

## Finish

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

```
[student@workstation ~]$ lab finish operate-integrate
```



### Important

Do not delete any resources created in this exercise. They are used in upcoming sections.

This concludes the section.

# Perform Multicluster Vulnerability Management with RHACS

## Objectives

- Analyze the vulnerabilities affecting the cluster fleet using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) vulnerability dashboard and reporting, and create a policy to prevent the deployment of vulnerable images.

## Introducing Vulnerability Management

Workloads that run on the cluster fleet use container images that contain all the software required for the workload. This software might have bugs that can cause security vulnerabilities. For example, a bug can cause a process to terminate on malformed input. A malicious attacker can feed malformed input to a service that is running an image affected by this bug and make the service unavailable.

Many different bugs can cause vulnerabilities. Usually, software maintainers document vulnerabilities that are detected in their software, including which versions contain the issue and which versions contain a fix. Organizations collect this information to create public databases of vulnerabilities. These databases often contain further information, such as the severity of the issue and mitigation options. For example, Red Hat Errata publishes security advisories that describe security fixes included in software updates.

Red Hat Advanced Cluster Security for Kubernetes (RHACS) analyzes workloads running on secured clusters for vulnerabilities. RHACS examines container images, enumerates versions of the software contained in the image, and correlates this information with vulnerability databases.

By using this information, you can identify vulnerabilities in workloads and take actions to maintain an adequate security level. You can update, replace, or remove the affected workloads, or mitigate vulnerabilities by other means, such as limiting access.

Additionally, in version 3.0.55.0 and later, RHACS enables an admission control webhook by default that can prevent the deployment of vulnerable images.

In addition to RHACS, you can also apply further strategies that reduce the risk of vulnerabilities.

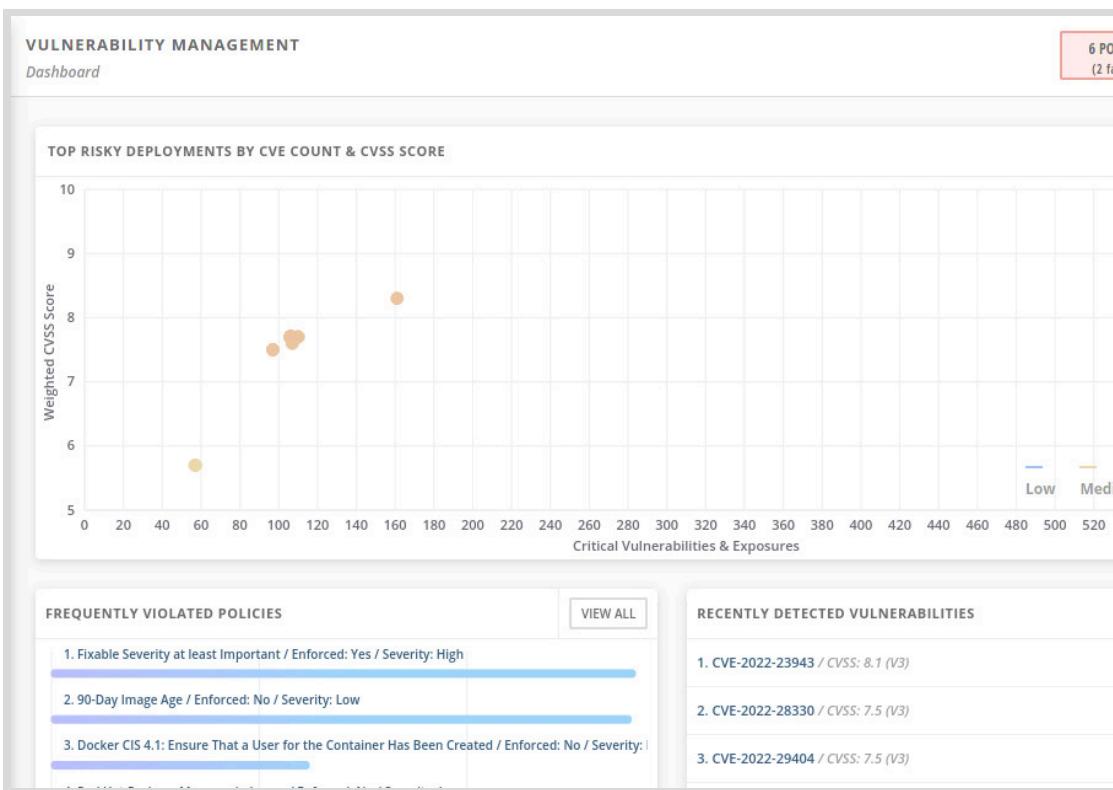
- Favor the use of container images with clear update policies that are aligned with the organization's needs. Choose base images and applications that receive updates to address vulnerabilities.
- Streamline or automate deployments and implement automated testing. This helps keep workloads running the latest available versions of software that contain the latest patches for vulnerabilities.

Performing frequent updates might require a significant effort. However, streamlined deployments can yield further benefits, such as faster time-to-market.

## Using the Vulnerability Dashboard

RHACS presents information about vulnerabilities in the **Vulnerability Management** section of the web console.

The Dashboard page shows an overview of this information with several different classifications.



**Figure 9.10: The vulnerability management dashboard**

For example, you can review the RECENTLY DETECTED VULNERABILITIES panel periodically to identify new vulnerabilities in the cluster. Using this panel, you can react to new threats quickly.

If you want to identify the workloads that present the biggest risks, then you can use the DEPLOYMENTS WITH MOST SEVERE POLICY VIOLATIONS. Using this panel, you can identify workloads to update, replace, or remove.

RHACS describes vulnerabilities by using the following concepts.

### Components

A component is a piece of software. RHACS can identify components that are installed in images as RPM packages or installed by using software, such as the npm package manager.

### Images

Images contain specific versions of components.

### Vulnerabilities

A vulnerability is an identified issue in specific versions of a component.

The RHACS console sometimes refers to vulnerabilities as CVEs. CVE stands for Common Vulnerabilities and Exposures, a popular vulnerability database. Vulnerabilities receive a CVE identifier such as CVE-2022-23943, to help reference vulnerabilities. CVE is also a common way to refer to a CVE identifier. However, CVE lists in RHACM can contain items such as RHSA-2021:4033, pertaining to Red Hat Errata.

When examining RHACM vulnerability information, you can navigate to identify components, images, and vulnerabilities. Then, you can navigate from images to deployments to act on the vulnerabilities.

Alternatively, you can schedule periodic reports about vulnerabilities. These reports can help you communicate with teams that are responsible for vulnerable deployments.

## Creating Policies

A different approach to managing vulnerabilities is to prevent deployments with vulnerable images.

You can define policies to prevent users from creating workloads that match specific criteria. RHACS uses an admission control webhook to enforce policies. Kubernetes uses admission control webhooks before creating any workload. If the webhook rejects the workload, then Kubernetes does not create the workload.

To create a policy, navigate to **Platform Configuration > Policies** and click **Create policy**. You can define the new policy in the wizard that opens.

The screenshot shows the 'Create policy' wizard. Step 1, 'Policy details', is selected. The page title is 'Create policy' with the subtitle 'Design custom security policies for your environment'. A dropdown menu shows '1 Policy details'. The main section is titled 'Policy details' with the sub-instruction 'Describe general information about your policy.' It contains fields for 'Name \*' (with placeholder 'Provide a descriptive and unique policy name') and 'Severity \*' (radio buttons for Low, Medium, High, Critical, with 'Low' selected). To the right, under 'Attach notifiers', it says 'Forward policy violations to external tooling by selecting one or more notifiers from existing integrations.' Below that, a note says 'No notifiers found. Add notifiers in the Integrations Page to add them to this policy.'

Figure 9.11: The first step creating a policy

On the **Create policy** page, you define the policy name and add categorization details, such as severity. You can also define notifications for the policy by using a configured notifier integration.

The screenshot shows the 'Create policy' interface in RHACS. At the top, there's a breadcrumb navigation: Policies > Create policy. The main title is 'Create policy' with the subtitle 'Design custom security policies for your environment'. A step indicator '2 Policy behavior' is shown. The 'Policy behavior' section asks to 'Select which stage of a container lifecycle this policy applies. Event sources can only be chosen for policies that apply at runtime.' Below this, an 'Info' section provides details about build-time, deploy-time, and runtime policies. Under 'Lifecycle stages', there are checkboxes for Build, Deploy, and Runtime. A note says 'Choose lifecycle stage to which your policy is applicable. You can select more than one stage.' Under 'Event sources (Runtime lifecycle only)', there are radio buttons for Deployment and Audit logs. Finally, the 'Response method' section has a radio button for 'Inform' (which is selected) and another for 'Inform and enforce'. A note states 'Inform will always include violations for this policy in the violations list.'

**Figure 9.12: The second step creating a policy**

On the **Policy behavior** page, you define when the policy activates and whether the policy prevents creating the workload or only informs. You can also choose the applicable lifecycle stage to which the policy applies. Depending on the stage, you can use different criteria for the policy. Runtime stage policies can respond to Kubernetes events by using audit logs as the event source. Other policies use criteria related to the properties of the workload.

**Create policy**

Design custom security policies for your environment

**3 Policy criteria**

**Policy criteria**  
Chain criteria with boolean logic. [Add condition](#)

|                             |                  |                              |  |
|-----------------------------|------------------|------------------------------|--|
| 1                           | Policy Section 1 |                              |  |
| Image pulled from registry: |                  | <input type="checkbox"/> Not |  |

**Drag out policy fields**

- > [Image registry](#)
- > [Image contents](#)
- > [Container configuration](#)
- > [Deployment metadata](#)

**Figure 9.13: The third step creating a policy**

On the **Policy criteria** page, you create the set of conditions that the policy uses. You can combine different conditions by dragging and dropping fields and adding conditions based on those fields.

**Create policy**

Design custom security policies for your environment

**4 Policy scope**

**Policy scope**  
Create scopes to restrict or exclude your policy from entities within your environment.

**Restrict by scope**  
Use Restrict by scope to enable this policy only for a specific cluster, namespace, or label. You can add multiple scope and also use regular expressions (RE2 syntax) for namespaces and labels. [Add inclusion scope](#)

**Figure 9.14: The fourth step creating a policy**

On the **Policy scope** page, you can restrict a policy scope by adding an inclusion scope. If you define an inclusion scope, then the policy only affects workloads matching the scope. Scopes can filter on clusters, namespaces, and labels and can use regular expressions.

You can also use an exclusion scope. If you define an exclusion scope, then the policy affects all workloads except those matching the scope.

You can also exclude container images from the policy when creating a build stage.

The screenshot shows the 'Create policy' interface in RHACS. At the top, there's a breadcrumb navigation: 'Policies > Create policy'. Below it, a section titled 'Create policy' with the sub-instruction 'Design custom security policies for your environment'. A numbered step '5 Review policy' is highlighted. The main content area is divided into two sections: 'Review policy' on the left and 'Preview violations' on the right. The 'Review policy' section contains the text 'Review policy settings and violations.' and a button labeled 'Preview policy violations'. The 'Preview violations' section contains the text 'The policy settings you have selected will generate violations for the Build or Deploy lifecycle stages.'

**Figure 9.15: The final step creating a policy**

The **Review policy** page displays all the details in the policy. Click **Preview policy violations** to display affected workloads.

After creating a policy, RHACS applies the policy. Navigate to the **Violations** page to view policy violations. To enforce policies, Kubernetes creates events when the policy affects a workload.



## References

For more information about admission controllers, refer to the *Using admission controller enforcement* chapter of the RHACS Operating Guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_advanced\\_cluster\\_security\\_for\\_kubernetes/3.69/html-single/operating/index#use-admission-controller-enforcement](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_security_for_kubernetes/3.69/html-single/operating/index#use-admission-controller-enforcement)

For more information about admission controllers, refer to the *Managing vulnerabilities* chapter of the RHACS Operating Guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_advanced\\_cluster\\_security\\_for\\_kubernetes/3.69/html-single/operating/index#managing-vulnerabilities](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_security_for_kubernetes/3.69/html-single/operating/index#managing-vulnerabilities)

For more information about security policies, refer to the *Managing security policies* chapter of the RHACS Operating Guide at [https://access.redhat.com/documentation/en-us/red\\_hat\\_advanced\\_cluster\\_security\\_for\\_kubernetes/3.69/html-single/operating/index#manage-security-policies](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_security_for_kubernetes/3.69/html-single/operating/index#manage-security-policies)

### Explaining Red Hat Errata (RHSA, RHBA, and RHEA)

<https://access.redhat.com/articles/2130961>

## ► Guided Exercise

# Perform Multicluster Vulnerability Management with RHACS

- Analyze the vulnerabilities affecting the cluster fleet by using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) vulnerability dashboard and reporting, and you create a policy to prevent the deployment of vulnerable images.

## Outcomes

- Explore the RHACS vulnerability dashboard to locate risky images.
- Create an RHACS policy to prevent the deployment of vulnerable images.
- Verify that RHACS blocks the deployment of vulnerable images.

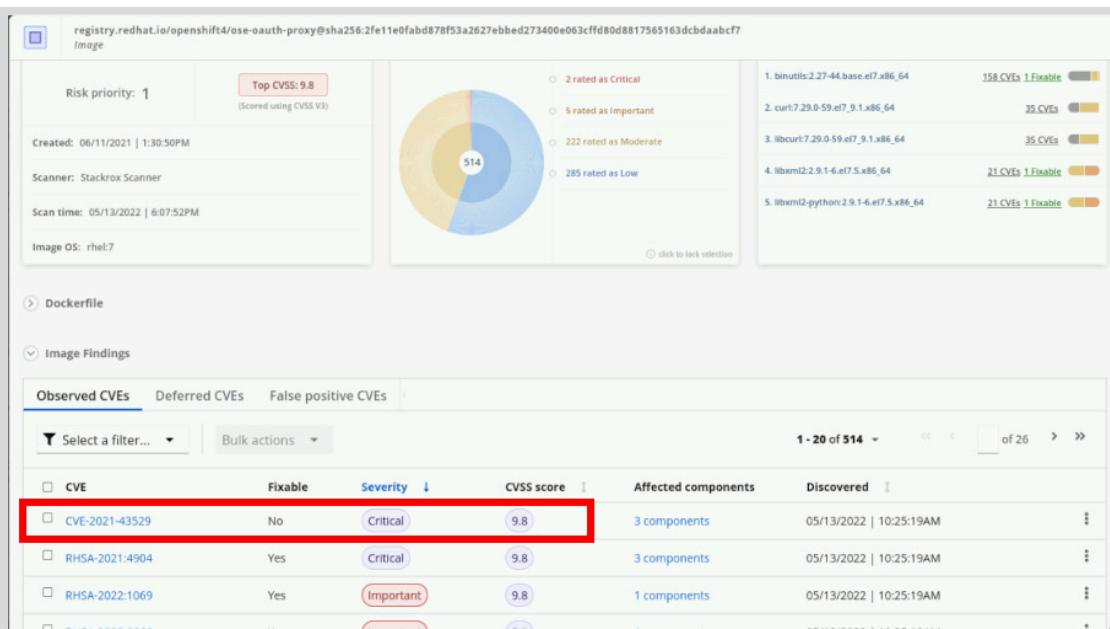
## Before You Begin

To perform this exercise, ensure that you have completed *Guided Exercise: Installing and Configuring RHACS* and *Guided Exercise: Importing Managed Clusters into RHACS*.

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

```
[student@workstation ~]$ lab start operate-vulnerability
```

1. Log in to the RHACS web console.
  1. From the `workstation` machine, navigate to the RHACS web console at `https://central-stackrox.apps.ocp4.example.com`.
  2. Select the `cluster` authentication provider and click **Log in**.
  3. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password. If prompted to authorize access, click **Allow selected permissions**.
2. Locate the Common Vulnerabilities and Exposures (CVE) with the highest score for the top riskiest image.
  1. In the left pane, click **Vulnerability Management > Dashboard**.
  2. In the **TOP RISKIEST IMAGES** dashboard, click **VIEW ALL**. The images are sorted by **Risk Priority**, which is calculated by RHACS.
  3. Click the image in the first row. In this example, the image is `registry.redhat.io/openshift4/ose-oauth-proxy@sha256:2fe11e0fabd878f53a2627ebbed273400e063cffd80d8817565163dcbdaabcf7`. A detailed view of the CVEs affecting the image opens.



In this example, the CVE with the highest score is CVE-2021-43529.



### Note

The CVE with the highest score is subject to change. Use the CVE with the highest score in your lab environment to complete the next steps.

2.4. Click CVE-2021-43529. Review the description and details of this CVE.

► 3. Create a highly-vulnerable deployment.

3.1. Open a terminal and log in to the ocp4 cluster as the `admin` user with the `redhat` password. The API server address 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...
```

3.2. Change to the `~/D0480/labs/operate-vulnerability/` directory.

```
[student@workstation ~]$ cd ~/D0480/labs/operate-vulnerability/
[student@workstation operate-vulnerability]$
```

3.3. Review the `etherpad-deployment.yaml` file with the deployment definition that uses a highly-vulnerable image.

```
...output omitted...
apiVersion: apps/v1
kind: Deployment
...output omitted...
 containers:
```

```

- env:
 - name: TITLE
 value: OpenShift 4 Etherpad
 - name: DEFAULT_PAD_TEXT
 value: OpenShift 4 Etherpad
 name: etherpad
 securityContext:
 {}
 image: "quay.io/redhattraining/etherpad:latest"
...output omitted...

```

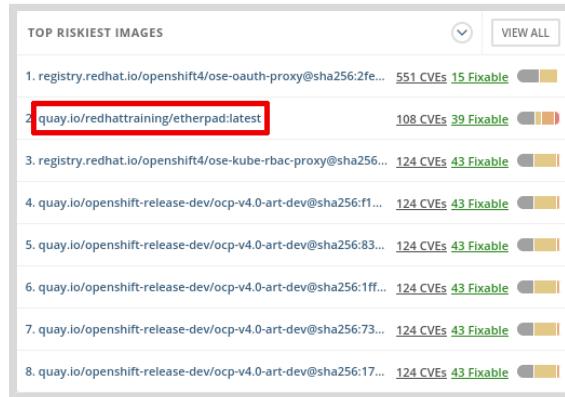
The YAML file also contains instructions to create the `etherpad` namespace, service, route, and persistent volume claim. The deployment uses the `quay.io/redhattraining/etherpad:latest` image.

- 3.4. Use the `oc` command to create the elements described in the `etherpad-deployment.yaml` file.

```
[student@workstation operate-vulnerability]$ oc create -f etherpad-deployment.yaml
namespace/etherpad created
service/etherpad created
route.route.openshift.io/etherpad created
persistentvolumeclaim/etherpad created
deployment.apps/etherpad created
[student@workstation operate-vulnerability]$
```

- 4. Analyze the details of the highly-vulnerable deployment from the RHACS web console.

- 4.1. From Firefox, open the RHACS web console.
- 4.2. In the left pane, click **Vulnerability Management > Dashboard**.
- 4.3. Locate the `quay.io/redhattraining/etherpad:latest` image in the **TOP RISKIEST IMAGES** dashboard.

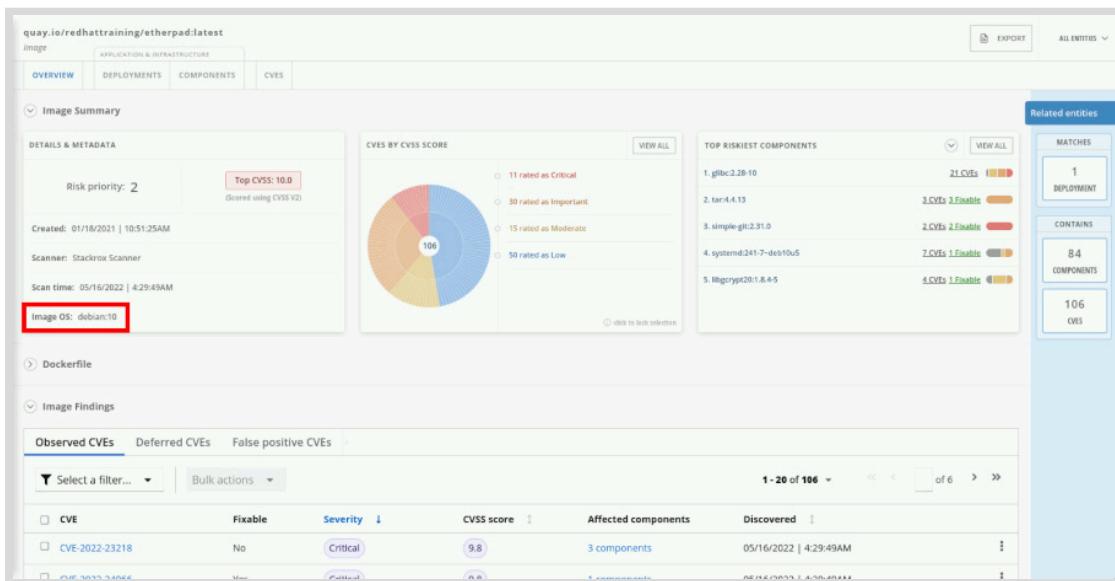


### Note

If the dashboard does not list the `quay.io/redhattraining/etherpad:latest` image, reload the page from the web browser.

- 4.4. In the **TOP RISKIEST IMAGES** dashboard, click `quay.io/redhattraining/etherpad:latest`.

4.5. Review the details of the vulnerabilities affecting this image.



The image contains several high risk vulnerabilities. Among other issues, the image uses an old operating system, `debian:10`. At this point, nothing prevents the vulnerable image from being deployed in the secured cluster.

- 5. Create an RHACS policy to prevent the deployment of images that use the `debian:10` operating system.

- 5.1. From the RHACS web console, navigate to Platform Configuration > Policies.
- 5.2. Click **Create policy**.
- 5.3. In the **Policy details** page, add the following parameters:

| Field name | Value                        |
|------------|------------------------------|
| Name       | <code>debian10-policy</code> |
| Severity   | High                         |
| Categories | Vulnerability Management     |

Click **Next**.

- 5.4. On the **Policy behavior** page, add the following parameters:

| Field name                     | Value              |
|--------------------------------|--------------------|
| Lifecycle stages               | Deploy             |
| Response method                | Inform and enforce |
| Configure enforcement behavior | Enforce on Deploy  |

Policies > debian10-policy

### debian10-policy

Design custom security policies for your environment

Event sources (Runtime lifecycle only)

Deployment  Audit logs

**Response method**

Select a method to address violations of this policy.

Inform  Inform and enforce

Inform and enforce will execute enforcement behavior at the stages you select.

**Configure enforcement behavior**

Based on the fields selected in your policy configuration, you may choose to apply enforcement at the following stages.

**Build**  Enforce on Build

If enabled, ACS will fail your CI builds when images match the conditions of this policy. Download the CLI to get started.

[Download CLI](#)

**Deploy**  Enforce on Deploy

If enabled, ACS will automatically block creation of deployments that match the conditions of this policy. In clusters with the ACS admissions controller enabled, the Kubernetes API server will block noncompliant deployments to prevent pods from being scheduled.

**Runtime**  Enforce on Runtime

If enabled, ACS will either kill the offending pod or block the action taken on the pod. Executions within a pod that match the conditions of the policy will result in the pod being killed. Actions taken through the API server that match policy criteria will be blocked.

[Next](#) [Back](#) [Cancel](#)

Click Next.

- 5.5. In the **Policy criteria** page, locate the **Image OS** drag out policy field in the right pane. You can find it under **Image contents**. Drag the **Image OS** policy field in to the **Policy Section 1** box and type **debian:10** in the **Image operating system:** text field.

Policies > Create policy

### Create policy

Design custom security policies for your environment

1 Policy details  
2 Policy behavior  
**3 Policy criteria**  
4 Policy scope  
5 Review policy

**Policy criteria**

Chain criteria with boolean logic.

**1 Policy Section 1**

Image operating system:  Not

debian:10

+  
- and -  
Drop a policy field inside

[Next](#) [Back](#) [Cancel](#)

Click Next.

- 5.6. On the **Policy scope** page, click **Next**.
- 5.7. On the **Review policy** page, verify that the **etherpad** image shows in the **Preview violations** right pane.

Policies > Create policy

### Create policy

Design custom security policies for your environment

- 1 Policy details
- 2 Policy behavior
- 3 Policy criteria
- 4 Policy scope
- 5 Review policy**

**Review policy**  
Review policy settings and violations.

**debian10-policy**

|             |                          |
|-------------|--------------------------|
| Severity    | High                     |
| Categories  | Vulnerability Management |
| Type        | User generated           |
| Description |                          |
| Rationale   |                          |
| Guidance    |                          |

**MITRE ATT&CK**  
Policy has no MITRE ATT&CK vectors

**Preview violations**  
The policy settings you have selected will generate violations for the Build or Deploy lifecycle stages. Runtime violations are not available in this preview because they are generated in response to future events.

Before you save the policy, verify that the violations seem accurate.

**etherpad**

- Container 'etherpad' has image with base OS 'debian:10'

**Save** **Back** **Cancel**

Click **Save** to create the policy.

- ▶ 6. Attempt to create a new **etherpad** deployment and verify that the policy blocks it.
- 6.1. From the terminal, use the **oc** command to delete all elements in the **etherpad-deployment.yaml** file.

```
[student@workstation operate-vulnerability]$ oc delete -f etherpad-deployment.yaml
namespace "etherpad" deleted
service "etherpad" deleted
route.route.openshift.io "etherpad" deleted
persistentvolumeclaim "etherpad" deleted
deployment.apps "etherpad" deleted
[student@workstation operate-vulnerability]$
```

- 6.2. Use the **oc** command to recreate the elements described in the **etherpad-deployment** YAML file.

```
[student@workstation operate-vulnerability]$ oc create -f etherpad-deployment.yaml
namespace/etherpad created
service/etherpad created
route.route.openshift.io/etherpad created
persistentvolumeclaim/etherpad created
deployment.apps/etherpad created
[student@workstation operate-vulnerability]$
```

- 6.3. Verify that the **debian10-policy** scales down the deployment to zero replicas.

```
[student@workstation ~]$ oc get events -n etherpad | grep debian10-policy
2m27s Warning StackRox enforcement deployment/etherpad
Deployment violated StackRox policy "debian10-policy" and was scaled down
```

- 6.4. Change to the home directory.

```
[student@workstation operate-vulnerability]$ cd ~
[student@workstation ~]$
```

- 6.5. In the left pane of the RHACS web console, click **Violations**.

- 6.6. In the search field, type **debian10-policy** in the **Policy** field.

- 6.7. Click **debian10-policy** and go to the **Enforcement** tab.

The screenshot shows the RHACS web console interface. In the top navigation bar, there is a link to 'Violations' and a breadcrumb trail showing 'Violations > debian10-policy'. Below this, there is a back button labeled '< Back to violations'. The main content area displays the details of the 'debian10-policy' violation, which is associated with the 'etherpad' deployment. The 'Enforcement' tab is selected, indicated by a blue underline. The page content includes a message stating 'Enforcement on this deployment is enabled for this policy', followed by another message about deployment data being evaluated against the policy and scaled to 0 replicas. A note at the bottom suggests reading more on remediation and resolution.

The RHACS web console provides information about the policy violation and the actions taken.

## ► 7. Delete the **debian10-policy**.

- 7.1. From the RHACS web console, navigate to **Platform Configuration > Policies**.

- 7.2. In the **Filter policies** text field, type **Policy**, press Tab, type **debian10-policy** and press Tab.

The screenshot shows the 'Policies' page in the RHACS web console. At the top, there are buttons for 'Create policy' and 'Import policy'. Below this is a search bar with the text 'Policy: debian10-policy'. The main table lists one policy entry: 'debian10-policy'. The table columns include 'Policy', 'Description', 'Status', 'Notifiers', 'Severity', 'Lifecycle', and 'Deploy'. The 'Status' column for the entry shows 'Enabled' with a checked checkbox. The 'Severity' column is marked as 'High'. To the right of the table, there are buttons for 'Reassess all' and '1 - 1 of 1'. The entire row for the 'debian10-policy' entry is highlighted with a red box.

- 7.3. Click the vertical ellipsis (⋮) menu to the right of the policy field, and then click **Delete policy**.

The screenshot shows the 'Policies' page in the RHACS interface. A search bar at the top left contains the text 'Policy' and 'debian10-policy'. To the right are buttons for 'Create policy' and 'Import policy'. The main area displays a table with one row. The columns are 'Policy' (with a dropdown arrow), 'Description' (containing a minus sign), 'Status' (with a radio button set to 'Enabled'), 'Notifiers' (empty), 'Severity' (set to 'High'), and 'Lifecycle' (empty). On the far right of the row, there is a context menu with options: 'Edit policy', 'Clone policy', 'Disable policy', 'Export policy to JSON', and 'Delete policy' (which is highlighted with a red oval).

7.4. When prompted, click **Delete** to confirm.

The page displays the **Successfully deleted policy** message.

## Finish

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

```
[student@workstation ~]$ lab finish operate-vulnerability
```



### Important

Do not delete any resources created in this exercise. They are used in upcoming sections.

This concludes the section.

# Multicloud Configuration Management with RHACS

---

## Objectives

- Detect security policy violations and application or infrastructure misconfigurations by using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) Configuration Management view.

## Introducing Configuration Management Policies

Kubernetes provides a uniform model for configuration and workloads on clusters. Pods, deployments, nodes, and other objects are all representable as YAML or JSON files. These objects have fields, for example, the `kind`, `metadata`, or `spec` fields. Furthermore, depending on the type of object, the `spec` field must follow a certain structure.

The Kubernetes API prevents creating objects without the correct structure.

Although Kubernetes prevents you from creating malformed objects, you can still create undesirable objects. For example, defining requests and limits for CPU and memory is a best practice, but nothing prevents you from creating a deployment without such limits.

Given the uniformity of Kubernetes resources, a single rules system can detect undesirable objects of all kinds.

Red Hat Advanced Cluster Security for Kubernetes (RHACS) includes a system to define rules to identify undesirable configurations. Optionally, this system can prevent creation of these configurations altogether.

RHACS uses the term *configuration management* to describe this system. Configuration management commonly refers to any process that helps keep a system correctly configured. The previous example about ensuring all deployments have resource limits represents the role of configuration management, because a deployment without resource limits is an incorrect configuration.

## RHACS Built-in Policies

RHACS comes preconfigured with many default configuration management rules. Navigate to **Platform Configuration > Policies** to list the preconfigured rules, called policies.

The screenshot shows the RHACS interface with the title "Advanced Cluster Security for Kubernetes". The left sidebar has a dropdown menu set to "Platform Configuration" and a link to "Policies" which is highlighted with a red box. The main area is titled "Policies" and contains a table with the following data:

|                          | Policy                           | Description                     | Status                                      |
|--------------------------|----------------------------------|---------------------------------|---------------------------------------------|
| <input type="checkbox"/> | 30-Day Scan Age                  | Alert on deployments with i...  | <input checked="" type="checkbox"/> Enabled |
| <input type="checkbox"/> | 90-Day Image Age                 | Alert on deployments with i...  | <input checked="" type="checkbox"/> Enabled |
| <input type="checkbox"/> | ADD Command used instead of COPY | Alert on deployments using ...  | Disabled                                    |
| <input type="checkbox"/> | Alpine Linux Package Manager     | Alert on deployments with th... | <input checked="" type="checkbox"/> Enabled |

Figure 9.25: RHACS policies

RHACS categorizes policies around topics, such as privileges, security best practices, and DevOps best practices.

You can examine the policy described previously by searching for the `No resource requests or limits specified` policy. This policy has medium severity, a description and rationale to describe the policy, and guidance to help address violations.

The screenshot shows the "Policy details" page for the "No resource requests or limits specified" policy. The top bar indicates the policy is "Enabled". The "Actions" button is visible. The "Policy details" section contains the following information:

|             |                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------|
| Severity    | Medium                                                                                                        |
| Categories  | DevOps Best Practices, Docker CIS                                                                             |
| Type        | System default                                                                                                |
| Description | Alert on deployments that have containers without resource requests and limits                                |
| Rationale   | If a container does not have resource requests or limits specified then the host may become over-provisioned. |
| Guidance    | Specify the requests and limits of CPU and Memory for your deployment.                                        |

Figure 9.26: The no resource requests or limits specified policy details

You can also review the criteria that define the policy. This policy examines container configuration fields and verifies that resource limit fields are set.

Policy criteria

1 Container CPU limit:  
Is equal to 0. ... OR  
Container CPU request:  
Is equal to 0. ...

2 Container memory limit:  
Is equal to 0. ... OR  
Container memory request:  
Is equal to 0. ...

Figure 9.27: The no resource requests or limits specified policy criteria

Furthermore, this policy excludes some system namespaces from the scope to prevent alerts on system deployments that you do not manage.

RHACS policies are the same mechanism that you used to prevent creating deployments based on images with known vulnerabilities, as described in *Perform Multicluster Vulnerability Management with RHACS*.

## Reviewing Configuration Policies Status

RHACS presents information about policy status in the **Configuration Management** section of the web console.

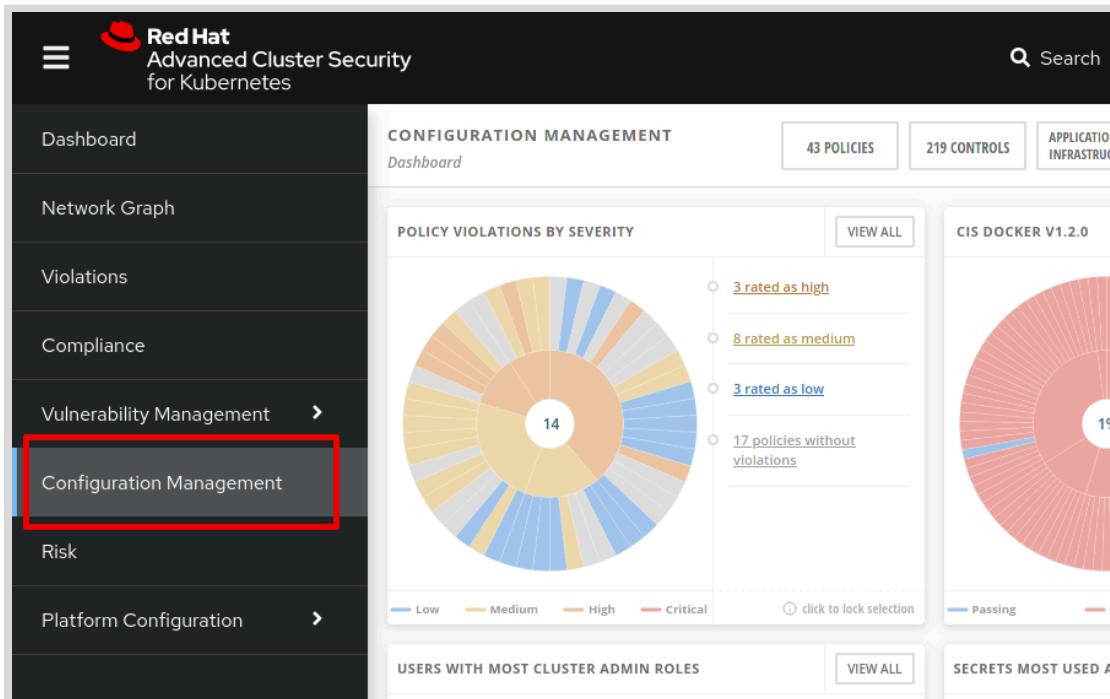


Figure 9.28: The configuration management dashboard

You can view this information from different perspectives.

For example, the **POLICY VIOLATIONS BY SEVERITY** panel in the dashboard shows all policies and their status. The items in the **APPLICATION & INFRASTRUCTURE** menu display information about clusters, namespaces, nodes, deployments, images, and secrets. If you want to find the most severe policy violations in the secured clusters, then the **POLICY VIOLATIONS BY SEVERITY** panel helps you locate them. To examine policy violations in a deployment, navigate to **APPLICATION & INFRASTRUCTURE > Deployments**.

## Additional Configuration Management Information

The Configuration Management section of the web console contains further information that can help you detect misconfigurations.

Controls are rules, similar to policies, that implement standards established by the Center for Internet Security (CIS). RHACS 3.69 provides controls for CIS Docker v1.2.0 and CIS Kubernetes v1.5.

The items in the **APPLICATION & INFRASTRUCTURE** menu list clusters, namespaces, nodes, deployments, images, and secrets in the secured clusters. Besides policy status for each of these objects, the subsequent pages display additional configuration management details. For example, the deployment list displays information about images and secrets used in the deployment and the associated service account.

The menu items in the **RBAC VISIBILITY & CONFIGURATION** menu list users and groups, service accounts, and roles in the secured clusters. In addition to each object's policy status, the subsequent pages display additional configuration management details. You can filter these objects with different criteria, such as the cluster, namespace, or role.

## Adding New Policies

As described in *Perform Multicloud Vulnerability Management with RHACS*, you can configure new policies to identify and prevent misconfigurations.

The following list enumerates the kinds of validations that you can perform in policies, in addition to those related to image vulnerabilities.

- Container configuration, such as privileged containers, environment variables, or memory and CPU requests
- Deployment configuration, such as labels or annotations
- Storage, such as volume type, writable host mounts or volumes, or mount propagation
- Networking, such as exposed ports
- Process activity, such as process names or arguments
- Kubernetes access, such as service account or minimum RBAC permissions
- Kubernetes events, such as user, group, or action

Note that many default policies in RHACS are disabled by default. Some of these policies, such as the `Wget in Image` policy, are not useful in all environments, but you can activate those policies easily if you consider them valuable. Some others, such as the `Required Image Label` policy, are templates that you can clone to provide a starter point for creating your own custom policies.



## References

For more information about security policies, refer to the *Managing security policies* chapter of the RHACS *Operating* Guide at  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_advanced\\_cluster\\_security\\_for\\_kubernetes/3.69/html-single/operating/index#manage-security-policies](https://access.redhat.com/documentation/en-us/red_hat_advanced_cluster_security_for_kubernetes/3.69/html-single/operating/index#manage-security-policies)

## ► Guided Exercise

# Multicloud Configuration Management with RHACS

- Use the Red Hat Advanced Cluster Security for Kubernetes (RHACS) configuration management view to identify a deployment that does not set Kubernetes requests and limits. Then, you modify the deployment and verify that the configuration issue is resolved.

## Outcomes

- Explore the RHACS configuration management dashboard to locate a deployment.
- Identify the configuration management issues affecting a deployment.
- Verify that the configuration issue is resolved.

## Before You Begin

To perform this exercise, ensure that you have completed *Guided Exercise: Installing and Configuring RHACS* and *Guided Exercise: Importing Managed Clusters into RHACS*.

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

```
[student@workstation ~]$ lab start operate-management
```

### ► 1. Create the etherpad deployment.

1. From the workstation machine, open a terminal and log in to the ocp4 cluster as the admin user with the redhat password. The API server address 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...
```

2. Change to the `~/D0480/labs/operate-management/` directory.

```
[student@workstation ~]$ cd ~/D0480/labs/operate-management/
[student@workstation operate-management]$
```

3. Review the `etherpad-deployment.yaml` file.

```
...output omitted...
apiVersion: apps/v1
kind: Deployment
```

```
...output omitted...
 containers:
 - env:
 - name: TITLE
 value: OpenShift 4 Etherpad
 - name: DEFAULT_PAD_TEXT
 value: OpenShift 4 Etherpad
 name: etherpad
 securityContext:
 {}
 image: "quay.io/redhattraining/etherpad:latest"
...output omitted...
```

The YAML file also contains instructions to create the `etherpad` namespace, service, route, and persistent volume claim. The deployment uses the `quay.io/redhattraining/etherpad:latest` image. There are no configuration parameters regarding the Kubernetes resources for the pod.

14. Use the `oc` command to create the elements described in the `etherpad-deployment.yaml` file.

```
[student@workstation operate-management]$ oc create -f etherpad-deployment.yaml
namespace/etherpad created
service/etherpad created
route.route.openshift.io/etherpad created
persistentvolumeclaim/etherpad created
deployment.apps/etherpad created
[student@workstation operate-management]$
```

▶ 2. Log in to the RHACS web console.

- 2.1. Use Firefox to navigate to the RHACS web console at <https://central-stackrox.apps.ocp4.example.com>.
- 2.2. Select the `cluster` authentication provider and click **Log in**.
- 2.3. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password. If prompted to authorize access, click **Allow selected permissions**.

▶ 3. Locate the `etherpad` deployment on the Configuration Management page.

- 3.1. In the left pane, click **Configuration Management**.
- 3.2. In the top right corner, click **APPLICATION & INFRASTRUCTURE**, and then **Deployments**. The deployments list is displayed.
- 3.3. Click **Namespace** to sort by namespace in ascending order.
- 3.4. Click the `etherpad` deployment row. A detailed view opens.
- 3.5. Scroll down to the **Deployment Findings**. Locate the policy **No resource requests or limits specified**.



The screenshot shows a table titled "Deployment Findings" with the heading "6 POLICIES FAILED ACROSS THIS DEPLOYMENT". The table has four columns: Policy, Enforced, Severity, and Categories. The rows represent the following policies:

| Policy                                          | Enforced | Severity | Categories                                     |
|-------------------------------------------------|----------|----------|------------------------------------------------|
| 90-Day Image Age                                | Yes      | Low      | DevOps Best Practices, Security Best Practices |
| Latest tag                                      | Yes      | Low      | DevOps Best Practices                          |
| Ubuntu Package Manager in Image                 | Yes      | Low      | Security Best Practices                        |
| No resource requests or limits specified        | Yes      | Medium   | DevOps Best Practices, Docker CIS              |
| Fixable Severity at least Important             | Yes      | High     | Vulnerability Management                       |
| Pod Service Account Token Automatically Mounted | Yes      | Medium   | Security Best Practices, Privileges            |

This policy applies to the DevOps Best Practices and Docker CIS categories.

► 4. Remove the etherpad deployment.

- 4.1. From the terminal, use the oc command to delete all the elements contained in the etherpad-deployment.yaml file.

```
[student@workstation operate-management]$ oc delete -f etherpad-deployment.yaml
namespace "etherpad" deleted
service "etherpad" deleted
route.route.openshift.io "etherpad" deleted
persistentvolumeclaim "etherpad" deleted
deployment.apps "etherpad" deleted
[student@workstation operate-management]$
```

► 5. Create an etherpad deployment with configured limits and requests.

- 5.1. Review the etherpad-deployment-limits.yaml file. Notice that this file contains configuration parameters about Kubernetes requests and limits.

```
[student@workstation operate-management]$ cat etherpad-deployment-limits.yaml
...output omitted...

resources:
 requests:
 memory: "128Mi"
 cpu: "250m"
 limits:
 memory: "256Mi"
 cpu: "500m"

...output omitted...
```

- 5.2. Use the oc command to create all the elements contained in the etherpad-deployment-limits.yaml file.

```
[student@workstation operate-management]$ oc create -f \
 etherpad-deployment-limits.yaml
namespace/etherpad created
service/etherpad created
route.route.openshift.io/etherpad created
persistentvolumeclaim/etherpad created
deployment.apps/etherpad created
```

5.3. Change to the home directory.

```
[student@workstation operate-management]$ cd ~
[student@workstation ~]$
```

► 6. Verify that RHACS does not report the configuration issue related to requests and limits.

- 6.1. From the RHACS web console, navigate to **Configuration Management**.
- 6.2. In the top right corner, click **APPLICATION & INFRASTRUCTURE**, then click **Deployments**. The deployments list is displayed.
- 6.3. Type **etherpad** in the **Deployment** field.
- 6.4. Click the **etherpad** deployment row. A detailed view opens.
- 6.5. Scroll down to the **Deployment Findings**. Verify that the **No resource requests or limits specified** policy is no longer shown.

| 5 POLICIES FAILED ACROSS THIS DEPLOYMENT        |          |          |                                                |  |
|-------------------------------------------------|----------|----------|------------------------------------------------|--|
| Policy                                          | Enforced | Severity | Categories                                     |  |
| 90-Day Image Age                                | Yes      | Low      | DevOps Best Practices, Security Best Practices |  |
| Latest tag                                      | Yes      | Low      | DevOps Best Practices                          |  |
| Ubuntu Package Manager in Image                 | Yes      | Low      | Security Best Practices                        |  |
| Fixable Severity at least Important             | Yes      | High     | Vulnerability Management                       |  |
| Pod Service Account Token Automatically Mounted | Yes      | Medium   | Security Best Practices, Privileges            |  |



### Note

If the deployment does not exist, or the page does not show the expected result, reload the web browser page.

## Finish

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

```
[student@workstation ~]$ lab finish operate-management
```

This concludes the section.

## ▶ Lab

# Multicloud Operational Security Using RHACS

- Configure Red Hat Advanced Cluster Security for Kubernetes (RHACS), generate an init bundle, and import a managed cluster. Then, you create a policy to prevent the deployment of vulnerable images. Finally, you deploy an application by using an image hosted in the private Red Hat Quay registry.

## Outcomes

- Generate a cluster init bundle by using the RHACS dashboard.
- Create cluster init secrets in all the managed clusters.
- Verify the import of all the clusters from the RHACS web console.
- Add a secret to the service account to pull images from the private Red Hat Quay registry.
- Create an RHACS policy to prevent the deployment of vulnerable images.
- Verify that RHACS blocks the deployment of vulnerable images.

## 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 RHACS and Quay are available.

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

- As the `cloudadmin` user with the `redhat` password, push the `quay.io/redhattraining/do480-hello-app` image to the `hello-app` private repository of the `cloudadmin` user.  
Push the image `quay.io/redhattraining/do480-hello-app` for both the `v1.0` and `latest` tags. The central quay registry URL is `central-quay-registry.apps.ocp4.example.com`.
- Create a secret named `quaysecret` in the `operate-review` namespace for the central Quay registry authentication. Add the secret to the default service account to pull the image from the Quay registry.
- Using the `htpasswd_provider` option, log in to the OpenShift web console as the `admin` user with the `redhat` password. Retrieve the default `admin` password for the RHACS central dashboard by using the OpenShift web console. As the `admin` user, log in to the RHACS central dashboard.  
The OpenShift console URL is `https://console-openshift-console.apps.ocp4.example.com`
- Create an integration for the `Cluster Init Bundle` bundle.
- Create the `operate-import` cluster secrets on both the `ocp4-mng` and `ocp4` clusters.

6. Verify that the local-cluster and managed-cluster clusters are imported successfully in RHACS.
7. Create an RHACS policy to prevent the deployment of images by using the latest tag from the central registry.
8. Deploy the hello-quay application to the production environment by using the ~/D0480/labs/operate-review/hello-quay.yaml file.  
The hello-quay application is using the quay.io/redhattraining/do480-hello-app image with the latest tag. Verify that the application is deployed and the application URL, hello-quay.apps.ocp4.example.com, is working.
9. Deploy the hello-central application by using the ~/D0480/labs/operate-review/hello-central.yaml file.  
The hello-central application is using central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app image with the latest tag. Verify that the latest-tag-policy RHACS policy scaled down the deployment.
10. Remove the elements described in the hello-central YAML file. Update the hello-central.yaml file to use the v1.0 tag. Verify that the application is deployed and the application URL, hello-central.apps.ocp4.example.com, is working.

## 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 operate-review
```

## Finish

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

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

This concludes the section.

## ► Solution

# Multicluster Operational Security Using RHACS

- Configure Red Hat Advanced Cluster Security for Kubernetes (RHACS), generate an init bundle, and import a managed cluster. Then, you create a policy to prevent the deployment of vulnerable images. Finally, you deploy an application by using an image hosted in the private Red Hat Quay registry.

## Outcomes

- Generate a cluster init bundle by using the RHACS dashboard.
- Create cluster init secrets in all the managed clusters.
- Verify the import of all the clusters from the RHACS web console.
- Add a secret to the service account to pull images from the private Red Hat Quay registry.
- Create an RHACS policy to prevent the deployment of vulnerable images.
- Verify that RHACS blocks the deployment of vulnerable images.

## 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 RHACS and Quay are available.

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

- As the `cloudadmin` user with the `redhat` password, push the `quay.io/redhattraining/do480-hello-app` image to the `hello-app` private repository of the `cloudadmin` user.  
Push the image `quay.io/redhattraining/do480-hello-app` for both the `v1.0` and `latest` tags. The central quay registry URL is `central-quay-registry.apps.ocp4.example.com`.
  - From the `workstation` machine, open a terminal and log in as the `cloudadmin` user with the `redhat` password.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
```

- Push the `quay.io/redhattraining/do480-hello-app:v1.0` image to the private repository of the `cloudadmin` user by using the `skopeo` command.

```
[student@workstation ~]$ skopeo copy \
docker://quay.io/redhattraining/do480-hello-app:v1.0 \
docker://central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app:v1.0
Getting image source signatures
...output omitted...
```

- 1.3. Push the quay.io/redhattraining/do480-hello-app:latest image to the private repository of the cloudadmin user by using the skopeo command.

```
[student@workstation ~]$ skopeo copy \
docker://quay.io/redhattraining/do480-hello-app:latest \
docker://central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app:latest
Getting image source signatures
...output omitted...
```

2. Create a secret named quaysecret in the operate-review namespace for the central Quay registry authentication. Add the secret to the default service account to pull the image from the Quay registry.
  - 2.1. Log in to the ocp4 cluster as the admin user with the redhat password. The API server address 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...
```

- 2.2. Create the operate-review project.

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

- 2.3. Review the /run/user/1000/containers/auth.json file. The JSON file contains authentication details for the central-quay-registry.apps.ocp4.example.com registry.

```
[student@workstation ~]$ cat /run/user/1000/containers/auth.json
{
 "auths": {
 "central-quay-registry.apps.ocp4.example.com": {
 "auth": "Y2xvdWRhZG1pbjpyZWRoYXQ="
 }
 }
}
```

**Note**

If authentication details are unavailable in the /run/user/1000/containers/auth.json file, then run the podman login command.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
central-quay-registry.apps.ocp4.example.com
```

- 2.4. Create the quaysecret secret for the central Quay registry authentication.

```
[student@workstation ~]$ oc create secret generic quaysecret \
--from-file=.dockerconfigjson=/run/user/1000/containers/auth.json \
--type=kubernetes.io/dockerconfigjson -n operate-review
secret/quaysecret created
```

- 2.5. Add the quaysecret secret to the default service account named default.

```
[student@workstation ~]$ oc secrets link default quaysecret \
--for=pull -n operate-review
```

3. Using the htpasswd\_provider option, log in to the OpenShift web console as the admin user with the redhat password. Retrieve the default admin password for the RHACS central dashboard by using the OpenShift web console. As the admin user, log in to the RHACS central dashboard.

The OpenShift console URL is https://console-openshift-console.apps.ocp4.example.com

- 3.1. From the workstation machine, open Firefox and access https://console-openshift-console.apps.ocp4.example.com.
- 3.2. Click **htpasswd\_provider** and log in as the admin user with the redhat password.
- 3.3. Navigate to **Home > Projects** to display the **Projects** page, and then click **stackrox** to select the stackrox namespace.
- 3.4. Navigate to **Operators > Installed Operators** to display installed operators, and then click **Central** to display the **Centrals** page.

| Name                                     | Managed Namespaces | Status                                                       | Provided APIs              |
|------------------------------------------|--------------------|--------------------------------------------------------------|----------------------------|
| Red Hat Quay                             | All Namespaces     | <span style="color: green;">✓ Succeeded</span><br>Up to date | Quay Registry              |
| Advanced Cluster Security for Kubernetes | All Namespaces     | <span style="color: green;">✓ Succeeded</span><br>Up to date | Central<br>Secured Cluster |

- 3.5. Click **stackrox-central-services** to display **stackrox-central-services** page. Copy the command from the **Admin Credentials Info** section to retrieve the default password for the **admin** user.

**Central overview**

|                                                                                                                                                                                                                                                                                  |                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| <b>Name</b>                                                                                                                                                                                                                                                                      | stackrox-central-services |
| <b>Namespace</b>                                                                                                                                                                                                                                                                 | NS stackrox               |
| <b>Labels</b>                                                                                                                                                                                                                                                                    | No labels                 |
| <b>Annotations</b>                                                                                                                                                                                                                                                               |                           |
| <b>Central</b>                                                                                                                                                                                                                                                                   |                           |
| <b>Admin Credentials Info</b>                                                                                                                                                                                                                                                    |                           |
| A password for the 'admin' user has been automatically generated and stored in the "password" entry of the central-htpasswd secret. To view the password, run <code>oc -n stackrox get secret central-htpasswd -o go-template='{{index .data "password"   base64decode}}'</code> |                           |
| <b>Product Version</b>                                                                                                                                                                                                                                                           | 3.69.0                    |

- 3.6. Retrieve the default password for the **admin** user from the command line.

```
[student@workstation ~]$ oc -n stackrox get secret central-htpasswd \
-o go-template='{{index .data "password" | base64decode}}'
WcRPchXqaP6KAoLbyCSTR9v9d
```

- 3.7. Run the `oc get route` command to retrieve the URL of the RHACS web interface.

```
[student@workstation ~]$ oc get route -n stackrox
NAME HOST/PORT
TERMINATION WILDCARD
central central-stackrox.apps.ocp4.example.com
passthrough None
...output omitted...
```

3.8. Open Firefox and access <https://central-stackrox.apps.ocp4.example.com>.



### Note

If Firefox displays a warning message, then add exceptions to permit the CENTRAL\_SERVICE certificate.

3.9. From the list of authentication providers, select **Login with username/password**. Log in as the **admin** user with the password retrieved in the preceding step.

**4.** Create an integration for the **Cluster Init Bundle** bundle.

**4.1.** On the RHACS web console, navigate to **Platform Configuration > Integrations** to display the **Integrations** page. Click **Cluster Init Bundle** in the **Authentication Tokens** section.

4.2. On the **Integrations** page, click **Generate bundle**.

4.3. Enter the **operate-import** value in the **Cluster init bundle name** field and click **Generate**.

4.4. On the **Configure Cluster Init Bundle Integration** page, click **Download Kubernetes secrets file** to download the secrets file. In the confirmation window, select **Save File** and click **OK**.

**Integrations > Cluster Init Bundle > Create Integration**

## Configure Cluster Init Bundle Integration

Integration was saved successfully

Please copy the generated cluster init bundle YAML file and store it safely. You will not be able to access it again after you leave this screen.

[Download Helm values file](#)

[Download Kubernetes secrets file](#)

|      |                |
|------|----------------|
| Name | operate-import |
|------|----------------|

5. Create the `operate-import` cluster secrets on both the `ocp4-mng` and `ocp4` clusters.
  - 5.1. Return to the terminal and log in to the `ocp4-mng` cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 5.2. Review the `operate-import-cluster-init-secrets.yaml` cluster init secrets file.

```
[student@workstation ~]$ cat ~/Downloads/operate-import-cluster-init-secrets.yaml
...output omitted...
apiVersion: v1
kind: Secret
metadata:
 annotations:
 init-bundle.stackrox.io/created-at: "2022-05-04T10:34:27.020155453Z"
 init-bundle.stackrox.io/expires-at: "2023-05-04T10:34:00Z"
 init-bundle.stackrox.io/id: e456829c-e586-4538-8732-cdceb7de2f40
 init-bundle.stackrox.io/name: operate-import
 creationTimestamp: null
 name: collector-tls
stringData:
 ca.pem: |
 ...output omitted...
apiVersion: v1
kind: Secret
metadata:
 annotations:
 init-bundle.stackrox.io/created-at: "2022-05-04T10:34:27.020155453Z"
 init-bundle.stackrox.io/expires-at: "2023-05-04T10:34:00Z"
 init-bundle.stackrox.io/id: e456829c-e586-4538-8732-cdceb7de2f40
 init-bundle.stackrox.io/name: operate-import
```

```

creationTimestamp: null
name: sensor-tls
stringData:
 ca.pem: |
...output omitted...
apiVersion: v1
kind: Secret
metadata:
 annotations:
 init-bundle.stackrox.io/created-at: "2022-05-04T10:34:27.020155453Z"
 init-bundle.stackrox.io/expires-at: "2023-05-04T10:34:00Z"
 init-bundle.stackrox.io/id: e456829c-e586-4538-8732-cdceb7de2f40
 init-bundle.stackrox.io/name: operate-import
 creationTimestamp: null
 name: admission-control-tls
 stringData:
 admission-control-cert.pem: |
...output omitted...

```

The file contains three secrets: `collector-tls`, `sensor-tls`, and `admission-control-tls`.

- 5.3. Create the secrets in the `stackrox` namespace by using the `operate-import-cluster-init-secrets.yaml` file.

```
[student@workstation ~]$ oc create -f \
~/Downloads/operate-import-cluster-init-secrets.yaml -n stackrox
secret/collector-tls created
secret/sensor-tls created
secret/admission-control-tls created
```

- 5.4. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 5.5. Create the secrets in the `stackrox` namespace by using the `operate-import-cluster-init-secrets.yaml` file.

```
[student@workstation ~]$ oc create -f \
~/Downloads/operate-import-cluster-init-secrets.yaml -n stackrox
secret/collector-tls created
secret/sensor-tls created
secret/admission-control-tls created
```

6. Verify that the `local-cluster` and `managed-cluster` clusters are imported successfully in RHACS.

- 6.1. From the RHACS web console, navigate to **Platform Configuration > Clusters** to display the **CLUSTERS** page. The **CLUSTERS** page displays both the `local-cluster` and `managed-cluster` clusters.

**Note**

It takes 15-20 minutes for clusters to import to RHACS successfully and show a **Healthy** status.

7. Create an RHACS policy to prevent the deployment of images by using the `latest` tag from the `central` registry.
  - 7.1. From the RHACS web console, navigate to **Platform Configuration > Policies**.
  - 7.2. Click **Create policy**.
  - 7.3. On the **Policy details** page, add the following parameters:

| Field name | Value                          |
|------------|--------------------------------|
| Name       | <code>latest-tag-policy</code> |
| Severity   | High                           |
| Categories | Vulnerability Management       |

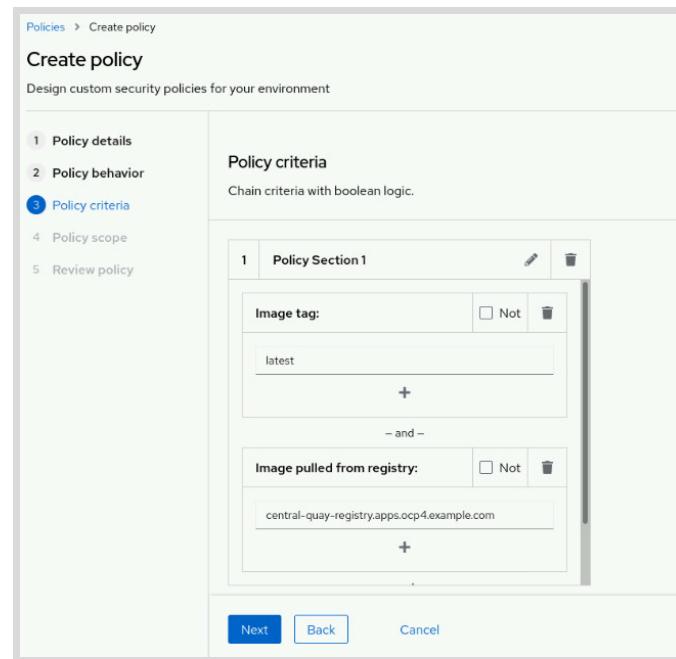
And click **Next**.

- 7.4. On the **Policy behavior** page, add the following parameters:

| Field name                     | Value              |
|--------------------------------|--------------------|
| Lifecycle stages               | Deploy             |
| Response method                | Inform and enforce |
| Configure enforcement behavior | Enforce on Deploy  |

Click **Next**.

- 7.5. In the **Drag out policy field** pane, expand the **Image registry** list. Drag the **Image tag** tile to the **Policy Section 1** list and type `latest` in the **Image tag** text field.
- 7.6. Drag the **Image registry** tile to the **Policy Section 1** list and type `central-quay-registry.apps.ocp4.example.com` in the **Image pulled from registry** text field.



Click **Next**.

- 7.7. On the **Policy scope** page, click **Next**.
- 7.8. On the **Review policy** page, click **Save** to create the policy.
8. Deploy the **hello-quay** application to the production environment by using the `~/D0480/labs/operate-review/hello-quay.yaml` file.

The **hello-quay** application is using the `quay.io/redhattraining/do480-hello-app` image with the `latest` tag. Verify that the application is deployed and the application URL, `hello-quay.apps.ocp4.example.com`, is working.

- 8.1. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 8.2. Change to the `~/D0480/labs/operate-review/` directory.

```
[student@workstation ~]$ cd ~/D0480/labs/operate-review/
[student@workstation operate-review]$
```

- 8.3. Review the `hello-quay.yaml` file. The YAML file uses an image from the `quay.io` registry with the `latest` tag.

```
[student@workstation operate-review]$ cat hello-quay.yaml
...output omitted...
spec:
 containers:
 - name: hello-quay
 image: quay.io/redhattraining/do480-hello-app:latest
...output omitted...
```

- 8.4. Use the `oc` command to create the elements described in the `hello-quay` YAML file.

```
[student@workstation operate-review]$ oc create -f hello-quay.yaml
deployment.apps/hello-quay created
service/hello-quay created
route.route.openshift.io/hello-quay created
```

- 8.5. Verify the status of the `hello-quay` application.

```
[student@workstation operate-review]$ oc get deployment \
hello-quay -n operate-review
NAME READY UP-TO-DATE AVAILABLE AGE
hello-quay 1/1 1 1 53s
```

- 8.6. Test the `hello-quay` application deployment.

```
[student@workstation operate-review]$ curl hello-quay.apps.ocp4.example.com
Hello Application!
Image version : latest
```

9. Deploy the `hello-central` application by using the `~/D0480/labs/operate-review/hello-central.yaml` file.

The `hello-central` application is using `central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app` image with the `latest` tag. Verify that the `latest-tag-policy` RHACS policy scaled down the deployment.

- 9.1. Review the `hello-central.yaml` file. The YAML file uses an image from `central-quay-registry.apps.ocp4.example.com` registry with the `latest` tag.

```
[student@workstation operate-review]$ cat hello-central.yaml
...output omitted...
spec:
 containers:
 - name: hello-central
 image: central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-
app:latest
...output omitted...
```

- 9.2. Use the `oc` command to create the elements described in the `hello-central` YAML file.

```
[student@workstation operate-review]$ oc create -f hello-central.yaml
deployment.apps/hello-central created
service/hello-central created
route.route.openshift.io/hello-central created
```

9.3. Verify that the `latest-tag-policy` scales down the deployment to zero replicas.

```
[student@workstation ~]$ oc get events -n operate-review | grep latest-tag-policy
11s Warning StackRox enforcement deployment/hello-central
Deployment violated StackRox policy "latest-tag-policy" and was scaled down
```

10. Remove the elements described in the `hello-central` YAML file. Update the `hello-central.yaml` file to use the `v1.0` tag. Verify that the application is deployed and the application URL, `hello-central.apps.ocp4.example.com`, is working.

10.1. Use the `oc` command to delete all elements in the `hello-central.yaml` file.

```
[student@workstation operate-review]$ oc delete -f hello-central.yaml
deployment.apps "hello-central" deleted
service "hello-central" deleted
route.route.openshift.io "hello-central" deleted
```

10.2. Change the image tag from `latest` to `v1.0` in the `hello-central.yaml` file.

```
[student@workstation operate-review]$ vim hello-central.yaml
...output omitted...
spec:
 containers:
 - image: central-quay-registry.apps.ocp4.example.com/clouadmin/hello-
app:v1.0
 imagePullPolicy: Always
 name: hello-central
...output omitted...
```

10.3. Use the `oc` command to create the elements described in the `hello-central` YAML file.

```
[student@workstation operate-review]$ oc create -f hello-central.yaml
deployment.apps/hello-central created
service/hello-central created
route.route.openshift.io/hello-central created
```

10.4. Verify the status of the `hello-central` application.

```
[student@workstation operate-review]$ oc get deployment \
hello-central -n operate-review
NAME READY UP-TO-DATE AVAILABLE AGE
hello-central 1/1 1 1 10m
```

10.5. Test the `hello-central` application deployment.

```
[student@workstation operate-review]$ curl hello-central.apps.ocp4.example.com
Hello Application!
Image version : 1.0
```

## 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 operate-review
```

## Finish

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

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

This concludes the section.

# Summary

---

- You can integrate Red Hat Advanced Cluster Security for Kubernetes (RHACS) with several image registries for vulnerability management.
- RHACS scans the deployments to detect secrets with credentials to image registries and automatically creates integrations with those registries.
- You can see the vulnerabilities affecting the cluster fleet from the RHACS vulnerability management dashboard.
- You can use RHACS policies to avoid the deployment of vulnerable images.
- You can analyze the configuration status of the cluster fleet from the RHACS configuration management dashboard.
- You can use RHACS policies to follow recommended practices about configuration management.

## Chapter 10

# Comprehensive Review

### Goal

Review tasks from *Multicluster Management with Red Hat OpenShift Platform Plus*

### Objectives

- Review tasks from *Multicluster Management with Red Hat OpenShift Platform Plus*

### Sections

- Comprehensive Review

### Lab

- Configure RHACS, Red Hat Quay, and RHACM Observability Stack
- Deploy and Update an Application to Align with the RHACS Policy

# Comprehensive Review

---

## Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills that you learned in Multicluster Management with Red Hat OpenShift Platform Plus.

### Reviewing Multicluster Management with Red Hat OpenShift Platform Plus

Before beginning the comprehensive review for this course, you should be comfortable with the topics covered in each chapter. Do not hesitate to ask the instructor for extra guidance or clarification on these topics.

#### ***Managing a Multicluster Kubernetes Architecture***

Describe multicluster architectures and use Red Hat OpenShift Platform Plus to solve their challenges.

- Identify common use cases for multiple Kubernetes clusters and the challenges presented by multicluster architectures.
- Identify the tools provided by OpenShift Platform Plus to facilitate the end-to-end management, security, and compliance of Kubernetes fleets on hybrid clouds.
- Install the RHACM operator from the operator hub in a Red Hat OpenShift hub cluster.

#### ***Inspecting Resources from Multiple Clusters Using the RHACM Web Console***

Describe and navigate the Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console. Configure role-based access control (RBAC) and search for resources across multiple clusters by using the RHACM search engine.

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

#### ***Deploying and Managing Policies for Multiple Clusters with RHACM***

Deploy and manage policies in a multicluster environment by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance.

- Deploy policies on multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.

- Deploy the compliance operator policy and view compliance reports from multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.
- Deploy the gatekeeper policy and gatekeeper constraints to multiple clusters by using the command line and the Red Hat Advanced Cluster Management for Kubernetes (RHACM) governance dashboard.

## ***Installing and Customizing the RHACM Observability Stack***

Gain insight into the fleet of managed clusters by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability components.

- Enable the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability stack and describe the architecture and the benefits of observability in a multicluster environment.
- Customize the Red Hat Advanced Cluster Management for Kubernetes (RHACM) observability stack.

## ***Deploying Applications Across Multiple Clusters with RHACM***

Deploy and manage applications in a multicluster environment with Red Hat Advanced Cluster Management for Kubernetes GitOps.

- Describe and define GitOps concepts and the resources of the Red Hat Advanced Cluster Management for Kubernetes (RHACM) application model.
- Deliver applications into multiple clusters by using Red Hat Advanced Cluster Management for Kubernetes (RHACM) GitOps.
- Describe Kustomize concepts and features, and deploy new customizations with the Kustomize command-line tool.

## ***Installing and Configuring Red Hat Quay***

Install and configure Red Hat Quay on Red Hat OpenShift Container Platform (RHOCP).

- Describe the architecture and identify the main benefits of Red Hat Quay.
- Install the Quay operator and deploy and configure a Quay registry using the command line.
- Describe the Red Hat Quay concepts for image registry organization, and manage role-based access control (RBAC) by using LDAP

## ***Integrating Red Hat Quay with Red Hat OpenShift and RHACM***

Describe Red Hat Quay use cases in a multicluster environment, and use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to deploy applications and control the image sources allowed in the cluster fleet.

- Identify the different Red Hat Quay use cases and architectures in a multicluster environment, and the benefits of using a central registry.
- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to restrict container image source registries in your cluster fleet.
- Use Red Hat Advanced Cluster Management for Kubernetes (RHACM) to deploy an application using a central Red Hat Quay registry.

## **Installing and Configuring RHACS**

Install and configure Red Hat Advanced Cluster Security for Kubernetes (RHACS) and learn how it can help organizations with security in multicluster environments.

- Identify security concerns in Kubernetes multicluster environments and potential fixes offered by Red Hat Advanced Cluster Security for Kubernetes (RHACS).
- Install and configure Red Hat Advanced Cluster Security for Kubernetes (RHACS) and become familiar with the RHACS architecture, installation methods, and available customizations.
- Import managed clusters into Red Hat Advanced Cluster Security for Kubernetes (RHACS) and retrieve basic security information from the cluster fleet using the RHACS console.

## **Multicluster Operational Security Using RHACS**

Manage the operational security of a Kubernetes cluster fleet using Red Hat Advanced Cluster Security for Kubernetes (RHACS), and integrate RHACS with external services.

- Integrate Red Hat Advanced Cluster Security for Kubernetes (RHACS) with Red Hat Quay and list the benefits of integrating RHACS with other tools.
- Analyze the vulnerabilities affecting the cluster fleet using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) vulnerability dashboard and reporting, and create a policy to prevent the deployment of vulnerable images.
- Detect security policy violations and application or infrastructure misconfigurations by using the Red Hat Advanced Cluster Security for Kubernetes (RHACS) Configuration Management view.

## ▶ Lab

# Configure RHACS, Red Hat Quay, and RHACM Observability Stack

- Install and configure a Red Hat Advanced Cluster Security for Kubernetes (RHACS) secured cluster to import all the clusters managed by Red Hat Advanced Cluster Management for Kubernetes (RHACM) into RHACS by using an RHACM policy. Then, you create a Prometheus custom alerting rule and push an image to the Red Hat Quay private registry.

## Outcomes

- Push an image to the private Red Hat Quay registry.
- Add a secret to a service account to pull images from the private Red Hat Quay registry.
- Generate a cluster `init bundle` by using the RHACS dashboard.
- Create cluster init secrets in all the managed clusters.
- Create the RHACS `SecuredCluster` custom resource in all the clusters that use an RHACM policy.
- Verify the import of all the clusters from the RHACS web console.
- Create a Prometheus custom alerting rule.

## Before You Begin

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

This command prepares your environment and ensures that all required resources are available.

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

## Specifications

- Push the `hello-app` image to the central private Quay registry of the `cloudadmin` user:
  - The image is available at `quay.io/redhattraining/do480-hello`.
  - The central quay registry URL is `central-quay-registry.apps.ocp4.example.com`.
  - Push the image for `v10`, `v11`, and `latest` tag.
  - Push the image as the `cloudadmin` user with the `redhat` password.
- Create a secret named `compsecret` from the authentication details of the `cloudadmin` user for the central registry, and add the secret to the default service account to pull the image from the Quay registry on both `ocp4` and `ocp4-mng` clusters:

- Create `compsecret` in the `comprevue-install` namespace.
  - The API server address is `https://api.ocp4.example.com:6443` for the `ocp4` cluster.
  - The API server address is `https://api.ocp4-mng.example.com:6443` for the `ocp4-mng` cluster.
  - Log in to the `ocp4` and `ocp4-mng` clusters as the `admin` user with the `redhat` password.
- Retrieve the default `admin` password for the RHACS central dashboard and log in to the RHACS central dashboard:
    - The OpenShift console URL is `https://console-openshift-console.apps.ocp4.example.com`.
    - Use the `htpasswd_provider` option to log in to the OpenShift web console as the `admin` user with the `redhat` password.
  - Create a `comprevue-import` integration for the `Cluster Init Bundle`.
  - Create the `comprevue-import` cluster secrets on both the `ocp4-mng` and `ocp4` clusters.
    - Create the secrets in the `stackrox` namespace.
  - Create an RHACM policy named `policy-advanced-managed-cluster-security` and verify that the policy status is `Compliant`:
    - This policy installs the RHACS `SecuredCluster` instance in the `stackrox` namespace in the `ocp4` and `ocp4-mng` clusters.
    - The `SecuredCluster` endpoint is `central-stackrox.apps.ocp4.example.com:443`.
    - The policy also ensures that the `stackrox` namespace exists in all the clusters.
    - The `ocp4` cluster is named `local-cluster`, and the `ocp4-mng` cluster is named `managed-cluster` in the RHACM.
    - Create the policy in the `rhacs-install` namespace.
    - The `rhacs-secured-cluster.yaml` policy template file is available at `~/D0480/labs/comprevue-install/`.
    - The RHACM web console URL is `https://multicloud-console.apps.ocp4.example.com`.
    - Use the `htpasswd_provider` option to log in to the RHACM web console as the `admin` user with the `redhat` password.
    - Verify that the `local-cluster` and `managed-cluster` are imported successfully in RHACS.
  - Create a Prometheus alerting rule so that cluster administrators receive alerts when the memory usage of a cluster exceeds 55%.
    - The template file `custom-rules.yaml` for the `ConfigMap` object is available at `~/D0480/labs/comprevue-install/`.

## Evaluation

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

```
[student@workstation ~]$ lab grade compreview-install
```

## Finish

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

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



### Important

Do not delete any resources created in this exercise. They are used in upcoming sections.

This concludes the section.

## ► Solution

# Configure RHACS, Red Hat Quay, and RHACM Observability Stack

- Install and configure a Red Hat Advanced Cluster Security for Kubernetes (RHACS) secured cluster to import all the clusters managed by Red Hat Advanced Cluster Management for Kubernetes (RHACM) into RHACS by using an RHACM policy. Then, you create a Prometheus custom alerting rule and push an image to the Red Hat Quay private registry.

## Outcomes

- Push an image to the private Red Hat Quay registry.
- Add a secret to a service account to pull images from the private Red Hat Quay registry.
- Generate a cluster `init bundle` by using the RHACS dashboard.
- Create cluster init secrets in all the managed clusters.
- Create the RHACS `SecuredCluster` custom resource in all the clusters that use an RHACM policy.
- Verify the import of all the clusters from the RHACS web console.
- Create a Prometheus custom alerting rule.

## Before You Begin

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

This command prepares your environment and ensures that all required resources are available.

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

- As the `cloudadmin` user with the `redhat` password, push the `quay.io/redhattraining/do480-hello` image to the `hello-app` private repository of the `cloudadmin` user.  
Push the image `quay.io/redhattraining/do480-hello` for the `v10`, `v11`, and `latest` tags. The central quay registry URL is `central-quay-registry.apps.ocp4.example.com`.
  - From the `workstation` machine, open a terminal and log in as the `cloudadmin` user with the `redhat` password.

**Warning**

This exercise uses a plain text password for brevity. Commands such as podman store passwords in the file system unencrypted. If a malicious actor obtains access to the file system, then the actor can impersonate your account.

In real-world situations, always use encrypted passwords as described elsewhere in this course.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
central-quay-registry.apps.ocp4.example.com
Login Succeeded!
```

- Push the quay.io/redhattraining/do480-hello:v10 image to the private repository of the cloudadmin user by using the skopeo command.

```
[student@workstation ~]$ skopeo copy \
docker://quay.io/redhattraining/do480-hello:v10 \
docker://central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app:v10
Getting image source signatures
...output omitted...
```

- Push the quay.io/redhattraining/do480-hello:v11 image to the private repository of the cloudadmin user by using the skopeo command.

```
[student@workstation ~]$ skopeo copy \
docker://quay.io/redhattraining/do480-hello:v11 \
docker://central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app:v11
Getting image source signatures
...output omitted...
```

- Push the quay.io/redhattraining/do480-hello:latest image to the private repository of the cloudadmin user by using the skopeo command.

```
[student@workstation ~]$ skopeo copy \
docker://quay.io/redhattraining/do480-hello:latest \
docker://central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-app:latest
Getting image source signatures
...output omitted...
```

- Create a secret named compsecret in the compreview-install namespace for the central Quay registry authentication on both ocp4 and ocp4-mng clusters. Add the secret to the default service account to pull the image from the Quay registry on both the ocp4 and ocp4-mng clusters.
  - Log in to the ocp4 cluster as the admin user with the redhat password. The API server address 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...
```

2.2. Create the compreview-install project.

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

2.3. Review the /run/user/1000/containers/auth.json file. The JSON file contains authentication details for the central-quay-registry.apps.ocp4.example.com registry.

```
[student@workstation ~]$ cat /run/user/1000/containers/auth.json
{
 "auths": {
 "central-quay-registry.apps.ocp4.example.com": {
 "auth": "Y2xvdWRhZG1pbjpyZWRoYXQ="
 }
 }
}
```



**Note**

If authentication details are unavailable in the /run/user/1000/containers/auth.json file, then run the podman login command.

```
[student@workstation ~]$ podman login -u=cloudadmin -p=redhat \
central-quay-registry.apps.ocp4.example.com
```

2.4. Create the compsecret secret for the central Quay registry authentication.

```
[student@workstation ~]$ oc create secret generic compsecret \
--from-file=dockerconfigjson=/run/user/1000/containers/auth.json \
--type=kubernetes.io/dockerconfigjson -n compreview-install
secret/compsecret created
```

2.5. Add the compsecret secret to the default service account named default.

```
[student@workstation ~]$ oc secrets link default compsecret \
--for=pull -n compreview-install
```

2.6. Log in to the ocp4-mng cluster as the admin user with the redhat password. The API server address 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...
```

- 2.7. Create the `comprevew-install` project.

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

- 2.8. Create the `compsecret` secret for the central Quay registry authentication.

```
[student@workstation ~]$ oc create secret generic compsecret \
--from-file=dockerconfigjson=/run/user/1000/containers/auth.json \
--type=kubernetes.io/dockerconfigjson -n compreview-install
secret/compsecret created
```

- 2.9. Add the `compsecret` secret to the default service account named `default`.

```
[student@workstation ~]$ oc secrets link default compsecret \
--for=pull -n compreview-install
```

3. Using the `htpasswd_provider` option, log in to the OpenShift web console as the `admin` user with the `redhat` password. Retrieve the default `admin` password for the RHACS central dashboard by using the OpenShift web console. As the `admin` user, log in to the RHACS central dashboard.

The OpenShift console URL is <https://console-openshift-console.apps.ocp4.example.com>.

- 3.1. From the `workstation` machine, open Firefox and access <https://console-openshift-console.apps.ocp4.example.com>.
- 3.2. Click `htpasswd_provider` and log in as the `admin` user with the `redhat` password.
- 3.3. Navigate to **Home > Projects** to display the **Projects** page, and then click **stackrox** to select the **stackrox** namespace.
- 3.4. Navigate to **Operators > Installed Operators** to display installed operators, and then click **Central** to display the **Centrals** page.

Project: stackrox

## Installed Operators

Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the [Understanding Operators documentation](#). Or create an Operator and ClusterServiceVersion using the [Operator SDK](#).

| Name                                                                   | Managed Namespaces | Status                                                       | Provided APIs              |
|------------------------------------------------------------------------|--------------------|--------------------------------------------------------------|----------------------------|
| Red Hat Quay<br>3.6.4 provided by Red Hat                              | All Namespaces     | <span style="color: green;">✓ Succeeded</span><br>Up to date | Quay Registry              |
| Advanced Cluster Security for Kubernetes<br>3.69.0 provided by Red Hat | All Namespaces     | <span style="color: green;">✓ Succeeded</span><br>Up to date | Central<br>Secured Cluster |

- 3.5. Click **stackrox-central-services** to display **stackrox-central-services** page. Copy the command from the **Admin Credentials Info** section to retrieve the default password for the **admin** user.

Project: stackrox

Installed Operators > [rhacs-operator.v3.69.0](#) > Central details

### stackrox-central-services

Actions ▾

- [Details](#)
- [YAML](#)
- [Resources](#)
- [Events](#)

#### Central overview

|             |                           |                                                                                                                                                                                                                                                                                  |
|-------------|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name        | stackrox-central-services | Central                                                                                                                                                                                                                                                                          |
| Namespace   | NS stackrox               | <b>Admin Credentials Info</b>                                                                                                                                                                                                                                                    |
| Labels      | No labels                 | A password for the 'admin' user has been automatically generated and stored in the "password" entry of the central-htpasswd secret. To view the password, run <code>oc -n stackrox get secret central-htpasswd -o go-template='{{index .data "password"   base64decode}}'</code> |
| Annotations |                           | Product Version                                                                                                                                                                                                                                                                  |
|             |                           | 3.69.0                                                                                                                                                                                                                                                                           |

- 3.6. Log in to the ocp4 cluster as the **admin** user with the **redhat** password. The API server address 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...
```

- 3.7. Retrieve the default password for the **admin** user from the command line.

```
[student@workstation ~]$ oc -n stackrox get secret central-htpasswd \
-o go-template='{{index .data "password" | base64decode}}'
WcRPchXqaP6KAoLbyCStr9v9d
```

- 3.8. Run the `oc get route` command to retrieve the URL of the RHACS web console.

```
[student@workstation ~]$ oc get route -n stackrox
NAME HOST/PORT PATH SERVICES PORT
TERMINATION WILDCARD
central central-stackrox.apps.ocp4.example.com central https
passthrough None
...output omitted...
```

- 3.9. Open Firefox and access <https://central-stackrox.apps.ocp4.example.com>.



### Note

If Firefox displays a warning message, then add exceptions to permit the CENTRAL\_SERVICE certificate.

- 3.10. From the list of authentication providers, select **Login with username/password**. Log in as the **admin** user with the password retrieved in the preceding step.

#### 4. Create an integration for the Cluster Init Bundle.

- 4.1. On the RHACS web console, navigate to **Platform Configuration > Integrations** to display the **Integration** page. Click **Cluster Init Bundle** in the **Authentication Tokens** section.

The screenshot shows the RHACS Integration page. On the left, a sidebar menu includes Compliance, Vulnerability Management, Configuration Management, Risk, Platform Configuration (with Clusters and Policies), and Integrations (which is selected and highlighted with a red box). The main content area is titled "Backup Integrations" and shows icons for Amazon S3 and Google Cloud Storage. Below this, under "Authentication Tokens", there are two entries: "StackRox" and "HELM". The "HELM" entry is also highlighted with a red box.

- 4.2. In the **Integrations** page, click **Generate bundle**.
- 4.3. Enter the `comprevew-import` value in the **Cluster init bundle name** field and click **Generate**.
- 4.4. On the **Configure Cluster Init Bundle Integration** page, click **Download Kubernetes secrets file** to download the secret file. In the confirmation window, select **Save File** and click **OK**.
5. Create the `comprevew-import` cluster secrets on both the `ocp4-mng` and `ocp4` clusters.
- 5.1. Return to the terminal and log in to the `ocp4-mng` cluster as the `admin` user with the `redhat` password. The API server address 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...
```

- 5.2. Create the secrets in the stackrox namespace by using the comprevew-import-cluster-init-secrets.yaml file.

```
[student@workstation ~]$ oc create -f \
 ~/Downloads/comprevew-import-cluster-init-secrets.yaml -n stackrox
secret/collector-tls created
secret/sensor-tls created
secret/admission-control-tls created
```

- 5.3. Log in to the ocp4 cluster as the admin user with the redhat password. The API server address 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...
```

- 5.4. Create the secrets in the stackrox namespace by using the comprevew-import-cluster-init-secrets.yaml file.

```
[student@workstation ~]$ oc create -f \
 ~/Downloads/comprevew-import-cluster-init-secrets.yaml -n stackrox
secret/collector-tls created
secret/sensor-tls created
secret/admission-control-tls created
```

6. Create an RHACM policy named policy-advanced-managed-cluster-security in the rhacs-install namespace. This policy installs the RHACS SecuredCluster instance in the stackrox namespace in the ocp4 and ocp4-mng clusters. The ocp4 cluster is named local-cluster and the ocp4-mng cluster is named managed-cluster in RHACM.

- 6.1. Change to the ~/D0480/labs/comprevew-install/ directory.

```
[student@workstation ~]$ cd ~/D0480/labs/comprevew-install/
[student@workstation comprevew-install]$
```

- 6.2. Open the rhacs-secured-cluster.yaml file and make the following edits to the YAML file:

```
[student@workstation comprevew-install]$ vim rhacs-secured-cluster.yaml
...output omitted...
spec:
 remediationAction: enforce
 disabled: false
 policy-templates:
 - objectDefinition:
```

```

apiVersion: policy.open-cluster-management.io/v1
kind: ConfigurationPolicy
metadata:
 name: managed-cluster-security-ns
...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: v1
 kind: Namespace
 metadata:
 name: stackrox
...output omitted...
 object-templates:
 - complianceType: musthave
 objectDefinition:
 apiVersion: platform.stackrox.io/v1alpha1
 kind: SecuredCluster
 metadata:
 namespace: stackrox
 name: stackrox-secured-cluster-services
 spec:
 clusterName: |
 {{ fromSecret "open-cluster-management-agent" "hub-kubeconfig-
secret" "cluster-name" | base64dec }}
 auditLogs:
 collection: Auto
 centralEndpoint: |
 central-stackrox.apps.ocp4.example.com:443
 admissionControl:
 listenOnCreates: false
 listenOnEvents: true
 listenOnUpdates: false
 perNode:
 collector:
 collection: KernelModule
 imageFlavor: Regular
 taintToleration: TolerateTaints
...output omitted...
 clusterSelector:
 matchExpressions: []

```

The policy also ensures that the `stackrox` namespace exists in all the clusters. It creates the RHACS `SecuredCluster` instance in the `stackrox` namespace.

- 6.3. Log in to the `ocp4` cluster as the `admin` user with the `redhat` password. The API server address is `https://api.ocp4.example.com:6443`.

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

- 6.4. Use the `oc` command to create the `policy-advanced-managed-cluster-security` policy in the `rhacs-install` namespace.

```
[student@workstation compreview-install]$ oc create -f \
 rhacs-secured-cluster.yaml -n rhacs-install
policy.policy.open-cluster-management.io/policy-advanced-managed-cluster-security
 created
placementbinding.policy.open-cluster-management.io/binding-policy-advanced-
 managed-cluster-security created
placementrule.apps.open-cluster-management.io/placement-policy-advanced-managed-
 cluster-security created
```

- 6.5. From the **workstation** machine, open Firefox and navigate to the RHACM web console. The URL is <https://multicloud-console.apps.ocp4.example.com>.
- 6.6. Click **htpasswd\_provider** and log in as the **admin** user with the **redhat** password.
- 6.7. In the left pane, click **Governance**. The dashboard shows the **policy-advanced-managed-cluster-security** policy. Click on the policy and then click **Clusters** to verify the status.
- 6.8. Verify that the **Clusters** tab shows two templates, **managed-cluster-security-ns**, and **managed-cluster-security-endpoints** for each cluster.

| Cluster Type    | Status    | Template                           | Description                                                                                                                                                         | Last Update       | Action                       |
|-----------------|-----------|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------------|
| managed-cluster | Compliant | managed-cluster-security-endpoints | Compliant; notification - securedclusters [stackrox-secured-cluster-services] in namespace stackrox found as specified, therefore this Object template is compliant | a minute ago      | <a href="#">View history</a> |
| managed-cluster | Compliant | managed-cluster-security-ns        | Compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant                                                     | a minute ago      | <a href="#">View history</a> |
| local-cluster   | Compliant | managed-cluster-security-endpoints | Compliant; notification - securedclusters [stackrox-secured-cluster-services] in namespace stackrox found as specified, therefore this Object template is compliant | a few seconds ago | <a href="#">View history</a> |
| local-cluster   | Compliant | managed-cluster-security-ns        | Compliant; notification - namespaces [stackrox] found as specified, therefore this Object template is compliant                                                     | a minute ago      | <a href="#">View history</a> |

**Note**

It takes 2-3 minutes for templates to show the Compliant status.

- 6.9. From the RHACS web console, navigate to **Platform Configuration > Clusters** to display the **CLUSTERS** page. The **CLUSTERS** page displays both the **local-cluster** and **managed-cluster** clusters.

The screenshot shows the RHACS interface with a sidebar containing 'Compliance', 'Vulnerability Management', 'Configuration Management', 'Risk', and 'Platform Configuration' sections. Under 'Platform Configuration', 'Clusters' is selected, showing two entries: 'local-cluster' and 'managed-cluster'. Each cluster entry includes a checkbox, 'Cloud Provider' (Not applicable), 'Cluster Status' (Healthy), 'Sensor Upgrade' (Up to date with Central), and 'Credential Expiration' (in 12 months). Below each status, there are three green checkmarks for 'Collector', 'Sensor', and 'AdmissionControl'.

**Note**

It takes 5-10 minutes for clusters to import to RHACS successfully with a **Healthy** status.

7. Create a Prometheus alerting rule so that cluster administrators receive alerts when the memory usage of a cluster exceeds 55%.
  - 7.1. From the terminal, open the file named `custom_rules.yaml` containing the `ConfigMap` object with the instructions in Prometheus Querying Language (PromQL) and make the following edits to the YAML file:

```
[student@workstation compreview-install]$ vim custom-rules.yaml
kind: ConfigMap
apiVersion: v1
metadata:
 name: thanos-ruler-custom-rules
data:
 custom_rules.yaml: |
 groups:
 - name: cluster-health
 rules:
 - alert: MemoryRequested-55
 annotations:
 summary: Notify when the total memory requested in the clusters is greater than 55%.
 description: "The cluster {{ $labels.cluster }} has more than 55% of the memory requested."
 expr: |
 cluster:memory_requested:ratio > 0.55
 for: 5s
 labels:
 cluster: "{{ $labels.cluster }}"
 severity: warning
```

- 7.2. Create the `thanos-ruler-custom-rules` ConfigMap object in the `open-cluster-management-observability` namespace.

```
[student@workstation compreview-install]$ 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.

- 7.3. Verify that the new configuration applies. Use the `watch` command to monitor the restart of the pods in the open-cluster-management-observability namespace.

```
[student@workstation compreview-install]$ 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...
```

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

After the `observability-thanos-rule-X` pods restart, the alerting rule is visible from the RHACM Grafana dashboard.

## Evaluation

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

```
[student@workstation ~]$ lab grade compreview-install
```

## Finish

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

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



### Important

Do not delete any resources created in this exercise. They are used in upcoming sections.

This concludes the section.

## ▶ Lab

# Deploy and Update an Application to Align with the RHACS Policy

- Deploy an application from the Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console and create a Red Hat Advanced Cluster Security for Kubernetes (RHACS) policy to prevent the deployment of vulnerable images. Then, you modify the deployment to align the application with the RHACS policy.

## Outcomes

- Label the clusters by using the RHACM web console.
- Create an application with the GitHub repository by using the RHACM web console.
- Create an RHACS policy to prevent the deployment of vulnerable images.
- Troubleshoot and update the application to comply with the RHACS policy.

## Before You Begin

To perform this exercise, ensure that you have completed *Lab: Configure RHACS, Red Hat Quay, and RHACM Observability Stack*.

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

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start compreview-deploy
```

## Specifications

- Label the clusters as stage and production environments.
  - Add `env=stage` and `env=prod` labels to the `local-cluster` and `managed-cluster` clusters, respectively, by using the RHACM web console.
  - The RHACM web console URL is <https://multicloud-console.apps.ocp4.example.com>.
  - Use the `htpasswd_provider` option to log in to the RHACM web console as the `admin` user with the `redhat` password.
- Fork the `D0480-apps` GitHub repository:
  - The GitHub repository is available at <https://github.com/redhattraining/D0480-apps/>.
- Create a `hello-app` application from the forked `D0480-apps` GitHub repository by using the RHACM web console.
  - Deploy the application on the stage environment.

- Use the `comprevew-stage` branch and the `comprevew` path.
- Deploy the application on the `comprevew-install` namespace.
- Create a `comprevew-policy` RHACS policy:
  - To block the creation of deployments that match the condition of this policy.
  - To prevent the deployment of images by using the `latest` tag from the central registry.
  - To prevent the deployment of images by using the `debian:10` OS from the central registry.
  - Select `Deploy` lifecycle stage for the policy.
  - Verify that `hello-app` application violates the `comprevew-policy` RHACS policy.
  - The RHACS web console URL is <https://central-stackrox.apps.ocp4.example.com>.
  - Use the **Login with username/password** option to log in to the RHACS web console as the `admin` user with the password retrieved in the previous exercise.
- Update the `hello-app` application to comply with the `comprevew-policy` RHACS policy:
  - The `hello-app:v10` image in the central registry uses `debian:10` OS.
  - The `hello-app:v11` and `hello-app:latest` images in the central registry use `debian:11` OS.
  - Verify that the updated application complies with the `comprevew-policy` RHACS policy.
- Deploy the `hello-app` application on the production environment:
  - Edit the `hello-app` application to create a second subscription for the production environment.
  - Use the `comprevew-prod` branch and the `comprevew` path.
  - Verify that the `hello-app` deployment is scaled down to zero in the production environment.
- Identify the issue with the `hello-app` application in the production environment by using the RHACS web console:
  - Verify that the `hello-app` application deployed on the `env=prod` labeled cluster violates the `comprevew-policy` RHACS policy.
  - Use the RHACS configuration management feature to identify the issue.
- Update the `hello-app` to comply with the `comprevew-policy` RHACS policy:
  - Verify that the updated application is deploying in the production environment successfully.

## 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 comprevew-deploy
```

## Finish

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

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

This concludes the section.

## ► Solution

# Deploy and Update an Application to Align with the RHACS Policy

- Deploy an application from the Red Hat Advanced Cluster Management for Kubernetes (RHACM) web console and create a Red Hat Advanced Cluster Security for Kubernetes (RHACS) policy to prevent the deployment of vulnerable images. Then, you modify the deployment to align the application with the RHACS policy.

## Outcomes

- Label the clusters by using the RHACM web console.
- Create an application with the GitHub repository by using the RHACM web console.
- Create an RHACS policy to prevent the deployment of vulnerable images.
- Troubleshoot and update the application to comply with the RHACS policy.

## Before You Begin

To perform this exercise, ensure that you have completed *Lab: Configure RHACS, Red Hat Quay, and RHACM Observability Stack*.

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

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start compreview-deploy
```

- Use the RHACM web console to add the `env=stage` and `env=prod` labels to the `local-cluster` and `managed-cluster` clusters, respectively. The web console URL is <https://multicloud-console.apps.ocp4.example.com>.
  - Navigate to the RHACM web console at <https://multicloud-console.apps.ocp4.example.com>. When prompted, click `httpasswd_provider` and log in as the `admin` user with the `redhat` password.
  - On the left navigation menu, click **Clusters** to add new labels to the `local-cluster` and `managed-cluster` clusters.
  - Click the vertical ellipsis (⋮) menu at the end of the `local-cluster` row and then click **Edit labels**.
  - Type `env=stage` for the new label and then click **Save**.
  - Click the vertical ellipsis menu at the end of the `managed-cluster` row and then click **Edit labels**.

- 1.6. Type env=prod for the new label and then click **Save**.
2. Fork the DO480-apps GitHub repository at <https://github.com/redhattraining/DO480-apps/>. This repository contains the YAML files required to build the application.

**Note**

If you have already performed this exercise or made other changes to the fork, then you must delete your fork to avoid problems during this exercise.

To delete the fork, navigate to <https://github.com/your-fork-name/DO480-apps/>. Click **Settings**. Click **Delete this repository** and enter the name of the repository to confirm the deletion.

- 2.1. From the **workstation** machine, navigate to the DO480-apps repository at <https://github.com/redhattraining/DO480-apps/>.
- 2.2. In the upper-right corner of the GitHub repository page, click **Fork** to create a fork for your repository.
- 2.3. Ensure that the **Copy the main branch only** option is disabled. You cannot perform this exercise if you copy only the **main** branch of the repository.
3. Use the RHACM web console to create a new **hello-app** application based on the following criteria.

| Field            | Value                                                                                                     |
|------------------|-----------------------------------------------------------------------------------------------------------|
| Name             | hello-app                                                                                                 |
| Namespace        | compreview-install                                                                                        |
| Repository types | Git                                                                                                       |
| URL              | <a href="https://github.com/your-fork-name/DO480-apps/">https://github.com/your-fork-name/DO480-apps/</a> |
| Branch           | compreview-stage                                                                                          |
| Path             | compreview                                                                                                |

- 3.1. On the left navigation bar, navigate to **Applications**, click **Create application** and then click **Subscription**.
- 3.2. On the **Applications** page, set **YAML: On** to view the YAML in the console as you create the application.
- 3.3. In the **Name** and **Namespace** fields, type **hello-app** and **compreview-install** respectively.
- 3.4. Click **Repository location for resources** and choose **Git** for the repository type to specify the location of your deployable resources.  
Enter the URL of your forked DO480-apps repository for the channel source in the **URL** field.

```

apiVersion: v1
kind: Namespace
metadata:
 name: compreview-install
...
apiVersion: app.openshift.io/v1beta1
kind: Application
metadata:
 name: hello-app
 namespace: compreview-install
spec:
 componentKinds:
 - group: apps.open-cluster-management.io
 kind: Subscription
 descriptor: {}
 selector:
 matchExpressions:
 - key: app
 operator: In
 values:
 - hello-app
...
apiVersion: v1
kind: Namespace
metadata:

```

- 3.5. Type **compreview-stage** and **compreview** for the **Branch** and **Path** fields, respectively.
- 3.6. Scroll down and click **Select clusters to deploy to**. Select **Deploy application resources only on clusters matching specified labels** to configure a placement rule for the application. Type **env** in the **Label** field and **stage** in the **Value** field, then click **Save**.
- 3.7. On the **Topology** page, click the **Route** pod, and then click the **Location URL** `hello-app.apps.ocp4.example.com` to view the application in the stage environment.
4. Create an RHACS policy to prevent the deployment of images by using the `latest` tag or `debian:10` OS from the central registry.

**Note**

If the password for the `admin` user is unavailable, then you can retrieve the password with the following commands:

```
[student@workstation ~]$ oc login -u admin -p redhat \
 https://api.ocp4.example.com:6443
Login successful.
...output omitted...
[student@workstation ~]$ oc -n stackrox get secret central-htpasswd \
 -o go-template='{{index .data "password" | base64decode}}'
WcRPchXqaP6KAoLbyCSTR9v9d
```

- 4.1. From the RHACS web console, navigate to **Platform Configuration > Policies**.
- 4.2. Click **Create policy**.
- 4.3. In the **Policy details** page, add the following parameters:

| Field name | Value                    |
|------------|--------------------------|
| Name       | comprevew-policy         |
| Severity   | High                     |
| Categories | Vulnerability Management |

And click **Next**.

- 4.4. On the **Policy behavior** page, add the following parameters:

| Field name                     | Value              |
|--------------------------------|--------------------|
| Lifecycle stages               | Deploy             |
| Response method                | Inform and enforce |
| Configure enforcement behavior | Enforce on Deploy  |

Click **Next**.

- 4.5. In the **Drag out policy field** pane, expand the **Image registry** list. Drag the **Image tag** tile to the **Policy Section 1** list and type **latest** in the **Image tag** text field.
- 4.6. Drag the **Image registry** tile to the **Policy Section 1** list and type **central-quay-registry.apps.ocp4.example.com** in the **Image pulled from registry** text field.
- 4.7. Click **Add condition**.
- 4.8. In the **Drag out policy field** pane, expand the **Image contents** list. Drag the **Image OS** tile to the **Policy Section 2** list and type **debian:10** in the **Image operating system** text field.
- 4.9. Drag the **Image registry** tile to the **Policy Section 2** list and type **central-quay-registry.apps.ocp4.example.com** in the **Image pulled from registry** text field.

The screenshot shows the 'Create policy' interface with the following details:

- Policies > Create policy**
- Create policy**
- Design custom security policies for your environment**
- Policy criteria**: Chain criteria with boolean logic.
- Policy Section 1** (Left):
  - Image tag:** latest
  - Image pulled from registry:** central-quay-registry.apps.ocp4.example.com
- OR**
- Policy Section 2** (Right):
  - Image operating system:** debian:10
  - Image pulled from registry:** central-quay-registry.apps.ocp4.example.com
- Add condition** button
- Drag out policy fields** pane on the right:
  - Image registry** (selected):
    - Image registry
    - Image remote
    - Image tag
  - Image contents** (selected):
    - Image age
    - Image scan age
    - Image user
    - Dockerfile line
- Next**, **Back**, **Cancel** buttons at the bottom

Click **Next**.

4.10. On the **Policy scope** page, click **Next**.

4.11. On the **Review policy** page, verify that the **hello-app** image shows in the **Preview violations** right pane.

The screenshot shows the 'Create policy' interface with the 'Review policy' tab selected. The left sidebar lists steps 1 through 5, with step 5 ('Review policy') highlighted. The main area shows policy details for 'compreviwer-policy': Severity is High, Categories are Vulnerability Management, Type is User generated, and there is no description or rationale. Under 'MITRE ATT&CK', it states 'Policy has no MITRE ATT&CK vectors'. To the right, the 'Preview violations' section shows two violations for the 'hello-app' container: 'Container "hello-app" has image with base OS "debian:10"' and 'Container "hello-app" has image with registry "central-quay-registry.apps.ocp4.example.com"'. Buttons at the bottom include 'Save', 'Back', and 'Cancel'.

4.12. Click **Save** to create the policy.

5. Update **hello-app** to comply with the **compreviwer-policy** RHACS policy.

5.1. From the **workstation** machine, open a terminal and navigate to the **~/D0480/labs/compreviwer-deploy** directory.

```
[student@workstation ~]$ cd D0480/labs/compreviwer-deploy
[student@workstation compreviwer-deploy]$
```

5.2. Use Git to clone your forked **D0480-apps** repository to access the common set of YAML resource files.

```
[student@workstation compreviwer-deploy]$ git clone \
 https://github.com/your-fork-name/D0480-apps
Cloning into 'D0480-apps'...
remote: Enumerating objects: 469, done.
remote: Counting objects: 100% (149/149), done.
remote: Compressing objects: 100% (81/81), done.
remote: Total 469 (delta 98), reused 95 (delta 68), pack-reused 320
Receiving objects: 100% (469/469), 59.00 KiB | 1.90 MiB/s, done.
Resolving deltas: 100% (212/212), done.
```

5.3. Navigate to the **D0480-apps** directory and checkout the **compreviwer-stage** branch, which contains the **hello-app** deployment YAML files.

```
[student@workstation compreview-deploy]$ cd DO480-apps
[student@workstation DO480-apps]$ git checkout compreview-stage
Branch 'compreview-stage' set up to track remote branch 'compreview-stage' from
'origin'.
Switched to a new branch 'compreview-stage'
```

- 5.4. Update the hello-app image tag to v11 in the deployment.yaml file.

```
[student@workstation compreview-deploy]$ vim compreview/deployment.yaml
...output omitted...
spec:
 containers:
 - name: hello-app
 image: central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-
app:v11
```

- 5.5. Add and commit the updated YAML file.

```
[student@workstation DO480-apps]$ git add .
[student@workstation DO480-apps]$ git commit -m "updating image tag"
...output omitted...
```

- 5.6. Push the changes to your Git repository.

```
[student@workstation DO480-apps]$ git push
...output omitted...
```

- 5.7. From the RHACM web console, navigate to Applications and then click hello-app.

- 5.8. On the Overview page, click Sync and then click Synchronize to synchronize application resources with the source repository.



### Note

It takes 2-3 minutes for hello-app application to update the resources from the source repository.

- 5.9. On the Topology page, click the Route pod, and then click the Location URL hello-app.apps.ocp4.example.com to view the application in the stage environment with the updated v11 image tag.



### Note

If you do not see the v11 image tag on the web page, then reload the page in your browser.

6. Update the hello-app application.

- 6.1. On the left navigation bar, navigate to Applications and then click hello-app.

- 6.2. Click Editor.

- 6.3. Scroll down and click **Add another repository**.
- 6.4. Click **Repository location for resources** and choose **Git** for the repository type to specify the location of your deployable resources.  
Enter the URL of your forked D0480-apps repository for the channel source in the **URL** field..
- 6.5. Type **comprevew-prod** and **comprevew** for the **Branch** and **Path** fields, respectively.
- 6.6. Scroll down and click **Select clusters to deploy to**. Select **Deploy application resources only on clusters matching specified labels** to configure a placement rule for the application. Type **env** in the **Label** field and **prod** in the **Value** field, then click **Update**.
- 6.7. On the **Topology** page, navigate to **Subscription > hello-app-subscription-2**. Click the **Deployment** pod. The **Cluster deploy status** for pods shows **0/0** for the **managed-cluster** cluster.
7. Locate the **hello-app** deployment in the **Configuration Management** page and verify the **comprevew-policy** RHACS policy violation.
- 7.1. In the left pane of the RHACS web console, click **Configuration Management**.
  - 7.2. In the top right corner, click **APPLICATION & INFRASTRUCTURE** and **Deployments**. The deployments list is displayed.
  - 7.3. On the **DEPLOYMENTS** page, type **Cluster:managed-cluster** and **Deployment:hello-app** in the search field to search the **hello-app** deployment in the **managed-cluster** cluster.
  - 7.4. Search results show the **hello-app** deployment with the **Fail Policy Status**.
  - 7.5. Click the **hello-app** deployment row. A detailed view opens.
  - 7.6. Scroll down to the **Deployment Findings** section. Locate the **comprevew-policy** policy.
  - 7.7. Click **comprevew-policy**. Scroll down to view the violation. The message shows **Container 'hello-app' has image with registry 'central-quay-registry.apps.ocp4.example.com' and tag 'latest'**.
8. Update the **hello-app** application to comply with the **comprevew-policy** RHACS policy.
- 8.1. Return to terminal and checkout the **comprevew-prod** branch, which contains the **hello-app** deployment **YAML** files.

```
[student@workstation D0480-apps]$ git checkout comprevew-prod
Branch 'comprevew-prod' set up to track remote branch 'comprevew-prod' from
'origin'.
Switched to a new branch 'comprevew-prod'
```

- 8.2. Update the **hello-app** image tag to **v11** in the **deployment.yaml** file.

```
[student@workstation compreview-deploy]$ vim compreview/deployment.yaml
...output omitted...
spec:
 containers:
 - name: hello-app
 image: central-quay-registry.apps.ocp4.example.com/cloudadmin/hello-
app:v11
```

8.3. Add and commit the updated YAML file.

```
[student@workstation D0480-apps]$ git add .
[student@workstation D0480-apps]$ git commit -m "updating image tag"
...output omitted...
```

8.4. Push the changes to your Git repository.

```
[student@workstation D0480-apps]$ git push
...output omitted...
```

- 8.5. From the RHACM web console, navigate to **Applications** and then click **hello-app**.
- 8.6. On the **Overview** page, click **Sync** and then click **Synchronize** to synchronize application resources with the source repository.
- 8.7. Navigate to **Subscription > hello-app-subscription-2**. Click the **Deployment** pod. The **Cluster deploy status for pods** shows 1/1 for the **managed-cluster** cluster.
- 8.8. Click the **Route** pod, and then click the **Location URL** `hello-app.apps.ocp4-mng.example.com` to view the application in the production environment.

## Evaluation

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

```
[student@workstation ~]$ lab grade compreview-deploy
```

## Finish

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

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

This concludes the section.

