OpenShift Container Platform 4.13

Security and compliance

Learning about and managing security for OpenShift Container Platform

Last Updated: 2026-01-30

OpenShift Container Platform 4.13 Security and compliance

Learning about and managing security for OpenShift Container Platform

Legal Notice

Abstract

This document discusses container security, configuring certificates, and enabling encryption to help secure the cluster.

Table of Contents

CHAPTER 1. OPENSHIFT CONTAINER PLATFORM SECURITY AND COMPLIANCE

1.1. SECURITY OVERVIEW

It is important to understand how to properly secure various aspects of your OpenShift Container Platform cluster.

Container security

A good starting point to understanding OpenShift Container Platform security is to review the concepts in Understanding container security. This and subsequent sections provide a high-level walkthrough of the container security measures available in OpenShift Container Platform, including solutions for the host layer, the container and orchestration layer, and the build and application layer. These sections also include information on the following topics:

- Why container security is important and how it compares with existing security standards.
- Which container security measures are provided by the host (RHCOS and RHEL) layer and which are provided by OpenShift Container Platform.
- How to evaluate your container content and sources for vulnerabilities.
- How to design your build and deployment process to proactively check container content.
- How to control access to containers through authentication and authorization.
- How networking and attached storage are secured in OpenShift Container Platform.
- Containerized solutions for API management and SSO.

Auditing

OpenShift Container Platform auditing provides a security-relevant chronological set of records documenting the sequence of activities that have affected the system by individual users, administrators, or other components of the system. Administrators can configure the audit log policy and view audit logs.

Certificates

Certificates are used by various components to validate access to the cluster. Administrators can replace the default ingress certificate, add API server certificates, or add a service certificate.

You can also review more details about the types of certificates used by the cluster:

- User-provided certificates for the API server
- Proxy certificates
- Service CA certificates
- Node certificates
- Bootstrap certificates
- etcd certificates
- OLM certificates

- Aggregated API client certificates
- Machine Config Operator certificates
- User-provided certificates for default ingress
- Ingress certificates
- Monitoring and cluster logging Operator component certificates
- Control plane certificates

Encrypting data

You can enable etcd encryption for your cluster to provide an additional layer of data security. For example, it can help protect the loss of sensitive data if an etcd backup is exposed to the incorrect parties.

Vulnerability scanning

Administrators can use the Red Hat Quay Container Security Operator to run vulnerability scans and review information about detected vulnerabilities.

1.2. COMPLIANCE OVERVIEW

For many OpenShift Container Platform customers, regulatory readiness, or compliance, on some level is required before any systems can be put into production. That regulatory readiness can be imposed by national standards, industry standards, or the organization's corporate governance framework.

Compliance checking

Administrators can use the Compliance Operator to run compliance scans and recommend remediations for any issues found. The **oc-compliance** plugin is an OpenShift CLI (**oc**) plugin that provides a set of utilities to easily interact with the Compliance Operator.

File integrity checking

Administrators can use the File Integrity Operator to continually run file integrity checks on cluster nodes and provide a log of files that have been modified.

1.3. ADDITIONAL RESOURCES

- Understanding authentication
- Configuring the internal OAuth server
- Understanding identity provider configuration
- Using RBAC to define and apply permissions
- Managing security context constraints

CHAPTER 2. CONTAINER SECURITY

2.1. UNDERSTANDING CONTAINER SECURITY

Securing a containerized application relies on multiple levels of security:

- Container security begins with a trusted base container image and continues through the container build process as it moves through your CI/CD pipeline.

IMPORTANT

Image streams by default do not automatically update. This default behavior might create a security issue because security updates to images referenced by an image stream do not automatically occur. For information about how to override this default behavior, see Configuring periodic importing of imagestreamtags.

- When a container is deployed, its security depends on it running on secure operating systems and networks, and establishing firm boundaries between the container itself and the users and hosts that interact with it.
- Continued security relies on being able to scan container images for vulnerabilities and having an efficient way to correct and replace vulnerable images.

Beyond what a platform such as OpenShift Container Platform offers out of the box, your organization will likely have its own security demands. Some level of compliance verification might be needed before you can even bring OpenShift Container Platform into your data center.

Likewise, you may need to add your own agents, specialized hardware drivers, or encryption features to OpenShift Container Platform, before it can meet your organization's security standards.

This guide provides a high-level walkthrough of the container security measures available in OpenShift Container Platform, including solutions for the host layer, the container and orchestration layer, and the build and application layer. It then points you to specific OpenShift Container Platform documentation to help you achieve those security measures.

This guide contains the following information:

- Why container security is important and how it compares with existing security standards.
- Which container security measures are provided by the host (RHCOS and RHEL) layer and which are provided by OpenShift Container Platform.
- How to evaluate your container content and sources for vulnerabilities.
- How to design your build and deployment process to proactively check container content.
- How to control access to containers through authentication and authorization.
- How networking and attached storage are secured in OpenShift Container Platform.
- Containerized solutions for API management and SSO.

The goal of this guide is to understand the incredible security benefits of using OpenShift Container Platform for your containerized workloads and how the entire Red Hat ecosystem plays a part in making

and keeping containers secure. It will also help you understand how you can engage with the OpenShift Container Platform to achieve your organization's security goals.

2.1.1. What are containers?

Containers package an application and all its dependencies into a single image that can be promoted from development, to test, to production, without change. A container might be part of a larger application that works closely with other containers.

Containers provide consistency across environments and multiple deployment targets: physical servers, virtual machines (VMs), and private or public cloud.

Some of the benefits of using containers include:

Infrastructure	Applications
Sandboxed application processes on a shared Linux operating system kernel	Package my application and all of its dependencies
Simpler, lighter, and denser than virtual machines	Deploy to any environment in seconds and enable CI/CD
Portable across different environments	Easily access and share containerized components

See Understanding Linux containers from the Red Hat Customer Portal to find out more about Linux containers. To learn about RHEL container tools, see Building, running, and managing containers in the RHEL product documentation.

2.1.2. What is OpenShift Container Platform?

Automating how containerized applications are deployed, run, and managed is the job of a platform such as OpenShift Container Platform. At its core, OpenShift Container Platform relies on the Kubernetes project to provide the engine for orchestrating containers across many nodes in scalable data centers.

Kubernetes is a project, which can run using different operating systems and add-on components that offer no guarantees of supportability from the project. As a result, the security of different Kubernetes platforms can vary.

OpenShift Container Platform is designed to lock down Kubernetes security and integrate the platform with a variety of extended components. To do this, OpenShift Container Platform draws on the extensive Red Hat ecosystem of open source technologies that include the operating systems, authentication, storage, networking, development tools, base container images, and many other components.

OpenShift Container Platform can leverage Red Hat's experience in uncovering and rapidly deploying fixes for vulnerabilities in the platform itself as well as the containerized applications running on the platform. Red Hat's experience also extends to efficiently integrating new components with OpenShift Container Platform as they become available and adapting technologies to individual customer needs.

Additional resources

- OpenShift Container Platform architecture

- OpenShift Security Guide

2.2. UNDERSTANDING HOST AND VM SECURITY

Both containers and virtual machines provide ways of separating applications running on a host from the operating system itself. Understanding RHCOS, which is the operating system used by OpenShift Container Platform, will help you see how the host systems protect containers and hosts from each other.

2.2.1. Securing containers on Red Hat Enterprise Linux CoreOS (RHCOS)

Containers simplify the act of deploying many applications to run on the same host, using the same kernel and container runtime to spin up each container. The applications can be owned by many users and, because they are kept separate, can run different, and even incompatible, versions of those applications at the same time without issue.

In Linux, containers are just a special type of process, so securing containers is similar in many ways to securing any other running process. An environment for running containers starts with an operating system that can secure the host kernel from containers and other processes running on the host, as well as secure containers from each other.

Because OpenShift Container Platform 4.13 runs on RHCOS hosts, with the option of using Red Hat Enterprise Linux (RHEL) as worker nodes, the following concepts apply by default to any deployed OpenShift Container Platform cluster. These RHEL security features are at the core of what makes running containers in OpenShift Container Platform more secure:

- *Linux namespaces* enable creating an abstraction of a particular global system resource to make it appear as a separate instance to processes within a namespace. Consequently, several containers can use the same computing resource simultaneously without creating a conflict. Container namespaces that are separate from the host by default include mount table, process table, network interface, user, control group, UTS, and IPC namespaces. Those containers that need direct access to host namespaces need to have elevated permissions to request that access. See Overview of Containers in Red Hat Systems from the RHEL 8 container documentation for details on the types of namespaces.

- *SELinux* provides an additional layer of security to keep containers isolated from each other and from the host. SELinux allows administrators to enforce mandatory access controls (MAC) for every user, application, process, and file.

WARNING

Disabling SELinux on RHCOS is not supported.

- *CGroups* (control groups) limit, account for, and isolate the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. CGroups are used to ensure that containers on the same host are not impacted by each other.

- *Secure computing mode (seccomp)* profiles can be associated with a container to restrict available system calls. See page 94 of the OpenShift Security Guide for details about seccomp.

- Deploying containers using *RHCOS* reduces the attack surface by minimizing the host environment and tuning it for containers. The CRI-O container engine further reduces that attack surface by implementing only those features required by Kubernetes and OpenShift Container Platform to run and manage containers, as opposed to other container engines that implement desktop-oriented standalone features.

RHCOS is a version of Red Hat Enterprise Linux (RHEL) that is specially configured to work as control plane (master) and worker nodes on OpenShift Container Platform clusters. So RHCOS is tuned to efficiently run container workloads, along with Kubernetes and OpenShift Container Platform services.

NOTE

To further protect RHCOS systems in OpenShift Container Platform clusters, most containers, except those managing or monitoring the host system itself, should run as a non-root user. Dropping the privilege level or creating containers with the least amount of privileges possible is recommended best practice for protecting your own OpenShift Container Platform clusters.

Additional resources

- How nodes enforce resource constraints
- Managing security context constraints
- Supported platforms for OpenShift clusters
- Requirements for a cluster with user-provisioned infrastructure
- Choosing how to configure RHCOS
- Ignition
- Kernel arguments
- Kernel modules
- Disk encryption
- Chrony time service
- About the OpenShift Update Service

2.2.2. Comparing virtualization and containers

Traditional virtualization provides another way to keep application environments separate on the same physical host. However, virtual machines work in a different way than containers. Virtualization relies on a hypervisor spinning up guest virtual machines (VMs), each of which has its own operating system (OS), represented by a running kernel, as well as the running application and its dependencies.

With VMs, the hypervisor isolates the guests from each other and from the host kernel. Fewer individuals and processes have access to the hypervisor, reducing the attack surface on the physical server. That said, security must still be monitored: one guest VM might be able to use hypervisor bugs to gain access to another VM or the host kernel. And, when the OS needs to be patched, it must be patched on all guest VMs using that OS.

Containers can be run inside guest VMs, and there might be use cases where this is desirable. For example, you might be deploying a traditional application in a container, perhaps to lift-and-shift an application to the cloud.

Container separation on a single host, however, provides a more lightweight, flexible, and easier-to-scale deployment solution. This deployment model is particularly appropriate for cloud-native applications. Containers are generally much smaller than VMs and consume less memory and CPU.

See Linux Containers Compared to KVM Virtualization in the RHEL 7 container documentation to learn about the differences between container and VMs.

2.2.3. Securing OpenShift Container Platform

When you deploy OpenShift Container Platform, you have the choice of an installer-provisioned infrastructure (there are several available platforms) or your own user-provisioned infrastructure. Some low-level security-related configuration, such as adding kernel modules required at first boot, might benefit from a user-provisioned infrastructure. Likewise, user-provisioned infrastructure is appropriate for disconnected OpenShift Container Platform deployments.

Keep in mind that, when it comes to making security enhancements and other configuration changes to OpenShift Container Platform, the goals should include:

- Keeping the underlying nodes as generic as possible. You want to be able to easily throw away and spin up similar nodes quickly and in prescriptive ways.
- Managing modifications to nodes through OpenShift Container Platform as much as possible, rather than making direct, one-off changes to the nodes.

In pursuit of those goals, most node changes should be done during installation through Ignition or later using MachineConfigs that are applied to sets of nodes by the Machine Config Operator. Examples of security-related configuration changes you can do in this way include:

- Adding kernel arguments
- Adding kernel modules
- Configuring disk encryption
- Configuring the chrony time service

Besides the Machine Config Operator, there are several other Operators available to configure OpenShift Container Platform infrastructure that are managed by the Cluster Version Operator (CVO). The CVO is able to automate many aspects of OpenShift Container Platform cluster updates.

2.3. HARDENING RHCOS

RHCOS was created and tuned to be deployed in OpenShift Container Platform with few if any changes needed to RHCOS nodes. Every organization adopting OpenShift Container Platform has its own requirements for system hardening. As a RHEL system with OpenShift-specific modifications and features added (such as Ignition, ostree, and a read-only /**usr** to provide limited immutability), RHCOS can be hardened just as you would any RHEL system. Differences lie in the ways you manage the hardening.

A key feature of OpenShift Container Platform and its Kubernetes engine is to be able to quickly scale applications and infrastructure up and down as needed. Unless it is unavoidable, you do not want to make direct changes to RHCOS by logging into a host and adding software or changing settings. You

want to have the OpenShift Container Platform installer and control plane manage changes to RHCOS so new nodes can be spun up without manual intervention.

So, if you are setting out to harden RHCOS nodes in OpenShift Container Platform to meet your security needs, you should consider both what to harden and how to go about doing that hardening.

2.3.1. Choosing what to harden in RHCOS

For information on how to approach security for any RHEL system, see the RHEL 9 Security Hardening guide.

Use this guide to learn how to approach cryptography, evaluate vulnerabilities, and assess threats to various services. Likewise, you can learn how to scan for compliance standards, check file integrity, perform auditing, and encrypt storage devices.

With the knowledge of what features you want to harden, you can then decide how to harden them in RHCOS.

2.3.2. Choosing how to harden RHCOS

Direct modification of RHCOS systems in OpenShift Container Platform is discouraged. Instead, you should think of modifying systems in pools of nodes, such as worker nodes and control plane nodes. When a new node is needed, in non-bare metal installs, you can request a new node of the type you want and it will be created from an RHCOS image plus the modifications you created earlier.

There are opportunities for modifying RHCOS before installation, during installation, and after the cluster is up and running.

2.3.2.1. Hardening before installation

For bare metal installations, you can add hardening features to RHCOS before beginning the OpenShift Container Platform installation. For example, you can add kernel options when you boot the RHCOS installer to turn security features on or off, such as various SELinux booleans or low-level settings, such as symmetric multithreading.

WARNING

Disabling SELinux on RHCOS nodes is not supported.

Although bare metal RHCOS installations are more difficult, they offer the opportunity of getting operating system changes in place before starting the OpenShift Container Platform installation. This can be important when you need to ensure that certain features, such as disk encryption or special networking settings, be set up at the earliest possible moment.

2.3.2.2. Hardening during installation

You can interrupt the OpenShift Container Platform installation process and change Ignition configs. Through Ignition configs, you can add your own files and systemd services to the RHCOS nodes. You can also make some basic security-related changes to the **install-config.yaml** file used for installation. Contents added in this way are available at each node's first boot.

2.3.2.3. Hardening after the cluster is running

After the OpenShift Container Platform cluster is up and running, there are several ways to apply hardening features to RHCOS:

- Daemon set: If you need a service to run on every node, you can add that service with a Kubernetes **DaemonSet** object.
- Machine config: **MachineConfig** objects contain a subset of Ignition configs in the same format. By applying machine configs to all worker or control plane nodes, you can ensure that the next node of the same type that is added to the cluster has the same changes applied.

All of the features noted here are described in the OpenShift Container Platform product documentation.

Additional resources

- OpenShift Security Guide
- Choosing how to configure RHCOS
- Modifying Nodes
- Manually creating the installation configuration file
- Creating the Kubernetes manifest and Ignition config files
- Installing RHCOS by using an ISO image
- Customizing nodes
- Adding kernel arguments to Nodes
- Installation configuration parameters
- RHEL core crypto components

2.4. CONTAINER IMAGE SIGNATURES

Red Hat delivers signatures for the images in the Red Hat Container Registries. Those signatures can be automatically verified when being pulled to OpenShift Container Platform 4 clusters by using the Machine Config Operator (MCO).

Quay.io serves most of the images that make up OpenShift Container Platform, and only the release image is signed. Release images refer to the approved OpenShift Container Platform images, offering a degree of protection against supply chain attacks. However, some extensions to OpenShift Container Platform, such as logging, monitoring, and service mesh, are shipped as Operators from the Operator Lifecycle Manager (OLM). Those images ship from the Red Hat Ecosystem Catalog Container images registry.

To verify the integrity of those images between Red Hat registries and your infrastructure, enable signature verification.

2.4.1. Enabling signature verification for Red Hat Container Registries

Enabling container signature validation for Red Hat Container Registries requires writing a signature verification policy file specifying the keys to verify images from these registries. For RHEL8 nodes, the registries are already defined in **/etc/containers/registries.d** by default.

Procedure

1. Create a Butane config file, **51-worker-rh-registry-trust.bu**, containing the necessary configuration for the worker nodes.

NOTE

The Butane version you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.13.0**. See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.13.0
metadata:
  name: 51-worker-rh-registry-trust
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/containers/policy.json
    mode: 0644
    overwrite: true
    contents:
      inline: |
        {
          "default": [
            {
              "type": "insecureAcceptAnything"
            }
          ],
          "transports": {
            "docker": {
              "registry.access.redhat.com": [
                {
                  "type": "signedBy",
                  "keyType": "GPGKeys",
                  "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
              ],
              "registry.redhat.io": [
                {
                  "type": "signedBy",
                  "keyType": "GPGKeys",
                  "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
              ]
            },
            "docker-daemon": {
              "": [
                {
                  "type": "insecureAcceptAnything"
```

```
          }
        ]
      }
    }
  }
```

2. Use Butane to generate a machine config YAML file, **51-worker-rh-registry-trust.yaml**, containing the file to be written to disk on the worker nodes:

```
$ butane 51-worker-rh-registry-trust.bu -o 51-worker-rh-registry-trust.yaml
```

3. Apply the created machine config:

```
$ oc apply -f 51-worker-rh-registry-trust.yaml
```

4. Check that the worker machine config pool has rolled out with the new machine config:

   a. Check that the new machine config was created:

```
$ oc get mc
```

Sample output

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
00-worker                                          a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
01-master-container-runtime
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
01-master-kubelet                                  a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
01-worker-container-runtime
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
01-worker-kubelet                                  a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
51-master-rh-registry-trust                                      3.2.0             13s
51-worker-rh-registry-trust                                      3.2.0             53s (1)
99-master-generated-crio-seccomp-use-default                                       3.2.0
25m
99-master-generated-registries
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
99-master-ssh                                                    3.2.0             28m
99-worker-generated-crio-seccomp-use-default                                       3.2.0
25m
99-worker-generated-registries
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
99-worker-ssh                                                    3.2.0             28m
rendered-master-af1e7ff78da0a9c851bab4be2777773b
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             8s
rendered-master-cd51fd0c47e91812bfef2765c52ec7e6
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             24m
rendered-worker-2b52f75684fbc711bd1652dd86fd0b82
```

```
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0          24m
rendered-worker-be3b3bce4f4aa52a62902304bac9da3c
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0          48s (2)
```

(1) New machine config

(2) New rendered machine config

b. Check that the worker machine config pool is updating with the new machine config:

```
$ oc get mcp
```

Sample output

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-af1e7ff78da0a9c851bab4be2777773b   True      False
False      3              3                   3                   0                   30m
worker   rendered-worker-be3b3bce4f4aa52a62902304bac9da3c   False     True
False      3              0                   0                   0                   30m (1)
```

(1) When the **UPDATING** field is **True**, the machine config pool is updating with the new machine config. When the field becomes **False**, the worker machine config pool has rolled out to the new machine config.

5. If your cluster uses any RHEL7 worker nodes, when the worker machine config pool is updated, create YAML files on those nodes in the **/etc/containers/registries.d** directory, which specify the location of the detached signatures for a given registry server. The following example works only for images hosted in **registry.access.redhat.com** and **registry.redhat.io**.

a. Start a debug session to each RHEL7 worker node:

```
$ oc debug node/<node_name>
```

b. Change your root directory to **/host**:

```
sh-4.2# chroot /host
```

c. Create a **/etc/containers/registries.d/registry.redhat.io.yaml** file that contains the following:

```
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
```

d. Create a **/etc/containers/registries.d/registry.access.redhat.com.yaml** file that contains the following:

```
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

e. Exit the debug session.

2.4.2. Verifying the signature verification configuration

After you apply the machine configs to the cluster, the Machine Config Controller detects the new **MachineConfig** object and generates a new **rendered-worker-<hash>** version.

Prerequisites

- You enabled signature verification by using a machine config file.

Procedure

1. On the command line, run the following command to display information about a desired worker:

```
$ oc describe machineconfigpool/worker
```

Example output of initial worker monitoring

```
Name:         worker
Namespace:
Labels:       machineconfiguration.openshift.io/mco-built-in=
Annotations:  <none>
API Version:  machineconfiguration.openshift.io/v1
Kind:         MachineConfigPool
Metadata:
  Creation Timestamp:  2019-12-19T02:02:12Z
  Generation:          3
  Resource Version:    16229
  Self Link:           /apis/machineconfiguration.openshift.io/v1/machineconfigpools/worker
  UID:                 92697796-2203-11ea-b48c-fa163e3940e5
Spec:
  Configuration:
    Name:  rendered-worker-f6819366eb455a401c42f8d96ab25c02
    Source:
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         00-worker
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         01-worker-container-runtime
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         01-worker-kubelet
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         51-worker-rh-registry-trust
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
```

```
    API Version:  machineconfiguration.openshift.io/v1
    Kind:         MachineConfig
    Name:         99-worker-ssh
  Machine Config Selector:
    Match Labels:
      machineconfiguration.openshift.io/role:  worker
  Node Selector:
    Match Labels:
      node-role.kubernetes.io/worker:
  Paused:                          false
Status:
  Conditions:
    Last Transition Time:  2019-12-19T02:03:27Z
    Message:
    Reason:
    Status:                False
    Type:                  RenderDegraded
    Last Transition Time:  2019-12-19T02:03:43Z
    Message:
    Reason:
    Status:                False
    Type:                  NodeDegraded
    Last Transition Time:  2019-12-19T02:03:43Z
    Message:
    Reason:
    Status:                False
    Type:                  Degraded
    Last Transition Time:  2019-12-19T02:28:23Z
    Message:
    Reason:
    Status:                False
    Type:                  Updated
    Last Transition Time:  2019-12-19T02:28:23Z
    Message:               All nodes are updating to rendered-worker-
f6819366eb455a401c42f8d96ab25c02
    Reason:
    Status:                True
    Type:                  Updating
  Configuration:
    Name:  rendered-worker-d9b3f4ffcfd65c30dcf591a0e8cf9b2e
    Source:
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  00-worker
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  01-worker-container-runtime
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  01-worker-kubelet
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  99-worker-ssh
```

```
  Degraded Machine Count:     0
  Machine Count:              1
  Observed Generation:        3
  Ready Machine Count:        0
  Unavailable Machine Count:  1
  Updated Machine Count:      0
Events:                       <none>
```

2. Run the **oc describe** command again:

```
$ oc describe machineconfigpool/worker
```

Example output after the worker is updated

```
...
    Last Transition Time:  2019-12-19T04:53:09Z
    Message:               All nodes are updated with rendered-worker-
f6819366eb455a401c42f8d96ab25c02
    Reason:
    Status:                True
    Type:                  Updated
    Last Transition Time:  2019-12-19T04:53:09Z
    Message:
    Reason:
    Status:                False
    Type:                  Updating
  Configuration:
    Name:  rendered-worker-f6819366eb455a401c42f8d96ab25c02
    Source:
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  00-worker
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  01-worker-container-runtime
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  01-worker-kubelet
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  51-worker-rh-registry-trust
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:           machineconfiguration.openshift.io/v1
      Kind:                  MachineConfig
      Name:                  99-worker-ssh
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        4
  Ready Machine Count:        3
  Unavailable Machine Count:  0
  Updated Machine Count:      3
...
```

NOTE

The **Observed Generation** parameter shows an increased count based on the generation of the controller-produced configuration. This controller updates this value even if it fails to process the specification and generate a revision. The **Configuration Source** value points to the **51-worker-rh-registry-trust** configuration.

3. Confirm that the **policy.json** file exists with the following command:

```
$ oc debug node/<node> -- chroot /host cat /etc/containers/policy.json
```

Example output

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
{
  "default": [
    {
      "type": "insecureAcceptAnything"
    }
  ],
  "transports": {
    "docker": {
      "registry.access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ],
      "registry.redhat.io": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ]
    },
    "docker-daemon": {
      "": [
        {
          "type": "insecureAcceptAnything"
        }
      ]
    }
  }
}
```

4. Confirm that the **registry.redhat.io.yaml** file exists with the following command:

```
$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.redhat.io.yaml
```

Example output

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
```

5. Confirm that the **registry.access.redhat.com.yaml** file exists with the following command:

```
$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.access.redhat.com.yaml
```

Example output

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

2.4.3. Understanding the verification of container images lacking verifiable signatures

Each OpenShift Container Platform release image is immutable and signed with a Red Hat production key. During an OpenShift Container Platform update or installation, a release image might deploy container images that do not have verifiable signatures. Each signed release image digest is immutable. Each reference in the release image is to the immutable digest of another image, so the contents can be trusted transitively. In other words, the signature on the release image validates all release contents.

For example, the image references lacking a verifiable signature are contained in the signed OpenShift Container Platform release image:

Example release info output

```
$ oc adm release info quay.io/openshift-release-dev/ocp-
release@sha256:2309578b68c5666dad62aed696f1f9d778ae1a089ee461060ba7b9514b7ca417 -o
pullspec 1
quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:9aafb914d5d7d0dec4edd800d02f811d7383a7d49e500af548eab5d00c1bffdb 2
```

1. Signed release image SHA.
2. Container image lacking a verifiable signature included in the release.

2.4.3.1. Automated verification during updates

Verification of signatures is automatic. The OpenShift Cluster Version Operator (CVO) verifies signatures on the release images during an OpenShift Container Platform update. This is an internal process. An OpenShift Container Platform installation or update fails if the automated verification fails.

Verification of signatures can also be done manually using the **skopeo** command-line utility.

Additional resources

- Introduction to OpenShift Updates

2.4.3.2. Using skopeo to verify signatures of Red Hat container images

You can verify the signatures for container images included in an OpenShift Container Platform release image by pulling those signatures from OCP release mirror site. Because the signatures on the mirror site are not in a format readily understood by Podman or CRI-O, you can use the **skopeo standalone-verify** command to verify that the your release images are signed by Red Hat.

Prerequisites

- You have installed the **skopeo** command-line utility.

Procedure

1. Get the full SHA for your release by running the following command:

```
$ oc adm release info <release_version>  \ 1
```

1 Substitute <release_version> with your release number, for example, **4.14.3**.

Example output snippet

```
---
Pull From: quay.io/openshift-release-dev/ocp-release@sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55
---
```

2. Pull down the Red Hat release key by running the following command:

```
$ curl -o pub.key https://access.redhat.com/security/data/fd431d51.txt
```

3. Get the signature file for the specific release that you want to verify by running the following command:

```
$ curl -o signature-1 https://mirror.openshift.com/pub/openshift-v4/signatures/openshift-release-dev/ocp-release/sha256=<sha_from_version>/signature-1 \ 1
```

1 Replace **<sha_from_version>** with SHA value from the full link to the mirror site that matches the SHA of your release. For example, the link to the signature for the 4.12.23 release is **https://mirror.openshift.com/pub/openshift-v4/signatures/openshift-release-dev/ocp-release/sha256=e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55/signature-1**, and the SHA value is **e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55**.

4. Get the manifest for the release image by running the following command:

```
$ skopeo inspect --raw docker://<quay_link_to_release> > manifest.json \ 1
```

1 Replace **<quay_link_to_release>** with the output of the **oc adm release info** command. For example, **quay.io/openshift-release-dev/ocp-release@sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55**.

5. Use skopeo to verify the signature:

```
$ skopeo standalone-verify manifest.json quay.io/openshift-release-dev/ocp-release:<release_number>-<arch> any signature-1 --public-key-file pub.key
```

where:

<release_number>

Specifies the release number, for example **4.14.3**.

<arch>

Specifies the architecture, for example **x86_64**.

Example output

```
Signature verified using fingerprint 567E347AD0044ADE55BA8A5F199E2F91FD431D51, digest sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55
```

2.4.4. Additional resources

- Machine Config Overview

2.5. UNDERSTANDING COMPLIANCE

For many OpenShift Container Platform customers, regulatory readiness, or compliance, on some level is required before any systems can be put into production. That regulatory readiness can be imposed by national standards, industry standards or the organization's corporate governance framework.

2.5.1. Understanding compliance and risk management

To understand Red Hat's view of OpenShift Container Platform compliance frameworks, refer to the Risk Management and Regulatory Readiness chapter of the OpenShift Security Guide Book .

2.6. SECURING CONTAINER CONTENT

To ensure the security of the content inside your containers you need to start with trusted base images, such as Red Hat Universal Base Images, and add trusted software. To check the ongoing security of your container images, there are both Red Hat and third-party tools for scanning images.

2.6.1. Securing inside the container

Applications and infrastructures are composed of readily available components, many of which are open source packages such as, the Linux operating system, JBoss Web Server, PostgreSQL, and Node.js.

Containerized versions of these packages are also available. However, you need to know where the packages originally came from, what versions are used, who built them, and whether there is any malicious code inside them.

Some questions to answer include:

- Will what is inside the containers compromise your infrastructure?
- Are there known vulnerabilities in the application layer?
- Are the runtime and operating system layers current?

By building your containers from Red Hat Universal Base Images (UBI) you are assured of a foundation for your container images that consists of the same RPM-packaged software that is included in Red Hat Enterprise Linux. No subscriptions are required to either use or redistribute UBI images.

To assure ongoing security of the containers themselves, security scanning features, used directly from RHEL or added to OpenShift Container Platform, can alert you when an image you are using has vulnerabilities. OpenSCAP image scanning is available in RHEL and the Red Hat Quay Container Security Operator can be added to check container images used in OpenShift Container Platform.

2.6.2. Creating redistributable images with UBI

To create containerized applications, you typically start with a trusted base image that offers the components that are usually provided by the operating system. These include the libraries, utilities, and other features the application expects to see in the operating system's file system.

Red Hat Universal Base Images (UBI) were created to encourage anyone building their own containers to start with one that is made entirely from Red Hat Enterprise Linux rpm packages and other content. These UBI images are updated regularly to keep up with security patches and free to use and redistribute with container images built to include your own software.

Search the Red Hat Ecosystem Catalog to both find and check the health of different UBI images. As someone creating secure container images, you might be interested in these two general types of UBI images:

- **UBI**: There are standard UBI images for RHEL 7, 8, and 9 (**ubi7/ubi**, **ubi8/ubi**, and **ubi9/ubi**), as well as minimal images based on those systems (**ubi7/ubi-minimal**, **ubi8/ubi-mimimal**, and ubi9/ubi-minimal). All of these images are preconfigured to point to free repositories of RHEL software that you can add to the container images you build, using standard **yum** and **dnf** commands.

NOTE

Red Hat encourages people to use these images on other distributions, such as Fedora and Ubuntu.

- **Red Hat Software Collections**: Search the Red Hat Ecosystem Catalog for **rhscl/** to find images created to use as base images for specific types of applications. For example, there are Apache httpd (**rhscl/httpd-***), Python (**rhscl/python-***), Ruby (**rhscl/ruby-***), Node.js (**rhscl/nodejs-***) and Perl (**rhscl/perl-***) rhscl images.

Keep in mind that while UBI images are freely available and redistributable, Red Hat support for these images is only available through Red Hat product subscriptions.

See Using Red Hat Universal Base Images in the Red Hat Enterprise Linux documentation for information on how to use and build on standard, minimal and init UBI images.

2.6.3. Security scanning in RHEL

For Red Hat Enterprise Linux (RHEL) systems, OpenSCAP scanning is available from the **openscap-utils** package. In RHEL, you can use the **openscap-podman** command to scan images for vulnerabilities. See Scanning containers and container images for vulnerabilities in the Red Hat Enterprise Linux documentation.

OpenShift Container Platform enables you to leverage RHEL scanners with your CI/CD process. For example, you can integrate static code analysis tools that test for security flaws in your source code and software composition analysis tools that identify open source libraries to provide metadata on those libraries such as known vulnerabilities.

2.6.3.1. Scanning OpenShift images

For the container images that are running in OpenShift Container Platform and are pulled from Red Hat Quay registries, you can use an Operator to list the vulnerabilities of those images. The Red Hat Quay Container Security Operator can be added to OpenShift Container Platform to provide vulnerability reporting for images added to selected namespaces.

Container image scanning for Red Hat Quay is performed by the Clair. In Red Hat Quay, Clair can search for and report vulnerabilities in images built from RHEL, CentOS, Oracle, Alpine, Debian, and Ubuntu operating system software.

2.6.4. Integrating external scanning

OpenShift Container Platform makes use of object annotations to extend functionality. External tools, such as vulnerability scanners, can annotate image objects with metadata to summarize results and control pod execution. This section describes the recognized format of this annotation so it can be reliably used in consoles to display useful data to users.

2.6.4.1. Image metadata

There are different types of image quality data, including package vulnerabilities and open source software (OSS) license compliance. Additionally, there may be more than one provider of this metadata. To that end, the following annotation format has been reserved:

```
quality.images.openshift.io/<qualityType>.<providerId>: {}
```

Table 2.1. Annotation key format

Component	Description	Acceptable values
qualityType	Metadata type	**vulnerability** **license** **operations** **policy**

Component	Description	Acceptable values
providerId	Provider ID string	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

2.6.4.1.1. Example annotation keys

```
quality.images.openshift.io/vulnerability.blackduck: {}
quality.images.openshift.io/vulnerability.jfrog: {}
quality.images.openshift.io/license.blackduck: {}
quality.images.openshift.io/vulnerability.openscap: {}
```

The value of the image quality annotation is structured data that must adhere to the following format:

Table 2.2. Annotation value format

Field	Required?	Description	Type
name	Yes	Provider display name	String
timestamp	Yes	Scan timestamp	String
description	No	Short description	String
reference	Yes	URL of information source or more details. Required so user may validate the data.	String
scannerVersion	No	Scanner version	String
compliant	No	Compliance pass or fail	Boolean
summary	No	Summary of issues found	List (see table below)

The **summary** field must adhere to the following format:

Table 2.3. Summary field value format

Field	Description	Type

Field	Description	Type
label	Display label for component (for example, "critical," "important," "moderate," "low," or "health")	String
data	Data for this component (for example, count of vulnerabilities found or score)	String
severityIndex	Component index allowing for ordering and assigning graphical representation. The value is range **0..3** where **0** = low.	Integer
reference	URL of information source or more details. Optional.	String

2.6.4.1.2. Example annotation values

This example shows an OpenSCAP annotation for an image with vulnerability summary data and a compliance boolean:

OpenSCAP annotation

```
{
  "name": "OpenSCAP",
  "description": "OpenSCAP vulnerability score",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://www.open-scap.org/930492",
  "compliant": true,
  "scannerVersion": "1.2",
  "summary": [
    { "label": "critical", "data": "4", "severityIndex": 3, "reference": null },
    { "label": "important", "data": "12", "severityIndex": 2, "reference": null },
    { "label": "moderate", "data": "8", "severityIndex": 1, "reference": null },
    { "label": "low", "data": "26", "severityIndex": 0, "reference": null }
  ]
}
```

This example shows the Container images section of the Red Hat Ecosystem Catalog annotation for an image with health index data with an external URL for additional details:

Red Hat Ecosystem Catalog annotation

```
{
  "name": "Red Hat Ecosystem Catalog",
  "description": "Container health index",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://access.redhat.com/errata/RHBA-2016:1566",
  "compliant": null,
  "scannerVersion": "1.2",
```

```
  "summary": [
    { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null }
  ]
}
```

2.6.4.2. Annotating image objects

While image stream objects are what an end user of OpenShift Container Platform operates against, image objects are annotated with security metadata. Image objects are cluster-scoped, pointing to a single image that may be referenced by many image streams and tags.

2.6.4.2.1. Example annotate CLI command

Replace **<image>** with an image digest, for example **sha256:401e359e0f45bfdcf004e258b72e253fd07fba8cc5c6f2ed4f4608fb119ecc2**:

```
$ oc annotate image <image> \
    quality.images.openshift.io/vulnerability.redhatcatalog='{ \
    "name": "Red Hat Ecosystem Catalog", \
    "description": "Container health index", \
    "timestamp": "2020-06-01T05:04:46Z", \
    "compliant": null, \
    "scannerVersion": "1.2", \
    "reference": "https://access.redhat.com/errata/RHBA-2020:2347", \
    "summary": "[ \
      { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null } ]" }'
```

2.6.4.3. Controlling pod execution

Use the **images.openshift.io/deny-execution** image policy to programmatically control if an image can be run.

2.6.4.3.1. Example annotation

```
annotations:
  images.openshift.io/deny-execution: true
```

2.6.4.4. Integration reference

In most cases, external tools such as vulnerability scanners develop a script or plugin that watches for image updates, performs scanning, and annotates the associated image object with the results. Typically this automation calls the OpenShift Container Platform 4.13 REST APIs to write the annotation. See OpenShift Container Platform REST APIs for general information on the REST APIs.

2.6.4.4.1. Example REST API call

The following example call using **curl** overrides the value of the annotation. Be sure to replace the values for **<token>**, **<openshift_server>**, **<image_id>**, and **<image_annotation>**.

Patch API call

```
$ curl -X PATCH \
  -H "Authorization: Bearer <token>" \
```

```
-H "Content-Type: application/merge-patch+json" \
https://<openshift_server>:6443/apis/image.openshift.io/v1/images/<image_id> \
--data '{ <image_annotation> }'
```

The following is an example of **PATCH** payload data:

Patch call data

```
{
"metadata": {
  "annotations": {
    "quality.images.openshift.io/vulnerability.redhatcatalog":
       "{ 'name': 'Red Hat Ecosystem Catalog', 'description': 'Container health index', 'timestamp': '2020-
06-01T05:04:46Z', 'compliant': null, 'reference': 'https://access.redhat.com/errata/RHBA-2020:2347',
'summary': [{'label': 'Health index', 'data': '4', 'severityIndex': 1, 'reference': null}] }"
    }
  }
}
```

Additional resources

- Image stream objects

2.7. USING CONTAINER REGISTRIES SECURELY

Container registries store container images to:

- Make images accessible to others
- Organize images into repositories that can include multiple versions of an image
- Optionally limit access to images, based on different authentication methods, or make them publicly available

There are public container registries, such as Quay.io and Docker Hub where many people and organizations share their images. The Red Hat Registry offers supported Red Hat and partner images, while the Red Hat Ecosystem Catalog offers detailed descriptions and health checks for those images. To manage your own registry, you could purchase a container registry such as Red Hat Quay.

From a security standpoint, some registries provide special features to check and improve the health of your containers. For example, Red Hat Quay offers container vulnerability scanning with Clair security scanner, build triggers to automatically rebuild images when source code changes in GitHub and other locations, and the ability to use role-based access control (RBAC) to secure access to images.

2.7.1. Knowing where containers come from?

There are tools you can use to scan and track the contents of your downloaded and deployed container images. However, there are many public sources of container images. When using public container registries, you can add a layer of protection by using trusted sources.

2.7.2. Immutable and certified containers

Consuming security updates is particularly important when managing *immutable containers*. Immutable containers are containers that will never be changed while running. When you deploy immutable

containers, you do not step into the running container to replace one or more binaries. From an operational standpoint, you rebuild and redeploy an updated container image to replace a container instead of changing it.

Red Hat certified images are:

- Free of known vulnerabilities in the platform components or layers
- Compatible across the RHEL platforms, from bare metal to cloud
- Supported by Red Hat

The list of known vulnerabilities is constantly evolving, so you must track the contents of your deployed container images, as well as newly downloaded images, over time. You can use Red Hat Security Advisories (RHSAs) to alert you to any newly discovered issues in Red Hat certified container images, and direct you to the updated image. Alternatively, you can go to the Red Hat Ecosystem Catalog to look up that and other security-related issues for each Red Hat image.

2.7.3. Getting containers from Red Hat Registry and Ecosystem Catalog

Red Hat lists certified container images for Red Hat products and partner offerings from the Container Images section of the Red Hat Ecosystem Catalog. From that catalog, you can see details of each image, including CVE, software packages listings, and health scores.

Red Hat images are actually stored in what is referred to as the *Red Hat Registry*, which is represented by a public container registry (**registry.access.redhat.com**) and an authenticated registry (**registry.redhat.io**). Both include basically the same set of container images, with **registry.redhat.io** including some additional images that require authentication with Red Hat subscription credentials.

Container content is monitored for vulnerabilities by Red Hat and updated regularly. When Red Hat releases security updates, such as fixes to *glibc*, DROWN, or Dirty Cow, any affected container images are also rebuilt and pushed to the Red Hat Registry.

Red Hat uses a **health index** to reflect the security risk for each container provided through the Red Hat Ecosystem Catalog. Because containers consume software provided by Red Hat and the errata process, old, stale containers are insecure whereas new, fresh containers are more secure.

To illustrate the age of containers, the Red Hat Ecosystem Catalog uses a grading system. A freshness grade is a measure of the oldest and most severe security errata available for an image. "A" is more up to date than "F". See Container Health Index grades as used inside the Red Hat Ecosystem Catalog for more details on this grading system.

See the Red Hat Product Security Center for details on security updates and vulnerabilities related to Red Hat software. Check out Red Hat Security Advisories to search for specific advisories and CVEs.

2.7.4. OpenShift Container Registry

OpenShift Container Platform includes the *OpenShift Container Registry*, a private registry running as an integrated component of the platform that you can use to manage your container images. The OpenShift Container Registry provides role-based access controls that allow you to manage who can pull and push which container images.

OpenShift Container Platform also supports integration with other private registries that you might already be using, such as Red Hat Quay.

Additional resources

- Integrated OpenShift image registry

2.7.5. Storing containers using Red Hat Quay

Red Hat Quay is an enterprise-quality container registry product from Red Hat. Development for Red Hat Quay is done through the upstream Project Quay. Red Hat Quay is available to deploy on-premise or through the hosted version of Red Hat Quay at Quay.io.

Security-related features of Red Hat Quay include:

- **Time machine**: Allows images with older tags to expire after a set period of time or based on a user-selected expiration time.
- **Repository mirroring**: Lets you mirror other registries for security reasons, such hosting a public repository on Red Hat Quay behind a company firewall, or for performance reasons, to keep registries closer to where they are used.
- **Action log storage**: Save Red Hat Quay logging output to Elasticsearch storage or Splunk to allow for later search and analysis.
- **Clair**: Scan images against a variety of Linux vulnerability databases, based on the origins of each container image.
- **Internal authentication**: Use the default local database to handle RBAC authentication to Red Hat Quay or choose from LDAP, Keystone (OpenStack), JWT Custom Authentication, or External Application Token authentication.
- **External authorization (OAuth)**: Allow authorization to Red Hat Quay from GitHub, GitHub Enterprise, or Google Authentication.
- **Access settings**: Generate tokens to allow access to Red Hat Quay from docker, rkt, anonymous access, user-created accounts, encrypted client passwords, or prefix username autocompletion.

Ongoing integration of Red Hat Quay with OpenShift Container Platform continues, with several OpenShift Container Platform Operators of particular interest. The Quay Bridge Operator lets you replace the internal OpenShift image registry with Red Hat Quay. The Red Hat Quay Container Security Operator lets you check vulnerabilities of images running in OpenShift Container Platform that were pulled from Red Hat Quay registries.

2.8. SECURING THE BUILD PROCESS

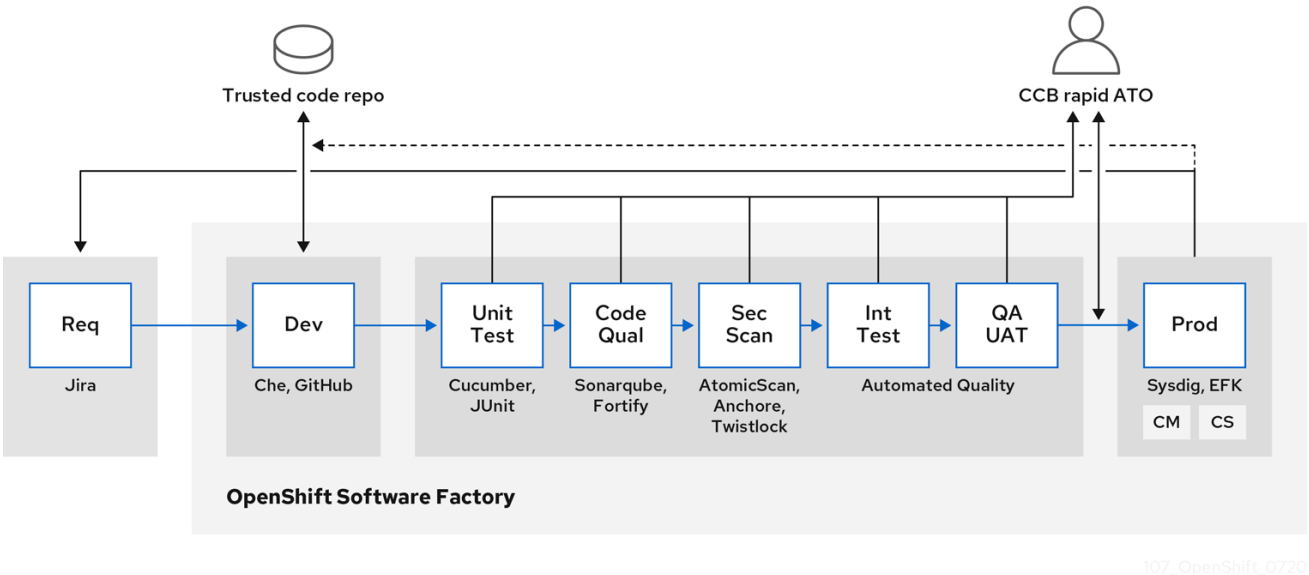
In a container environment, the software build process is the stage in the life cycle where application code is integrated with the required runtime libraries. Managing this build process is key to securing the software stack.

2.8.1. Building once, deploying everywhere

Using OpenShift Container Platform as the standard platform for container builds enables you to guarantee the security of the build environment. Adhering to a "build once, deploy everywhere" philosophy ensures that the product of the build process is exactly what is deployed in production.

It is also important to maintain the immutability of your containers. You should not patch running containers, but rebuild and redeploy them.

As your software moves through the stages of building, testing, and production, it is important that the tools making up your software supply chain be trusted. The following figure illustrates the process and tools that could be incorporated into a trusted software supply chain for containerized software:



OpenShift Container Platform can be integrated with trusted code repositories (such as GitHub) and development platforms (such as Che) for creating and managing secure code. Unit testing could rely on Cucumber and JUnit.

You can inspect your containers for vulnerabilities and configuration issues at build, deploy, or runtime with Red Hat Advanced Cluster Security for Kubernetes . For images stored in Quay, you can use the Clair scanner to inspect images at rest. In addition, there are certified vulnerability scanners available in the Red Hat ecosystem catalog.

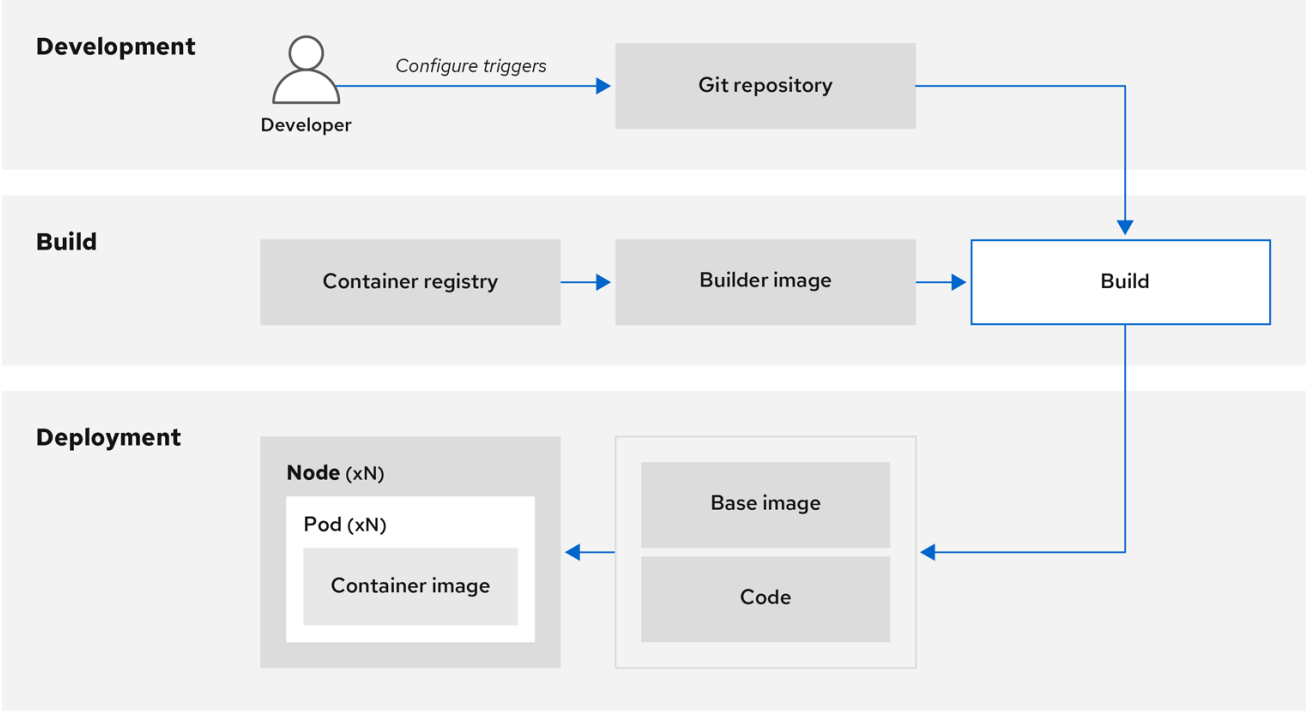
Tools such as Sysdig can provide ongoing monitoring of your containerized applications.

2.8.2. Managing builds

You can use Source-to-Image (S2I) to combine source code and base images. *Builder images* make use of S2I to enable your development and operations teams to collaborate on a reproducible build environment. With Red Hat S2I images available as Universal Base Image (UBI) images, you can now freely redistribute your software with base images built from real RHEL RPM packages. Red Hat has removed subscription restrictions to allow this.

When developers commit code with Git for an application using build images, OpenShift Container Platform can perform the following functions:

- Trigger, either by using webhooks on the code repository or other automated continuous integration (CI) process, to automatically assemble a new image from available artifacts, the S2I builder image, and the newly committed code.
- Automatically deploy the newly built image for testing.
- Promote the tested image to production where it can be automatically deployed using a CI process.



You can use the integrated OpenShift Container Registry to manage access to final images. Both S2I and native build images are automatically pushed to your OpenShift Container Registry.

In addition to the included Jenkins for CI, you can also integrate your own build and CI environment with OpenShift Container Platform using RESTful APIs, as well as use any API-compliant image registry.

2.8.3. Securing inputs during builds

In some scenarios, build operations require credentials to access dependent resources, but it is undesirable for those credentials to be available in the final application image produced by the build. You can define input secrets for this purpose.

For example, when building a Node.js application, you can set up your private mirror for Node.js modules. To download modules from that private mirror, you must supply a custom **.npmrc** file for the build that contains a URL, user name, and password. For security reasons, you do not want to expose your credentials in the application image.

Using this example scenario, you can add an input secret to a new **BuildConfig** object:

1. Create the secret, if it does not exist:

```
$ oc create secret generic secret-npmrc --from-file=.npmrc=~/.npmrc
```

   This creates a new secret named **secret-npmrc**, which contains the base64 encoded content of the **~/.npmrc** file.

2. Add the secret to the **source** section in the existing **BuildConfig** object:

```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
```

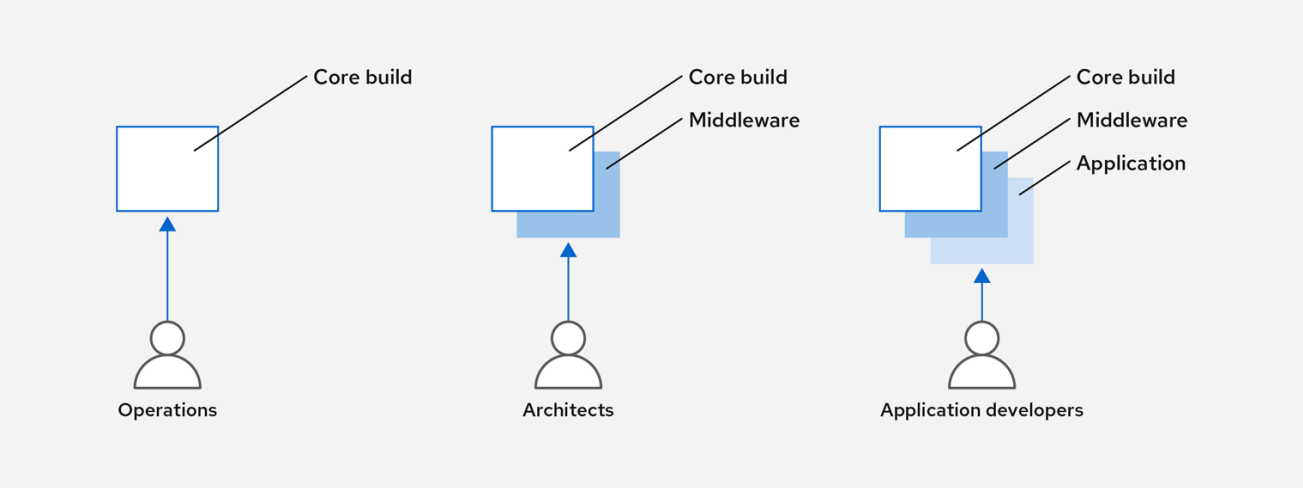
```
- destinationDir: .
  secret:
    name: secret-npmrc
```

3. To include the secret in a new **BuildConfig** object, run the following command:

```
$ oc new-build \
    openshift/nodejs-010-centos7~https://github.com/sclorg/nodejs-ex.git \
    --build-secret secret-npmrc
```

2.8.4. Designing your build process

You can design your container image management and build process to use container layers so that you can separate control.



For example, an operations team manages base images, while architects manage middleware, runtimes, databases, and other solutions. Developers can then focus on application layers and focus on writing code.

Because new vulnerabilities are identified daily, you need to proactively check container content over time. To do this, you should integrate automated security testing into your build or CI process. For example:

- SAST / DAST – Static and Dynamic security testing tools.
- Scanners for real-time checking against known vulnerabilities. Tools like these catalog the open source packages in your container, notify you of any known vulnerabilities, and update you when new vulnerabilities are discovered in previously scanned packages.

Your CI process should include policies that flag builds with issues discovered by security scans so that your team can take appropriate action to address those issues. You should sign your custom built containers to ensure that nothing is tampered with between build and deployment.

Using GitOps methodology, you can use the same CI/CD mechanisms to manage not only your application configurations, but also your OpenShift Container Platform infrastructure.

2.8.5. Building Knative serverless applications

Relying on Kubernetes and Kourier, you can build, deploy, and manage serverless applications by using OpenShift Serverless in OpenShift Container Platform.

As with other builds, you can use S2I images to build your containers, then serve them using Knative services. View Knative application builds through the **Topology** view of the OpenShift Container Platform web console.

2.8.6. Additional resources

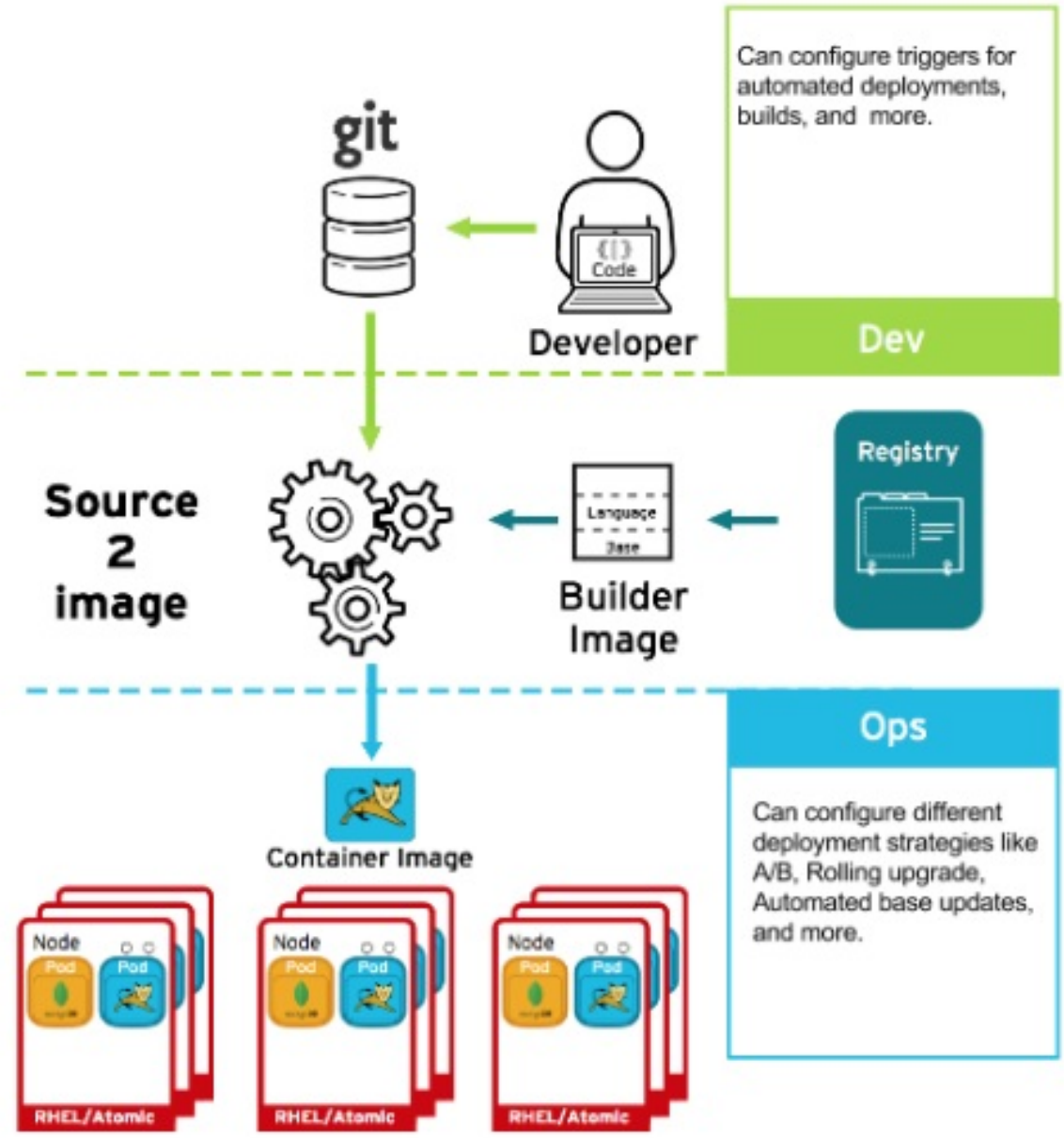
- Understanding image builds
- Triggering and modifying builds
- Creating build inputs
- Input secrets and config maps
- OpenShift Serverless overview
- Viewing application composition using the Topology view

2.9. DEPLOYING CONTAINERS

You can use a variety of techniques to make sure that the containers you deploy hold the latest production-quality content and that they have not been tampered with. These techniques include setting up build triggers to incorporate the latest code and using signatures to ensure that the container comes from a trusted source and has not been modified.

2.9.1. Controlling container deployments with triggers

If something happens during the build process, or if a vulnerability is discovered after an image has been deployed, you can use tooling for automated, policy-based deployment to remediate. You can use triggers to rebuild and replace images, ensuring the immutable containers process, instead of patching running containers, which is not recommended.



For example, you build an application using three container image layers: core, middleware, and applications. An issue is discovered in the core image and that image is rebuilt. After the build is complete, the image is pushed to your OpenShift Container Registry. OpenShift Container Platform detects that the image has changed and automatically rebuilds and deploys the application image, based on the defined triggers. This change incorporates the fixed libraries and ensures that the production code is identical to the most current image.

You can use the **oc set triggers** command to set a deployment trigger. For example, to set a trigger for a deployment called deployment-example:

```
$ oc set triggers deploy/deployment-example \
    --from-image=example:latest \
    --containers=web
```

2.9.2. Controlling what image sources can be deployed

It is important that the intended images are actually being deployed, that the images including the contained content are from trusted sources, and they have not been altered. Cryptographic signing provides this assurance. OpenShift Container Platform enables cluster administrators to apply security

policy that is broad or narrow, reflecting deployment environment and security requirements. Two parameters define this policy:

- one or more registries, with optional project namespace
- trust type, such as accept, reject, or require public key(s)

You can use these policy parameters to allow, deny, or require a trust relationship for entire registries, parts of registries, or individual images. Using trusted public keys, you can ensure that the source is cryptographically verified. The policy rules apply to nodes. Policy may be applied uniformly across all nodes or targeted for different node workloads (for example, build, zone, or environment).

Example image signature policy file

```
{
    "default": [{"type": "reject"}],
    "transports": {
        "docker": {
            "access.redhat.com": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ]
        },
        "atomic": {
            "172.30.1.1:5000/openshift": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ],
            "172.30.1.1:5000/production": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/example.com/pubkey"
                }
            ],
            "172.30.1.1:5000": [{"type": "reject"}]
        }
    }
}
```

The policy can be saved onto a node as **/etc/containers/policy.json**. Saving this file to a node is best accomplished using a new **MachineConfig** object. This example enforces the following rules:

- Require images from the Red Hat Registry (**registry.access.redhat.com**) to be signed by the Red Hat public key.
- Require images from your OpenShift Container Registry in the **openshift** namespace to be signed by the Red Hat public key.

- Require images from your OpenShift Container Registry in the **production** namespace to be signed by the public key for **example.com**.
- Reject all other registries not specified by the global **default** definition.

2.9.3. Using signature transports

A signature transport is a way to store and retrieve the binary signature blob. There are two types of signature transports.

- **atomic**: Managed by the OpenShift Container Platform API.
- **docker**: Served as a local file or by a web server.

The OpenShift Container Platform API manages signatures that use the **atomic** transport type. You must store the images that use this signature type in your OpenShift Container Registry. Because the docker/distribution **extensions** API auto-discovers the image signature endpoint, no additional configuration is required.

Signatures that use the **docker** transport type are served by local file or web server. These signatures are more flexible; you can serve images from any container image registry and use an independent server to deliver binary signatures.

However, the **docker** transport type requires additional configuration. You must configure the nodes with the URI of the signature server by placing arbitrarily-named YAML files into a directory on the host system, **/etc/containers/registries.d** by default. The YAML configuration files contain a registry URI and a signature server URI, or *sigstore*:

Example registries.d file

```
docker:
    access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

In this example, the Red Hat Registry, **access.redhat.com**, is the signature server that provides signatures for the **docker** transport type. Its URI is defined in the **sigstore** parameter. You might name this file **/etc/containers/registries.d/redhat.com.yaml** and use the Machine Config Operator to automatically place the file on each node in your cluster. No service restart is required since policy and **registries.d** files are dynamically loaded by the container runtime.

2.9.4. Creating secrets and config maps

The **Secret** object type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, **dockercfg** files, and private source repository credentials. Secrets decouple sensitive content from pods. You can mount secrets into containers using a volume plugin or the system can use secrets to perform actions on behalf of a pod.

For example, to add a secret to your deployment configuration so that it can access a private image repository, do the following:

Procedure

1. Log in to the OpenShift Container Platform web console.

2. Create a new project.

3. Navigate to **Resources → Secrets** and create a new secret. Set **Secret Type** to **Image Secret** and **Authentication Type** to **Image Registry Credentials** to enter credentials for accessing a private image repository.

4. When creating a deployment configuration (for example, from the **Add to Project → Deploy Image** page), set the **Pull Secret** to your new secret.

Config maps are similar to secrets, but are designed to support working with strings that do not contain sensitive information. The **ConfigMap** object holds key-value pairs of configuration data that can be consumed in pods or used to store configuration data for system components such as controllers.

2.9.5. Automating continuous deployment

You can integrate your own continuous deployment (CD) tooling with OpenShift Container Platform.

By leveraging CI/CD and OpenShift Container Platform, you can automate the process of rebuilding the application to incorporate the latest fixes, testing, and ensuring that it is deployed everywhere within the environment.

Additional resources

- Input secrets and config maps

2.10. SECURING THE CONTAINER PLATFORM

OpenShift Container Platform and Kubernetes APIs are key to automating container management at scale. APIs are used to:

- Validate and configure the data for pods, services, and replication controllers.
- Perform project validation on incoming requests and invoke triggers on other major system components.

Security-related features in OpenShift Container Platform that are based on Kubernetes include:

- Multitenancy, which combines Role-Based Access Controls and network policies to isolate containers at multiple levels.
- Admission plugins, which form boundaries between an API and those making requests to the API.

OpenShift Container Platform uses Operators to automate and simplify the management of Kubernetes-level security features.

2.10.1. Isolating containers with multitenancy

Multitenancy allows applications on an OpenShift Container Platform cluster that are owned by multiple users, and run across multiple hosts and namespaces, to remain isolated from each other and from outside attacks. You obtain multitenancy by applying role-based access control (RBAC) to Kubernetes namespaces.

In Kubernetes, *namespaces* are areas where applications can run in ways that are separate from other applications. OpenShift Container Platform uses and extends namespaces by adding extra annotations,

including MCS labeling in SELinux, and identifying these extended namespaces as *projects*. Within the scope of a project, users can maintain their own cluster resources, including service accounts, policies, constraints, and various other objects.

RBAC objects are assigned to projects to authorize selected users to have access to those projects. That authorization takes the form of rules, roles, and bindings:

- Rules define what a user can create or access in a project.
- Roles are collections of rules that you can bind to selected users or groups.
- Bindings define the association between users or groups and roles.

Local RBAC roles and bindings attach a user or group to a particular project. Cluster RBAC can attach cluster-wide roles and bindings to all projects in a cluster. There are default cluster roles that can be assigned to provide **admin**, **basic-user**, **cluster-admin**, and **cluster-status** access.

2.10.2. Protecting control plane with admission plugins

While RBAC controls access rules between users and groups and available projects, *admission plugins* define access to the OpenShift Container Platform master API. Admission plugins form a chain of rules that consist of:

- Default admissions plugins: These implement a default set of policies and resources limits that are applied to components of the OpenShift Container Platform control plane.
- Mutating admission plugins: These plugins dynamically extend the admission chain. They call out to a webhook server and can both authenticate a request and modify the selected resource.
- Validating admission plugins: These validate requests for a selected resource and can both validate the request and ensure that the resource does not change again.

API requests go through admissions plugins in a chain, with any failure along the way causing the request to be rejected. Each admission plugin is associated with particular resources and only responds to requests for those resources.

2.10.2.1. Security context constraints (SCCs)

You can use *security context constraints* (SCCs) to define a set of conditions that a pod must run with to be accepted into the system.

Some aspects that can be managed by SCCs include:

- Running of privileged containers
- Capabilities a container can request to be added
- Use of host directories as volumes
- SELinux context of the container
- Container user ID

If you have the required permissions, you can adjust the default SCC policies to be more permissive, if required.

2.10.2.2. Granting roles to service accounts

You can assign roles to service accounts, in the same way that users are assigned role-based access. There are three default service accounts created for each project. A service account:

- is limited in scope to a particular project
- derives its name from its project
- is automatically assigned an API token and credentials to access the OpenShift Container Registry

Service accounts associated with platform components automatically have their keys rotated.

2.10.3. Authentication and authorization

2.10.3.1. Controlling access using OAuth

You can use API access control via authentication and authorization for securing your container platform. The OpenShift Container Platform master includes a built-in OAuth server. Users can obtain OAuth access tokens to authenticate themselves to the API.

As an administrator, you can configure OAuth to authenticate using an *identity provider*, such as LDAP, GitHub, or Google. The identity provider is used by default for new OpenShift Container Platform deployments, but you can configure this at initial installation time or postinstallation.

2.10.3.2. API access control and management

Applications can have multiple, independent API services which have different endpoints that require management. OpenShift Container Platform includes a containerized version of the 3scale API gateway so that you can manage your APIs and control access.

3scale gives you a variety of standard options for API authentication and security, which can be used alone or in combination to issue credentials and control access: standard API keys, application ID and key pair, and OAuth 2.0.

You can restrict access to specific endpoints, methods, and services and apply access policy for groups of users. Application plans allow you to set rate limits for API usage and control traffic flow for groups of developers.

For a tutorial on using APIcast v2, the containerized 3scale API Gateway, see Running APIcast on Red Hat OpenShift in the 3scale documentation.

2.10.3.3. Red Hat Single Sign-On

The Red Hat Single Sign-On server enables you to secure your applications by providing web single sign-on capabilities based on standards, including SAML 2.0, OpenID Connect, and OAuth 2.0. The server can act as a SAML or OpenID Connect–based identity provider (IdP), mediating with your enterprise user directory or third-party identity provider for identity information and your applications using standards-based tokens. You can integrate Red Hat Single Sign-On with LDAP-based directory services including Microsoft Active Directory and Red Hat Enterprise Linux Identity Management.

2.10.3.4. Secure self-service web console

OpenShift Container Platform provides a self-service web console to ensure that teams do not access other environments without authorization. OpenShift Container Platform ensures a secure multitenant master by providing the following:

- Access to the master uses Transport Layer Security (TLS)
- Access to the API Server uses X.509 certificates or OAuth access tokens
- Project quota limits the damage that a rogue token could do
- The etcd service is not exposed directly to the cluster

2.10.4. Managing certificates for the platform

OpenShift Container Platform has multiple components within its framework that use REST-based HTTPS communication leveraging encryption via TLS certificates. OpenShift Container Platform's installer configures these certificates during installation. There are some primary components that generate this traffic:

- masters (API server and controllers)
- etcd
- nodes
- registry
- router

2.10.4.1. Configuring custom certificates

You can configure custom serving certificates for the public hostnames of the API server and web console during initial installation or when redeploying certificates. You can also use a custom CA.

Additional resources

- Introduction to OpenShift Container Platform
- Using RBAC to define and apply permissions
- About admission plugins
- Managing security context constraints
- SCC reference commands
- Examples of granting roles to service accounts
- Configuring the internal OAuth server
- Understanding identity provider configuration
- Certificate types and descriptions
- Proxy certificates

2.11. SECURING NETWORKS

Network security can be managed at several levels. At the pod level, network namespaces can prevent containers from seeing other pods or the host system by restricting network access. Network policies give you control over allowing and rejecting connections. You can manage ingress and egress traffic to and from your containerized applications.

2.11.1. Using network namespaces

OpenShift Container Platform uses software-defined networking (SDN) to provide a unified cluster network that enables communication between containers across the cluster.

Network policy mode, by default, makes all pods in a project accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create **NetworkPolicy** objects in that project to indicate the allowed incoming connections. Using multitenant mode, you can provide project-level isolation for pods and services.

2.11.2. Isolating pods with network policies

Using *network policies*, you can isolate pods from each other in the same project. Network policies can deny all network access to a pod, only allow connections for the Ingress Controller, reject connections from pods in other projects, or set similar rules for how networks behave.

Additional resources

- About network policy

2.11.3. Using multiple pod networks

Each running container has only one network interface by default. The Multus CNI plugin lets you create multiple CNI networks, and then attach any of those networks to your pods. In that way, you can do things like separate private data onto a more restricted network and have multiple network interfaces on each node.

Additional resources

- Using multiple networks

2.11.4. Isolating applications

OpenShift Container Platform enables you to segment network traffic on a single cluster to make multitenant clusters that isolate users, teams, applications, and environments from non-global resources.

Additional resources

- Configuring network isolation using OpenShiftSDN

2.11.5. Securing ingress traffic

There are many security implications related to how you configure access to your Kubernetes services from outside of your OpenShift Container Platform cluster. Besides exposing HTTP and HTTPS routes, ingress routing allows you to set up NodePort or LoadBalancer ingress types. NodePort exposes an

application's service API object from each cluster worker. LoadBalancer lets you assign an external load balancer to an associated service API object in your OpenShift Container Platform cluster.

Additional resources

- Configuring ingress cluster traffic

2.11.6. Securing egress traffic

OpenShift Container Platform provides the ability to control egress traffic using either a router or firewall method. For example, you can use IP whitelisting to control database access. A cluster administrator can assign one or more egress IP addresses to a project in an OpenShift Container Platform SDN network provider. Likewise, a cluster administrator can prevent egress traffic from going outside of an OpenShift Container Platform cluster using an egress firewall.

By assigning a fixed egress IP address, you can have all outgoing traffic assigned to that IP address for a particular project. With the egress firewall, you can prevent a pod from connecting to an external network, prevent a pod from connecting to an internal network, or limit a pod's access to specific internal subnets.

Additional resources

- Configuring an egress firewall to control access to external IP addresses
- Configuring egress IPs for a project

2.12. SECURING ATTACHED STORAGE

OpenShift Container Platform supports multiple types of storage, both for on-premise and cloud providers. In particular, OpenShift Container Platform can use storage types that support the Container Storage Interface.

2.12.1. Persistent volume plugins

Containers are useful for both stateless and stateful applications. Protecting attached storage is a key element of securing stateful services. Using the Container Storage Interface (CSI), OpenShift Container Platform can incorporate storage from any storage back end that supports the CSI interface.

OpenShift Container Platform provides plugins for multiple types of storage, including:

- Red Hat OpenShift Data Foundation *
- AWS Elastic Block Stores (EBS) *
- AWS Elastic File System (EFS) *
- Azure Disk *
- Azure File *
- OpenStack Cinder *
- GCE Persistent Disks *
- VMware vSphere *

- Network File System (NFS)
- FlexVolume
- Fibre Channel
- iSCSI

Plugins for those storage types with dynamic provisioning are marked with an asterisk (*). Data in transit is encrypted via HTTPS for all OpenShift Container Platform components communicating with each other.

You can mount a persistent volume (PV) on a host in any way supported by your storage type. Different types of storage have different capabilities and each PV's access modes are set to the specific modes supported by that particular volume.

For example, NFS can support multiple read/write clients, but a specific NFS PV might be exported on the server as read-only. Each PV has its own set of access modes describing that specific PV's capabilities, such as **ReadWriteOnce**, **ReadOnlyMany**, and **ReadWriteMany**.

2.12.2. Shared storage

For shared storage providers like NFS, the PV registers its group ID (GID) as an annotation on the PV resource. Then, when the PV is claimed by the pod, the annotated GID is added to the supplemental groups of the pod, giving that pod access to the contents of the shared storage.

2.12.3. Block storage

For block storage providers like AWS Elastic Block Store (EBS), GCE Persistent Disks, and iSCSI, OpenShift Container Platform uses SELinux capabilities to secure the root of the mounted volume for non-privileged pods, making the mounted volume owned by and only visible to the container with which it is associated.

Additional resources

- Understanding persistent storage
- Configuring CSI volumes
- Dynamic provisioning
- Persistent storage using NFS
- Persistent storage using AWS Elastic Block Store
- Persistent storage using GCE Persistent Disk

2.13. MONITORING CLUSTER EVENTS AND LOGS

The ability to monitor and audit an OpenShift Container Platform cluster is an important part of safeguarding the cluster and its users against inappropriate usage.

There are two main sources of cluster-level information that are useful for this purpose: events and logging.

2.13.1. Watching cluster events

Cluster administrators are encouraged to familiarize themselves with the **Event** resource type and review the list of system events to determine which events are of interest. Events are associated with a namespace, either the namespace of the resource they are related to or, for cluster events, the **default** namespace. The default namespace holds relevant events for monitoring or auditing a cluster, such as node events and resource events related to infrastructure components.

The master API and **oc** command do not provide parameters to scope a listing of events to only those related to nodes. A simple approach would be to use **grep**:

```
$ oc get event -n default | grep Node
```

Example output

```
1h         20h         3         origin-node-1.example.local   Node      Normal    NodeHasDiskPressure   ...
```

A more flexible approach is to output the events in a form that other tools can process. For example, the following example uses the **jq** tool against JSON output to extract only **NodeHasDiskPressure** events:

```
$ oc get events -n default -o json \
  | jq '.items[] | select(.involvedObject.kind == "Node" and .reason == "NodeHasDiskPressure")'
```

Example output

```
{
  "apiVersion": "v1",
  "count": 3,
  "involvedObject": {
    "kind": "Node",
    "name": "origin-node-1.example.local",
    "uid": "origin-node-1.example.local"
  },
  "kind": "Event",
  "reason": "NodeHasDiskPressure",
  ...
}
```

Events related to resource creation, modification, or deletion can also be good candidates for detecting misuse of the cluster. The following query, for example, can be used to look for excessive pulling of images:

```
$ oc get events --all-namespaces -o json \
  | jq '[.items[] | select(.involvedObject.kind == "Pod" and .reason == "Pulling")] | length'
```

Example output

```
4
```

NOTE

When a namespace is deleted, its events are deleted as well. Events can also expire and are deleted to prevent filling up etcd storage. Events are not stored as a permanent record and frequent polling is necessary to capture statistics over time.

2.13.2. Logging

Using the **oc log** command, you can view container logs, build configs and deployments in real time. Different can users have access different access to logs:

- Users who have access to a project are able to see the logs for that project by default.
- Users with admin roles can access all container logs.

To save your logs for further audit and analysis, you can enable the **cluster-logging** add-on feature to collect, manage, and view system, container, and audit logs. You can deploy, manage, and upgrade OpenShift Logging through the OpenShift Elasticsearch Operator and Red Hat OpenShift Logging Operator.

2.13.3. Audit logs

With *audit logs*, you can follow a sequence of activities associated with how a user, administrator, or other OpenShift Container Platform component is behaving. API audit logging is done on each server.

Additional resources

- List of system events
- Understanding OpenShift Logging
- Viewing audit logs

CHAPTER 3. CONFIGURING CERTIFICATES

3.1. REPLACING THE DEFAULT INGRESS CERTIFICATE

3.1.1. Understanding the default ingress certificate

By default, OpenShift Container Platform uses the Ingress Operator to create an internal CA and issue a wildcard certificate that is valid for applications under the **.apps** sub-domain. Both the web console and CLI use this certificate as well.

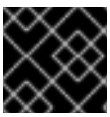
The internal infrastructure CA certificates are self-signed. While this process might be perceived as bad practice by some security or PKI teams, any risk here is minimal. The only clients that implicitly trust these certificates are other components within the cluster. Replacing the default wildcard certificate with one that is issued by a public CA already included in the CA bundle as provided by the container userspace allows external clients to connect securely to applications running under the **.apps** sub-domain.

3.1.2. Replacing the default ingress certificate

You can replace the default ingress certificate for all applications under the **.apps** subdomain. After you replace the certificate, all applications, including the web console and CLI, will have encryption provided by specified certificate.

Prerequisites

- You must have a wildcard certificate for the fully qualified **.apps** subdomain and its corresponding private key. Each should be in a separate PEM format file.
- The private key must be unencrypted. If your key is encrypted, decrypt it before importing it into OpenShift Container Platform.
- The certificate must include the **subjectAltName** extension showing ***.apps.<clustername>.<domain>**.
- The certificate file can contain one or more certificates in a chain. The wildcard certificate must be the first certificate in the file. It can then be followed with any intermediate certificates, and the file should end with the root CA certificate.
- Copy the root CA certificate into an additional PEM format file.
- Verify that all certificates which include **-----END CERTIFICATE-----** also end with one carriage return after that line.



IMPORTANT

Updating the certificate authority (CA) causes the nodes in your cluster to reboot.

Procedure

1. Create a config map that includes only the root CA certificate used to sign the wildcard certificate:

```
$ oc create configmap custom-ca \
    --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
    -n openshift-config
```

1 **</path/to/example-ca.crt>** is the path to the root CA certificate file on your local file system.

2. Update the cluster-wide proxy configuration with the newly created config map:

```
$ oc patch proxy/cluster \
    --type=merge \
    --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'
```

3. Create a secret that contains the wildcard certificate chain and key:

```
$ oc create secret tls <secret> \ 1
    --cert=</path/to/cert.crt> \ 2
    --key=</path/to/cert.key> \ 3
    -n openshift-ingress
```

1 **<secret>** is the name of the secret that will contain the certificate chain and private key.

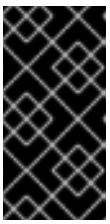
2 **</path/to/cert.crt>** is the path to the certificate chain on your local file system.

3 **</path/to/cert.key>** is the path to the private key associated with this certificate.

4. Update the Ingress Controller configuration with the newly created secret:

```
$ oc patch ingresscontroller.operator default \
    --type=merge -p \
    '{"spec":{"defaultCertificate": {"name": "<secret>"}}}' \ 1
    -n openshift-ingress-operator
```

1 Replace **<secret>** with the name used for the secret in the previous step.



IMPORTANT

To trigger the Ingress Operator to perform a rolling update, you must update the name of the secret. Because the kubelet automatically propagates changes to the secret in the volume mount, updating the secret contents does not trigger a rolling update. For more information, see this Red Hat Knowledgebase Solution.

Additional resources

- Replacing the CA Bundle certificate
- Proxy certificate customization

3.2. ADDING API SERVER CERTIFICATES

The default API server certificate is issued by an internal OpenShift Container Platform cluster CA. Clients outside of the cluster will not be able to verify the API server's certificate by default. This certificate can be replaced by one that is issued by a CA that clients trust.

3.2.1. Add an API server named certificate

The default API server certificate is issued by an internal OpenShift Container Platform cluster CA. You can add one or more alternative certificates that the API server will return based on the fully qualified domain name (FQDN) requested by the client, for example when a reverse proxy or load balancer is used.

Prerequisites

- You must have a certificate for the FQDN and its corresponding private key. Each should be in a separate PEM format file.
- The private key must be unencrypted. If your key is encrypted, decrypt it before importing it into OpenShift Container Platform.
- The certificate must include the **subjectAltName** extension showing the FQDN.
- The certificate file can contain one or more certificates in a chain. The certificate for the API server FQDN must be the first certificate in the file. It can then be followed with any intermediate certificates, and the file should end with the root CA certificate.

WARNING

Do not provide a named certificate for the internal load balancer (host name **api-int.<cluster_name>.<base_domain>**). Doing so will leave your cluster in a degraded state.

Procedure

1. Login to the new API as the **kubeadmin** user.

```
$ oc login -u kubeadmin -p <password> https://FQDN:6443
```

2. Get the **kubeconfig** file.

```
$ oc config view --flatten > kubeconfig-newapi
```

3. Create a secret that contains the certificate chain and private key in the **openshift-config** namespace.

```
$ oc create secret tls <secret> \ 1
    --cert=</path/to/cert.crt> \ 2
    --key=</path/to/cert.key> \ 3
    -n openshift-config
```

1. **<secret>** is the name of the secret that will contain the certificate chain and private key.
2. **</path/to/cert.crt>** is the path to the certificate chain on your local file system.
3. **</path/to/cert.key>** is the path to the private key associated with this certificate.

4. Update the API server to reference the created secret.

```
$ oc patch apiserver cluster \
    --type=merge -p \
    '{"spec":{"servingCerts": {"namedCertificates":
    [{"names": ["<FQDN>"],
    "servingCertificate": {"name": "<secret>"}}]}}}'
```

where:

<FQDN>
: Replace **<FQDN>** with the FQDN that the API server should provide the certificate for. Do not include the port number.

<secret>
: Replace **<secret>** with the name used for the secret in the previous step.

5. Examine the **apiserver/cluster** object and confirm the secret is now referenced.

```
$ oc get apiserver cluster -o yaml
```

Example output

```
...
spec:
  servingCerts:
    namedCertificates:
    - names:
      - <FQDN>
      servingCertificate:
        name: <secret>
...
```

6. Check the **kube-apiserver** operator, and verify that a new revision of the Kubernetes API server rolls out. It may take a minute for the operator to detect the configuration change and trigger a new deployment. While the new revision is rolling out, **PROGRESSING** will report **True**.

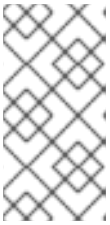
```
$ oc get clusteroperators kube-apiserver
```

Do not continue to the next step until **PROGRESSING** is listed as **False**, as shown in the following output:

Example output

```
NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
kube-apiserver   4.13.0    True        False         False      145m
```

If **PROGRESSING** is showing **True**, wait a few minutes and try again.



NOTE

A new revision of the Kubernetes API server only rolls out if the API server named certificate is added for the first time. When the API server named certificate is renewed, a new revision of the Kubernetes API server does not roll out because the **kube-apiserver** pods dynamically reload the updated certificate.

3.3. SECURING SERVICE TRAFFIC USING SERVICE SERVING CERTIFICATE SECRETS

3.3.1. Understanding service serving certificates

Service serving certificates are intended to support complex middleware applications that require encryption. These certificates are issued as TLS web server certificates.

The **service-ca** controller uses the **x509.SHA256WithRSA** signature algorithm to generate service certificates.

The generated certificate and key are in PEM format, stored in **tls.crt** and **tls.key** respectively, within a created secret. The certificate and key are automatically replaced when they get close to expiration.

The service CA certificate, which issues the service certificates, is valid for 26 months and is automatically rotated when there is less than 13 months validity left. After rotation, the previous service CA configuration is still trusted until its expiration. This allows a grace period for all affected services to refresh their key material before the expiration. If you do not upgrade your cluster during this grace period, which restarts services and refreshes their key material, you might need to manually restart services to avoid failures after the previous service CA expires.

NOTE

You can use the following command to manually restart all pods in the cluster. Be aware that running this command causes a service interruption, because it deletes every running pod in every namespace. These pods will automatically restart after they are deleted.

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
    do oc delete pods --all -n $I; \
    sleep 1; \
    done
```

3.3.2. Add a service certificate

To secure communication to your service, generate a signed serving certificate and key pair into a secret in the same namespace as the service.

The generated certificate is only valid for the internal service DNS name **<service.name>.<service.namespace>.svc**, and is only valid for internal communications. If your service is a headless service (no **clusterIP** value set), the generated certificate also contains a wildcard subject in the format of ***.<service.name>.<service.namespace>.svc**.

IMPORTANT

Because the generated certificates contain wildcard subjects for headless services, you must not use the service CA if your client must differentiate between individual pods. In this case:

- Generate individual TLS certificates by using a different CA.
- Do not accept the service CA as a trusted CA for connections that are directed to individual pods and must not be impersonated by other pods. These connections must be configured to trust the CA that was used to generate the individual TLS certificates.

Prerequisites

- You must have a service defined.

Procedure

1. Annotate the service with **service.beta.openshift.io/serving-cert-secret-name**:

```
$ oc annotate service <service_name> \ 1
    service.beta.openshift.io/serving-cert-secret-name=<secret_name> 2
```

1 Replace **<service_name>** with the name of the service to secure.

2 **<secret_name>** will be the name of the generated secret containing the certificate and key pair.

NOTE

For convenience, it is recommended that this value be the same as **<service_name>**.

For example, use the following command to annotate the service **test1**:

```
$ oc annotate service test1 service.beta.openshift.io/serving-cert-secret-name=test1
```

2. Examine the service to confirm that the annotations are present:

```
$ oc describe service <service_name>
```

Example output

```
...
Annotations:              service.beta.openshift.io/serving-cert-secret-name: <service_name>
                          service.beta.openshift.io/serving-cert-signed-by: openshift-service-serving-signer@1556850837
...
```

3. After the cluster generates a secret for your service, your **Pod** spec can mount it, and the pod will run after it becomes available.

Additional resources

- You can use a service certificate to configure a secure route using reencrypt TLS termination. For more information, see Creating a re-encrypt route with a custom certificate .

3.3.3. Add the service CA bundle to a config map

A pod can access the service Certificate Authority (CA) certificate by mounting a **ConfigMap** object that has the **service.beta.openshift.io/inject-cabundle=true** annotation. After annotating the config map, the cluster automatically injects the service CA certificate into the **service-ca.crt** key on the config map. Access to this CA certificate allows TLS clients to verify connections to services by using service serving certificates.

IMPORTANT

After adding this annotation to a config map, the OpenShift Service CA Operator deletes all the data in the config map. Consider using a separate config map to contain the **service-ca.crt**, instead of using the same config map that stores your pod configuration.

Procedure

1. Annotate the config map with the **service.beta.openshift.io/inject-cabundle=true** annotation by entering the following command:

```
$ oc annotate configmap <config_map_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 Replace **<config_map_name>** with the name of the config map to annotate.

NOTE

Explicitly referencing the **service-ca.crt** key in a volume mount prevents a pod from starting until the config map has been injected with the CA bundle. You can override this behavior by setting the **optional** parameter to **true** in the serving certificate configuration of the volume.

2. View the config map to ensure that the service CA bundle has been injected:

```
$ oc get configmap <config_map_name> -o yaml
```

The CA bundle is displayed as the value of the **service-ca.crt** key in the YAML output:

```
apiVersion: v1
data:
  service-ca.crt: |
    -----BEGIN CERTIFICATE-----
...
```

3. Mount the config map as a volume to each container that exists in a pod by configuring your **Deployment** object.

Example Deployment object that defines the volume for the mounted config map

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
          volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
      volumes:
      - name: trusted-ca
        configMap:
          name: <config_map_name> 1
          items:
            - key: ca-bundle.crt 2
              path: tls-ca-bundle.pem 3
# ...
```

1. Specify the name of the config map that you annotated in an earlier step of the procedure.
2. **ca-bundle.crt** is required as the ConfigMap key.
3. **tls-ca-bundle.pem** is required as the ConfigMap path.

3.3.4. Add the service CA bundle to an API service

You can annotate an **APIService** object with **service.beta.openshift.io/inject-cabundle=true** to have its **spec.caBundle** field populated with the service CA bundle. This allows the Kubernetes API server to validate the service CA certificate used to secure the targeted endpoint.

Procedure

1. Annotate the API service with **service.beta.openshift.io/inject-cabundle=true**:

```
$ oc annotate apiservice <api_service_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1. Replace **<api_service_name>** with the name of the API service to annotate.

For example, use the following command to annotate the API service **test1**:

```
$ oc annotate apiservice test1 service.beta.openshift.io/inject-cabundle=true
```

2. View the API service to ensure that the service CA bundle has been injected:

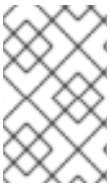
```
$ oc get apiservice <api_service_name> -o yaml
```

The CA bundle is displayed in the **spec.caBundle** field in the YAML output:

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
spec:
  caBundle: <CA_BUNDLE>
...
```

3.3.5. Add the service CA bundle to a custom resource definition

You can annotate a **CustomResourceDefinition** (CRD) object with **service.beta.openshift.io/inject-cabundle=true** to have its **spec.conversion.webhook.clientConfig.caBundle** field populated with the service CA bundle. This allows the Kubernetes API server to validate the service CA certificate used to secure the targeted endpoint.



NOTE

The service CA bundle will only be injected into the CRD if the CRD is configured to use a webhook for conversion. It is only useful to inject the service CA bundle if a CRD's webhook is secured with a service CA certificate.

Procedure

1. Annotate the CRD with **service.beta.openshift.io/inject-cabundle=true**:

```
$ oc annotate crd <crd_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 Replace **<crd_name>** with the name of the CRD to annotate.

For example, use the following command to annotate the CRD **test1**:

```
$ oc annotate crd test1 service.beta.openshift.io/inject-cabundle=true
```

2. View the CRD to ensure that the service CA bundle has been injected:

```
$ oc get crd <crd_name> -o yaml
```

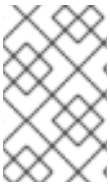
The CA bundle is displayed in the **spec.conversion.webhook.clientConfig.caBundle** field in the YAML output:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
```

```
spec:
  conversion:
    strategy: Webhook
    webhook:
      clientConfig:
        caBundle: <CA_BUNDLE>
...
```

3.3.6. Add the service CA bundle to a mutating webhook configuration

You can annotate a **MutatingWebhookConfiguration** object with **service.beta.openshift.io/inject-cabundle=true** to have the **clientConfig.caBundle** field of each webhook populated with the service CA bundle. This allows the Kubernetes API server to validate the service CA certificate used to secure the targeted endpoint.



NOTE

Do not set this annotation for admission webhook configurations that need to specify different CA bundles for different webhooks. If you do, then the service CA bundle will be injected for all webhooks.

Procedure

1. Annotate the mutating webhook configuration with **service.beta.openshift.io/inject-cabundle=true**:

```
$ oc annotate mutatingwebhookconfigurations <mutating_webhook_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 Replace **<mutating_webhook_name>** with the name of the mutating webhook configuration to annotate.

For example, use the following command to annotate the mutating webhook configuration **test1**:

```
$ oc annotate mutatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. View the mutating webhook configuration to ensure that the service CA bundle has been injected:

```
$ oc get mutatingwebhookconfigurations <mutating_webhook_name> -o yaml
```

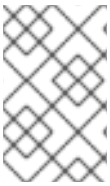
The CA bundle is displayed in the **clientConfig.caBundle** field of all webhooks in the YAML output:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
```

```
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

3.3.7. Add the service CA bundle to a validating webhook configuration

You can annotate a **ValidatingWebhookConfiguration** object with **service.beta.openshift.io/inject-cabundle=true** to have the **clientConfig.caBundle** field of each webhook populated with the service CA bundle. This allows the Kubernetes API server to validate the service CA certificate used to secure the targeted endpoint.



NOTE

Do not set this annotation for admission webhook configurations that need to specify different CA bundles for different webhooks. If you do, then the service CA bundle will be injected for all webhooks.

Procedure

1. Annotate the validating webhook configuration with **service.beta.openshift.io/inject-cabundle=true**:

```
$ oc annotate validatingwebhookconfigurations <validating_webhook_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 Replace **<validating_webhook_name>** with the name of the validating webhook configuration to annotate.

For example, use the following command to annotate the validating webhook configuration **test1**:

```
$ oc annotate validatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. View the validating webhook configuration to ensure that the service CA bundle has been injected:

```
$ oc get validatingwebhookconfigurations <validating_webhook_name> -o yaml
```

The CA bundle is displayed in the **clientConfig.caBundle** field of all webhooks in the YAML output:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
webhooks:
```

```
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

3.3.8. Manually rotate the generated service certificate

You can rotate the service certificate by deleting the associated secret. Deleting the secret results in a new one being automatically created, resulting in a new certificate.

Prerequisites

- A secret containing the certificate and key pair must have been generated for the service.

Procedure

1. Examine the service to determine the secret containing the certificate. This is found in the **serving-cert-secret-name** annotation, as seen below.

```
$ oc describe service <service_name>
```

Example output

```
...
service.beta.openshift.io/serving-cert-secret-name: <secret>
...
```

2. Delete the generated secret for the service. This process will automatically recreate the secret.

```
$ oc delete secret <secret> 1
```

1 Replace **<secret>** with the name of the secret from the previous step.

3. Confirm that the certificate has been recreated by obtaining the new secret and examining the **AGE**.

```
$ oc get secret <service_name>
```

Example output

```
NAME            TYPE                DATA   AGE
<service.name>  kubernetes.io/tls   2      1s
```

3.3.9. Manually rotate the service CA certificate

The service CA is valid for 26 months and is automatically refreshed when there is less than 13 months validity left.

If necessary, you can manually refresh the service CA by using the following procedure.

WARNING

A manually-rotated service CA does not maintain trust with the previous service CA. You might experience a temporary service disruption until the pods in the cluster are restarted, which ensures that pods are using service serving certificates issued by the new service CA.

Prerequisites

- You must be logged in as a cluster admin.

Procedure

1. View the expiration date of the current service CA certificate by using the following command.

```
$ oc get secrets/signing-key -n openshift-service-ca \
    -o template='{{index .data "tls.crt"}}' \
    | base64 --decode \
    | openssl x509 -noout -enddate
```

2. Manually rotate the service CA. This process generates a new service CA which will be used to sign the new service certificates.

```
$ oc delete secret/signing-key -n openshift-service-ca
```

3. To apply the new certificates to all services, restart all the pods in your cluster. This command ensures that all services use the updated certificates.

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
    do oc delete pods --all -n $I; \
    sleep 1; \
    done
```

WARNING

This command will cause a service interruption, as it goes through and deletes every running pod in every namespace. These pods will automatically restart after they are deleted.

3.4. UPDATING THE CA BUNDLE

IMPORTANT

Updating the certificate authority (CA) will cause the nodes of your cluster to reboot.

3.4.1. Understanding the CA Bundle certificate

Proxy certificates allow users to specify one or more custom certificate authority (CA) used by platform components when making egress connections.

The **trustedCA** field of the Proxy object is a reference to a config map that contains a user-provided trusted certificate authority (CA) bundle. This bundle is merged with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle and injected into the trust store of platform components that make egress HTTPS calls. For example, **image-registry-operator** calls an external image registry to download images. If **trustedCA** is not specified, only the RHCOS trust bundle is used for proxied HTTPS connections. Provide custom CA certificates to the RHCOS trust bundle if you want to use your own certificate infrastructure.

The **trustedCA** field should only be consumed by a proxy validator. The validator is responsible for reading the certificate bundle from required key **ca-bundle.crt** and copying it to a config map named **trusted-ca-bundle** in the **openshift-config-managed** namespace. The namespace for the config map referenced by **trustedCA** is **openshift-config**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

3.4.2. Replacing the CA Bundle certificate

Procedure

1. Create a config map that includes the root CA certificate used to sign the wildcard certificate:

```
$ oc create configmap custom-ca \
    --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
    -n openshift-config
```

1 **</path/to/example-ca.crt>** is the path to the CA certificate bundle on your local file system.

2. Update the cluster-wide proxy configuration with the newly created config map:

```
$ oc patch proxy/cluster \
    --type=merge \
    --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'
```

Additional resources

- Replacing the default ingress certificate
- Enabling the cluster-wide proxy

- Proxy certificate customization

CHAPTER 4. CERTIFICATE TYPES AND DESCRIPTIONS

4.1. USER-PROVIDED CERTIFICATES FOR THE API SERVER

4.1.1. Purpose

The API server is accessible by clients external to the cluster at **api.<cluster_name>.<base_domain>**. You might want clients to access the API server at a different hostname or without the need to distribute the cluster-managed certificate authority (CA) certificates to the clients. The administrator must set a custom default certificate to be used by the API server when serving content.

4.1.2. Location

The user-provided certificates must be provided in a **kubernetes.io/tls** type **Secret** in the **openshift-config** namespace. Update the API server cluster configuration, the **apiserver/cluster** resource, to enable the use of the user-provided certificate.

4.1.3. Management

User-provided certificates are managed by the user.

4.1.4. Expiration

API server client certificate expiration is less than five minutes.

User-provided certificates are managed by the user.

4.1.5. Customization

Update the secret containing the user-managed certificate as needed.

Additional resources

- Adding API server certificates

4.2. PROXY CERTIFICATES

4.2.1. Purpose

Proxy certificates allow users to specify one or more custom certificate authority (CA) certificates used by platform components when making egress connections.

The **trustedCA** field of the Proxy object is a reference to a config map that contains a user-provided trusted certificate authority (CA) bundle. This bundle is merged with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle and injected into the trust store of platform components that make egress HTTPS calls. For example, **image-registry-operator** calls an external image registry to download images. If **trustedCA** is not specified, only the RHCOS trust bundle is used for proxied HTTPS connections. Provide custom CA certificates to the RHCOS trust bundle if you want to use your own certificate infrastructure.

The **trustedCA** field should only be consumed by a proxy validator. The validator is responsible for reading the certificate bundle from required key **ca-bundle.crt** and copying it to a config map named

trusted-ca-bundle in the **openshift-config-managed** namespace. The namespace for the config map referenced by **trustedCA** is **openshift-config**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

Additional resources

- Configuring the cluster-wide proxy

4.2.2. Managing proxy certificates during installation

The **additionalTrustBundle** value of the installer configuration is used to specify any proxy-trusted CA certificates during installation. For example:

```
$ cat install-config.yaml
```

Example output

```
...
proxy:
  httpProxy: http://<username:password@proxy.example.com:123/>
  httpsProxy: http://<username:password@proxy.example.com:123/>
  noProxy: <123.example.com,10.88.0.0/16>
additionalTrustBundle: |
    -----BEGIN CERTIFICATE-----
   <MY_HTTPS_PROXY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...
```

4.2.3. Location

The user-provided trust bundle is represented as a config map. The config map is mounted into the file system of platform components that make egress HTTPS calls. Typically, Operators mount the config map to **/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**, but this is not required by the proxy. A proxy can modify or inspect the HTTPS connection. In either case, the proxy must generate and sign a new certificate for the connection.

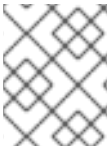
Complete proxy support means connecting to the specified proxy and trusting any signatures it has generated. Therefore, it is necessary to let the user specify a trusted root, such that any certificate chain connected to that trusted root is also trusted.

If you use the RHCOS trust bundle, place CA certificates in **/etc/pki/ca-trust/source/anchors**. For more information, see Using shared system certificates in the Red Hat Enterprise Linux (RHEL) *Securing networks* document.

4.2.4. Expiration

The user sets the expiration term of the user-provided trust bundle.

The default expiration term is defined by the CA certificate itself. It is up to the CA administrator to configure this for the certificate before it can be used by OpenShift Container Platform or RHCOS.



NOTE

Red Hat does not monitor for when CAs expire. However, due to the long life of CAs, this is generally not an issue. However, you might need to periodically update the trust bundle.

4.2.5. Services

By default, all platform components that make egress HTTPS calls will use the RHCOS trust bundle. If **trustedCA** is defined, it will also be used.

Any service that is running on the RHCOS node is able to use the trust bundle of the node.

4.2.6. Management

These certificates are managed by the system and not the user.

4.2.7. Customization

Updating the user-provided trust bundle consists of either:

- updating the PEM-encoded certificates in the config map referenced by **trustedCA,** or
- creating a config map in the namespace **openshift-config** that contains the new trust bundle and updating **trustedCA** to reference the name of the new config map.

The mechanism for writing CA certificates to the RHCOS trust bundle is exactly the same as writing any other file to RHCOS, which is done through the use of machine configs. When the Machine Config Operator (MCO) applies the new machine config that contains the new CA certificates, the node is rebooted. During the next boot, the service **coreos-update-ca-trust.service** runs on the RHCOS nodes, which automatically update the trust bundle with the new CA certificates. For example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-examplecorp-ca-cert
spec:
  config:
    ignition:
      version: 3.1.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVORENDQXh5Z0F3SUJBZ0lKQU5
1bkkwRDY2MmNuTUEwR0NTcUdTSWIzRFFFQkN3VUFNSUdsTVFzd0NRWUQKV1FRR0V3SlZVek
VYTUJVR0ExVUVDQXdPVG05eWRHZ2dRMkZ5YjJ4cGJtRXhFREFPQmdOVkJBY01CMUpoYkdWcA
```

pBMmd4RmpBVUJnTlZCQW9NRFZKbFpDQklZWFFzSUVsdVl5NHhFekFSQmdOVkJBc01DbEpsWk
NCSVlYUWdTVlF4Ckh6QVpCZ05WQkFNTUVsSmxaQ0JJWVhRZ1NWUWdVbTl2ZENCRFFURWhN
QjhHQ1NxR1NJYjNEUUVKQVJZU2FXNW0KWGpDQnBURUxNQWtHQTFVRUJoTUNWVk14RnpBV
kJnTlZCQWdNRGs1dmNuUm9JRU5oY205c2FXNWhNUkF3RGdZRApXUVFIREFkU1lXeGxhV2RvTV
JZd0ZBWURWUVFLREExU1pXUWdTR0YwTENCSmJtTXVNUk13RVFZRFZRUUxEQXBTCkFXUWd
TR0YwSUVsVU1Sc3dHUVlEVlFRRERCSlNaV1FnU0dGMElFbFVJRkp2YjNRZ1EwRXhJVEFmQmdrc
WhraUcKMHcwQkNRRVdFbWx1Wm05elpXTkFjbVZrYUdGMExtTnZiVENDQVNJd0RRWUpLb1pJaH
ZjTkFRRUJCUUFEZ2dFUApCRENDQVFvQ2dnRUJBTFF0OU9KUWg2R0M1TFQxZzgwcU5oMHU1
MEJRNHNaL3laOGFFVHh0KzVsblBWWDZNSEt6CmQvaTdsRHFUZlRjZkxMMm55VUJkMmZRRGsx
QjBmeHJza2hHSUlaM2lmUDFQczRsdFRrdjhoUlNvYjNWdE5xU28KSHhrS2Z2RDJQS2pUUHhEUFdZ
eXJ1eTlpckxaaW9NZmZpM2kvZ0N1dDBaV3RBeU8zTVZlNXFXRi9lbkt3Z1BFUwpZOXBvK1RkQ3ZS
Qi9SVU9iQmFNNzYxRWNyTFNNMUdxSE51ZVNmcW5obzNBakxRNmRCblBXbG82MzhabTFWZWJ
LCkNFTHloa0xXTVNGa0t3RG1uZTBqUTAyWTRnMDc1dkNLdkNzQ0F3RUFBYU5qTUdFd0hRWUR
WUjBPQkJZRUZIN1IKNXlDK1VlaElJUGV1TDhacXczUHpiZ2NaTUI4R0ExVWRJd1FZTUJhQUZIN1I0
eUMrVWVoSUlQZXVMOFpxdzNQegpjZ2NaTUE4R0ExVWRFd0VCL3dRRk1BTUJBZjh3RGdZRFZS
MFBBUUgvQkFRREFnR0dNQTBHQ1NxR1NJYjNEUUVCCkR3VUFBNElCQVFCRE52RDJWbTlzQT
VBOUFsT0pSOCtlbjVYejloWGN4SkI1cGh4Y1pROGpGb0cwNFZzaHZkMGUKTUVuVXJNY2ZGZ0laN
G5qTUtUUUNNNFpGVVBBaWV5THg0ZjUySHVEb3BwM2U1SnlJTWZXK0tGY05JcEt3Q3NhawpwU2
9LdElVT3NVSks3cUJWWnhjckl5ZVFWMnFjWU9lWmh0UzV3QnFJd09BaEZ3bENFVDdaZTU4UUhtUz
Q4c2xqCjVlVGtSaml2QWxFeHJGektjbGpDNGF4S1Fsbk92VkF6eitHbTMyVTB4UEJGNEJ5ZVBWeEN
KVUh3MVRzeVRtZWwKU3hORXA3eUhvWGN3bitmWG5hK3Q1SldoMWd4VVp0eTMKLS0tLS1FTkQ
gQ0VSVElGSUNBVEUtLS0tLQo=
    mode: 0644
    overwrite: true
    path: /etc/pki/ca-trust/source/anchors/examplecorp-ca.crt

The trust store of machines must also support updating the trust store of nodes.

4.2.8. Renewal

There are no Operators that can auto-renew certificates on the RHCOS nodes.

NOTE

Red Hat does not monitor for when CAs expire. However, due to the long life of CAs, this is generally not an issue. However, you might need to periodically update the trust bundle.

4.3. SERVICE CA CERTIFICATES

4.3.1. Purpose

service-ca is an Operator that creates a self-signed CA when an OpenShift Container Platform cluster is deployed.

4.3.2. Expiration

A custom expiration term is not supported. The self-signed CA is stored in a secret with qualified name **service-ca/signing-key** in fields **tls.crt** (certificate(s)), **tls.key** (private key), and **ca-bundle.crt** (CA bundle).

Other services can request a service serving certificate by annotating a service resource with **service.beta.openshift.io/serving-cert-secret-name: <secret name>**. In response, the Operator generates a new certificate, as **tls.crt**, and private key, as **tls.key** to the named secret. The certificate is valid for two years.

Other services can request that the CA bundle for the service CA be injected into API service or config map resources by annotating with **service.beta.openshift.io/inject-cabundle: true** to support validating certificates generated from the service CA. In response, the Operator writes its current CA bundle to the **CABundle** field of an API service or as **service-ca.crt** to a config map.

As of OpenShift Container Platform 4.3.5, automated rotation is supported and is backported to some 4.2.z and 4.3.z releases. For any release supporting automated rotation, the service CA is valid for 26 months and is automatically refreshed when there is less than 13 months validity left. If necessary, you can manually refresh the service CA.

The service CA expiration of 26 months is longer than the expected upgrade interval for a supported OpenShift Container Platform cluster, such that non-control plane consumers of service CA certificates will be refreshed after CA rotation and prior to the expiration of the pre-rotation CA.

WARNING

A manually-rotated service CA does not maintain trust with the previous service CA. You might experience a temporary service disruption until the pods in the cluster are restarted, which ensures that pods are using service serving certificates issued by the new service CA.

IMPORTANT

Applications using the **service-ca** certificate must be capable of dynamically reloading CA certificates. Otherwise, when automated rotation occurs, the application pods might require a restart in order to rebuild certificate trust.

4.3.3. Management

These certificates are managed by the system and not the user.

4.3.4. Services

Services that use service CA certificates include:

- cluster-autoscaler-operator
- cluster-monitoring-operator
- cluster-authentication-operator
- cluster-image-registry-operator
- cluster-ingress-operator
- cluster-kube-apiserver-operator
- cluster-kube-controller-manager-operator
- cluster-kube-scheduler-operator

- cluster-networking-operator
- cluster-openshift-apiserver-operator
- cluster-openshift-controller-manager-operator
- cluster-samples-operator
- machine-config-operator
- console-operator
- insights-operator
- machine-api-operator
- operator-lifecycle-manager

This is not a comprehensive list.

Additional resources

- Manually rotate service serving certificates
- Securing service traffic using service serving certificate secrets

4.4. NODE CERTIFICATES

4.4.1. Purpose

Node certificates are signed by the cluster and allow the kubelet to communicate with the Kubernetes API server. They come from the kubelet CA certificate, which is generated by the bootstrap process.

4.4.2. Location

The kubelet CA certificate is located in the **kube-apiserver-to-kubelet-signer** secret in the **openshift-kube-apiserver-operator** namespace.

4.4.3. Management

These certificates are managed by the system and not the user.

4.4.4. Expiration

Node certificates are automatically rotated after 30 days.

4.4.5. Renewal

The Kubernetes API Server Operator automatically generates a new **kube-apiserver-to-kubelet-signer** CA certificate at 292 days. The old CA certificate is removed after 365 days. Nodes are not rebooted when a kubelet CA certificate is renewed or removed.

Cluster administrators can manually renew the kubelet CA certificate by running the following command:

```
$ oc annotate -n openshift-kube-apiserver-operator secret kube-apiserver-to-kubelet-signer
auth.openshift.io/certificate-not-after-
```

Additional resources

- Working with nodes

4.5. BOOTSTRAP CERTIFICATES

4.5.1. Purpose

The kubelet, in OpenShift Container Platform 4 and later, uses the bootstrap certificate located in **/etc/kubernetes/kubeconfig** to initially bootstrap. This is followed by the bootstrap initialization process and authorization of the kubelet to create a CSR .

In that process, the kubelet generates a CSR while communicating over the bootstrap channel. The controller manager signs the CSR, resulting in a certificate that the kubelet manages.

4.5.2. Management

These certificates are managed by the system and not the user.

4.5.3. Expiration

This bootstrap certificate is valid for 10 years.

The kubelet-managed certificate is valid for one year and rotates automatically at around the 80 percent mark of that one year.

NOTE

OpenShift Lifecycle Manager (OLM) does not update the bootstrap certificate.

4.5.4. Customization

You cannot customize the bootstrap certificates.

4.6. ETCD CERTIFICATES

4.6.1. Purpose

etcd certificates are signed by the etcd-signer; they come from a certificate authority (CA) that is generated by the bootstrap process.

4.6.2. Expiration

The CA certificates are valid for 10 years. The peer, client, and server certificates are valid for three years.

4.6.3. Management

These certificates are only managed by the system and are automatically rotated.

4.6.4. Services

etcd certificates are used for encrypted communication between etcd member peers, as well as encrypted client traffic. The following certificates are generated and used by etcd and other processes that communicate with etcd:

- Peer certificates: Used for communication between etcd members.
- Client certificates: Used for encrypted server-client communication. Client certificates are currently used by the API server only, and no other service should connect to etcd directly except for the proxy. Client secrets (**etcd-client**, **etcd-metric-client**, **etcd-metric-signer**, and **etcd-signer**) are added to the **openshift-config**, **openshift-monitoring**, and **openshift-kube-apiserver** namespaces.
- Server certificates: Used by the etcd server for authenticating client requests.
- Metric certificates: All metric consumers connect to proxy with metric-client certificates.

Additional resources

- Restoring to a previous cluster state

4.7. OLM CERTIFICATES

4.7.1. Management

All certificates for Operator Lifecycle Manager (OLM) components (**olm-operator**, **catalog-operator**, **packageserver**, and **marketplace-operator**) are managed by the system.

When installing Operators that include webhooks or API services in their **ClusterServiceVersion** (CSV) object, OLM creates and rotates the certificates for these resources. Certificates for resources in the **openshift-operator-lifecycle-manager** namespace are managed by OLM.

OLM will not update the certificates of Operators that it manages in proxy environments. These certificates must be managed by the user using the subscription config.

Next steps

- Configuring proxy support in Operator Lifecycle Manager

4.7.2. Additional resources

- Proxy certificates
- Replacing the default ingress certificate
- Updating the CA bundle

4.8. AGGREGATED API CLIENT CERTIFICATES

4.8.1. Purpose

Aggregated API client certificates are used to authenticate the KubeAPIServer when connecting to the Aggregated API Servers.

4.8.2. Management

These certificates are managed by the system and not the user.

4.8.3. Expiration

This CA is valid for 30 days.

The managed client certificates are valid for 30 days.

CA and client certificates are rotated automatically through the use of controllers.

4.8.4. Customization

You cannot customize the aggregated API server certificates.

4.9. MACHINE CONFIG OPERATOR CERTIFICATES

4.9.1. Purpose

This certificate authority is used to secure connections from nodes to Machine Config Server (MCS) during initial provisioning.

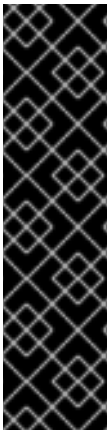
There are two certificates:

- A self-signed CA, the MCS CA
- A derived certificate, the MCS cert

4.9.1.1. Provisioning details

OpenShift Container Platform installations that use Red Hat Enterprise Linux CoreOS (RHCOS) are installed by using Ignition. This process is split into two parts:

1. An Ignition config is created that references a URL for the full configuration served by the MCS.
2. For user-provisioned infrastucture installation methods, the Ignition config manifests as a **worker.ign** file created by the **openshift-install** command. For installer-provisioned infrastructure installation methods that use the Machine API Operator, this configuration appears as the **worker-user-data** secret.



IMPORTANT

Currently, there is no supported way to block or restrict the machine config server endpoint. The machine config server must be exposed to the network so that newly-provisioned machines, which have no existing configuration or state, are able to fetch their configuration. In this model, the root of trust is the certificate signing requests (CSR) endpoint, which is where the kubelet sends its certificate signing request for approval to join the cluster. Because of this, machine configs should not be used to distribute sensitive information, such as secrets and certificates.

To ensure that the machine config server endpoints, ports 22623 and 22624, are secured in bare metal scenarios, customers must configure proper network policies.

Additional resources

- Understanding the Machine Config Operator .
- About the OpenShift SDN network plugin .

4.9.1.2. Provisioning chain of trust

The MCS CA is injected into the Ignition configuration under the **security.tls.certificateAuthorities** configuration field. The MCS then provides the complete configuration using the MCS cert presented by the web server.

The client validates that the MCS cert presented by the server has a chain of trust to an authority it recognizes. In this case, the MCS CA is that authority, and it signs the MCS cert. This ensures that the client is accessing the correct server. The client in this case is Ignition running on a machine in the initramfs.

4.9.1.3. Key material inside a cluster

The MCS CA appears in the cluster as a config map in the **kube-system** namespace, **root-ca** object, with **ca.crt** key. The private key is not stored in the cluster and is discarded after the installation completes.

The MCS cert appears in the cluster as a secret in the **openshift-machine-config-operator** namespace and **machine-config-server-tls** object with the **tls.crt** and **tls.key** keys.

4.9.2. Management

At this time, directly modifying either of these certificates is not supported.

4.9.3. Expiration

The MCS CA is valid for 10 years.

The issued serving certificates are valid for 10 years.

4.9.4. Customization

You cannot customize the Machine Config Operator certificates.

4.10. USER-PROVIDED CERTIFICATES FOR DEFAULT INGRESS

4.10.1. Purpose

Applications are usually exposed at **<route_name>.apps.<cluster_name>.<base_domain>**. The **<cluster_name>** and **<base_domain>** come from the installation config file. **<route_name>** is the host field of the route, if specified, or the route name. For example, **hello-openshift-default.apps.username.devcluster.openshift.com**. **hello-openshift** is the name of the route and the route is in the default namespace. You might want clients to access the applications without the need to distribute the cluster-managed CA certificates to the clients. The administrator must set a custom default certificate when serving application content.

WARNING

The Ingress Operator generates a default certificate for an Ingress Controller to serve as a placeholder until you configure a custom default certificate. Do not use operator-generated default certificates in production clusters.

4.10.2. Location

The user-provided certificates must be provided in a **tls** type **Secret** resource in the **openshift-ingress** namespace. Update the **IngressController** CR in the **openshift-ingress-operator** namespace to enable the use of the user-provided certificate. For more information on this process, see Setting a custom default certificate.

4.10.3. Management

User-provided certificates are managed by the user.

4.10.4. Expiration

User-provided certificates are managed by the user.

4.10.5. Services

Applications deployed on the cluster use user-provided certificates for default ingress.

4.10.6. Customization

Update the secret containing the user-managed certificate as needed.

Additional resources

- Replacing the default ingress certificate

4.11. INGRESS CERTIFICATES

4.11.1. Purpose

The Ingress Operator uses certificates for:

- Securing access to metrics for Prometheus.
- Securing access to routes.

4.11.2. Location

To secure access to Ingress Operator and Ingress Controller metrics, the Ingress Operator uses service serving certificates. The Operator requests a certificate from the **service-ca** controller for its own metrics, and the **service-ca** controller puts the certificate in a secret named **metrics-tls** in the **openshift-ingress-operator** namespace. Additionally, the Ingress Operator requests a certificate for each Ingress Controller, and the **service-ca** controller puts the certificate in a secret named **router-metrics-certs-<name>**, where **<name>** is the name of the Ingress Controller, in the **openshift-ingress** namespace.

Each Ingress Controller has a default certificate that it uses for secured routes that do not specify their own certificates. Unless you specify a custom certificate, the Operator uses a self-signed certificate by default. The Operator uses its own self-signed signing certificate to sign any default certificate that it generates. The Operator generates this signing certificate and puts it in a secret named **router-ca** in the **openshift-ingress-operator** namespace. When the Operator generates a default certificate, it puts the default certificate in a secret named **router-certs-<name>** (where **<name>** is the name of the Ingress Controller) in the **openshift-ingress** namespace.

WARNING

The Ingress Operator generates a default certificate for an Ingress Controller to serve as a placeholder until you configure a custom default certificate. Do not use Operator-generated default certificates in production clusters.

4.11.3. Workflow

Figure 4.1. Custom certificate workflow

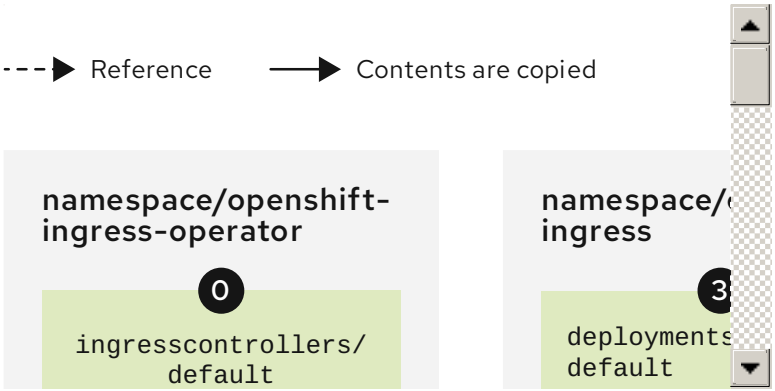
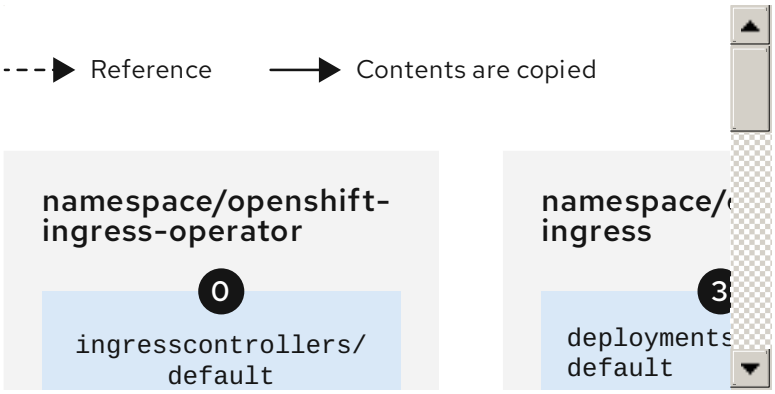


Figure 4.2. Default certificate workflow



0 An empty **defaultCertificate** field causes the Ingress Operator to use its self-signed CA to generate a serving certificate for the specified domain.

1 The default CA certificate and key generated by the Ingress Operator. Used to sign Operator-generated default serving certificates.

2 In the default workflow, the wildcard default serving certificate, created by the Ingress Operator and signed using the generated default CA certificate. In the custom workflow, this is the user-provided certificate.

3 The router deployment. Uses the certificate in **secrets/router-certs-default** as its default front-end server certificate.

4 In the default workflow, the contents of the wildcard default serving certificate (public and private parts) are copied here to enable OAuth integration. In the custom workflow, this is the user-provided certificate.

5 The public (certificate) part of the default serving certificate. Replaces the **configmaps/router-ca** resource.

6 The user updates the cluster proxy configuration with the CA certificate that signed the **ingresscontroller** serving certificate. This enables components like **auth**, **console**, and the registry to trust the serving certificate.

7 The cluster-wide trusted CA bundle containing the combined Red Hat Enterprise Linux CoreOS (RHCOS) and user-provided CA bundles or an RHCOS-only bundle if a user bundle is not provided.

8 The custom CA certificate bundle, which instructs other components (for example, **auth** and **console**) to trust an **ingresscontroller** configured with a custom certificate.

9 The **trustedCA** field is used to reference the user-provided CA bundle.

10 The Cluster Network Operator injects the trusted CA bundle into the **proxy-ca** config map.

11 OpenShift Container Platform 4.13 and newer use **default-ingress-cert**.

4.11.4. Expiration

The expiration terms for the Ingress Operator's certificates are as follows:

- The expiration date for metrics certificates that the **service-ca** controller creates is two years after the date of creation.
- The expiration date for the Operator's signing certificate is two years after the date of creation.
- The expiration date for default certificates that the Operator generates is two years after the date of creation.

You cannot specify custom expiration terms on certificates that the Ingress Operator or **service-ca** controller creates.

You cannot specify expiration terms when installing OpenShift Container Platform for certificates that the Ingress Operator or **service-ca** controller creates.

4.11.5. Services

Prometheus uses the certificates that secure metrics.

The Ingress Operator uses its signing certificate to sign default certificates that it generates for Ingress Controllers for which you do not set custom default certificates.

Cluster components that use secured routes may use the default Ingress Controller's default certificate.

Ingress to the cluster via a secured route uses the default certificate of the Ingress Controller by which the route is accessed unless the route specifies its own certificate.

4.11.6. Management

Ingress certificates are managed by the user. See Replacing the default ingress certificate for more information.

4.11.7. Renewal

The **service-ca** controller automatically rotates the certificates that it issues. However, it is possible to use **oc delete secret <secret>** to manually rotate service serving certificates.

The Ingress Operator does not rotate its own signing certificate or the default certificates that it generates. Operator-generated default certificates are intended as placeholders for custom default certificates that you configure.

4.12. MONITORING AND OPENSHIFT LOGGING OPERATOR COMPONENT CERTIFICATES

4.12.1. Expiration

Monitoring components secure their traffic with service CA certificates. These certificates are valid for 2 years and are replaced automatically on rotation of the service CA, which is every 13 months.

If the certificate lives in the **openshift-monitoring** or **openshift-logging** namespace, it is system managed and rotated automatically.

4.12.2. Management

These certificates are managed by the system and not the user.

4.13. CONTROL PLANE CERTIFICATES

4.13.1. Location

Control plane certificates are included in these namespaces:

- openshift-config-managed
- openshift-kube-apiserver
- openshift-kube-apiserver-operator
- openshift-kube-controller-manager
- openshift-kube-controller-manager-operator
- openshift-kube-scheduler

4.13.2. Management

Control plane certificates are managed by the system and rotated automatically.

In the rare case that your control plane certificates have expired, see Recovering from expired control plane certificates.

CHAPTER 5. COMPLIANCE OPERATOR

5.1. COMPLIANCE OPERATOR OVERVIEW

The OpenShift Container Platform Compliance Operator assists users by automating the inspection of numerous technical implementations and compares those against certain aspects of industry standards, benchmarks, and baselines; the Compliance Operator is not an auditor. In order to be compliant or certified under these various standards, you need to engage an authorized auditor such as a Qualified Security Assessor (QSA), Joint Authorization Board (JAB), or other industry recognized regulatory authority to assess your environment.

The Compliance Operator makes recommendations based on generally available information and practices regarding such standards and may assist with remediations, but actual compliance is your responsibility. You are required to work with an authorized auditor to achieve compliance with a standard. For the latest updates, see the Compliance Operator release notes. For more information on compliance support for all Red Hat products, see Product Compliance.

5.1.1. Compliance Operator concepts

Understanding the Compliance Operator

Understanding the Custom Resource Definitions

5.1.2. Compliance Operator management

Installing the Compliance Operator

Updating the Compliance Operator

Managing the Compliance Operator

Uninstalling the Compliance Operator

5.1.3. Compliance Operator scan management

Supported compliance profiles

Compliance Operator scans

Tailoring the Compliance Operator

Retrieving Compliance Operator raw results

Managing Compliance Operator remediation

Performing advanced Compliance Operator tasks

Troubleshooting the Compliance Operator

Using the oc-compliance plugin

5.2. COMPLIANCE OPERATOR RELEASE NOTES

The Compliance Operator lets OpenShift Container Platform administrators describe the required compliance state of a cluster and provides them with an overview of gaps and ways to remediate them.

These release notes track the development of the Compliance Operator in the OpenShift Container Platform.

For an overview of the Compliance Operator, see Understanding the Compliance Operator.

To access the latest release, see Updating the Compliance Operator.

For more information on compliance support for all Red Hat products, see Product Compliance.

5.2.1. OpenShift Compliance Operator 1.6.2

The following advisory is available for the OpenShift Compliance Operator 1.6.2:

- RHBA-2025:2659 - OpenShift Compliance Operator 1.6.2 update

CVE-2024-45338 is resolved in the Compliance Operator 1.6.2 release. (**CVE-2024-45338**)

5.2.2. OpenShift Compliance Operator 1.6.1

The following advisory is available for the OpenShift Compliance Operator 1.6.1:

- RHBA-2024:10367 - OpenShift Compliance Operator 1.6.1 update

This update includes upgraded dependencies in underlying base images.

5.2.3. OpenShift Compliance Operator 1.6.0

The following advisory is available for the OpenShift Compliance Operator 1.6.0:

- RHBA-2024:6761 - OpenShift Compliance Operator 1.6.0 bug fix and enhancement update

5.2.3.1. New features and enhancements

- The Compliance Operator now contains supported profiles for Payment Card Industry Data Security Standard (PCI-DSS) version 4. For more information, see Supported compliance profiles.
- The Compliance Operator now contains supported profiles for Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) V2R1. For more information, see Supported compliance profiles.
- A **must-gather** extension is now available for the Compliance Operator installed on **x86**, **ppc64le**, and **s390x** architectures. The **must-gather** tool provides crucial configuration details to Red Hat Customer Support and engineering. For more information, see Using the must-gather tool for the Compliance Operator.

5.2.3.2. Bug fixes

- Before this release, a misleading description in the **ocp4-route-ip-whitelist** rule resulted in misunderstanding, causing potential for misconfigurations. With this update, the rule is now more clearly defined. (**CMP-2485**)

- Previously, the reporting of all of the **ComplianceCheckResults** for a **DONE** status **ComplianceScan** was incomplete. With this update, annotation has been added to report the number of total **ComplianceCheckResults** for a **ComplianceScan** with a **DONE** status. (CMP-2615)

- Previously, the **ocp4-cis-scc-limit-container-allowed-capabilities** rule description contained ambiguous guidelines, leading to confusion among users. With this update, the rule description and actionable steps are clarified. (OCPBUGS-17828)

- Before this update, sysctl configurations caused certain auto remediations for RHCOS4 rules to fail scans in affected clusters. With this update, the correct sysctl settings are applied and RHCOS4 rules for FedRAMP High profiles pass scans correctly. (OCPBUGS-19690)

- Before this update, an issue with a **jq** filter caused errors with the **rhacs-operator-controller-manager** deployment during compliance checks. With this update, the **jq** filter expression is updated and the **rhacs-operator-controller-manager** deployment is exempt from compliance checks pertaining to container resource limits, eliminating false positive results. (OCPBUGS-19690)

- Before this update, **rhcos4-high** and **rhcos4-moderate** profiles checked values of an incorrectly titled configuration file. As a result, some scan checks could fail. With this update, the **rhcos4** profiles now check the correct configuration file and scans pass correctly. (OCPBUGS-31674)

- Previously, the **accessokenInactivityTimeoutSeconds** variable used in the **oauthclient-inactivity-timeout** rule was immutable, leading to a **FAIL** status when performing DISA STIG scans. With this update, proper enforcement of the **accessTokenInactivityTimeoutSeconds** variable operates correctly and a **PASS** status is now possible. (OCPBUGS-32551)

- Before this update, some annotations for rules were not updated, displaying the incorrect control standards. With this update, annotations for rules are updated correctly, ensuring the correct control standards are displayed. (OCPBUGS-34982)

- Previously, when upgrading to Compliance Operator 1.5.1, an incorrectly referenced secret in a **ServiceMonitor** configuration caused integration issues with the Prometheus Operator. With this update, the Compliance Operator will accurately reference the secret containing the token for **ServiceMonitor** metrics. (OCPBUGS-39417)

5.2.4. OpenShift Compliance Operator 1.5.1

The following advisory is available for the OpenShift Compliance Operator 1.5.1:

- RHBA-2024:5956 - OpenShift Compliance Operator 1.5.1 bug fix and enhancement update

5.2.5. OpenShift Compliance Operator 1.5.0

The following advisory is available for the OpenShift Compliance Operator 1.5.0:

- RHBA-2024:3533 - OpenShift Compliance Operator 1.5.0 bug fix and enhancement update

5.2.5.1. New features and enhancements

- With this update, the Compliance Operator provides a unique profile ID for easier programmatic use. (CMP-2450)

- With this release, the Compliance Operator is now tested and supported on the ROSA HCP environment. The Compliance Operator loads only Node profiles when running on ROSA HCP. This is because a Red Hat managed platform restricts access to the control plane, which makes Platform profiles irrelevant to the operator's function.(CMP-2581)

5.2.5.2. Bug fixes

- CVE-2024-2961 is resolved in the Compliance Operator 1.5.0 release. (CVE-2024-2961)
- Previously, for ROSA HCP systems, profile listings were incorrect. This update allows the Compliance Operator to provide correct profile output. (OCPBUGS-34535)
- With this release, namespaces can be excluded from the **ocp4-configure-network-policies-namespaces** check by setting the **ocp4-var-network-policies-namespaces-exempt-regex** variable in the tailored profile. (CMP-2543)

5.2.6. OpenShift Compliance Operator 1.4.1

The following advisory is available for the OpenShift Compliance Operator 1.4.1:

- RHBA-2024:1830 - OpenShift Compliance Operator bug fix and enhancement update

5.2.6.1. New features and enhancements

- As of this release, the Compliance Operator now provides the CIS OpenShift 1.5.0 profile rules. (CMP-2447)
- With this update, the Compliance Operator now provides **OCP4 STIG ID** and **SRG** with the profile rules. (CMP-2401)
- With this update, obsolete rules being applied to **s390x** have been removed. (CMP-2471)

5.2.6.2. Bug fixes

- Previously, for Red Hat Enterprise Linux CoreOS (RHCOS) systems using Red Hat Enterprise Linux (RHEL) 9, application of the **ocp4-kubelet-enable-protect-kernel-sysctl-file-exist** rule failed. This update replaces the rule with **ocp4-kubelet-enable-protect-kernel-sysctl**. Now, after auto remediation is applied, RHEL 9-based RHCOS systems will show **PASS** upon the application of this rule. (OCPBUGS-13589)
- Previously, after applying compliance remediations using profile **rhcos4-e8**, the nodes were no longer accessible using SSH to the core user account. With this update, nodes remain accessible through SSH using the `sshkey1 option. (OCPBUGS-18331)
- Previously, the **STIG** profile was missing rules from CaC that fulfill requirements on the published **STIG** for OpenShift Container Platform. With this update, upon remediation, the cluster satisfies **STIG** requirements that can be remediated using Compliance Operator. (OCPBUGS-26193)
- Previously, creating a **ScanSettingBinding** object with profiles of different types for multiple products bypassed a restriction against multiple products types in a binding. With this update, the product validation now allows multiple products regardless of the of profile types in the **ScanSettingBinding** object. (OCPBUGS-26229)

- Previously, running the **rhcos4-service-debug-shell-disabled** rule showed as **FAIL** even after auto-remediation was applied. With this update, running the **rhcos4-service-debug-shell-disabled** rule now shows **PASS** after auto-remediation is applied. (OCPBUGS-28242)

- With this update, instructions for the use of the **rhcos4-banner-etc-issue** rule are enhanced to provide more detail. (OCPBUGS-28797)

- Previously the **api_server_api_priority_flowschema_catch_all** rule provided **FAIL** status on OpenShift Container Platform 4.16 clusters. With this update, the **api_server_api_priority_flowschema_catch_all** rule provides **PASS** status on OpenShift Container Platform 4.16 clusters. (OCPBUGS-28918)

- Previously, when a profile was removed from a completed scan shown in a **ScanSettingBinding** (SSB) object, the Compliance Operator did not remove the old scan. Afterward, when launching a new SSB using the deleted profile, the Compliance Operator failed to update the result. With this release of the Compliance Operator, the new SSB now shows the new compliance check result. (OCPBUGS-29272)

- Previously, on **ppc64le** architecture, the metrics service was not created. With this update, when deploying the Compliance Operator v1.4.1 on **ppc64le** architecture, the metrics service is now created correctly. (OCPBUGS-32797)

- Previously, on a HyperShift hosted cluster, a scan with the **ocp4-pci-dss profile** will run into an unrecoverable error due to a **filter cannot iterate** issue. With this release, the scan for the **ocp4-pci-dss** profile will reach **done** status and return either a **Compliance** or **Non-Compliance** test result. (OCPBUGS-33067)

5.2.7. OpenShift Compliance Operator 1.4.0

The following advisory is available for the OpenShift Compliance Operator 1.4.0:

- RHBA-2023:7658 - OpenShift Compliance Operator bug fix and enhancement update

5.2.7.1. New features and enhancements

- With this update, clusters which use custom node pools outside the default **worker** and **master** node pools no longer need to supply additional variables to ensure Compliance Operator aggregates the configuration file for that node pool.

- Users can now pause scan schedules by setting the **ScanSetting.suspend** attribute to **True**. This allows users to suspend a scan schedule and reactivate it without the need to delete and re-create the **ScanSettingBinding**. This simplifies pausing scan schedules during maintenance periods. (CMP-2123)

- Compliance Operator now supports an optional **version** attribute on **Profile** custom resources. (CMP-2125)

- Compliance Operator now supports profile names in **ComplianceRules**. (CMP-2126)

- Compliance Operator compatibility with improved **cronjob** API improvements is available in this release. (CMP-2310)

5.2.7.2. Bug fixes

- Previously, on a cluster with Windows nodes, some rules will FAIL after auto remediation is applied because the Windows nodes were not skipped by the compliance scan. With this release, Windows nodes are correctly skipped when scanning. (OCPBUGS-7355)
- With this update, **rprivate** default mount propagation is now handled correctly for root volume mounts of pods that rely on multipathing. (OCPBUGS-17494)
- Previously, the Compliance Operator would generate a remediation for **coreos_vsyscall_kernel_argument** without reconciling the rule even while applying the remediation. With release 1.4.0, the **coreos_vsyscall_kernel_argument** rule properly evaluates kernel arguments and generates an appropriate remediation.(OCPBUGS-8041)
- Before this update, rule **rhcos4-audit-rules-login-events-faillock** would fail even after auto-remediation has been applied. With this update, **rhcos4-audit-rules-login-events-faillock** failure locks are now applied correctly after auto-remediation. (OCPBUGS-24594)
- Previously, upgrades from Compliance Operator 1.3.1 to Compliance Operator 1.4.0 would cause OVS rules scan results to go from **PASS** to **NOT-APPLICABLE**. With this update, OVS rules scan results now show **PASS** (OCPBUGS-25323)

5.2.8. OpenShift Compliance Operator 1.3.1

The following advisory is available for the OpenShift Compliance Operator 1.3.1:

- RHBA-2023:5669 - OpenShift Compliance Operator bug fix and enhancement update

This update addresses a CVE in an underlying dependency.

IMPORTANT

It is recommended to update the Compliance Operator to version 1.3.1 or later before updating your OpenShift Container Platform cluster to version 4.14 or later.

5.2.8.1. New features and enhancements

- You can install and use the Compliance Operator in an OpenShift Container Platform cluster running in FIPS mode.

IMPORTANT

To enable FIPS mode for your cluster, you must run the installation program from a RHEL computer configured to operate in FIPS mode. For more information about configuring FIPS mode on RHEL, see Installing the system in FIPS mode .

5.2.8.2. Known issue

- On a cluster with Windows nodes, some rules will FAIL after auto remediation is applied because the Windows nodes are not skipped by the compliance scan. This differs from the expected results because the Windows nodes must be skipped when scanning. (OCPBUGS-7355)

5.2.9. OpenShift Compliance Operator 1.3.0

The following advisory is available for the OpenShift Compliance Operator 1.3.0:

- RHBA-2023:5102 - OpenShift Compliance Operator enhancement update

5.2.9.1. New features and enhancements

- The Defense Information Systems Agency Security Technical Implementation Guide (DISA-STIG) for OpenShift Container Platform is now available from Compliance Operator 1.3.0. See Supported compliance profiles for additional information.
- Compliance Operator 1.3.0 now supports IBM Power and IBM Z for NIST 800-53 Moderate-Impact Baseline for OpenShift Container Platform platform and node profiles.

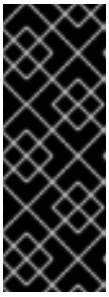
5.2.10. OpenShift Compliance Operator 1.2.0

The following advisory is available for the OpenShift Compliance Operator 1.2.0:

- RHBA-2023:4245 - OpenShift Compliance Operator enhancement update

5.2.10.1. New features and enhancements

- The CIS OpenShift Container Platform 4 Benchmark v1.4.0 profile is now available for platform and node applications. To locate the CIS OpenShift Container Platform v4 Benchmark, go to CIS Benchmarks and click **Download Latest CIS Benchmark**, where you can then register to download the benchmark.



IMPORTANT

Upgrading to Compliance Operator 1.2.0 will overwrite the CIS OpenShift Container Platform 4 Benchmark 1.1.0 profiles.

If your OpenShift Container Platform environment contains existing **cis** and **cis-node** remediations, there might be some differences in scan results after upgrading to Compliance Operator 1.2.0.

- Additional clarity for auditing security context constraints (SCCs) is now available for the **scc-limit-container-allowed-capabilities** rule.

5.2.10.2. Known issues

- When using the CIS OpenShift Container Platform 4 Benchmark v1.4.0 profile, some controls might fail due to tighter permissions in the CIS profile than in OpenShift Container Platform. For more information, see Solution article #7024725.

5.2.11. OpenShift Compliance Operator 1.1.0

The following advisory is available for the OpenShift Compliance Operator 1.1.0:

- RHBA-2023:3630 - OpenShift Compliance Operator bug fix and enhancement update

5.2.11.1. New features and enhancements

- A start and end timestamp is now available in the **ComplianceScan** custom resource definition (CRD) status.

- The Compliance Operator can now be deployed on Hosted Control Planes using the OperatorHub by creating a **Subscription** file. For more information, see Installing the Compliance Operator on Hosted Control Planes.

5.2.11.2. Bug fixes

- Before this update, some Compliance Operator rule instructions were not present. After this update, instructions are improved for the following rules:
  - **classification_banner**
  - **oauth_login_template_set**
  - **oauth_logout_url_set**
  - **oauth_provider_selection_set**
  - **ocp_allowed_registries**
  - **ocp_allowed_registries_for_import** (OCPBUGS-10473)
- Before this update, check accuracy and rule instructions were unclear. After this update, the check accuracy and instructions are improved for the following **sysctl** rules:
  - **kubelet-enable-protect-kernel-sysctl**
  - **kubelet-enable-protect-kernel-sysctl-kernel-keys-root-maxbytes**
  - **kubelet-enable-protect-kernel-sysctl-kernel-keys-root-maxkeys**
  - **kubelet-enable-protect-kernel-sysctl-kernel-panic**
  - **kubelet-enable-protect-kernel-sysctl-kernel-panic-on-oops**
  - **kubelet-enable-protect-kernel-sysctl-vm-overcommit-memory**
  - **kubelet-enable-protect-kernel-sysctl-vm-panic-on-oom** (OCPBUGS-11334)
- Before this update, the **ocp4-alert-receiver-configured** rule did not include instructions. With this update, the **ocp4-alert-receiver-configured** rule now includes improved instructions. (OCPBUGS-7307)
- Before this update, the **rhcos4-sshd-set-loglevel-info** rule would fail for the **rhcos4-e8** profile. With this update, the remediation for the **sshd-set-loglevel-info** rule was updated to apply the correct configuration changes, allowing subsequent scans to pass after the remediation is applied. (OCPBUGS-7816)
- Before this update, a new installation of OpenShift Container Platform with the latest Compliance Operator install failed on the **scheduler-no-bind-address** rule. With this update, the **scheduler-no-bind-address** rule has been disabled on newer versions of OpenShift Container Platform since the parameter was removed. (OCPBUGS-8347)

5.2.12. OpenShift Compliance Operator 1.0.0

The following advisory is available for the OpenShift Compliance Operator 1.0.0:

- RHBA-2023:1682 - OpenShift Compliance Operator bug fix update

5.2.12.1. New features and enhancements

- The Compliance Operator is now stable and the release channel is upgraded to **stable**. Future releases will follow Semantic Versioning. To access the latest release, see Updating the Compliance Operator.

5.2.12.2. Bug fixes

- Before this update, the compliance_operator_compliance_scan_error_total metric had an ERROR label with a different value for each error message. With this update, the compliance_operator_compliance_scan_error_total metric does not increase in values. (OCPBUGS-1803)

- Before this update, the **ocp4-api-server-audit-log-maxsize** rule would result in a **FAIL** state. With this update, the error message has been removed from the metric, decreasing the cardinality of the metric in line with best practices. (OCPBUGS-7520)

- Before this update, the **rhcos4-enable-fips-mode** rule description was misleading that FIPS could be enabled after installation. With this update, the **rhcos4-enable-fips-mode** rule description clarifies that FIPS must be enabled at install time. (OCPBUGS-8358)

5.2.13. OpenShift Compliance Operator 0.1.61

The following advisory is available for the OpenShift Compliance Operator 0.1.61:

- RHBA-2023:0557 - OpenShift Compliance Operator bug fix update

5.2.13.1. New features and enhancements

- The Compliance Operator now supports timeout configuration for Scanner Pods. The timeout is specified in the **ScanSetting** object. If the scan is not completed within the timeout, the scan retries until the maximum number of retries is reached. See Configuring ScanSetting timeout for more information.

5.2.13.2. Bug fixes

- Before this update, Compliance Operator remediations required variables as inputs. Remediations without variables set were applied cluster-wide and resulted in stuck nodes, even though it appeared the remediation applied correctly. With this update, the Compliance Operator validates if a variable needs to be supplied using a **TailoredProfile** for a remediation. (OCPBUGS-3864)

- Before this update, the instructions for **ocp4-kubelet-configure-tls-cipher-suites** were incomplete, requiring users to refine the query manually. With this update, the query provided in **ocp4-kubelet-configure-tls-cipher-suites** returns the actual results to perform the audit steps. (OCPBUGS-3017)

- Before this update, system reserved parameters were not generated in kubelet configuration files, causing the Compliance Operator to fail to unpause the machine config pool. With this update, the Compliance Operator omits system reserved parameters during machine

configuration pool evaluation. (OCPBUGS-4445)

- Before this update, **ComplianceCheckResult** objects did not have correct descriptions. With this update, the Compliance Operator sources the **ComplianceCheckResult** information from the rule description. (OCPBUGS-4615)
- Before this update, the Compliance Operator did not check for empty kubelet configuration files when parsing machine configurations. As a result, the Compliance Operator would panic and crash. With this update, the Compliance Operator implements improved checking of the kubelet configuration data structure and only continues if it is fully rendered. (OCPBUGS-4621)
- Before this update, the Compliance Operator generated remediations for kubelet evictions based on machine config pool name and a grace period, resulting in multiple remediations for a single eviction rule. With this update, the Compliance Operator applies all remediations for a single rule. (OCPBUGS-4338)
- Before this update, a regression occurred when attempting to create a **ScanSettingBinding** that was using a **TailoredProfile** with a non-default **MachineConfigPool** marked the **ScanSettingBinding** as **Failed**. With this update, functionality is restored and custom **ScanSettingBinding** using a **TailoredProfile** performs correctly. (OCPBUGS-6827)
- Before this update, some kubelet configuration parameters did not have default values. With this update, the following parameters contain default values (OCPBUGS-6708):
  - **ocp4-cis-kubelet-enable-streaming-connections**
  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-imagefs-available**
  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-imagefs-inodesfree**
  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-memory-available**
  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-nodefs-available**
- Before this update, the **selinux_confinement_of_daemons** rule failed running on the kubelet because of the permissions necessary for the kubelet to run. With this update, the **selinux_confinement_of_daemons** rule is disabled. (OCPBUGS-6968)

5.2.14. OpenShift Compliance Operator 0.1.59

The following advisory is available for the OpenShift Compliance Operator 0.1.59:

- RHBA-2022:8538 - OpenShift Compliance Operator bug fix update

5.2.14.1. New features and enhancements

- The Compliance Operator now supports Payment Card Industry Data Security Standard (PCI-DSS) **ocp4-pci-dss** and **ocp4-pci-dss-node** profiles on the **ppc64le** architecture.

5.2.14.2. Bug fixes

- Previously, the Compliance Operator did not support the Payment Card Industry Data Security Standard (PCI DSS) **ocp4-pci-dss** and **ocp4-pci-dss-node** profiles on different architectures such as **ppc64le**. Now, the Compliance Operator supports **ocp4-pci-dss** and **ocp4-pci-dss-node** profiles on the **ppc64le** architecture. (OCPBUGS-3252)

- Previously, after the recent update to version 0.1.57, the **rerunner** service account (SA) was no longer owned by the cluster service version (CSV), which caused the SA to be removed during the Operator upgrade. Now, the CSV owns the **rerunner** SA in 0.1.59, and upgrades from any previous version will not result in a missing SA. (OCPBUGS-3452)

5.2.15. OpenShift Compliance Operator 0.1.57

The following advisory is available for the OpenShift Compliance Operator 0.1.57:

- RHBA-2022:6657 - OpenShift Compliance Operator bug fix update

5.2.15.1. New features and enhancements

- **KubeletConfig** checks changed from **Node** to **Platform** type. **KubeletConfig** checks the default configuration of the **KubeletConfig**. The configuration files are aggregated from all nodes into a single location per node pool. See Evaluating **KubeletConfig** rules against default configuration values.

- The **ScanSetting** Custom Resource now allows users to override the default CPU and memory limits of scanner pods through the **scanLimits** attribute. For more information, see Increasing Compliance Operator resource limits.

- A **PriorityClass** object can now be set through **ScanSetting**. This ensures the Compliance Operator is prioritized and minimizes the chance that the cluster falls out of compliance. For more information, see Setting **PriorityClass** for **ScanSetting** scans.

5.2.15.2. Bug fixes

- Previously, the Compliance Operator hard-coded notifications to the default **openshift-compliance** namespace. If the Operator were installed in a non-default namespace, the notifications would not work as expected. Now, notifications work in non-default **openshift-compliance** namespaces. (BZ#2060726)

- Previously, the Compliance Operator was unable to evaluate default configurations used by kubelet objects, resulting in inaccurate results and false positives. This new feature evaluates the kubelet configuration and now reports accurately. (BZ#2075041)

- Previously, the Compliance Operator reported the **ocp4-kubelet-configure-event-creation** rule in a **FAIL** state after applying an automatic remediation because the **eventRecordQPS** value was set higher than the default value. Now, the **ocp4-kubelet-configure-event-creation** rule remediation sets the default value, and the rule applies correctly. (BZ#2082416)

- The **ocp4-configure-network-policies** rule requires manual intervention to perform effectively. New descriptive instructions and rule updates increase applicability of the **ocp4-configure-network-policies** rule for clusters using Calico CNIs. (BZ#2091794)

- Previously, the Compliance Operator would not clean up pods used to scan infrastructure when using the **debug=true** option in the scan settings. This caused pods to be left on the cluster even after deleting the **ScanSettingBinding**. Now, pods are always deleted when a **ScanSettingBinding** is deleted.(BZ#2092913)

- Previously, the Compliance Operator used an older version of the **operator-sdk** command that caused alerts about deprecated functionality. Now, an updated version of the **operator-sdk** command is included and there are no more alerts for deprecated functionality. (BZ#2098581)

- Previously, the Compliance Operator would fail to apply remediations if it could not determine the relationship between kubelet and machine configurations. Now, the Compliance Operator has improved handling of the machine configurations and is able to determine if a kubelet configuration is a subset of a machine configuration. (BZ#2102511)

- Previously, the rule for **ocp4-cis-node-master-kubelet-enable-cert-rotation** did not properly describe success criteria. As a result, the requirements for **RotateKubeletClientCertificate** were unclear. Now, the rule for **ocp4-cis-node-master-kubelet-enable-cert-rotation** reports accurately regardless of the configuration present in the kubelet configuration file. (BZ#2105153)

- Previously, the rule for checking idle streaming timeouts did not consider default values, resulting in inaccurate rule reporting. Now, more robust checks ensure increased accuracy in results based on default configuration values. (BZ#2105878)

- Previously, the Compliance Operator would fail to fetch API resources when parsing machine configurations without Ignition specifications, which caused the **api-check-pods** processes to crash loop. Now, the Compliance Operator handles Machine Config Pools that do not have Ignition specifications correctly. (BZ#2117268)

- Previously, rules evaluating the **modprobe** configuration would fail even after applying remediations due to a mismatch in values for the **modprobe** configuration. Now, the same values are used for the **modprobe** configuration in checks and remediations, ensuring consistent results. (BZ#2117747)

5.2.15.3. Deprecations

- Specifying **Install into all namespaces in the cluster**or setting the **WATCH_NAMESPACES** environment variable to **""** no longer affects all namespaces. Any API resources installed in namespaces not specified at the time of Compliance Operator installation is no longer be operational. API resources might require creation in the selected namespace, or the **openshift-compliance** namespace by default. This change improves the Compliance Operator's memory usage.

5.2.16. OpenShift Compliance Operator 0.1.53

The following advisory is available for the OpenShift Compliance Operator 0.1.53:

- RHBA-2022:5537 - OpenShift Compliance Operator bug fix update

5.2.16.1. Bug fixes

- Previously, the **ocp4-kubelet-enable-streaming-connections** rule contained an incorrect variable comparison, resulting in false positive scan results. Now, the Compliance Operator provides accurate scan results when setting **streamingConnectionIdleTimeout**. (BZ#2069891)

- Previously, group ownership for **/etc/openvswitch/conf.db** was incorrect on IBM Z architectures, resulting in **ocp4-cis-node-worker-file-groupowner-ovs-conf-db** check failures. Now, the check is marked **NOT-APPLICABLE** on IBM Z architecture systems. (BZ#2072597)

- Previously, the **ocp4-cis-scc-limit-container-allowed-capabilities** rule reported in a **FAIL** state due to incomplete data regarding the security context constraints (SCC) rules in the deployment. Now, the result is **MANUAL**, which is consistent with other checks that require

human intervention. (BZ#2077916)

- Previously, the following rules failed to account for additional configuration paths for API servers and TLS certificates and keys, resulting in reported failures even if the certificates and keys were set properly:
  - **ocp4-cis-api-server-kubelet-client-cert**
  - **ocp4-cis-api-server-kubelet-client-key**
  - **ocp4-cis-kubelet-configure-tls-cert**
  - **ocp4-cis-kubelet-configure-tls-key**

  Now, the rules report accurately and observe legacy file paths specified in the kubelet configuration file. (BZ#2079813)

- Previously, the **content_rule_oauth_or_oauthclient_inactivity_timeout** rule did not account for a configurable timeout set by the deployment when assessing compliance for timeouts. This resulted in the rule failing even if the timeout was valid. Now, the Compliance Operator uses the **var_oauth_inactivity_timeout** variable to set valid timeout length. (BZ#2081952)

- Previously, the Compliance Operator used administrative permissions on namespaces not labeled appropriately for privileged use, resulting in warning messages regarding pod security-level violations. Now, the Compliance Operator has appropriate namespace labels and permission adjustments to access results without violating permissions. (BZ#2088202)

- Previously, applying auto remediations for **rhcos4-high-master-sysctl-kernel-yama-ptrace-scope** and **rhcos4-sysctl-kernel-core-pattern** resulted in subsequent failures of those rules in scan results, even though they were remediated. Now, the rules report **PASS** accurately, even after remediations are applied.(BZ#2094382)

- Previously, the Compliance Operator would fail in a **CrashLoopBackoff** state because of out-of-memory exceptions. Now, the Compliance Operator is improved to handle large machine configuration data sets in memory and function correctly. (BZ#2094854)

5.2.16.2. Known issue

- When **"debug":true** is set within the **ScanSettingBinding** object, the pods generated by the **ScanSettingBinding** object are not removed when that binding is deleted. As a workaround, run the following command to delete the remaining pods:

  ```
  $ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
  ```

  (BZ#2092913)

5.2.17. OpenShift Compliance Operator 0.1.52

The following advisory is available for the OpenShift Compliance Operator 0.1.52:

- RHBA-2022:4657 - OpenShift Compliance Operator bug fix update

5.2.17.1. New features and enhancements

- The FedRAMP high SCAP profile is now available for use in OpenShift Container Platform environments. For more information, See Supported compliance profiles.

5.2.17.2. Bug fixes

- Previously, the **OpenScap** container would crash due to a mount permission issue in a security environment where **DAC_OVERRIDE** capability is dropped. Now, executable mount permissions are applied to all users. (BZ#2082151)

- Previously, the compliance rule **ocp4-configure-network-policies** could be configured as **MANUAL**. Now, compliance rule **ocp4-configure-network-policies** is set to **AUTOMATIC**. (BZ#2072431)

- Previously, the Cluster Autoscaler would fail to scale down because the Compliance Operator scan pods were never removed after a scan. Now, the pods are removed from each node by default unless explicitly saved for debugging purposes. (BZ#2075029)

- Previously, applying the Compliance Operator to the **KubeletConfig** would result in the node going into a **NotReady** state due to unpausing the Machine Config Pools too early. Now, the Machine Config Pools are unpaused appropriately and the node operates correctly. (BZ#2071854)

- Previously, the Machine Config Operator used **base64** instead of **url-encoded** code in the latest release, causing Compliance Operator remediation to fail. Now, the Compliance Operator checks encoding to handle both **base64** and **url-encoded** Machine Config code and the remediation applies correctly. (BZ#2082431)

5.2.17.3. Known issue

- When **"debug":true** is set within the **ScanSettingBinding** object, the pods generated by the **ScanSettingBinding** object are not removed when that binding is deleted. As a workaround, run the following command to delete the remaining pods:

  ```
  $ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
  ```

  (BZ#2092913)

5.2.18. OpenShift Compliance Operator 0.1.49

The following advisory is available for the OpenShift Compliance Operator 0.1.49:

- RHBA-2022:1148 - OpenShift Compliance Operator bug fix and enhancement update

5.2.18.1. New features and enhancements

- The Compliance Operator is now supported on the following architectures:
  - IBM Power
  - IBM Z
  - IBM LinuxONE

5.2.18.2. Bug fixes

- Previously, the **openshift-compliance** content did not include platform-specific checks for network types. As a result, OVN- and SDN-specific checks would show as **failed** instead of **not-applicable** based on the network configuration. Now, new rules contain platform checks for networking rules, resulting in a more accurate assessment of network-specific checks. (BZ#1994609)

- Previously, the **ocp4-moderate-routes-protected-by-tls** rule incorrectly checked TLS settings that results in the rule failing the check, even if the connection secure SSL/TLS protocol. Now, the check properly evaluates TLS settings that are consistent with the networking guidance and profile recommendations. (BZ#2002695)

- Previously, **ocp-cis-configure-network-policies-namespace** used pagination when requesting namespaces. This caused the rule to fail because the deployments truncated lists of more than 500 namespaces. Now, the entire namespace list is requested, and the rule for checking configured network policies works for deployments with more than 500 namespaces. (BZ#2038909)

- Previously, remediations using the **sshd jinja** macros were hard-coded to specific sshd configurations. As a result, the configurations were inconsistent with the content the rules were checking for and the check would fail. Now, the sshd configuration is parameterized and the rules apply successfully. (BZ#2049141)

- Previously, the **ocp4-cluster-version-operator-verify-integrity** always checked the first entry in the Cluter Version Operator (CVO) history. As a result, the upgrade would fail in situations where subsequent versions of {product-name} would be verified. Now, the compliance check result for **ocp4-cluster-version-operator-verify-integrity** is able to detect verified versions and is accurate with the CVO history. (BZ#2053602)

- Previously, the **ocp4-api-server-no-adm-ctrl-plugins-disabled** rule did not check for a list of empty admission controller plugins. As a result, the rule would always fail, even if all admission plugins were enabled. Now, more robust checking of the **ocp4-api-server-no-adm-ctrl-plugins-disabled** rule accurately passes with all admission controller plugins enabled. (BZ#2058631)

- Previously, scans did not contain platform checks for running against Linux worker nodes. As a result, running scans against worker nodes that were not Linux-based resulted in a never ending scan loop. Now, the scan schedules appropriately based on platform type and labels complete successfully. (BZ#2056911)

5.2.19. OpenShift Compliance Operator 0.1.48

The following advisory is available for the OpenShift Compliance Operator 0.1.48:

- RHBA-2022:0416 - OpenShift Compliance Operator bug fix and enhancement update

5.2.19.1. Bug fixes

- Previously, some rules associated with extended Open Vulnerability and Assessment Language (OVAL) definitions had a **checkType** of **None**. This was because the Compliance Operator was not processing extended OVAL definitions when parsing rules. With this update, content from extended OVAL definitions is parsed so that these rules now have a **checkType** of either **Node** or **Platform**. (BZ#2040282)

- Previously, a manually created **MachineConfig** object for **KubeletConfig** prevented a **KubeletConfig** object from being generated for remediation, leaving the remediation in the **Pending** state. With this release, a **KubeletConfig** object is created by the remediation,

regardless if there is a manually created **MachineConfig** object for **KubeletConfig**. As a result, **KubeletConfig** remediations now work as expected. (BZ#2040401)

5.2.20. OpenShift Compliance Operator 0.1.47

The following advisory is available for the OpenShift Compliance Operator 0.1.47:

- RHBA-2022:0014 - OpenShift Compliance Operator bug fix and enhancement update

5.2.20.1. New features and enhancements

- The Compliance Operator now supports the following compliance benchmarks for the Payment Card Industry Data Security Standard (PCI DSS):
  - ocp4-pci-dss
  - ocp4-pci-dss-node
- Additional rules and remediations for FedRAMP moderate impact level are added to the OCP4-moderate, OCP4-moderate-node, and rhcos4-moderate profiles.
- Remediations for KubeletConfig are now available in node-level profiles.

5.2.20.2. Bug fixes

- Previously, if your cluster was running OpenShift Container Platform 4.6 or earlier, remediations for USBGuard-related rules would fail for the moderate profile. This is because the remediations created by the Compliance Operator were based on an older version of USBGuard that did not support drop-in directories. Now, invalid remediations for USBGuard-related rules are not created for clusters running OpenShift Container Platform 4.6. If your cluster is using OpenShift Container Platform 4.6, you must manually create remediations for USBGuard-related rules.
  Additionally, remediations are created only for rules that satisfy minimum version requirements. (BZ#1965511)
- Previously, when rendering remediations, the compliance operator would check that the remediation was well-formed by using a regular expression that was too strict. As a result, some remediations, such as those that render **sshd_config**, would not pass the regular expression check and therefore, were not created. The regular expression was found to be unnecessary and removed. Remediations now render correctly. (BZ#2033009)

5.2.21. OpenShift Compliance Operator 0.1.44

The following advisory is available for the OpenShift Compliance Operator 0.1.44:

- RHBA-2021:4530 - OpenShift Compliance Operator bug fix and enhancement update

5.2.21.1. New features and enhancements

- In this release, the **strictNodeScan** option is now added to the **ComplianceScan**, **ComplianceSuite** and **ScanSetting** CRs. This option defaults to **true** which matches the previous behavior, where an error occurred if a scan was not able to be scheduled on a node. Setting the option to **false** allows the Compliance Operator to be more permissive about

scheduling scans. Environments with ephemeral nodes can set the **strictNodeScan** value to false, which allows a compliance scan to proceed, even if some of the nodes in the cluster are not available for scheduling.

- You can now customize the node that is used to schedule the result server workload by configuring the **nodeSelector** and **tolerations** attributes of the **ScanSetting** object. These attributes are used to place the **ResultServer** pod, the pod that is used to mount a PV storage volume and store the raw Asset Reporting Format (ARF) results. Previously, the **nodeSelector** and the **tolerations** parameters defaulted to selecting one of the control plane nodes and tolerating the **node-role.kubernetes.io/master taint**. This did not work in environments where control plane nodes are not permitted to mount PVs. This feature provides a way for you to select the node and tolerate a different taint in those environments.

- The Compliance Operator can now remediate **KubeletConfig** objects.

- A comment containing an error message is now added to help content developers differentiate between objects that do not exist in the cluster compared to objects that cannot be fetched.

- Rule objects now contain two new attributes, **checkType** and **description**. These attributes allow you to determine if the rule pertains to a node check or platform check, and also allow you to review what the rule does.

- This enhancement removes the requirement that you have to extend an existing profile to create a tailored profile. This means the **extends** field in the **TailoredProfile** CRD is no longer mandatory. You can now select a list of rule objects to create a tailored profile. Note that you must select whether your profile applies to nodes or the platform by setting the **compliance.openshift.io/product-type:** annotation or by setting the **-node** suffix for the **TailoredProfile** CR.

- In this release, the Compliance Operator is now able to schedule scans on all nodes irrespective of their taints. Previously, the scan pods would only tolerated the **node-role.kubernetes.io/master taint**, meaning that they would either ran on nodes with no taints or only on nodes with the **node-role.kubernetes.io/master** taint. In deployments that use custom taints for their nodes, this resulted in the scans not being scheduled on those nodes. Now, the scan pods tolerate all node taints.

- In this release, the Compliance Operator supports the following North American Electric Reliability Corporation (NERC) security profiles:

  - ocp4-nerc-cip
  - ocp4-nerc-cip-node
  - rhcos4-nerc-cip

- In this release, the Compliance Operator supports the NIST 800-53 Moderate-Impact Baseline for the Red Hat OpenShift - Node level, ocp4-moderate-node, security profile.

5.2.21.2. Templating and variable use

- In this release, the remediation template now allows multi-value variables.

- With this update, the Compliance Operator can change remediations based on variables that are set in the compliance profile. This is useful for remediations that include deployment-specific values such as time outs, NTP server host names, or similar. Additionally, the

ComplianceCheckResult objects now use the label **compliance.openshift.io/check-has-value** that lists the variables a check has used.

5.2.21.3. Bug fixes

- Previously, while performing a scan, an unexpected termination occurred in one of the scanner containers of the pods. In this release, the Compliance Operator uses the latest OpenSCAP version 1.3.5 to avoid a crash.
- Previously, using **autoReplyRemediations** to apply remediations triggered an update of the cluster nodes. This was disruptive if some of the remediations did not include all of the required input variables. Now, if a remediation is missing one or more required input variables, it is assigned a state of **NeedsReview**. If one or more remediations are in a **NeedsReview** state, the machine config pool remains paused, and the remediations are not applied until all of the required variables are set. This helps minimize disruption to the nodes.
- The RBAC Role and Role Binding used for Prometheus metrics are changed to 'ClusterRole' and 'ClusterRoleBinding' to ensure that monitoring works without customization.
- Previously, if an error occurred while parsing a profile, rules or variables objects were removed and deleted from the profile. Now, if an error occurs during parsing, the **profileparser** annotates the object with a temporary annotation that prevents the object from being deleted until after parsing completes. (BZ#1988259)
- Previously, an error occurred if titles or descriptions were missing from a tailored profile. Because the XCCDF standard requires titles and descriptions for tailored profiles, titles and descriptions are now required to be set in **TailoredProfile** CRs.
- Previously, when using tailored profiles, **TailoredProfile** variable values were allowed to be set using only a specific selection set. This restriction is now removed, and **TailoredProfile** variables can be set to any value.

5.2.22. Release Notes for Compliance Operator 0.1.39

The following advisory is available for the OpenShift Compliance Operator 0.1.39:

- RHBA-2021:3214 - OpenShift Compliance Operator bug fix and enhancement update

5.2.22.1. New features and enhancements

- Previously, the Compliance Operator was unable to parse Payment Card Industry Data Security Standard (PCI DSS) references. Now, the Operator can parse compliance content that is provided with PCI DSS profiles.
- Previously, the Compliance Operator was unable to execute rules for AU-5 control in the moderate profile. Now, permission is added to the Operator so that it can read **Prometheusrules.monitoring.coreos.com** objects and run the rules that cover AU-5 control in the moderate profile.

5.2.23. Additional resources

- Understanding the Compliance Operator

5.3. COMPLIANCE OPERATOR SUPPORT

5.3.1. Compliance Operator lifecycle

The Compliance Operator is a "Rolling Stream" Operator, meaning updates are available asynchronously of OpenShift Container Platform releases. For more information, see OpenShift Operator Life Cycles on the Red Hat Customer Portal.

5.3.2. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal . From the Customer Portal, you can:

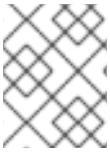
- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in OpenShift Cluster Manager Hybrid Cloud Console. Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a Jira issue for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

5.3.3. Using the must-gather tool for the Compliance Operator

Starting in Compliance Operator v1.6.0, you can collect data about the Compliance Operator resources by running the **must-gather** command with the Compliance Operator image.



NOTE

Consider using the **must-gather** tool when opening support cases or filing bug reports, as it provides additional details about the Operator configuration and logs.

Procedure

- Run the following command to collect data about the Compliance Operator:

```
$ oc adm must-gather --image=$(oc get csv compliance-operator.v1.6.0 -
o=jsonpath='{.spec.relatedImages[?(@.name=="must-gather")].image}')
```

5.3.4. Additional resources

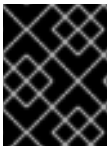
- About the must-gather tool
- Product Compliance

5.4. COMPLIANCE OPERATOR CONCEPTS

5.4.1. Understanding the Compliance Operator

The Compliance Operator lets OpenShift Container Platform administrators describe the required

compliance state of a cluster and provides them with an overview of gaps and ways to remediate them. The Compliance Operator assesses compliance of both the Kubernetes API resources of OpenShift Container Platform, as well as the nodes running the cluster. The Compliance Operator uses OpenSCAP, a NIST-certified tool, to scan and enforce security policies provided by the content.



IMPORTANT

The Compliance Operator is available for Red Hat Enterprise Linux CoreOS (RHCOS) deployments only.

5.4.1.1. Compliance Operator profiles

There are several profiles available as part of the Compliance Operator installation. You can use the **oc get** command to view available profiles, profile details, and specific rules.

- View the available profiles:

```
$ oc get profile.compliance -n openshift-compliance
```

Example output

```
NAME                   AGE     VERSION
ocp4-cis               3h49m   1.5.0
ocp4-cis-1-4           3h49m   1.4.0
ocp4-cis-1-5           3h49m   1.5.0
ocp4-cis-node          3h49m   1.5.0
ocp4-cis-node-1-4      3h49m   1.4.0
ocp4-cis-node-1-5      3h49m   1.5.0
ocp4-e8                3h49m
ocp4-high              3h49m   Revision 4
ocp4-high-node         3h49m   Revision 4
ocp4-high-node-rev-4   3h49m   Revision 4
ocp4-high-rev-4        3h49m   Revision 4
ocp4-moderate          3h49m   Revision 4
ocp4-moderate-node     3h49m   Revision 4
ocp4-moderate-node-rev-4   3h49m   Revision 4
ocp4-moderate-rev-4    3h49m   Revision 4
ocp4-nerc-cip          3h49m
ocp4-nerc-cip-node     3h49m
ocp4-pci-dss           3h49m   3.2.1
ocp4-pci-dss-3-2       3h49m   3.2.1
ocp4-pci-dss-4-0       3h49m   4.0.0
ocp4-pci-dss-node      3h49m   3.2.1
ocp4-pci-dss-node-3-2  3h49m   3.2.1
ocp4-pci-dss-node-4-0  3h49m   4.0.0
ocp4-stig              3h49m   V2R1
ocp4-stig-node         3h49m   V2R1
ocp4-stig-node-v1r1    3h49m   V1R1
ocp4-stig-node-v2r1    3h49m   V2R1
ocp4-stig-v1r1         3h49m   V1R1
ocp4-stig-v2r1         3h49m   V2R1
rhcos4-e8              3h49m
rhcos4-high            3h49m   Revision 4
rhcos4-high-rev-4      3h49m   Revision 4
rhcos4-moderate        3h49m   Revision 4
```

```
rhcos4-moderate-rev-4      3h49m   Revision 4
rhcos4-nerc-cip            3h49m
rhcos4-stig                3h49m   V2R1
rhcos4-stig-v1r1           3h49m   V1R1
rhcos4-stig-v2r1           3h49m   V2R1
```

These profiles represent different compliance benchmarks. Each profile has the product name that it applies to added as a prefix to the profile's name. **ocp4-e8** applies the Essential 8 benchmark to the OpenShift Container Platform product, while **rhcos4-e8** applies the Essential 8 benchmark to the Red Hat Enterprise Linux CoreOS (RHCOS) product.

- Run the following command to view the details of the **rhcos4-e8** profile:

```
$ oc get -n openshift-compliance -oyaml profiles.compliance rhcos4-e8
```

Example 5.1. Example output

```
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.
  A copy of the Essential Eight in Linux Environments guide can be found at the ACSC
  website: https://www.cyber.gov.au/acsc/view-all-content/publications/hardening-linux-
workstations-and-servers'
id: xccdf_org.ssgproject.content_profile_e8
kind: Profile
metadata:
  annotations:
    compliance.openshift.io/image-digest: pb-rhcos4hrdkm
    compliance.openshift.io/product: redhat_enterprise_linux_coreos_4
    compliance.openshift.io/product-type: Node
  creationTimestamp: "2022-10-19T12:06:49Z"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: rhcos4
  name: rhcos4-e8
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: rhcos4
    uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
  resourceVersion: "43699"
  uid: 86353f70-28f7-40b4-bf0e-6289ec33675b
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
- rhcos4-audit-rules-execution-setfiles
- rhcos4-audit-rules-execution-setsebool
- rhcos4-audit-rules-execution-seunshare
```

```
    - rhcos4-audit-rules-kernel-module-loading-delete
    - rhcos4-audit-rules-kernel-module-loading-finit
    - rhcos4-audit-rules-kernel-module-loading-init
    - rhcos4-audit-rules-login-events
    - rhcos4-audit-rules-login-events-faillock
    - rhcos4-audit-rules-login-events-lastlog
    - rhcos4-audit-rules-login-events-tallylog
    - rhcos4-audit-rules-networkconfig-modification
    - rhcos4-audit-rules-sysadmin-actions
    - rhcos4-audit-rules-time-adjtimex
    - rhcos4-audit-rules-time-clock-settime
    - rhcos4-audit-rules-time-settimeofday
    - rhcos4-audit-rules-time-stime
    - rhcos4-audit-rules-time-watch-localtime
    - rhcos4-audit-rules-usergroup-modification
    - rhcos4-auditd-data-retention-flush
    - rhcos4-auditd-freq
    - rhcos4-auditd-local-events
    - rhcos4-auditd-log-format
    - rhcos4-auditd-name-format
    - rhcos4-auditd-write-logs
    - rhcos4-configure-crypto-policy
    - rhcos4-configure-ssh-crypto-policy
    - rhcos4-no-empty-passwords
    - rhcos4-selinux-policytype
    - rhcos4-selinux-state
    - rhcos4-service-auditd-enabled
    - rhcos4-sshd-disable-empty-passwords
    - rhcos4-sshd-disable-gssapi-auth
    - rhcos4-sshd-disable-rhosts
    - rhcos4-sshd-disable-root-login
    - rhcos4-sshd-disable-user-known-hosts
    - rhcos4-sshd-do-not-permit-user-env
    - rhcos4-sshd-enable-strictmodes
    - rhcos4-sshd-print-last-log
    - rhcos4-sshd-set-loglevel-info
    - rhcos4-sysctl-kernel-dmesg-restrict
    - rhcos4-sysctl-kernel-kptr-restrict
    - rhcos4-sysctl-kernel-randomize-va-space
    - rhcos4-sysctl-kernel-unprivileged-bpf-disabled
    - rhcos4-sysctl-kernel-yama-ptrace-scope
    - rhcos4-sysctl-net-core-bpf-jit-harden
  title: Australian Cyber Security Centre (ACSC) Essential Eight
```

- Run the following command to view the details of the **rhcos4-audit-rules-login-events** rule:

```
$ oc get -n openshift-compliance -oyaml rules rhcos4-audit-rules-login-events
```

Example 5.2. Example output

```
apiVersion: compliance.openshift.io/v1alpha1
checkType: Node
description: |-
  The audit system already collects login information for all users and root. If the auditd
```

daemon is configured to use the augenrules program to read audit rules during daemon startup (the default), add the following lines to a file with suffix.rules in the directory /etc/audit/rules.d in order to watch for attempted manual edits of files involved in storing logon events:

```
  -w /var/log/tallylog -p wa -k logins
  -w /var/run/faillock -p wa -k logins
  -w /var/log/lastlog -p wa -k logins
```

If the auditd daemon is configured to use the auditctl utility to read audit rules during daemon startup, add the following lines to /etc/audit/audit.rules file in order to watch for unattempted manual edits of files involved in storing logon events:

```
  -w /var/log/tallylog -p wa -k logins
  -w /var/run/faillock -p wa -k logins
  -w /var/log/lastlog -p wa -k logins
id: xccdf_org.ssgproject.content_rule_audit_rules_login_events
kind: Rule
metadata:
  annotations:
    compliance.openshift.io/image-digest: pb-rhcos4hrdkm
    compliance.openshift.io/rule: audit-rules-login-events
    control.compliance.openshift.io/NIST-800-53: AU-2(d);AU-12(c);AC-6(9);CM-6(a)
    control.compliance.openshift.io/PCI-DSS: Req-10.2.3
    policies.open-cluster-management.io/controls: AU-2(d),AU-12(c),AC-6(9),CM-
6(a),Req-10.2.3
    policies.open-cluster-management.io/standards: NIST-800-53,PCI-DSS
  creationTimestamp: "2022-10-19T12:07:08Z"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: rhcos4
  name: rhcos4-audit-rules-login-events
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: rhcos4
    uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
  resourceVersion: "44819"
  uid: 75872f1f-3c93-40ca-a69d-44e5438824a4
rationale: Manual editing of these files may indicate nefarious activity, such as
  an attacker attempting to remove evidence of an intrusion.
severity: medium
title: Record Attempts to Alter Logon and Logout Events
warning: Manual editing of these files may indicate nefarious activity, such as an
  attacker attempting to remove evidence of an intrusion.
```

5.4.1.1.1. Compliance Operator profile types

Compliance Operator rules are organized into profiles. Profiles can target the Platform or Nodes for OpenShift Container Platform, and some benchmarks include **rhcos4** Node profiles.

Platform

Platform profiles evaluate your OpenShift Container Platform cluster components. For example, a Platform-level rule can confirm whether APIServer configurations are using strong encryption cyphers.

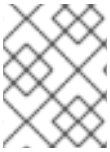
Node

Node profiles evaluate the OpenShift or RHCOS configuration of each host. You can use two Node profiles: **ocp4** Node profiles and **rhcos4** Node profiles. The **ocp4** Node profiles evaluate the OpenShift configuration of each host. For example, they can confirm whether **kubeconfig** files have the correct permissions to meet a compliance standard. The **rhcos4** Node profiles evaluate the Red Hat Enterprise Linux CoreOS (RHCOS) configuration of each host. For example, they can confirm whether the SSHD service is configured to disable password logins.

IMPORTANT

For benchmarks that have Node and Platform profiles, such as PCI-DSS, you must run both profiles in your OpenShift Container Platform environment.

For benchmarks that have **ocp4** Platform, **ocp4** Node, and **rhcos4** Node profiles, such as FedRAMP High, you must run all three profiles in your OpenShift Container Platform environment.



NOTE

In a cluster with many Nodes, both **ocp4** Node and **rhcos4** Node scans might take a long time to complete.

5.4.2. Understanding the Custom Resource Definitions

The Compliance Operator in the OpenShift Container Platform provides you with several Custom Resource Definitions (CRDs) to accomplish the compliance scans. To run a compliance scan, it leverages the predefined security policies, which are derived from the ComplianceAsCode community project. The Compliance Operator converts these security policies into CRDs, which you can use to run compliance scans and get remediations for the issues found.

5.4.2.1. CRDs workflow

The CRD provides you the following workflow to complete the compliance scans:

1. Define your compliance scan requirements
2. Configure the compliance scan settings
3. Process compliance requirements with compliance scans settings
4. Monitor the compliance scans
5. Check the compliance scan results

5.4.2.2. Defining the compliance scan requirements

By default, the Compliance Operator CRDs include **ProfileBundle** and **Profile** objects, in which you can define and set the rules for your compliance scan requirements. You can also customize the default profiles by using a **TailoredProfile** object.

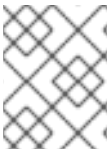
5.4.2.2.1. ProfileBundle object

When you install the Compliance Operator, it includes ready-to-run **ProfileBundle** objects. The Compliance Operator parses the **ProfileBundle** object and creates a **Profile** object for each profile in the bundle. It also parses **Rule** and **Variable** objects, which are used by the **Profile** object.

Example ProfileBundle object

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
  name: <profile bundle name>
  namespace: openshift-compliance
status:
  dataStreamStatus: VALID (1)
```

(1) Indicates whether the Compliance Operator was able to parse the content files.



NOTE

When the **contentFile** fails, an **errorMessage** attribute appears, which provides details of the error that occurred.

Troubleshooting

When you roll back to a known content image from an invalid image, the **ProfileBundle** object stops responding and displays **PENDING** state. As a workaround, you can move to a different image than the previous one. Alternatively, you can delete and re-create the **ProfileBundle** object to return to the working state.

5.4.2.2.2. Profile object

The **Profile** object defines the rules and variables that can be evaluated for a certain compliance standard. It contains parsed out details about an OpenSCAP profile, such as its XCCDF identifier and profile checks for a **Node** or **Platform** type. You can either directly use the **Profile** object or further customize it using a **TailorProfile** object.

NOTE

You cannot create or modify the **Profile** object manually because it is derived from a single **ProfileBundle** object. Typically, a single **ProfileBundle** object can include several **Profile** objects.

Example Profile object

```
apiVersion: compliance.openshift.io/v1alpha1
description: <description of the profile>
id: xccdf_org.ssgproject.content_profile_moderate (1)
kind: Profile
metadata:
  annotations:
    compliance.openshift.io/product: <product name>
```

```
    compliance.openshift.io/product-type: Node 2
  creationTimestamp: "YYYY-MM-DDTMM:HH:SSZ"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: <profile bundle name>
  name: rhcos4-moderate
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: <profile bundle name>
    uid: <uid string>
  resourceVersion: "<version number>"
  selfLink: /apis/compliance.openshift.io/v1alpha1/namespaces/openshift-compliance/profiles/rhcos4-
moderate
  uid: <uid string>
rules: 3
- rhcos4-account-disable-post-pw-expiration
- rhcos4-accounts-no-uid-except-zero
- rhcos4-audit-rules-dac-modification-chmod
- rhcos4-audit-rules-dac-modification-chown
title: <title of the profile>
```

1 Specify the XCCDF name of the profile. Use this identifier when you define a **ComplianceScan** object as the value of the profile attribute of the scan.

2 Specify either a **Node** or **Platform**. Node profiles scan the cluster nodes and platform profiles scan the Kubernetes platform.

3 Specify the list of rules for the profile. Each rule corresponds to a single check.

5.4.2.2.3. Rule object

The **Rule** object, which forms the profiles, are also exposed as objects. Use the **Rule** object to define your compliance check requirements and specify how it could be fixed.

Example Rule object

```
  apiVersion: compliance.openshift.io/v1alpha1
  checkType: Platform 1
  description: <description of the rule>
  id: xccdf_org.ssgproject.content_rule_configure_network_policies_namespaces 2
  instructions: <manual instructions for the scan>
  kind: Rule
  metadata:
    annotations:
      compliance.openshift.io/rule: configure-network-policies-namespaces
      control.compliance.openshift.io/CIS-OCP: 5.3.2
      control.compliance.openshift.io/NERC-CIP: CIP-003-3 R4;CIP-003-3 R4.2;CIP-003-3
        R5;CIP-003-3 R6;CIP-004-3 R2.2.4;CIP-004-3 R3;CIP-007-3 R2;CIP-007-3 R2.1;CIP-007-3
        R2.2;CIP-007-3 R2.3;CIP-007-3 R5.1;CIP-007-3 R6.1
      control.compliance.openshift.io/NIST-800-53: AC-4;AC-4(21);CA-3(5);CM-6;CM-6(1);CM-7;CM-
```

```
7(1);SC-7;SC-7(3);SC-7(5);SC-7(8);SC-7(12);SC-7(13);SC-7(18)
    labels:
      compliance.openshift.io/profile-bundle: ocp4
    name: ocp4-configure-network-policies-namespaces
    namespace: openshift-compliance
  rationale: <description of why this rule is checked>
  severity: high (3)
  title: <summary of the rule>
```

(1) Specify the type of check this rule executes. **Node** profiles scan the cluster nodes and **Platform** profiles scan the Kubernetes platform. An empty value indicates there is no automated check.

(2) Specify the XCCDF name of the rule, which is parsed directly from the datastream.

(3) Specify the severity of the rule when it fails.

NOTE

The **Rule** object gets an appropriate label for an easy identification of the associated **ProfileBundle** object. The **ProfileBundle** also gets specified in the **OwnerReferences** of this object.

5.4.2.2.4. TailoredProfile object

Use the **TailoredProfile** object to modify the default **Profile** object based on your organization requirements. You can enable or disable rules, set variable values, and provide justification for the customization. After validation, the **TailoredProfile** object creates a **ConfigMap**, which can be referenced by a **ComplianceScan** object.

TIP

You can use the **TailoredProfile** object by referencing it in a **ScanSettingBinding** object. For more information about **ScanSettingBinding**, see ScanSettingBinding object.

Example TailoredProfile object

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: rhcos4-with-usb
spec:
  extends: rhcos4-moderate (1)
  title: <title of the tailored profile>
  disableRules:
    - name: <name of a rule object to be disabled>
      rationale: <description of why this rule is checked>
status:
  id: xccdf_compliance.openshift.io_profile_rhcos4-with-usb (2)
  outputRef:
    name: rhcos4-with-usb-tp (3)
    namespace: openshift-compliance
  state: READY (4)
```

1. This is optional. Name of the **Profile** object upon which the **TailoredProfile** is built. If no value is set, a new profile is created from the **enableRules** list.
2. Specifies the XCCDF name of the tailored profile.
3. Specifies the **ConfigMap** name, which can be used as the value of the **tailoringConfigMap.name** attribute of a **ComplianceScan**.
4. Shows the state of the object such as **READY**, **PENDING**, and **FAILURE**. If the state of the object is **ERROR**, then the attribute **status.errorMessage** provides the reason for the failure.

With the **TailoredProfile** object, it is possible to create a new **Profile** object using the **TailoredProfile** construct. To create a new **Profile**, set the following configuration parameters :

- an appropriate title
- **extends** value must be empty
- scan type annotation on the **TailoredProfile** object:

```
compliance.openshift.io/product-type: Platform/Node
```

NOTE

If you have not set the **product-type** annotation, the Compliance Operator defaults to **Platform** scan type. Adding the **-node** suffix to the name of the **TailoredProfile** object results in **node** scan type.

5.4.2.3. Configuring the compliance scan settings

After you have defined the requirements of the compliance scan, you can configure it by specifying the type of the scan, occurrence of the scan, and location of the scan. To do so, Compliance Operator provides you with a **ScanSetting** object.

5.4.2.3.1. ScanSetting object

Use the **ScanSetting** object to define and reuse the operational policies to run your scans. By default, the Compliance Operator creates the following **ScanSetting** objects:

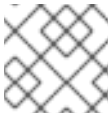
- **default** - it runs a scan every day at 1 AM on both master and worker nodes using a 1Gi Persistent Volume (PV) and keeps the last three results. Remediation is neither applied nor updated automatically.
- **default-auto-apply** - it runs a scan every day at 1AM on both control plane and worker nodes using a 1Gi Persistent Volume (PV) and keeps the last three results. Both **autoApplyRemediations** and **autoUpdateRemediations** are set to true.

Example ScanSetting object

```
apiVersion: compliance.openshift.io/v1alpha1
autoApplyRemediations: true 1
autoUpdateRemediations: true 2
kind: ScanSetting
```

```
maxRetryOnTimeout: 3
metadata:
  creationTimestamp: "2022-10-18T20:21:00Z"
  generation: 1
  name: default-auto-apply
  namespace: openshift-compliance
  resourceVersion: "38840"
  uid: 8cb0967d-05e0-4d7a-ac1c-08a7f7e89e84
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ""
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3 (3)
  size: 1Gi (4)
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
    operator: Exists
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  - effect: NoSchedule
    key: node.kubernetes.io/memory-pressure
    operator: Exists
roles: (5)
- master
- worker
scanTolerations:
- operator: Exists
schedule: 0 1 * * * (6)
showNotApplicable: false
strictNodeScan: true
timeout: 30m
```

1. Set to **true** to enable auto remediations. Set to **false** to disable auto remediations.
2. Set to **true** to enable auto remediations for content updates. Set to **false** to disable auto remediations for content updates.
3. Specify the number of stored scans in the raw result format. The default value is **3**. As the older results get rotated, the administrator must store the results elsewhere before the rotation happens.
4. Specify the storage size that should be created for the scan to store the raw results. The default value is **1Gi**
6. Specify how often the scan should be run in cron format.



NOTE

To disable the rotation policy, set the value to **0**.

5 Specify the **node-role.kubernetes.io** label value to schedule the scan for **Node** type. This value has to match the name of a **MachineConfigPool**.

5.4.2.4. Processing the compliance scan requirements with compliance scans settings

When you have defined the compliance scan requirements and configured the settings to run the scans, then the Compliance Operator processes it using the **ScanSettingBinding** object.

5.4.2.4.1. ScanSettingBinding object

Use the **ScanSettingBinding** object to specify your compliance requirements with reference to the **Profile** or **TailoredProfile** object. It is then linked to a **ScanSetting** object, which provides the operational constraints for the scan. Then the Compliance Operator generates the **ComplianceSuite** object based on the **ScanSetting** and **ScanSettingBinding** objects.

Example ScanSettingBinding object

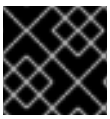
```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: <name of the scan>
profiles: 1
  # Node checks
  - name: rhcos4-with-usb
    kind: TailoredProfile
    apiGroup: compliance.openshift.io/v1alpha1
  # Cluster checks
  - name: ocp4-moderate
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef: 2
  name: my-companys-constraints
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

1 Specify the details of **Profile** or **TailoredProfile** object to scan your environment.

2 Specify the operational constraints, such as schedule and storage size.

The creation of **ScanSetting** and **ScanSettingBinding** objects results in the compliance suite. To get the list of compliance suite, run the following command:

```
$ oc get compliancesuites
```



IMPORTANT

If you delete **ScanSettingBinding**, then compliance suite also is deleted.

5.4.2.5. Tracking the compliance scans

After the creation of compliance suite, you can monitor the status of the deployed scans using the **ComplianceSuite** object.

5.4.2.5.1. ComplianceSuite object

The **ComplianceSuite** object helps you keep track of the state of the scans. It contains the raw settings to create scans and the overall result.

For **Node** type scans, you should map the scan to the **MachineConfigPool**, since it contains the remediations for any issues. If you specify a label, ensure it directly applies to a pool.

Example ComplianceSuite object

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: <name of the scan>
spec:
  autoApplyRemediations: false (1)
  schedule: "0 1 * * *" (2)
  scans: (3)
    - name: workers-scan
      scanType: Node
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:45dc...
      rule: "xccdf_org.ssgproject.content_rule_no_netrc_files"
      nodeSelector:
        node-role.kubernetes.io/worker: ""
status:
  Phase: DONE (4)
  Result: NON-COMPLIANT (5)
  scanStatuses:
  - name: workers-scan
    phase: DONE
    result: NON-COMPLIANT
```

1. Set to **true** to enable auto remediations. Set to **false** to disable auto remediations.
2. Specify how often the scan should be run in cron format.
3. Specify a list of scan specifications to run in the cluster.
4. Indicates the progress of the scans.
5. Indicates the overall verdict of the suite.

The suite in the background creates the **ComplianceScan** object based on the **scans** parameter. You can programmatically fetch the **ComplianceSuites** events. To get the events for the suite, run the following command:

```
$ oc get events --field-selector involvedObject.kind=ComplianceSuite,involvedObject.name=<name of
```

```
the suite>
```

IMPORTANT

You might create errors when you manually define the **ComplianceSuite**, since it contains the XCCDF attributes.

5.4.2.5.2. Advanced ComplianceScan Object

The Compliance Operator includes options for advanced users for debugging or integrating with existing tooling. While it is recommended that you not create a **ComplianceScan** object directly, you can instead manage it using a **ComplianceSuite** object.

Example Advanced ComplianceScan object

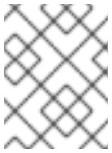
```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceScan
metadata:
  name: <name of the scan>
spec:
  scanType: Node 1
  profile: xccdf_org.ssgproject.content_profile_moderate 2
  content: ssg-ocp4-ds.xml
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:45dc... 3
  rule: "xccdf_org.ssgproject.content_rule_no_netrc_files" 4
  nodeSelector: 5
    node-role.kubernetes.io/worker: ""
status:
  phase: DONE 6
  result: NON-COMPLIANT 7
```

1. Specify either **Node** or **Platform**. Node profiles scan the cluster nodes and platform profiles scan the Kubernetes platform.
2. Specify the XCCDF identifier of the profile that you want to run.
3. Specify the container image that encapsulates the profile files.
4. It is optional. Specify the scan to run a single rule. This rule has to be identified with the XCCDF ID, and has to belong to the specified profile.

NOTE

If you skip the **rule** parameter, then scan runs for all the available rules of the specified profile.

5. If you are on the OpenShift Container Platform and wants to generate a remediation, then nodeSelector label has to match the **MachineConfigPool** label.



NOTE

If you do not specify **nodeSelector** parameter or match the **MachineConfig** label, scan will still run, but it will not create remediation.

6 Indicates the current phase of the scan.

7 Indicates the verdict of the scan.

IMPORTANT

If you delete a **ComplianceSuite** object, then all the associated scans get deleted.

When the scan is complete, it generates the result as Custom Resources of the **ComplianceCheckResult** object. However, the raw results are available in ARF format. These results are stored in a Persistent Volume (PV), which has a Persistent Volume Claim (PVC) associated with the name of the scan. You can programmatically fetch the **ComplianceScans** events. To generate events for the suite, run the following command:

```
oc get events --field-selector involvedObject.kind=ComplianceScan,involvedObject.name=<name of the suite>
```

5.4.2.6. Viewing the compliance results

When the compliance suite reaches the **DONE** phase, you can view the scan results and possible remediations.

5.4.2.6.1. ComplianceCheckResult object

When you run a scan with a specific profile, several rules in the profiles are verified. For each of these rules, a **ComplianceCheckResult** object is created, which provides the state of the cluster for a specific rule.

Example ComplianceCheckResult object

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceCheckResult
metadata:
  labels:
    compliance.openshift.io/check-severity: medium
    compliance.openshift.io/check-status: FAIL
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
  name: workers-scan-no-direct-root-logins
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceScan
    name: workers-scan
description: <description of scan check>
```

```
instructions: <manual instructions for the scan>
id: xccdf_org.ssgproject.content_rule_no_direct_root_logins
severity: medium 1
status: FAIL 2
```

1 Describes the severity of the scan check.

2 Describes the result of the check. The possible values are:

- PASS: check was successful.
- FAIL: check was unsuccessful.
- INFO: check was successful and found something not severe enough to be considered an error.
- MANUAL: check cannot automatically assess the status and manual check is required.
- INCONSISTENT: different nodes report different results.
- ERROR: check run successfully, but could not complete.
- NOTAPPLICABLE: check did not run as it is not applicable.

To get all the check results from a suite, run the following command:

```
oc get compliancecheckresults \
-l compliance.openshift.io/suite=workers-compliancesuite
```

5.4.2.6.2. ComplianceRemediation object

For a specific check you can have a datastream specified fix. However, if a Kubernetes fix is available, then the Compliance Operator creates a **ComplianceRemediation** object.

Example ComplianceRemediation object

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  labels:
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
    machineconfiguration.openshift.io/role: worker
  name: workers-scan-disable-users-coredumps
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
    name: workers-scan-disable-users-coredumps
    uid: <UID>
spec:
  apply: false 1
```

```
object:
  current: 2
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
    spec:
      config:
        ignition:
          version: 2.2.0
        storage:
          files:
          - contents:
              source: data:,%2A%20%20%20%20%20hard%20%20%20core%20%20%20%200
            filesystem: root
            mode: 420
            path: /etc/security/limits.d/75-disable_users_coredumps.conf
  outdated: {} 3
```

1 **true** indicates the remediation was applied. **false** indicates the remediation was not applied.

2 Includes the definition of the remediation.

3 Indicates remediation that was previously parsed from an earlier version of the content. The Compliance Operator still retains the outdated objects to give the administrator a chance to review the new remediations before applying them.

To get all the remediations from a suite, run the following command:

```
oc get complianceremediations \
-l compliance.openshift.io/suite=workers-compliancesuite
```

To list all failing checks that can be remediated automatically, run the following command:

```
oc get compliancecheckresults \
-l 'compliance.openshift.io/check-status in (FAIL),compliance.openshift.io/automated-remediation'
```

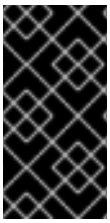
To list all failing checks that can be remediated manually, run the following command:

```
oc get compliancecheckresults \
-l 'compliance.openshift.io/check-status in (FAIL),!compliance.openshift.io/automated-remediation'
```

5.5. COMPLIANCE OPERATOR MANAGEMENT

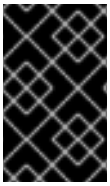
5.5.1. Installing the Compliance Operator

Before you can use the Compliance Operator, you must ensure it is deployed in the cluster.



IMPORTANT

The Compliance Operator might report incorrect results on managed platforms, such as OpenShift Dedicated, Red Hat OpenShift Service on AWS Classic, and Microsoft Azure Red Hat OpenShift. For more information, see the Knowledgebase article Compliance Operator reports incorrect results on Managed Services.



IMPORTANT

Before deploying the Compliance Operator, you are required to define persistent storage in your cluster to store the raw results output. For more information, see Persistant storage overview and Managing the default storage class.

5.5.1.1. Installing the Compliance Operator through the web console

Prerequisites

- You must have **admin** privileges.
- You must have a **StorageClass** resource configured.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Search for the Compliance Operator, then click **Install**.
3. Keep the default selection of **Installation mode** and **namespace** to ensure that the Operator will be installed to the **openshift-compliance** namespace.
4. Click **Install**.

Verification

To confirm that the installation is successful:

1. Navigate to the **Operators → Installed Operators** page.
2. Check that the Compliance Operator is installed in the **openshift-compliance** namespace and its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.
2. Navigate to the **Workloads → Pods** page and check the logs in any pods in the **openshift-compliance** project that are reporting issues.

IMPORTANT

If the **restricted** Security Context Constraints (SCC) have been modified to contain the **system:authenticated** group or has added **requiredDropCapabilities**, the Compliance Operator may not function properly due to permissions issues.

You can create a custom SCC for the Compliance Operator scanner pod service account. For more information, see Creating a custom SCC for the Compliance Operator .

5.5.1.2. Installing the Compliance Operator using the CLI

Prerequisites

- You must have **admin** privileges.
- You must have a **StorageClass** resource configured.

Procedure

1. Define a **Namespace** object:

   Example namespace-object.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-compliance
```

   (1) In OpenShift Container Platform 4.13, the pod security label must be set to **privileged** at the namespace level.

2. Create the **Namespace** object:

```
$ oc create -f namespace-object.yaml
```

3. Define an **OperatorGroup** object:

   Example operator-group-object.yaml

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

4. Create the **OperatorGroup** object:

```
$ oc create -f operator-group-object.yaml
```

5. Define a **Subscription** object:

   Example subscription-object.yaml

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
```

```
channel: "stable"
installPlanApproval: Automatic
name: compliance-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
```

6. Create the **Subscription** object:

```
$ oc create -f subscription-object.yaml
```

NOTE

If you are setting the global scheduler feature and enable **defaultNodeSelector**, you must create the namespace manually and update the annotations of the **openshift-compliance** namespace, or the namespace where the Compliance Operator was installed, with **openshift.io/node-selector: “”**. This removes the default node selector and prevents deployment failures.

Verification

1. Verify the installation succeeded by inspecting the CSV file:

```
$ oc get csv -n openshift-compliance
```

2. Verify that the Compliance Operator is up and running:

```
$ oc get deploy -n openshift-compliance
```

5.5.1.3. Installing the Compliance Operator on ROSA hosted control planes (HCP)

As of the Compliance Operator 1.5.0 release, the Operator is tested against Red Hat OpenShift Service on AWS using Hosted control planes.

Red Hat OpenShift Service on AWS Hosted control planes clusters have restricted access to the control plane, which is managed by Red Hat. By default, the Compliance Operator will schedule to nodes within the **master** node pool, which is not available in Red Hat OpenShift Service on AWS Hosted control planes installations. This requires you to configure the **Subscription** object in a way that allows the Operator to schedule on available node pools. This step is necessary for a successful installation on Red Hat OpenShift Service on AWS Hosted control planes clusters.

Prerequisites

- You must have **admin** privileges.
- You must have a **StorageClass** resource configured.

Procedure

1. Define a **Namespace** object:

   Example namespace-object.yaml file

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-compliance
```

(1) In OpenShift Container Platform 4.13, the pod security label must be set to **privileged** at the namespace level.

2. Create the **Namespace** object by running the following command:

```
$ oc create -f namespace-object.yaml
```

3. Define an **OperatorGroup** object:

Example operator-group-object.yaml file

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

4. Create the **OperatorGroup** object by running the following command:

```
$ oc create -f operator-group-object.yaml
```

5. Define a **Subscription** object:

Example subscription-object.yaml file

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: "" (1)
```

(1) Update the Operator deployment to deploy on **worker** nodes.

6. Create the **Subscription** object by running the following command:

```
$ oc create -f subscription-object.yaml
```

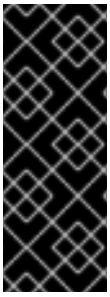
Verification

1. Verify that the installation succeeded by running the following command to inspect the cluster service version (CSV) file:

```
$ oc get csv -n openshift-compliance
```

2. Verify that the Compliance Operator is up and running by using the following command:

```
$ oc get deploy -n openshift-compliance
```



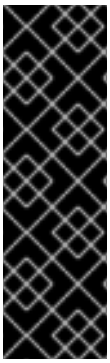
IMPORTANT

If the **restricted** Security Context Constraints (SCC) have been modified to contain the **system:authenticated** group or has added **requiredDropCapabilities**, the Compliance Operator may not function properly due to permissions issues.

You can create a custom SCC for the Compliance Operator scanner pod service account. For more information, see Creating a custom SCC for the Compliance Operator .

5.5.1.4. Installing the Compliance Operator on Hypershift hosted control planes

The Compliance Operator can be installed in Hosted control planes using the OperatorHub by creating a **Subscription** file.



IMPORTANT

Hosted control planes is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Prerequisites

- You must have **admin** privileges.

Procedure

1. Define a **Namespace** object similar to the following:

Example namespace-object.yaml

```
apiVersion: v1
kind: Namespace
```

```
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-compliance
```

(1) In OpenShift Container Platform 4.13, the pod security label must be set to **privileged** at the namespace level.

2. Create the **Namespace** object by running the following command:

```
$ oc create -f namespace-object.yaml
```

3. Define an **OperatorGroup** object:

Example operator-group-object.yaml

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

4. Create the **OperatorGroup** object by running the following command:

```
$ oc create -f operator-group-object.yaml
```

5. Define a **Subscription** object:

Example subscription-object.yaml

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
    env:
    - name: PLATFORM
      value: "HyperShift"
```

6. Create the **Subscription** object by running the following command:

```
$ oc create -f subscription-object.yaml
```

Verification

1. Verify the installation succeeded by inspecting the CSV file by running the following command:

```
$ oc get csv -n openshift-compliance
```

2. Verify that the Compliance Operator is up and running by running the following command:

```
$ oc get deploy -n openshift-compliance
```

Additional resources

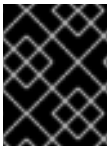
- Hosted control planes overview

5.5.1.5. Additional resources

- The Compliance Operator is supported in a restricted network environment. For more information, see Using Operator Lifecycle Manager on restricted networks .

5.5.2. Updating the Compliance Operator

As a cluster administrator, you can update the Compliance Operator on your OpenShift Container Platform cluster.



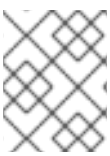
IMPORTANT

It is recommended to update the Compliance Operator to version 1.3.1 or later before updating your OpenShift Container Platform cluster to version 4.14 or later.

5.5.2.1. Preparing for an Operator update

The subscription of an installed Operator specifies an update channel that tracks and receives updates for the Operator. You can change the update channel to start tracking and receiving updates from a newer channel.

The names of update channels in a subscription can differ between Operators, but the naming scheme typically follows a common convention within a given Operator. For example, channel names might follow a minor release update stream for the application provided by the Operator (**1.2**, **1.3**) or a release frequency (**stable**, **fast**).



NOTE

You cannot change installed Operators to a channel that is older than the current channel.

Red Hat Customer Portal Labs include the following application that helps administrators prepare to update their Operators:

- Red Hat OpenShift Container Platform Operator Update Information Checker

You can use the application to search for Operator Lifecycle Manager-based Operators and verify the available Operator version per update channel across different versions of OpenShift Container Platform. Cluster Version Operator-based Operators are not included.

5.5.2.2. Changing the update channel for an Operator

You can change the update channel for an Operator by using the OpenShift Container Platform web console.

TIP

If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new Operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

Prerequisites

- An Operator previously installed using Operator Lifecycle Manager (OLM).

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Operators → Installed Operators**.
2. Click the name of the Operator you want to change the update channel for.
3. Click the **Subscription** tab.
4. Click the name of the update channel under **Update channel**.
5. Click the newer update channel that you want to change to, then click **Save**.
6. For subscriptions with an **Automatic** approval strategy, the update begins automatically. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the update. When complete, the status changes to **Succeeded** and **Up to date**.
   For subscriptions with a **Manual** approval strategy, you can manually approve the update from the **Subscription** tab.

5.5.2.3. Manually approving a pending Operator update

If an installed Operator has the approval strategy in its subscription set to **Manual**, when new updates are released in its current update channel, the update must be manually approved before installation can begin.

Prerequisites

- An Operator previously installed using Operator Lifecycle Manager (OLM).

Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.

2. Operators that have a pending update display a status with **Upgrade available**. Click the name of the Operator you want to update.

3. Click the **Subscription** tab. Any updates requiring approval are displayed next to **Upgrade status**. For example, it might display **1 requires approval**.

4. Click **1 requires approval**, then click **Preview Install Plan**.

5. Review the resources that are listed as available for update. When satisfied, click **Approve**.

6. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the update. When complete, the status changes to **Succeeded** and **Up to date**.

5.5.3. Managing the Compliance Operator

This section describes the lifecycle of security content, including how to use an updated version of compliance content and how to create a custom **ProfileBundle** object.

5.5.3.1. ProfileBundle CR example

The **ProfileBundle** object requires two pieces of information: the URL of a container image that contains the **contentImage** and the file that contains the compliance content. The **contentFile** parameter is relative to the root of the file system. You can define the built-in **rhcos4 ProfileBundle** object as shown in the following example:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  creationTimestamp: "2022-10-19T12:06:30Z"
  finalizers:
  - profilebundle.finalizers.compliance.openshift.io
  generation: 1
  name: rhcos4
  namespace: openshift-compliance
  resourceVersion: "46741"
  uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
spec:
  contentFile: ssg-rhcos4-ds.xml 1
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:900e...
2
status:
  conditions:
  - lastTransitionTime: "2022-10-19T12:07:51Z"
    message: Profile bundle successfully parsed
    reason: Valid
    status: "True"
    type: Ready
  dataStreamStatus: VALID
```

1. Location of the file containing the compliance content.
2. Content image location.

IMPORTANT

The base image used for the content images must include **coreutils**.

5.5.3.2. Updating security content

Security content is included as container images that the **ProfileBundle** objects refer to. To accurately track updates to **ProfileBundles** and the custom resources parsed from the bundles such as rules or profiles, identify the container image with the compliance content using a digest instead of a tag:

```
$ oc -n openshift-compliance get profilebundles rhcos4 -oyaml
```

Example output

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  creationTimestamp: "2022-10-19T12:06:30Z"
  finalizers:
  - profilebundle.finalizers.compliance.openshift.io
  generation: 1
  name: rhcos4
  namespace: openshift-compliance
  resourceVersion: "46741"
  uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
spec:
  contentFile: ssg-rhcos4-ds.xml
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:900e...
(1)
status:
  conditions:
  - lastTransitionTime: "2022-10-19T12:07:51Z"
    message: Profile bundle successfully parsed
    reason: Valid
    status: "True"
    type: Ready
  dataStreamStatus: VALID
```

(1) Security container image.

Each **ProfileBundle** is backed by a deployment. When the Compliance Operator detects that the container image digest has changed, the deployment is updated to reflect the change and parse the content again. Using the digest instead of a tag ensures that you use a stable and predictable set of profiles.

5.5.3.3. Additional resources

- The Compliance Operator is supported in a restricted network environment. For more information, see Using Operator Lifecycle Manager on restricted networks .

5.5.4. Uninstalling the Compliance Operator

You can remove the OpenShift Compliance Operator from your cluster by using the OpenShift Container Platform web console or the CLI.

5.5.4.1. Uninstalling the OpenShift Compliance Operator from OpenShift Container Platform using the web console

To remove the Compliance Operator, you must first delete the objects in the namespace. After the objects are removed, you can remove the Operator and its namespace by deleting the **openshift-compliance** project.

Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- The OpenShift Compliance Operator must be installed.

Procedure

To remove the Compliance Operator by using the OpenShift Container Platform web console:

1. Go to the **Operators → Installed Operators → Compliance Operator** page.
   a. Click **All instances**.
   b. In **All namespaces**, click the Options menu and delete all ScanSettingBinding, ComplainceSuite, ComplianceScan, and ProfileBundle objects.
2. Switch to the **Administration → Operators → Installed Operators** page.
3. Click the Options menu on the **Compliance Operator** entry and select **Uninstall Operator**.
4. Switch to the **Home → Projects** page.
5. Search for 'compliance'.
6. Click the Options menu next to the **openshift-compliance** project, and select **Delete Project**.
   a. Confirm the deletion by typing **openshift-compliance** in the dialog box, and click **Delete**.

5.5.4.2. Uninstalling the OpenShift Compliance Operator from OpenShift Container Platform using the CLI

To remove the Compliance Operator, you must first delete the objects in the namespace. After the objects are removed, you can remove the Operator and its namespace by deleting the **openshift-compliance** project.

Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- The OpenShift Compliance Operator must be installed.

Procedure

1. Delete all objects in the namespace.
   a. Delete the **ScanSettingBinding** objects:

```
$ oc delete ssb --all -n openshift-compliance
```

   b. Delete the **ScanSetting** objects:

```
$ oc delete ss --all -n openshift-compliance
```

   c. Delete the **ComplianceSuite** objects:

```
$ oc delete suite --all -n openshift-compliance
```

   d. Delete the **ComplianceScan** objects:

```
$ oc delete scan --all -n openshift-compliance
```

   e. Delete the **ProfileBundle** objects:

```
$ oc delete profilebundle.compliance --all -n openshift-compliance
```

2. Delete the Subscription object:

```
$ oc delete sub --all -n openshift-compliance
```

3. Delete the CSV object:

```
$ oc delete csv --all -n openshift-compliance
```

4. Delete the project:

```
$ oc delete project openshift-compliance
```

Example output

```
project.project.openshift.io "openshift-compliance" deleted
```

Verification

1. Confirm the namespace is deleted:

```
$ oc get project/openshift-compliance
```

Example output

```
Error from server (NotFound): namespaces "openshift-compliance" not found
```

5.6. COMPLIANCE OPERATOR SCAN MANAGEMENT

5.6.1. Supported compliance profiles

There are several profiles available as part of the Compliance Operator (CO) installation. While you can use the following profiles to assess gaps in a cluster, usage alone does not infer or guarantee compliance with a particular profile and is not an auditor.

In order to be compliant or certified under these various standards, you need to engage an authorized auditor such as a Qualified Security Assessor (QSA), Joint Authorization Board (JAB), or other industry recognized regulatory authority to assess your environment. You are required to work with an authorized auditor to achieve compliance with a standard.

For more information on compliance support for all Red Hat products, see Product Compliance.

IMPORTANT

The Compliance Operator might report incorrect results on some managed platforms, such as OpenShift Dedicated and Azure Red Hat OpenShift. For more information, see the Red Hat Knowledgebase Solution #6983418 .

5.6.1.1. Compliance profiles

The Compliance Operator provides profiles to meet industry standard benchmarks.

NOTE

The following tables reflect the latest available profiles in the Compliance Operator.

5.6.1.1.1. CIS compliance profiles

Table 5.1. Supported CIS compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-cis [1]	CIS Red Hat OpenShift Container Platform Benchmark v1.5.0	Platform	CIS Benchmarks ™ [1]	**x86_64 ppc64le s390x**	

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-cis-1-4 [3]	CIS Red Hat OpenShift Container Platform Benchmark v1.4.0	Platform	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x**	
ocp4-cis-1-5	CIS Red Hat OpenShift Container Platform Benchmark v1.5.0	Platform	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x**	
ocp4-cis-node [1]	CIS Red Hat OpenShift Container Platform Benchmark v1.5.0	Node [2]	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-cis-node-1-4 [3]	CIS Red Hat OpenShift Container Platform Benchmark v1.4.0	Node [2]	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-cis-node-1-5	CIS Red Hat OpenShift Container Platform Benchmark v1.5.0	Node [2]	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

1. The **ocp4-cis** and **ocp4-cis-node** profiles maintain the most up-to-date version of the CIS benchmark as it becomes available in the Compliance Operator. If you want to adhere to a specific version, such as CIS v1.4.0, use the **ocp4-cis-1-4** and **ocp4-cis-node-1-4** profiles.

2. Node profiles must be used with the relevant Platform profile. For more information, see *Compliance Operator profile types*.

3. CIS v1.4.0 is superceded by CIS v1.5.0. It is recommended to apply the latest profile to your environment.

4. To locate the CIS OpenShift Container Platform v4 Benchmark, go to CIS Benchmarks and click **Download Latest CIS Benchmark**, where you can then register to download the benchmark.

5.6.1.1.2. Essential Eight compliance profiles

Table 5.2. Supported Essential Eight compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	Platform	ACSC Hardening Linux Workstations and Servers	**x86_64**	
rhcos4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	Node	ACSC Hardening Linux Workstations and Servers	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

5.6.1.1.3. FedRAMP High compliance profiles

Table 5.3. Supported FedRAMP High compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-high [1]	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - Platform level	Platform	NIST SP-800-53 Release Search	**x86_64**	
ocp4-high-node [1]	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - Node level	Node [2]	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-high-node-rev-4	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - Node level	Node [2]	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-high-rev-4	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - Platform level	Platform	NIST SP-800-53 Release Search	**x86_64**	
rhcos4-high [1]	NIST 800-53 High-Impact Baseline for Red Hat Enterprise Linux CoreOS	Node	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
rhcos4-high-rev-4	NIST 800-53 High-Impact Baseline for Red Hat Enterprise Linux CoreOS	Node	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

1. The **ocp4-high**, **ocp4-high-node** and **rhcos4-high** profiles maintain the most up-to-date version of the FedRAMP High standard as it becomes available in the Compliance Operator. If you want to adhere to a specific version, such as FedRAMP high R4, use the **ocp4-high-rev-4** and **ocp4-high-node-rev-4** profiles.

2. Node profiles must be used with the relevant Platform profile. For more information, see *Compliance Operator profile types*.

5.6.1.1.4. FedRAMP Moderate compliance profiles

Table 5.4. Supported FedRAMP Moderate compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-moderate [1]	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Platform level	Platform	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x**	

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-moderate-node [1]	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Node level	Node [2]	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-moderate-node-rev-4	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Node level	Node [2]	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-moderate-rev-4	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Platform level	Platform	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x**	
rhcos4-moderate [1]	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	Node	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
rhcos4-moderate-rev-4	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	Node	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

1. The **ocp4-moderate**, **ocp4-moderate-node** and **rhcos4-moderate** profiles maintain the most up-to-date version of the FedRAMP Moderate standard as it becomes available in the Compliance Operator. If you want to adhere to a specific version, such as FedRAMP Moderate R4, use the **ocp4-moderate-rev-4** and **ocp4-moderate-node-rev-4** profiles.

2. Node profiles must be used with the relevant Platform profile. For more information, see *Compliance Operator profile types*.

5.6.1.1.5. NERC-CIP compliance profiles

Table 5.5. Supported NERC-CIP compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-nerc-cip	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for the OpenShift Container Platform - Platform level	Platform	NERC CIP Standards	**x86_64**	
ocp4-nerc-cip-node	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for the OpenShift Container Platform - Node level	Node [1]	NERC CIP Standards	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
rhcos4-nerc-cip	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for Red Hat Enterprise Linux CoreOS	Node	NERC CIP Standards	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

1. Node profiles must be used with the relevant Platform profile. For more information, see *Compliance Operator profile types*.

5.6.1.1.6. PCI-DSS compliance profiles

Table 5.6. Supported PCI-DSS compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-pci-dss [1]	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	Platform	PCI Security Standards ® Council Document Library	**x86_64**	
ocp4-pci-dss-3-2 [3]	PCI-DSS v3.2.1 Control Baseline for OpenShift Container Platform 4	Platform	PCI Security Standards ® Council Document Library	**x86_64**	
ocp4-pci-dss-4-0	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	Platform	PCI Security Standards ® Council Document Library	**x86_64**	
ocp4-pci-dss-node [1]	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	Node [2]	PCI Security Standards ® Council Document Library	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-pci-dss-node-3-2 [3]	PCI-DSS v3.2.1 Control Baseline for OpenShift Container Platform 4	Node [2]	PCI Security Standards ® Council Document Library	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-pci-dss-node-4-0	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	Node [2]	PCI Security Standards ® Council Document Library	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

1. The **ocp4-pci-dss** and **ocp4-pci-dss-node** profiles maintain the most up-to-date version of the PCI-DSS standard as it becomes available in the Compliance Operator. If you want to adhere to a specific version, such as PCI-DSS v3.2.1, use the **ocp4-pci-dss-3-2** and **ocp4-pci-dss-node-3-2** profiles.

2. Node profiles must be used with the relevant Platform profile. For more information, see *Compliance Operator profile types*.

3. PCI-DSS v3.2.1 is superceded by PCI-DSS v4. It is recommended to apply the latest profile to your environment.

5.6.1.1.7. STIG compliance profiles

Table 5.7. Supported STIG compliance profiles

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-stig [1]	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift	Platform	DISA-STIG	**x86_64**	
ocp4-stig-node [1]	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift	Node [2]	DISA-STIG	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-stig-node-v1r1 [3]	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift V1R1	Node [2]	DISA-STIG	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
ocp4-stig-node-v2r1	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift V2R1	Node [2]	DISA-STIG	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

Profile	Profile title	Application	Industry compliance benchmark	Supported architectures	Supported platforms
ocp4-stig-v1r1 [3]	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift V1R1	Platform	DISA-STIG	**x86_64**	
ocp4-stig-v2r1	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift V2R1	Platform	DISA-STIG	**x86_64**	
rhcos4-stig	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift	Node	DISA-STIG	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
rhcos4-stig-v1r1 [3]	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift V1R1	Node	DISA-STIG [3]	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)
rhcos4-stig-v2r1	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift V2R1	Node	DISA-STIG	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA HCP)

1. The **ocp4-stig**, **ocp4-stig-node** and **rhcos4-stig** profiles maintain the most up-to-date version

of the DISA-STIG benchmark as it becomes available in the Compliance Operator. If you want to adhere to a specific version, such as DISA-STIG V2R1, use the **ocp4-stig-v2r1** and **ocp4-stig-node-v2r1** profiles.

2. Node profiles must be used with the relevant Platform profile. For more information, see *Compliance Operator profile types*.

3. DISA-STIG V1R1 is superceded by DISA-STIG V2R1. It is recommended to apply the latest profile to your environment.

5.6.1.1.8. About extended compliance profiles

Some compliance profiles have controls that require following industry best practices, resulting in some profiles extending others. Combining the Center for Internet Security (CIS) best practices with National Institute of Standards and Technology (NIST) security frameworks establishes a path to a secure and compliant environment.

For example, the NIST High-Impact and Moderate-Impact profiles extend the CIS profile to achieve compliance. As a result, extended compliance profiles eliminate the need to run both profiles in a single cluster.

Table 5.8. Profile extensions

Profile	Extends
ocp4-pci-dss	ocp4-cis
ocp4-pci-dss-node	ocp4-cis-node
ocp4-high	ocp4-cis
ocp4-high-node	ocp4-cis-node
ocp4-moderate	ocp4-cis
ocp4-moderate-node	ocp4-cis-node
ocp4-nerc-cip	ocp4-moderate
ocp4-nerc-cip-node	ocp4-moderate-node

5.6.1.1.9. Compliance Operator profile types

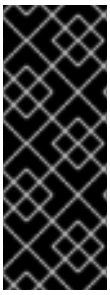
Compliance Operator rules are organized into profiles. Profiles can target the Platform or Nodes for OpenShift Container Platform, and some benchmarks include **rhcos4** Node profiles.

Platform

Platform profiles evaluate your OpenShift Container Platform cluster components. For example, a Platform-level rule can confirm whether APIServer configurations are using strong encryption cyphers.

Node

Node profiles evaluate the OpenShift or RHCOS configuration of each host. You can use two Node profiles: **ocp4** Node profiles and **rhcos4** Node profiles. The **ocp4** Node profiles evaluate the OpenShift configuration of each host. For example, they can confirm whether **kubeconfig** files have the correct permissions to meet a compliance standard. The **rhcos4** Node profiles evaluate the Red Hat Enterprise Linux CoreOS (RHCOS) configuration of each host. For example, they can confirm whether the SSHD service is configured to disable password logins.



IMPORTANT

For benchmarks that have Node and Platform profiles, such as PCI-DSS, you must run both profiles in your OpenShift Container Platform environment.

For benchmarks that have **ocp4** Platform, **ocp4** Node, and **rhcos4** Node profiles, such as FedRAMP High, you must run all three profiles in your OpenShift Container Platform environment.

NOTE

In a cluster with many Nodes, both **ocp4** Node and **rhcos4** Node scans might take a long time to complete.

5.6.2. Compliance Operator scans

The **ScanSetting** and **ScanSettingBinding** APIs are recommended to run compliance scans with the Compliance Operator. For more information on these API objects, run:

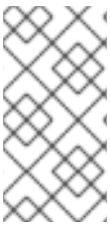
```
$ oc explain scansettings
```

or

```
$ oc explain scansettingbindings
```

5.6.2.1. Running compliance scans

You can run a scan using the Center for Internet Security (CIS) profiles. For convenience, the Compliance Operator creates a **ScanSetting** object with reasonable defaults on startup. This **ScanSetting** object is named **default**.



NOTE

For all-in-one control plane and worker nodes, the compliance scan runs twice on the worker and control plane nodes. The compliance scan might generate inconsistent scan results. You can avoid inconsistent results by defining only a single role in the **ScanSetting** object.

Procedure

1. Inspect the **ScanSetting** object by running the following command:

```
$ oc describe scansettings default -n openshift-compliance
```

Example output

```
Name:                default
Namespace:           openshift-compliance
Labels:              <none>
Annotations:         <none>
API Version:         compliance.openshift.io/v1alpha1
Kind:                ScanSetting
Max Retry On Timeout:  3
Metadata:
  Creation Timestamp:  2024-07-16T14:56:42Z
  Generation:          2
  Resource Version:    91655682
  UID:                 50358cf1-57a8-4f69-ac50-5c7a5938e402
Raw Result Storage:
  Node Selector:
    node-role.kubernetes.io/master:
  Pv Access Modes:
    ReadWriteOnce 1
  Rotation:            3 2
  Size:                1Gi 3
  Storage Class Name:  standard 4
  Tolerations:
    Effect:            NoSchedule
    Key:               node-role.kubernetes.io/master
    Operator:          Exists
    Effect:            NoExecute
    Key:               node.kubernetes.io/not-ready
    Operator:          Exists
    Toleration Seconds:  300
    Effect:            NoExecute
    Key:               node.kubernetes.io/unreachable
    Operator:          Exists
    Toleration Seconds:  300
    Effect:            NoSchedule
    Key:               node.kubernetes.io/memory-pressure
    Operator:          Exists
Roles:
  master 5
  worker 6
Scan Tolerations: 7
  Operator:          Exists
Schedule:            0 1 * * * 8
Show Not Applicable:  false
Strict Node Scan:     true
Suspend:              false
Timeout:              30m
Events:               <none>
```

1 The Compliance Operator creates a persistent volume (PV) that contains the results of the scans. By default, the PV will use access mode **ReadWriteOnce** because the Compliance Operator cannot make any assumptions about the storage classes configured on the cluster. Additionally, **ReadWriteOnce** access mode is available on most clusters. If you need to fetch the scan results, you can do so by using a helper pod, which also binds the volume. Volumes that use the **ReadWriteOnce** access mode can be mounted by only one pod at time, so it is important to remember to delete the helper pods. Otherwise, the

Compliance Operator will not be able to reuse the volume for subsequent scans.

2 The Compliance Operator keeps results of three subsequent scans in the volume; older scans are rotated.

3 The Compliance Operator will allocate one GB of storage for the scan results.

4 The **scansetting.rawResultStorage.storageClassName** field specifies the **storageClassName** value to use when creating the **PersistentVolumeClaim** object to store the raw results. The default value is null, which will attempt to use the default storage class configured in the cluster. If there is no default class specified, then you must set a default class.

5 6 If the scan setting uses any profiles that scan cluster nodes, scan these node roles.

7 The default scan setting object scans all the nodes.

8 The default scan setting object runs scans at 01:00 each day.

As an alternative to the default scan setting, you can use **default-auto-apply**, which has the following settings:

```
Name:                default-auto-apply
Namespace:           openshift-compliance
Labels:              <none>
Annotations:         <none>
API Version:         compliance.openshift.io/v1alpha1
Auto Apply Remediations:  true 1
Auto Update Remediations:  true 2
Kind:                ScanSetting
Metadata:
  Creation Timestamp:  2022-10-18T20:21:00Z
  Generation:          1
  Managed Fields:
    API Version:  compliance.openshift.io/v1alpha1
    Fields Type:  FieldsV1
    fieldsV1:
      f:autoApplyRemediations:
      f:autoUpdateRemediations:
      f:rawResultStorage:
        .:
        f:nodeSelector:
          .:
          f:node-role.kubernetes.io/master:
        f:pvAccessModes:
        f:rotation:
        f:size:
        f:tolerations:
      f:roles:
      f:scanTolerations:
      f:schedule:
      f:showNotApplicable:
      f:strictNodeScan:
    Manager:         compliance-operator
    Operation:       Update
```

```
    Time:         2022-10-18T20:21:00Z
  Resource Version:  38840
  UID:          8cb0967d-05e0-4d7a-ac1c-08a7f7e89e84
Raw Result Storage:
  Node Selector:
    node-role.kubernetes.io/master:
  Pv Access Modes:
    ReadWriteOnce
  Rotation:  3
  Size:      1Gi
  Tolerations:
    Effect:          NoSchedule
    Key:             node-role.kubernetes.io/master
    Operator:          Exists
    Effect:          NoExecute
    Key:             node.kubernetes.io/not-ready
    Operator:          Exists
    Toleration Seconds:  300
    Effect:          NoExecute
    Key:             node.kubernetes.io/unreachable
    Operator:          Exists
    Toleration Seconds:  300
    Effect:          NoSchedule
    Key:             node.kubernetes.io/memory-pressure
    Operator:          Exists
Roles:
  master
  worker
Scan Tolerations:
  Operator:        Exists
Schedule:          0 1 * * *
Show Not Applicable:  false
Strict Node Scan:     true
Events:           <none>
```

1 2 Setting **autoUpdateRemediations** and **autoApplyRemediations** flags to **true** allows you to easily create **ScanSetting** objects that auto-remediate without extra steps.

2. Create a **ScanSettingBinding** object that binds to the default **ScanSetting** object and scans the cluster using the **cis** and **cis-node** profiles. For example:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis-compliance
  namespace: openshift-compliance
profiles:
  - name: ocp4-cis-node
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
  - name: ocp4-cis
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef:
```

```
  name: default
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

3. Create the **ScanSettingBinding** object by running:

```
$ oc create -f <file-name>.yaml -n openshift-compliance
```

At this point in the process, the **ScanSettingBinding** object is reconciled and based on the **Binding** and the **Bound** settings. The Compliance Operator creates a **ComplianceSuite** object and the associated **ComplianceScan** objects.

4. Follow the compliance scan progress by running:

```
$ oc get compliancescan -w -n openshift-compliance
```

The scans progress through the scanning phases and eventually reach the **DONE** phase when complete. In most cases, the result of the scan is **NON-COMPLIANT**. You can review the scan results and start applying remediations to make the cluster compliant. See *Managing Compliance Operator remediation* for more information.

5.6.2.2. Setting custom storage size for results

While the custom resources such as **ComplianceCheckResult** represent an aggregated result of one check across all scanned nodes, it can be useful to review the raw results as produced by the scanner. The raw results are produced in the ARF format and can be large (tens of megabytes per node), it is impractical to store them in a Kubernetes resource backed by the **etcd** key-value store. Instead, every scan creates a persistent volume (PV) which defaults to 1GB size. Depending on your environment, you may want to increase the PV size accordingly. This is done using the **rawResultStorage.size** attribute that is exposed in both the **ScanSetting** and **ComplianceScan** resources.

A related parameter is **rawResultStorage.rotation** which controls how many scans are retained in the PV before the older scans are rotated. The default value is 3, setting the rotation policy to 0 disables the rotation. Given the default rotation policy and an estimate of 100MB per a raw ARF scan report, you can calculate the right PV size for your environment.

5.6.2.2.1. Using custom result storage values

Because OpenShift Container Platform can be deployed in a variety of public clouds or bare metal, the Compliance Operator cannot determine available storage configurations. By default, the Compliance Operator will try to create the PV for storing results using the default storage class of the cluster, but a custom storage class can be configured using the **rawResultStorage.StorageClassName** attribute.

IMPORTANT

If your cluster does not specify a default storage class, this attribute must be set.

Configure the **ScanSetting** custom resource to use a standard storage class and create persistent volumes that are 10GB in size and keep the last 10 results:

Example ScanSetting CR

```
apiVersion: compliance.openshift.io/v1alpha1
```

```
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  storageClassName: standard
  rotation: 10
  size: 10Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
```

5.6.2.3. Scheduling the result server pod on a worker node

The result server pod mounts the persistent volume (PV) that stores the raw Asset Reporting Format (ARF) scan results. The **nodeSelector** and **tolerations** attributes enable you to configure the location of the result server pod.

This is helpful for those environments where control plane nodes are not permitted to mount persistent volumes.

Procedure

- Create a **ScanSetting** custom resource (CR) for the Compliance Operator:
  a. Define the **ScanSetting** CR, and save the YAML file, for example, **rs-workers.yaml**:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: rs-on-workers
  namespace: openshift-compliance
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: "" (1)
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists (2)
roles:
- worker
- master
scanTolerations:
  - operator: Exists
schedule: 0 1 * * *
```

(1) The Compliance Operator uses this node to store scan results in ARF format.

2 The result server pod tolerates all taints.

b. To create the **ScanSetting** CR, run the following command:

```
$ oc create -f rs-workers.yaml
```

Verification

- To verify that the **ScanSetting** object is created, run the following command:

```
$ oc get scansettings rs-on-workers -n openshift-compliance -o yaml
```

Example output

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  creationTimestamp: "2021-11-19T19:36:36Z"
  generation: 1
  name: rs-on-workers
  namespace: openshift-compliance
  resourceVersion: "48305"
  uid: 43fdfc5f-15a7-445a-8bbc-0e4a160cd46e
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists
roles:
- worker
- master
scanTolerations:
- operator: Exists
schedule: 0 1 * * *
strictNodeScan: true
```

5.6.2.4. ScanSetting Custom Resource

The **ScanSetting** Custom Resource now allows you to override the default CPU and memory limits of scanner pods through the scan limits attribute. The Compliance Operator will use defaults of 500Mi memory, 100m CPU for the scanner container, and 200Mi memory with 100m CPU for the **api-resource-collector** container. To set the memory limits of the Operator, modify the **Subscription** object if installed through OLM or the Operator deployment itself.

To increase the default CPU and memory limits of the Compliance Operator, see *Increasing Compliance Operator resource limits*.

IMPORTANT

Increasing the memory limit for the Compliance Operator or the scanner pods is needed if the default limits are not sufficient and the Operator or scanner pods are ended by the Out Of Memory (OOM) process.

5.6.2.5. Configuring the Hosted control planes management cluster

If you are hosting your own Hosted control plane or Hypershift environment and want to scan a Hosted Cluster from the management cluster, you will need to set the name and prefix namespace for the target Hosted Cluster. You can achieve this by creating a **TailoredProfile**.

IMPORTANT

This procedure only applies to users managing their own Hosted control planes environment.

NOTE

Only **ocp4-cis** and **ocp4-pci-dss** profiles are supported in Hosted control planes management clusters.

Prerequisites

- The Compliance Operator is installed in the management cluster.

Procedure

1. Obtain the **name** and **namespace** of the hosted cluster to be scanned by running the following command:

```
$ oc get hostedcluster -A
```

Example output

```
NAMESPACE       NAME                VERSION   KUBECONFIG
PROGRESS    AVAILABLE   PROGRESSING   MESSAGE
local-cluster   79136a1bdb84b3c13217   4.13.5    79136a1bdb84b3c13217-admin-kubeconfig
  Completed   True        False         The hosted control plane is available
```

2. In the management cluster, create a **TailoredProfile** extending the scan Profile and define the name and namespace of the Hosted Cluster to be scanned:

Example management-tailoredprofile.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: hypershift-cisk57aw88gry
  namespace: openshift-compliance
spec:
  description: This profile test required rules
  extends: ocp4-cis (1)
```

```
title: Management namespace profile
setValues:
- name: ocp4-hypershift-cluster
  rationale: This value is used for HyperShift version detection
  value: 79136a1bdb84b3c13217 2
- name: ocp4-hypershift-namespace-prefix
  rationale: This value is used for HyperShift control plane namespace detection
  value: local-cluster 3
```

1. Variable. Only **ocp4-cis** and **ocp4-pci-dss** profiles are supported in Hosted control planes management clusters.
2. The **value** is the **NAME** from the output in the previous step.
3. The **value** is the **NAMESPACE** from the output in the previous step.

3. Create the **TailoredProfile**:

```
$ oc create -n openshift-compliance -f mgmt-tp.yaml
```

5.6.2.6. Applying resource requests and limits

When the kubelet starts a container as part of a Pod, the kubelet passes that container's requests and limits for memory and CPU to the container runtime. In Linux, the container runtime configures the kernel cgroups that apply and enforce the limits you defined.

The CPU limit defines how much CPU time the container can use. During each scheduling interval, the Linux kernel checks to see if this limit is exceeded. If so, the kernel waits before allowing the cgroup to resume execution.

If several different containers (cgroups) want to run on a contended system, workloads with larger CPU requests are allocated more CPU time than workloads with small requests. The memory request is used during Pod scheduling. On a node that uses cgroups v2, the container runtime might use the memory request as a hint to set **memory.min** and **memory.low** values.

If a container attempts to allocate more memory than this limit, the Linux kernel out-of-memory subsystem activates and intervenes by stopping one of the processes in the container that tried to allocate memory. The memory limit for the Pod or container can also apply to pages in memory-backed volumes, such as an emptyDir.

The kubelet tracks **tmpfs emptyDir** volumes as container memory is used, rather than as local ephemeral storage. If a container exceeds its memory request and the node that it runs on becomes short of memory overall, the Pod's container might be evicted.

IMPORTANT

A container may not exceed its CPU limit for extended periods. Container run times do not stop Pods or containers for excessive CPU usage. To determine whether a container cannot be scheduled or is being killed due to resource limits, see *Troubleshooting the Compliance Operator*.

5.6.2.7. Scheduling Pods with container resource requests

When a Pod is created, the scheduler selects a Node for the Pod to run on. Each node has a maximum capacity for each resource type in the amount of CPU and memory it can provide for the Pods. The scheduler ensures that the sum of the resource requests of the scheduled containers is less than the capacity nodes for each resource type.

Although memory or CPU resource usage on nodes is very low, the scheduler might still refuse to place a Pod on a node if the capacity check fails to protect against a resource shortage on a node.

For each container, you can specify the following resource limits and request:

```
spec.containers[].resources.limits.cpu
spec.containers[].resources.limits.memory
spec.containers[].resources.limits.hugepages-<size>
spec.containers[].resources.requests.cpu
spec.containers[].resources.requests.memory
spec.containers[].resources.requests.hugepages-<size>
```

Although you can specify requests and limits for only individual containers, it is also useful to consider the overall resource requests and limits for a pod. For a particular resource, a container resource request or limit is the sum of the resource requests or limits of that type for each container in the pod.

Example container resource requests and limits

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests: 1
        memory: "64Mi"
        cpu: "250m"
      limits: 2
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

1. The container is requesting 64 Mi of memory and 250 m CPU.
2. The container's limits are 128 Mi of memory and 500 m CPU.

5.6.3. Tailoring the Compliance Operator

While the Compliance Operator comes with ready-to-use profiles, they must be modified to fit the organizations' needs and requirements. The process of modifying a profile is called *tailoring*.

The Compliance Operator provides the **TailoredProfile** object to help tailor profiles.

5.6.3.1. Creating a new tailored profile

You can write a tailored profile from scratch by using the **TailoredProfile** object. Set an appropriate **title** and **description** and leave the **extends** field empty. Indicate to the Compliance Operator what type of scan this custom profile will generate:

- Node scan: Scans the Operating System.
- Platform scan: Scans the OpenShift Container Platform configuration.

Procedure

- Set the following annotation on the **TailoredProfile** object:

Example new-profile.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: new-profile
  annotations:
    compliance.openshift.io/product-type: Node 1
spec:
  extends: ocp4-cis-node 2
  description: My custom profile 3
  title: Custom profile 4
  enableRules:
    - name: ocp4-etcd-unique-ca
      rationale: We really need to enable this
  disableRules:
    - name: ocp4-file-groupowner-cni-conf
      rationale: This does not apply to the cluster
```

1. Set **Node** or **Platform** accordingly.
2. The **extends** field is optional.
3. Use the **description** field to describe the function of the new **TailoredProfile** object.
4. Give your **TailoredProfile** object a title with the **title** field.

NOTE

Adding the **-node** suffix to the **name** field of the **TailoredProfile** object is similar to adding the **Node** product type annotation and generates an Operating System scan.

5.6.3.2. Using tailored profiles to extend existing ProfileBundles

While the **TailoredProfile** CR enables the most common tailoring operations, the XCCDF standard allows even more flexibility in tailoring OpenSCAP profiles. In addition, if your organization has been using OpenScap previously, you may have an existing XCCDF tailoring file and can reuse it.

The **ComplianceSuite** object contains an optional **TailoringConfigMap** attribute that you can point to a custom tailoring file. The value of the **TailoringConfigMap** attribute is a name of a config map, which must contain a key called **tailoring.xml** and the value of this key is the tailoring contents.

Procedure

1. Browse the available rules for the Red Hat Enterprise Linux CoreOS (RHCOS) **ProfileBundle**:

```
$ oc get rules.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

2. Browse the available variables in the same **ProfileBundle**:

```
$ oc get variables.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

3. Create a tailored profile named **nist-moderate-modified**:

   a. Choose which rules you want to add to the **nist-moderate-modified** tailored profile. This example extends the **rhcos4-moderate** profile by disabling two rules and changing one value. Use the **rationale** value to describe why these changes were made:

Example new-profile-node.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: nist-moderate-modified
spec:
  extends: rhcos4-moderate
  description: NIST moderate profile
  title: My modified NIST moderate profile
  disableRules:
  - name: rhcos4-file-permissions-var-log-messages
    rationale: The file contains logs of error messages in the system
  - name: rhcos4-account-disable-post-pw-expiration
    rationale: No need to check this as it comes from the IdP
  setValues:
  - name: rhcos4-var-selinux-state
    rationale: Organizational requirements
    value: permissive
```

Table 5.9. Attributes for spec variables

Attribute	Description
extends	Name of the **Profile** object upon which this **TailoredProfile** is built.

Attribute	Description
title	Human-readable title of the **TailoredProfile**.
disableRules	A list of name and rationale pairs. Each name refers to a name of a rule object that is to be disabled. The rationale value is human-readable text describing why the rule is disabled.
manualRules	A list of name and rationale pairs. When a manual rule is added, the check result status will always be **manual** and remediation will not be generated. This attribute is automatic and by default has no values when set as a manual rule.
enableRules	A list of name and rationale pairs. Each name refers to a name of a rule object that is to be enabled. The rationale value is human-readable text describing why the rule is enabled.
description	Human-readable text describing the **TailoredProfile**.
setValues	A list of name, rationale, and value groupings. Each name refers to a name of the value set. The rationale is human-readable text describing the set. The value is the actual setting.

b. Add the **tailoredProfile.spec.manualRules** attribute:

Example tailoredProfile.spec.manualRules.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: ocp4-manual-scc-check
spec:
  extends: ocp4-cis
  description: This profile extends ocp4-cis by forcing the SCC check to always return
MANUAL
  title: OCP4 CIS profile with manual SCC check
  manualRules:
    - name: ocp4-scc-limit-container-allowed-capabilities
      rationale: We use third party software that installs its own SCC with extra privileges
```

c. Create the **TailoredProfile** object:

```
$ oc create -n openshift-compliance -f new-profile-node.yaml 1
```

1 The **TailoredProfile** object is created in the default **openshift-compliance** namespace.

Example output

```
tailoredprofile.compliance.openshift.io/nist-moderate-modified created
```

4. Define the **ScanSettingBinding** object to bind the new **nist-moderate-modified** tailored profile to the default **ScanSetting** object.

Example new-scansettingbinding.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: nist-moderate-modified
profiles:
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: Profile
    name: ocp4-moderate
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: TailoredProfile
    name: nist-moderate-modified
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default
```

5. Create the **ScanSettingBinding** object:

```
$ oc create -n openshift-compliance -f new-scansettingbinding.yaml
```

Example output

```
scansettingbinding.compliance.openshift.io/nist-moderate-modified created
```

5.6.4. Retrieving Compliance Operator raw results

When proving compliance for your OpenShift Container Platform cluster, you might need to provide the scan results for auditing purposes.

5.6.4.1. Obtaining Compliance Operator raw results from a persistent volume

Procedure

The Compliance Operator generates and stores the raw results in a persistent volume. These results are in Asset Reporting Format (ARF).

1. Explore the **ComplianceSuite** object:

```
$ oc get compliancesuites nist-moderate-modified \
-o json -n openshift-compliance | jq '.status.scanStatuses[].resultsStorage'
```

Example output

```
{
    "name": "ocp4-moderate",
    "namespace": "openshift-compliance"
}
```

```
{
    "name": "nist-moderate-modified-master",
    "namespace": "openshift-compliance"
}
{
    "name": "nist-moderate-modified-worker",
    "namespace": "openshift-compliance"
}
```

This shows the persistent volume claims where the raw results are accessible.

2. Verify the raw data location by using the name and namespace of one of the results:

```
$ oc get pvc -n openshift-compliance rhcos4-moderate-worker
```

Example output

```
NAME                    STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
rhcos4-moderate-worker   Bound pvc-548f6cfe-164b-42fe-ba13-a07cfbc77f3a   1Gi     RWO
gp2         92m
```

3. Fetch the raw results by spawning a pod that mounts the volume and copying the results:

```
$ oc create -n openshift-compliance -f pod.yaml
```

Example pod.yaml

```
apiVersion: "v1"
kind: Pod
metadata:
  name: pv-extract
spec:
  containers:
    - name: pv-extract-pod
      image: registry.access.redhat.com/ubi9/ubi
      command: ["sleep", "3000"]
      volumeMounts:
      - mountPath: "/workers-scan-results"
        name: workers-scan-vol
  volumes:
    - name: workers-scan-vol
      persistentVolumeClaim:
        claimName: rhcos4-moderate-worker
```

4. After the pod is running, download the results:

```
$ oc cp pv-extract:/workers-scan-results -n openshift-compliance .
```

IMPORTANT

Spawning a pod that mounts the persistent volume will keep the claim as **Bound**. If the volume's storage class in use has permissions set to **ReadWriteOnce**, the volume is only mountable by one pod at a time. You must delete the pod upon completion, or it will not be possible for the Operator to schedule a pod and continue storing results in this location.

5. After the extraction is complete, the pod can be deleted:

```
$ oc delete pod pv-extract -n openshift-compliance
```

5.6.5. Managing Compliance Operator result and remediation

Each **ComplianceCheckResult** represents a result of one compliance rule check. If the rule can be remediated automatically, a **ComplianceRemediation** object with the same name, owned by the **ComplianceCheckResult** is created. Unless requested, the remediations are not applied automatically, which gives an OpenShift Container Platform administrator the opportunity to review what the remediation does and only apply a remediation once it has been verified.

IMPORTANT

Full remediation for Federal Information Processing Standards (FIPS) compliance requires enabling FIPS mode for the cluster. To enable FIPS mode, you must run the installation program from a Red Hat Enterprise Linux (RHEL) computer configured to operate in FIPS mode. For more information about configuring FIPS mode on RHEL, see Installing the system in FIPS mode .

FIPS mode is supported on the following architectures:

- **x86_64**
- **ppc64le**
- **s390x**

5.6.5.1. Filters for compliance check results

By default, the **ComplianceCheckResult** objects are labeled with several useful labels that allow you to query the checks and decide on the next steps after the results are generated.

List checks that belong to a specific suite:

```
$ oc get -n openshift-compliance compliancecheckresults \
  -l compliance.openshift.io/suite=workers-compliancesuite
```

List checks that belong to a specific scan:

```
$ oc get -n openshift-compliance compliancecheckresults \
-l compliance.openshift.io/scan=workers-scan
```

Not all **ComplianceCheckResult** objects create **ComplianceRemediation** objects. Only **ComplianceCheckResult** objects that can be remediated automatically do. A

ComplianceCheckResult object has a related remediation if it is labeled with the **compliance.openshift.io/automated-remediation** label. The name of the remediation is the same as the name of the check.

List all failing checks that can be remediated automatically:

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/automated-remediation'
```

List all failing checks sorted by severity:

```
$ oc get compliancecheckresults -n openshift-compliance \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/check-severity=high'
```

Example output

```
NAME                                                STATUS   SEVERITY
nist-moderate-modified-master-configure-crypto-policy          FAIL     high
nist-moderate-modified-master-coreos-pti-kernel-argument       FAIL     high
nist-moderate-modified-master-disable-ctrlaltdel-burstaction   FAIL     high
nist-moderate-modified-master-disable-ctrlaltdel-reboot        FAIL     high
nist-moderate-modified-master-no-empty-passwords               FAIL     high
nist-moderate-modified-master-selinux-state                    FAIL     high
nist-moderate-modified-worker-configure-crypto-policy          FAIL     high
nist-moderate-modified-worker-coreos-pti-kernel-argument       FAIL     high
nist-moderate-modified-worker-disable-ctrlaltdel-burstaction   FAIL     high
nist-moderate-modified-worker-disable-ctrlaltdel-reboot        FAIL     high
nist-moderate-modified-worker-no-empty-passwords               FAIL     high
nist-moderate-modified-worker-selinux-state                    FAIL     high
ocp4-moderate-configure-network-policies-namespaces            FAIL     high
```

List all failing checks that must be remediated manually:

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,!compliance.openshift.io/automated-remediation'
```

The manual remediation steps are typically stored in the **description** attribute in the **ComplianceCheckResult** object.

Table 5.10. ComplianceCheckResult Status

ComplianceCheckResult Status	Description
PASS	Compliance check ran to completion and passed.
FAIL	Compliance check ran to completion and failed.
INFO	Compliance check ran to completion and found something not severe enough to be considered an error.

ComplianceCheckResult Status	Description
MANUAL	Compliance check does not have a way to automatically assess the success or failure and must be checked manually.
INCONSISTENT	Compliance check reports different results from different sources, typically cluster nodes.
ERROR	Compliance check ran, but could not complete properly.
NOT-APPLICABLE	Compliance check did not run because it is not applicable or not selected.

5.6.5.2. Reviewing a remediation

Review both the **ComplianceRemediation** object and the **ComplianceCheckResult** object that owns the remediation. The **ComplianceCheckResult** object contains human-readable descriptions of what the check does and the hardening trying to prevent, as well as other **metadata** like the severity and the associated security controls. The **ComplianceRemediation** object represents a way to fix the problem described in the **ComplianceCheckResult**. After first scan, check for remediations with the state **MissingDependencies**.

Below is an example of a check and a remediation called **sysctl-net-ipv4-conf-all-accept-redirects**. This example is redacted to only show **spec** and **status** and omits **metadata**:

```
spec:
  apply: false
  current:
  object:
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
    spec:
      config:
        ignition:
          version: 3.2.0
        storage:
          files:
            - path: /etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf
              mode: 0644
              contents:
                source: data:,net.ipv4.conf.all.accept_redirects%3D0
  outdated: {}
status:
  applicationState: NotApplied
```

The remediation payload is stored in the **spec.current** attribute. The payload can be any Kubernetes object, but because this remediation was produced by a node scan, the remediation payload in the

above example is a **MachineConfig** object. For Platform scans, the remediation payload is often a different kind of an object (for example, a **ConfigMap** or **Secret** object), but typically applying that remediation is up to the administrator, because otherwise the Compliance Operator would have required a very broad set of permissions to manipulate any generic Kubernetes object. An example of remediating a Platform check is provided later in the text.

To see exactly what the remediation does when applied, the **MachineConfig** object contents use the Ignition objects for the configuration. See the Ignition specification for further information about the format. In our example, **the spec.config.storage.files[0].path** attribute specifies the file that is being create by this remediation (**/etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf**) and the **spec.config.storage.files[0].contents.source** attribute specifies the contents of that file.

NOTE

The contents of the files are URL-encoded.

Use the following Python script to view the contents:

```
$ echo "net.ipv4.conf.all.accept_redirects%3D0" | python3 -c "import sys, urllib.parse;
print(urllib.parse.unquote(''.join(sys.stdin.readlines())))"
```

Example output

```
net.ipv4.conf.all.accept_redirects=0
```

IMPORTANT

The Compliance Operator does not automatically resolve dependency issues that can occur between remediations. Users should perform a rescan after remediations are applied to ensure accurate results.

5.6.5.3. Applying remediation when using customized machine config pools

When you create a custom **MachineConfigPool**, add a label to the **MachineConfigPool** so that **machineConfigPoolSelector** present in the **KubeletConfig** can match the label with **MachineConfigPool**.

IMPORTANT

Do not set **protectKernelDefaults: false** in the **KubeletConfig** file, because the **MachineConfigPool** object might fail to unpause unexpectedly after the Compliance Operator finishes applying remediation.

Procedure

1. List the nodes.

```
$ oc get nodes -n openshift-compliance
```

Example output

```
NAME                                         STATUS  ROLES  AGE    VERSION
ip-10-0-128-92.us-east-2.compute.internal  Ready   master 5h21m  v1.26.0
ip-10-0-158-32.us-east-2.compute.internal  Ready   worker 5h17m  v1.26.0
ip-10-0-166-81.us-east-2.compute.internal  Ready   worker 5h17m  v1.26.0
ip-10-0-171-170.us-east-2.compute.internal Ready   master 5h21m  v1.26.0
ip-10-0-197-35.us-east-2.compute.internal  Ready   master 5h22m  v1.26.0
```

2. Add a label to nodes.

```
$ oc -n openshift-compliance \
label node ip-10-0-166-81.us-east-2.compute.internal \
node-role.kubernetes.io/<machine_config_pool_name>=
```

Example output

```
node/ip-10-0-166-81.us-east-2.compute.internal labeled
```

3. Create custom **MachineConfigPool** CR.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: <machine_config_pool_name>
  labels:
    pools.operator.machineconfiguration.openshift.io/<machine_config_pool_name>: '' (1)
spec:
  machineConfigSelector:
  matchExpressions:
  - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,
<machine_config_pool_name>]}
  nodeSelector:
  matchLabels:
    node-role.kubernetes.io/<machine_config_pool_name>: ""
```

(1) The **labels** field defines label name to add for Machine config pool(MCP).

4. Verify MCP created successfully.

```
$ oc get mcp -w
```

5.6.5.4. Evaluating KubeletConfig rules against default configuration values

OpenShift Container Platform infrastructure might contain incomplete configuration files at run time, and nodes assume default configuration values for missing configuration options. Some configuration options can be passed as command-line arguments. As a result, the Compliance Operator cannot verify if the configuration file on the node is complete because it might be missing options used in the rule checks.

To prevent false negative results where the default configuration value passes a check, the Compliance Operator uses the Node/Proxy API to fetch the configuration for each node in a node pool, then all configuration options that are consistent across nodes in the node pool are stored in a file that

represents the configuration for all nodes within that node pool. This increases the accuracy of the scan results.

No additional configuration changes are required to use this feature with default **master** and **worker** node pools configurations.

5.6.5.5. Scanning custom node pools

The Compliance Operator does not maintain a copy of each node pool configuration. The Compliance Operator aggregates consistent configuration options for all nodes within a single node pool into one copy of the configuration file. The Compliance Operator then uses the configuration file for a particular node pool to evaluate rules against nodes within that pool.

Procedure

1. Add the **example** role to the **ScanSetting** object that will be stored in the **ScanSettingBinding** CR:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  rotation: 3
  size: 1Gi
roles:
- worker
- master
- example
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
```

2. Create a scan that uses the **ScanSettingBinding** CR:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis
  namespace: openshift-compliance
profiles:
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: ocp4-cis
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: ocp4-cis-node
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default
```

Verification

- The Platform KubeletConfig rules are checked through the **Node/Proxy** object. You can find those rules by running the following command:

```
$ oc get rules -o json | jq '.items[] | select(.checkType == "Platform") | select(.metadata.name
| contains("ocp4-kubelet-")) | .metadata.name'
```

5.6.5.6. Remediating KubeletConfig sub pools

KubeletConfig remediation labels can be applied to **MachineConfigPool** sub-pools.

Procedure

- Add a label to the sub-pool **MachineConfigPool** CR:

```
$ oc label mcp <sub-pool-name> pools.operator.machineconfiguration.openshift.io/<sub-
pool-name>=
```

5.6.5.7. Applying a remediation

The boolean attribute **spec.apply** controls whether the remediation should be applied by the Compliance Operator. You can apply the remediation by setting the attribute to **true**:

```
$ oc -n openshift-compliance \
patch complianceremediations/<scan-name>-sysctl-net-ipv4-conf-all-accept-redirects \
--patch '{"spec":{"apply":true}}' --type=merge
```

After the Compliance Operator processes the applied remediation, the **status.ApplicationState** attribute would change to **Applied** or to **Error** if incorrect. When a machine config remediation is applied, that remediation along with all other applied remediations are rendered into a **MachineConfig** object named **75-$scan-name-$suite-name**. That **MachineConfig** object is subsequently rendered by the Machine Config Operator and finally applied to all the nodes in a machine config pool by an instance of the machine control daemon running on each node.

Note that when the Machine Config Operator applies a new **MachineConfig** object to nodes in a pool, all the nodes belonging to the pool are rebooted. This might be inconvenient when applying multiple remediations, each of which re-renders the composite **75-$scan-name-$suite-name MachineConfig** object. To prevent applying the remediation immediately, you can pause the machine config pool by setting the **.spec.paused** attribute of a **MachineConfigPool** object to **true**.

The Compliance Operator can apply remediations automatically. Set **autoApplyRemediations: true** in the **ScanSetting** top-level object.

WARNING

Applying remediations automatically should only be done with careful consideration.

IMPORTANT

The Compliance Operator does not automatically resolve dependency issues that can occur between remediations. Users should perform a rescan after remediations are applied to ensure accurate results.

5.6.5.8. Remediating a platform check manually

Checks for Platform scans typically have to be remediated manually by the administrator for two reasons:

- It is not always possible to automatically determine the value that must be set. One of the checks requires that a list of allowed registries is provided, but the scanner has no way of knowing which registries the organization wants to allow.
- Different checks modify different API objects, requiring automated remediation to possess **root** or superuser access to modify objects in the cluster, which is not advised.

Procedure

1. The example below uses the **ocp4-ocp-allowed-registries-for-import** rule, which would fail on a default OpenShift Container Platform installation. Inspect the rule **oc get rule.compliance/ocp4-ocp-allowed-registries-for-import -oyaml**, the rule is to limit the registries the users are allowed to import images from by setting the **allowedRegistriesForImport** attribute, The *warning* attribute of the rule also shows the API object checked, so it can be modified and remediate the issue:

```
$ oc edit image.config.openshift.io/cluster
```

Example output

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2020-09-10T10:12:54Z"
  generation: 2
  name: cluster
  resourceVersion: "363096"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: 2dcb614e-2f8a-4a23-ba9a-8e33cd0ff77e
spec:
  allowedRegistriesForImport:
  - domainName: registry.redhat.io
status:
  externalRegistryHostnames:
  - default-route-openshift-image-registry.apps.user-cluster-09-10-12-
07.devcluster.openshift.com
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

2. Re-run the scan:

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

5.6.5.9. Updating remediations

When a new version of compliance content is used, it might deliver a new and different version of a remediation than the previous version. The Compliance Operator will keep the old version of the remediation applied. The OpenShift Container Platform administrator is also notified of the new version to review and apply. A ComplianceRemediation object that had been applied earlier, but was updated changes its status to **Outdated**. The outdated objects are labeled so that they can be searched for easily.

The previously applied remediation contents would then be stored in the **spec.outdated** attribute of a **ComplianceRemediation** object and the new updated contents would be stored in the **spec.current** attribute. After updating the content to a newer version, the administrator then needs to review the remediation. As long as the **spec.outdated** attribute exists, it would be used to render the resulting **MachineConfig** object. After the **spec.outdated** attribute is removed, the Compliance Operator re-renders the resulting **MachineConfig** object, which causes the Operator to push the configuration to the nodes.

Procedure

1. Search for any outdated remediations:

```
$ oc -n openshift-compliance get complianceremediations \
-l complianceoperator.openshift.io/outdated-remediation=
```

Example output

```
NAME                            STATE
workers-scan-no-empty-passwords   Outdated
```

The currently applied remediation is stored in the **Outdated** attribute and the new, unapplied remediation is stored in the **Current** attribute. If you are satisfied with the new version, remove the **Outdated** field. If you want to keep the updated content, remove the **Current** and **Outdated** attributes.

2. Apply the newer version of the remediation:

```
$ oc -n openshift-compliance patch complianceremediations workers-scan-no-empty-passwords \
--type json -p '[{"op":"remove", "path":/spec/outdated}]'
```

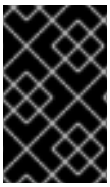
3. The remediation state will switch from **Outdated** to **Applied**:

```
$ oc get -n openshift-compliance complianceremediations workers-scan-no-empty-passwords
```

Example output

```
NAME                            STATE
workers-scan-no-empty-passwords   Applied
```

4. The nodes will apply the newer remediation version and reboot.



IMPORTANT

The Compliance Operator does not automatically resolve dependency issues that can occur between remediations. Users should perform a rescan after remediations are applied to ensure accurate results.

5.6.5.10. Unapplying a remediation

It might be required to unapply a remediation that was previously applied.

Procedure

1. Set the **apply** flag to **false**:

```
$ oc -n openshift-compliance \
patch complianceremediations/rhcos4-moderate-worker-sysctl-net-ipv4-conf-all-accept-redirects \
--patch '{"spec":{"apply":false}}' --type=merge
```

2. The remediation status will change to **NotApplied** and the composite **MachineConfig** object would be re-rendered to not include the remediation.

IMPORTANT

All affected nodes with the remediation will be rebooted.

IMPORTANT

The Compliance Operator does not automatically resolve dependency issues that can occur between remediations. Users should perform a rescan after remediations are applied to ensure accurate results.

5.6.5.11. Removing a KubeletConfig remediation

KubeletConfig remediations are included in node-level profiles. In order to remove a KubeletConfig remediation, you must manually remove it from the **KubeletConfig** objects. This example demonstrates how to remove the compliance check for the **one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** remediation.

Procedure

1. Locate the **scan-name** and compliance check for the **one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** remediation:

```
$ oc -n openshift-compliance get remediation \ one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available -o yaml
```

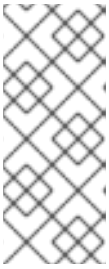
Example output

```
apiVersion: compliance.openshift.io/v1alpha1
```

```
kind: ComplianceRemediation
metadata:
  annotations:
    compliance.openshift.io/xccdf-value-used: var-kubelet-evictionhard-imagefs-available
  creationTimestamp: "2022-01-05T19:52:27Z"
  generation: 1
  labels:
    compliance.openshift.io/scan-name: one-rule-tp-node-master (1)
    compliance.openshift.io/suite: one-rule-ssb-node
  name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
    name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
    uid: fe8e1577-9060-4c59-95b2-3e2c51709adc
  resourceVersion: "84820"
  uid: 5339d21a-24d7-40cb-84d2-7a2ebb015355
spec:
  apply: true
  current:
    object:
      apiVersion: machineconfiguration.openshift.io/v1
      kind: KubeletConfig
      spec:
        kubeletConfig:
          evictionHard:
            imagefs.available: 10% (2)
  outdated: {}
  type: Configuration
status:
  applicationState: Applied
```

(1) The scan name of the remediation.

(2) The remediation that was added to the **KubeletConfig** objects.



NOTE

If the remediation invokes an **evictionHard** kubelet configuration, you must specify all of the **evictionHard** parameters: **memory.available**, **nodefs.available**, **nodefs.inodesFree**, **imagefs.available**, and **imagefs.inodesFree**. If you do not specify all parameters, only the specified parameters are applied and the remediation will not function properly.

2. Remove the remediation:

   a. Set **apply** to false for the remediation object:

```
$ oc -n openshift-compliance patch \
complianceremediations/one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available \
```

```
-p '{"spec":{"apply":false}}' --type=merge
```

b. Using the **scan-name**, find the **KubeletConfig** object that the remediation was applied to:

```
$ oc -n openshift-compliance get kubeletconfig \
--selector compliance.openshift.io/scan-name=one-rule-tp-node-master
```

Example output

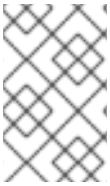
```
NAME                                AGE
compliance-operator-kubelet-master   2m34s
```

c. Manually remove the remediation, **imagefs.available: 10%**, from the **KubeletConfig** object:

```
$ oc edit -n openshift-compliance KubeletConfig compliance-operator-kubelet-master
```

IMPORTANT

All affected nodes with the remediation will be rebooted.



NOTE

You must also exclude the rule from any scheduled scans in your tailored profiles that auto-applies the remediation, otherwise, the remediation will be re-applied during the next scheduled scan.

5.6.5.12. Inconsistent ComplianceScan

The **ScanSetting** object lists the node roles that the compliance scans generated from the **ScanSetting** or **ScanSettingBinding** objects would scan. Each node role usually maps to a machine config pool.

IMPORTANT

It is expected that all machines in a machine config pool are identical and all scan results from the nodes in a pool should be identical.

If some of the results are different from others, the Compliance Operator flags a **ComplianceCheckResult** object where some of the nodes will report as **INCONSISTENT**. All **ComplianceCheckResult** objects are also labeled with **compliance.openshift.io/inconsistent-check**.

Because the number of machines in a pool might be quite large, the Compliance Operator attempts to find the most common state and list the nodes that differ from the common state. The most common state is stored in the **compliance.openshift.io/most-common-status** annotation and the annotation **compliance.openshift.io/inconsistent-source** contains pairs of **hostname:status** of check statuses that differ from the most common status. If no common state can be found, all the **hostname:status** pairs are listed in the **compliance.openshift.io/inconsistent-source annotation**.

If possible, a remediation is still created so that the cluster can converge to a compliant status. However, this might not always be possible and correcting the difference between nodes must be done manually. The compliance scan must be re-run to get a consistent result by annotating the scan with the

compliance.openshift.io/rescan= option:

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

5.6.5.13. Additional resources

- Modifying nodes.

5.6.6. Performing advanced Compliance Operator tasks

The Compliance Operator includes options for advanced users for the purpose of debugging or integration with existing tooling.

5.6.6.1. Using the ComplianceSuite and ComplianceScan objects directly

While it is recommended that users take advantage of the **ScanSetting** and **ScanSettingBinding** objects to define the suites and scans, there are valid use cases to define the **ComplianceSuite** objects directly:

- Specifying only a single rule to scan. This can be useful for debugging together with the **debug: true** attribute which increases the OpenSCAP scanner verbosity, as the debug mode tends to get quite verbose otherwise. Limiting the test to one rule helps to lower the amount of debug information.
- Providing a custom nodeSelector. In order for a remediation to be applicable, the nodeSelector must match a pool.
- Pointing the Scan to a bespoke config map with a tailoring file.
- For testing or development when the overhead of parsing profiles from bundles is not required.

The following example shows a **ComplianceSuite** that scans the worker machines with only a single rule:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:45dc...
      debug: true
      rule: xccdf_org.ssgproject.content_rule_no_direct_root_logins
      nodeSelector:
      node-role.kubernetes.io/worker: ""
```

The **ComplianceSuite** object and the **ComplianceScan** objects referred to above specify several attributes in a format that OpenSCAP expects.

To find out the profile, content, or rule values, you can start by creating a similar Suite from **ScanSetting** and **ScanSettingBinding** or inspect the objects parsed from the **ProfileBundle** objects like rules or

profiles. Those objects contain the **xccdf_org** identifiers you can use to refer to them from a **ComplianceSuite**.

5.6.6.2. Setting PriorityClass for ScanSetting scans

In large scale environments, the default **PriorityClass** object can be too low to guarantee Pods execute scans on time. For clusters that must maintain compliance or guarantee automated scanning, it is recommended to set the **PriorityClass** variable to ensure the Compliance Operator is always given priority in resource constrained situations.

Prerequisites

- Optional: You have created a **PriorityClass** object. For more information, see "Configuring priority and preemption" in the *Additional resources*.

Procedure

- Set the **PriorityClass** variable:

```
apiVersion: compliance.openshift.io/v1alpha1
strictNodeScan: true
metadata:
  name: default
  namespace: openshift-compliance
priorityClass: compliance-high-priority 1
kind: ScanSetting
showNotApplicable: false
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ''
  pvAccessModes:
    - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
    - effect: NoSchedule
      key: node.kubernetes.io/memory-pressure
      operator: Exists
schedule: 0 1 * * *
roles:
  - master
  - worker
scanTolerations:
  - operator: Exists
```

-

1 If the **PriorityClass** referenced in the **ScanSetting** cannot be found, the Operator will leave the **PriorityClass** empty, issue a warning, and continue scheduling scans without a **PriorityClass**.

Additional resources

- Configuring priority and preemption

5.6.6.3. Using raw tailored profiles

While the **TailoredProfile** CR enables the most common tailoring operations, the XCCDF standard allows even more flexibility in tailoring OpenSCAP profiles. In addition, if your organization has been using OpenScap previously, you may have an existing XCCDF tailoring file and can reuse it.

The **ComplianceSuite** object contains an optional **TailoringConfigMap** attribute that you can point to a custom tailoring file. The value of the **TailoringConfigMap** attribute is a name of a config map which must contain a key called **tailoring.xml** and the value of this key is the tailoring contents.

Procedure

1. Create the **ConfigMap** object from a file:

```
$ oc -n openshift-compliance \
create configmap nist-moderate-modified \
--from-file=tailoring.xml=/path/to/the/tailoringFile.xml
```

2. Reference the tailoring file in a scan that belongs to a suite:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  debug: true
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: registry.redhat.io/compliance/openshift-compliance-content-
rhel8@sha256:45dc...
      debug: true
  tailoringConfigMap:
      name: nist-moderate-modified
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

5.6.6.4. Performing a rescan

Typically you will want to re-run a scan on a defined schedule, like every Monday or daily. It can also be useful to re-run a scan once after fixing a problem on a node. To perform a single scan, annotate the scan with the **compliance.openshift.io/rescan=** option:

-

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

A rescan generates four additional **mc** for **rhcos-moderate** profile:

```
$ oc get mc
```

Example output

```
75-worker-scan-chronyd-or-ntpd-specify-remote-server
75-worker-scan-configure-usbguard-auditbackend
75-worker-scan-service-usbguard-enabled
75-worker-scan-usbguard-allow-hid-and-hub
```

IMPORTANT

When the scan setting **default-auto-apply** label is applied, remediations are applied automatically and outdated remediations automatically update. If there are remediations that were not applied due to dependencies, or remediations that had been outdated, rescanning applies the remediations and might trigger a reboot. Only remediations that use **MachineConfig** objects trigger reboots. If there are no updates or dependencies to be applied, no reboot occurs.

5.6.6.5. Setting custom storage size for results

While the custom resources such as **ComplianceCheckResult** represent an aggregated result of one check across all scanned nodes, it can be useful to review the raw results as produced by the scanner. The raw results are produced in the ARF format and can be large (tens of megabytes per node), it is impractical to store them in a Kubernetes resource backed by the **etcd** key-value store. Instead, every scan creates a persistent volume (PV) which defaults to 1GB size. Depending on your environment, you may want to increase the PV size accordingly. This is done using the **rawResultStorage.size** attribute that is exposed in both the **ScanSetting** and **ComplianceScan** resources.

A related parameter is **rawResultStorage.rotation** which controls how many scans are retained in the PV before the older scans are rotated. The default value is 3, setting the rotation policy to 0 disables the rotation. Given the default rotation policy and an estimate of 100MB per a raw ARF scan report, you can calculate the right PV size for your environment.

5.6.6.5.1. Using custom result storage values

Because OpenShift Container Platform can be deployed in a variety of public clouds or bare metal, the Compliance Operator cannot determine available storage configurations. By default, the Compliance Operator will try to create the PV for storing results using the default storage class of the cluster, but a custom storage class can be configured using the **rawResultStorage.StorageClassName** attribute.

IMPORTANT

If your cluster does not specify a default storage class, this attribute must be set.

Configure the **ScanSetting** custom resource to use a standard storage class and create persistent volumes that are 10GB in size and keep the last 10 results:

Example ScanSetting CR

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  storageClassName: standard
  rotation: 10
  size: 10Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
```

5.6.6.6. Applying remediations generated by suite scans

Although you can use the **autoApplyRemediations** boolean parameter in a **ComplianceSuite** object, you can alternatively annotate the object with **compliance.openshift.io/apply-remediations**. This allows the Operator to apply all of the created remediations.

Procedure

- Apply the **compliance.openshift.io/apply-remediations** annotation by running:

```
$ oc -n openshift-compliance \
annotate compliancesuites/workers-compliancesuite compliance.openshift.io/apply-remediations=
```

5.6.6.7. Automatically update remediations

In some cases, a scan with newer content might mark remediations as **OUTDATED**. As an administrator, you can apply the **compliance.openshift.io/remove-outdated** annotation to apply new remediations and remove the outdated ones.

Procedure

- Apply the **compliance.openshift.io/remove-outdated** annotation:

```
$ oc -n openshift-compliance \
annotate compliancesuites/workers-compliancesuite compliance.openshift.io/remove-outdated=
```

Alternatively, set the **autoUpdateRemediations** flag in a **ScanSetting** or **ComplianceSuite** object to update the remediations automatically.

5.6.6.8. Creating a custom SCC for the Compliance Operator

In some environments, you must create a custom Security Context Constraints (SCC) file to ensure the correct permissions are available to the Compliance Operator **api-resource-collector**.

Prerequisites

- You must have **admin** privileges.

Procedure

1. Define the SCC in a YAML file named **restricted-adjusted-compliance.yaml**:

SecurityContextConstraints object definition

```
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
kind: SecurityContextConstraints
metadata:
  name: restricted-adjusted-compliance
priority: 30 (1)
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- SETUID
- SETGID
- MKNOD
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:openshift-compliance:api-resource-collector (2)
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

(1) The priority of this SCC must be higher than any other SCC that applies to the **system:authenticated** group.

(2) Service Account used by Compliance Operator Scanner pod.

2. Create the SCC:

```
$ oc create -n openshift-compliance  -f restricted-adjusted-compliance.yaml
```

Example output

```
securitycontextconstraints.security.openshift.io/restricted-adjusted-compliance created
```

Verification

1. Verify the SCC was created:

```
$ oc get -n openshift-compliance scc restricted-adjusted-compliance
```

Example output

```
NAME                         PRIV    CAPS         SELINUX    RUNASUSER         FSGROUP
SUPGROUP   PRIORITY   READONLYROOTFS   VOLUMES
restricted-adjusted-compliance   false   <no value>   MustRunAs   MustRunAsRange
MustRunAs   RunAsAny   30         false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
```

5.6.6.9. Additional resources

- Managing security context constraints

5.6.7. Troubleshooting Compliance Operator scans

This section describes how to troubleshoot the Compliance Operator. The information can be useful either to diagnose a problem or provide information in a bug report. Some general tips:

- The Compliance Operator emits Kubernetes events when something important happens. You can either view all events in the cluster using the command:

```
 $ oc get events -n openshift-compliance
```

  Or view events for an object like a scan using the command:

```
$ oc describe -n openshift-compliance compliancescan/cis-compliance
```

- The Compliance Operator consists of several controllers, approximately one per API object. It could be useful to filter only those controllers that correspond to the API object having issues. If a **ComplianceRemediation** cannot be applied, view the messages from the **remediationctrl** controller. You can filter the messages from a single controller by parsing with **jq**:

```
$ oc -n openshift-compliance logs compliance-operator-775d7bddbd-gj58f \
    | jq -c 'select(.logger == "profilebundlectrl")'
```

- The timestamps are logged as seconds since UNIX epoch in UTC. To convert them to a human-readable date, use **date -d @timestamp --utc**, for example:

```
$ date -d @1596184628.955853 --utc
```

- Many custom resources, most importantly **ComplianceSuite** and **ScanSetting**, allow the **debug** option to be set. Enabling this option increases verbosity of the OpenSCAP scanner pods, as well as some other helper pods.
- If a single rule is passing or failing unexpectedly, it could be helpful to run a single scan or a suite with only that rule to find the rule ID from the corresponding **ComplianceCheckResult** object and use it as the **rule** attribute value in a **Scan** CR. Then, together with the **debug** option enabled, the **scanner** container logs in the scanner pod would show the raw OpenSCAP logs.

5.6.7.1. Anatomy of a scan

The following sections outline the components and stages of Compliance Operator scans.

5.6.7.1.1. Compliance sources

The compliance content is stored in **Profile** objects that are generated from a **ProfileBundle** object. The Compliance Operator creates a **ProfileBundle** object for the cluster and another for the cluster nodes.

```
$ oc get -n openshift-compliance profilebundle.compliance
```

```
$ oc get -n openshift-compliance profile.compliance
```

The **ProfileBundle** objects are processed by deployments labeled with the **Bundle** name. To troubleshoot an issue with the **Bundle**, you can find the deployment and view logs of the pods in a deployment:

```
$ oc logs -n openshift-compliance -lprofile-bundle=ocp4 -c profileparser
```

```
$ oc get -n openshift-compliance deployments,pods -lprofile-bundle=ocp4
```

```
$ oc logs -n openshift-compliance pods/<pod-name>
```

```
$ oc describe -n openshift-compliance pod/<pod-name> -c profileparser
```

5.6.7.1.2. The ScanSetting and ScanSettingBinding objects lifecycle and debugging

With valid compliance content sources, the high-level **ScanSetting** and **ScanSettingBinding** objects can be used to generate **ComplianceSuite** and **ComplianceScan** objects:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: my-companys-constraints
debug: true
# For each role, a separate scan will be created pointing
# to a node-role specified in roles
roles:
  - worker
---
```

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: my-companys-compliance-requirements
profiles:
  # Node checks
  - name: rhcos4-e8
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
  # Cluster checks
  - name: ocp4-e8
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef:
  name: my-companys-constraints
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

Both **ScanSetting** and **ScanSettingBinding** objects are handled by the same controller tagged with **logger=scansettingbindingctrl**. These objects have no status. Any issues are communicated in form of events:

```
Events:
  Type    Reason         Age    From                  Message
  ----    ------         ----   ----                  -------
  Normal  SuiteCreated   9m52s  scansettingbindingctrl  ComplianceSuite openshift-compliance/my-companys-compliance-requirements created
```

Now a **ComplianceSuite** object is created. The flow continues to reconcile the newly created **ComplianceSuite**.

5.6.7.1.3. ComplianceSuite custom resource lifecycle and debugging

The **ComplianceSuite** CR is a wrapper around **ComplianceScan** CRs. The **ComplianceSuite** CR is handled by controller tagged with **logger=suitectrl**. This controller handles creating scans from a suite, reconciling and aggregating individual Scan statuses into a single Suite status. If a suite is set to execute periodically, the **suitectrl** also handles creating a **CronJob** CR that re-runs the scans in the suite after the initial run is done:

```
$ oc get cronjobs
```

Example output

```
NAME                            SCHEDULE    SUSPEND   ACTIVE   LAST SCHEDULE   AGE
<cron_name>                       0 1 * * *   False     0        <none>          151m
```

For the most important issues, events are emitted. View them with **oc describe compliancesuites/<name>**. The **Suite** objects also have a **Status** subresource that is updated when any of **Scan** objects that belong to this suite update their **Status** subresource. After all expected scans are created, control is passed to the scan controller.

5.6.7.1.4. ComplianceScan custom resource lifecycle and debugging

The **ComplianceScan** CRs are handled by the **scanctrl** controller. This is also where the actual scans happen and the scan results are created. Each scan goes through several phases:

5.6.7.1.4.1. Pending phase

The scan is validated for correctness in this phase. If some parameters like storage size are invalid, the scan transitions to DONE with ERROR result, otherwise proceeds to the Launching phase.

5.6.7.1.4.2. Launching phase

In this phase, several config maps that contain either environment for the scanner pods or directly the script that the scanner pods will be evaluating. List the config maps:

```
$ oc -n openshift-compliance get cm \
-l compliance.openshift.io/scan-name=rhcos4-e8-worker,complianceoperator.openshift.io/scan-script=
```

These config maps will be used by the scanner pods. If you ever needed to modify the scanner behavior, change the scanner debug level or print the raw results, modifying the config maps is the way to go. Afterwards, a persistent volume claim is created per scan to store the raw ARF results:

```
$ oc get pvc -n openshift-compliance -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

The PVCs are mounted by a per-scan **ResultServer** deployment. A **ResultServer** is a simple HTTP server where the individual scanner pods upload the full ARF results to. Each server can run on a different node. The full ARF results might be very large and you cannot presume that it would be possible to create a volume that could be mounted from multiple nodes at the same time. After the scan is finished, the **ResultServer** deployment is scaled down. The PVC with the raw results can be mounted from another custom pod and the results can be fetched or inspected. The traffic between the scanner pods and the **ResultServer** is protected by mutual TLS protocols.

Finally, the scanner pods are launched in this phase; one scanner pod for a **Platform** scan instance and one scanner pod per matching node for a **node** scan instance. The per-node pods are labeled with the node name. Each pod is always labeled with the **ComplianceScan** name:

```
$ oc get pods -lcompliance.openshift.io/scan-name=rhcos4-e8-worker,workload=scanner --show-labels
```

Example output

```
NAME                                                READY   STATUS      RESTARTS   AGE   LABELS
rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod   0/2     Completed   0          39m   compliance.openshift.io/scan-name=rhcos4-e8-worker,targetNode=ip-10-0-169-90.eu-north-1.compute.internal,workload=scanner
```

+ The scan then proceeds to the Running phase.

5.6.7.1.4.3. Running phase

The running phase waits until the scanner pods finish. The following terms and processes are in use in the running phase:

- **init container**: There is one init container called **content-container**. It runs the **contentImage** container and executes a single command that copies the **contentFile** to the **/content** directory shared with the other containers in this pod.

- **scanner**: This container runs the scan. For node scans, the container mounts the node filesystem as **/host** and mounts the content delivered by the init container. The container also mounts the **entrypoint ConfigMap** created in the Launching phase and executes it. The default script in the entrypoint **ConfigMap** executes OpenSCAP and stores the result files in the **/results** directory shared between the pod's containers. Logs from this pod can be viewed to determine what the OpenSCAP scanner checked. More verbose output can be viewed with the **debug** flag.

- **logcollector**: The logcollector container waits until the scanner container finishes. Then, it uploads the full ARF results to the **ResultServer** and separately uploads the XCCDF results along with scan result and OpenSCAP result code as a **ConfigMap.** These result config maps are labeled with the scan name (**compliance.openshift.io/scan-name=rhcos4-e8-worker**):

```
$ oc describe cm/rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
```

Example output

```
Name:         rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
Namespace:    openshift-compliance
Labels:       compliance.openshift.io/scan-name-scan=rhcos4-e8-worker
              complianceoperator.openshift.io/scan-result=
Annotations:  compliance-remediations/processed:
              compliance.openshift.io/scan-error-msg:
              compliance.openshift.io/scan-result: NON-COMPLIANT
              OpenSCAP-scan-result/node: ip-10-0-169-90.eu-north-1.compute.internal

Data
====
exit-code:
----
2
results:
----
<?xml version="1.0" encoding="UTF-8"?>
...
```

Scanner pods for **Platform** scans are similar, except:

- There is one extra init container called **api-resource-collector** that reads the OpenSCAP content provided by the content-container init, container, figures out which API resources the content needs to examine and stores those API resources to a shared directory where the **scanner** container would read them from.

- The **scanner** container does not need to mount the host file system.

When the scanner pods are done, the scans move on to the Aggregating phase.

5.6.7.1.4.4. Aggregating phase

In the aggregating phase, the scan controller spawns yet another pod called the aggregator pod. Its

purpose it to take the result **ConfigMap** objects, read the results and for each check result create the corresponding Kubernetes object. If the check failure can be automatically remediated, a **ComplianceRemediation** object is created. To provide human-readable metadata for the checks and remediations, the aggregator pod also mounts the OpenSCAP content using an init container.

When a config map is processed by an aggregator pod, it is labeled the **compliance-remediations/processed** label. The result of this phase are **ComplianceCheckResult** objects:

```
$ oc get compliancecheckresults -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

Example output

```
NAME                                           STATUS   SEVERITY
rhcos4-e8-worker-accounts-no-uid-except-zero    PASS     high
rhcos4-e8-worker-audit-rules-dac-modification-chmod     FAIL     medium
```

and **ComplianceRemediation** objects:

```
$ oc get complianceremediations -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

Example output

```
NAME                                           STATE
rhcos4-e8-worker-audit-rules-dac-modification-chmod     NotApplied
rhcos4-e8-worker-audit-rules-dac-modification-chown     NotApplied
rhcos4-e8-worker-audit-rules-execution-chcon            NotApplied
rhcos4-e8-worker-audit-rules-execution-restorecon       NotApplied
rhcos4-e8-worker-audit-rules-execution-semanage         NotApplied
rhcos4-e8-worker-audit-rules-execution-setfiles         NotApplied
```

After these CRs are created, the aggregator pod exits and the scan moves on to the Done phase.

5.6.7.1.4.5. Done phase

In the final scan phase, the scan resources are cleaned up if needed and the **ResultServer** deployment is either scaled down (if the scan was one-time) or deleted if the scan is continuous; the next scan instance would then recreate the deployment again.

It is also possible to trigger a re-run of a scan in the Done phase by annotating it:

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

After the scan reaches the Done phase, nothing else happens on its own unless the remediations are set to be applied automatically with **autoApplyRemediations: true**. The OpenShift Container Platform administrator would now review the remediations and apply them as needed. If the remediations are set to be applied automatically, the **ComplianceSuite** controller takes over in the Done phase, pauses the machine config pool to which the scan maps to and applies all the remediations in one go. If a remediation is applied, the **ComplianceRemediation** controller takes over.

5.6.7.1.5. ComplianceRemediation controller lifecycle and debugging

The example scan has reported some findings. One of the remediations can be enabled by toggling its **apply** attribute to **true**:

```
$ oc patch complianceremediations/rhcos4-e8-worker-audit-rules-dac-modification-chmod --patch
'{"spec":{"apply":true}}' --type=merge
```

The **ComplianceRemediation** controller (**logger=remediationctrl**) reconciles the modified object. The result of the reconciliation is change of status of the remediation object that is reconciled, but also a change of the rendered per-suite **MachineConfig** object that contains all the applied remediations.

The **MachineConfig** object always begins with **75-** and is named after the scan and the suite:

```
$ oc get mc | grep 75-
```

Example output

```
75-rhcos4-e8-worker-my-companys-compliance-requirements                 3.2.0
2m46s
```

The remediations the **mc** currently consists of are listed in the machine config's annotations:

```
$ oc describe mc/75-rhcos4-e8-worker-my-companys-compliance-requirements
```

Example output

```
Name:         75-rhcos4-e8-worker-my-companys-compliance-requirements
Labels:       machineconfiguration.openshift.io/role=worker
Annotations:  remediation/rhcos4-e8-worker-audit-rules-dac-modification-chmod:
```

The **ComplianceRemediation** controller's algorithm works like this:

- All currently applied remediations are read into an initial remediation set.
- If the reconciled remediation is supposed to be applied, it is added to the set.
- A **MachineConfig** object is rendered from the set and annotated with names of remediations in the set. If the set is empty (the last remediation was unapplied), the rendered **MachineConfig** object is removed.
- If and only if the rendered machine config is different from the one already applied in the cluster, the applied MC is updated (or created, or deleted).
- Creating or modifying a **MachineConfig** object triggers a reboot of nodes that match the **machineconfiguration.openshift.io/role** label - see the Machine Config Operator documentation for more details.

The remediation loop ends once the rendered machine config is updated, if needed, and the reconciled remediation object status is updated. In our case, applying the remediation would trigger a reboot. After the reboot, annotate the scan to re-run it:

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

The scan will run and finish. Check for the remediation to pass:

```
$ oc -n openshift-compliance \
get compliancecheckresults/rhcos4-e8-worker-audit-rules-dac-modification-chmod
```

Example output

```
NAME                                                STATUS   SEVERITY
rhcos4-e8-worker-audit-rules-dac-modification-chmod   PASS     medium
```

5.6.7.1.6. Useful labels

Each pod that is spawned by the Compliance Operator is labeled specifically with the scan it belongs to and the work it does. The scan identifier is labeled with the **compliance.openshift.io/scan-name** label. The workload identifier is labeled with the **workload** label.

The Compliance Operator schedules the following workloads:

- **scanner**: Performs the compliance scan.
- **resultserver**: Stores the raw results for the compliance scan.
- **aggregator**: Aggregates the results, detects inconsistencies and outputs result objects (checkresults and remediations).
- **suitererunner**: Will tag a suite to be re-run (when a schedule is set).
- **profileparser**: Parses a datastream and creates the appropriate profiles, rules and variables.

When debugging and logs are required for a certain workload, run:

```
$ oc logs -l workload=<workload_name> -c <container_name>
```

5.6.7.2. Increasing Compliance Operator resource limits

In some cases, the Compliance Operator might require more memory than the default limits allow. The best way to mitigate this issue is to set custom resource limits.

To increase the default memory and CPU limits of scanner pods, see *`ScanSetting` Custom resource*.

Procedure

1. To increase the Operator's memory limits to 500 Mi, create the following patch file named **co-memlimit-patch.yaml**:

```
spec:
  config:
    resources:
      limits:
        memory: 500Mi
```

2. Apply the patch file:

```
$ oc patch sub compliance-operator -nopenshift-compliance --patch-file co-memlimit-
patch.yaml --type=merge
```

5.6.7.3. Configuring Operator resource constraints

The **resources** field defines Resource Constraints for all the containers in the Pod created by the Operator Lifecycle Manager (OLM).

NOTE

Resource Constraints applied in this process overwrites the existing resource constraints.

Procedure

- Inject a request of 0.25 cpu and 64 Mi of memory, and a limit of 0.5 cpu and 128 Mi of memory in each container by editing the **Subscription** object:

```
kind: Subscription
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  package: package-name
  channel: stable
  config:
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

5.6.7.4. Configuring ScanSetting resources

When using the Compliance Operator in a cluster that contains more than 500 MachineConfigs, the **ocp4-pci-dss-api-checks-pod** pod may pause in the **init** phase when performing a **Platform** scan.

NOTE

Resource constraints applied in this process overwrites the existing resource constraints.

Procedure

1. Confirm the **ocp4-pci-dss-api-checks-pod** pod is stuck in the **Init:OOMKilled** status:

```
$ oc get pod ocp4-pci-dss-api-checks-pod -w
```

Example output

```
NAME                          READY   STATUS     RESTARTS        AGE
ocp4-pci-dss-api-checks-pod   0/2     Init:1/2   8 (5m56s ago)   25m
ocp4-pci-dss-api-checks-pod   0/2     Init:OOMKilled   8 (6m19s ago)   26m
```

2. Edit the **scanLimits** attribute in the **ScanSetting** CR to increase the available memory for the **ocp4-pci-dss-api-checks-pod** pod:

```
timeout: 30m
strictNodeScan: true
metadata:
  name: default
  namespace: openshift-compliance
kind: ScanSetting
showNotApplicable: false
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ''
  pvAccessModes:
    - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
    - effect: NoSchedule
      key: node.kubernetes.io/memory-pressure
      operator: Exists
schedule: 0 1 * * *
roles:
  - master
  - worker
apiVersion: compliance.openshift.io/v1alpha1
maxRetryOnTimeout: 3
scanTolerations:
  - operator: Exists
scanLimits:
  memory: 1024Mi (1)
```

(1) The default setting is **500Mi**.

3. Apply the **ScanSetting** CR to your cluster:

```
$ oc apply -f scansetting.yaml
```

5.6.7.5. Configuring ScanSetting timeout

The **ScanSetting** object has a timeout option that can be specified in the **ComplianceScanSetting** object as a duration string, such as **1h30m**. If the scan does not finish within the specified timeout, the scan reattempts until the **maxRetryOnTimeout** limit is reached.

Procedure

- To set a **timeout** and **maxRetryOnTimeout** in ScanSetting, modify an existing **ScanSetting** object:

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  rotation: 3
  size: 1Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
timeout: '10m0s' 1
maxRetryOnTimeout: 3 2
```

1. The **timeout** variable is defined as a duration string, such as **1h30m**. The default value is **30m**. To disable the timeout, set the value to **0s**.
2. The **maxRetryOnTimeout** variable defines how many times a retry is attempted. The default value is **3**.

5.6.7.6. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal. From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in OpenShift Cluster Manager Hybrid Cloud Console. Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a Jira issue for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

5.6.8. Using the oc-compliance plugin

Although the Compliance Operator automates many of the checks and remediations for the cluster, the full process of bringing a cluster into compliance often requires administrator interaction with the Compliance Operator API and other components. The **oc-compliance** plugin makes the process easier.

5.6.8.1. Installing the oc-compliance plugin

Procedure

1. Extract the **oc-compliance** image to get the **oc-compliance** binary:

```
$ podman run --rm -v ~/.local/bin:/mnt/out:Z registry.redhat.io/compliance/oc-compliance-rhel8:stable /bin/cp /usr/bin/oc-compliance /mnt/out/
```

Example output

```
W0611 20:35:46.486903   11354 manifest.go:440] Chose linux/amd64 manifest from the manifest list.
```

You can now run **oc-compliance**.

5.6.8.2. Fetching raw results

When a compliance scan finishes, the results of the individual checks are listed in the resulting **ComplianceCheckResult** custom resource (CR). However, an administrator or auditor might require the complete details of the scan. The OpenSCAP tool creates an Advanced Recording Format (ARF) formatted file with the detailed results. This ARF file is too large to store in a config map or other standard Kubernetes resource, so a persistent volume (PV) is created to contain it.

Procedure

- Fetching the results from the PV with the Compliance Operator is a four-step process. However, with the **oc-compliance** plugin, you can use a single command:

```
$ oc compliance fetch-raw <object-type> <object-name> -o <output-path>
```

- **<object-type>** can be either **scansettingbinding**, **compliancescan** or **compliancesuite**, depending on which of these objects the scans were launched with.
- **<object-name>** is the name of the binding, suite, or scan object to gather the ARF file for, and **<output-path>** is the local directory to place the results.
  For example:

```
$ oc compliance fetch-raw scansettingbindings my-binding -o /tmp/
```

Example output

```
Fetching results for my-binding scans: ocp4-cis, ocp4-cis-node-worker, ocp4-cis-node-master
Fetching raw compliance results for scan 'ocp4-cis'.....
The raw compliance results are available in the following directory: /tmp/ocp4-cis
Fetching raw compliance results for scan 'ocp4-cis-node-worker'.....
```

```
The raw compliance results are available in the following directory: /tmp/ocp4-cis-node-worker
Fetching raw compliance results for scan 'ocp4-cis-node-master'.....
The raw compliance results are available in the following directory: /tmp/ocp4-cis-node-master
```

View the list of files in the directory:

```
$ ls /tmp/ocp4-cis-node-master/
```

Example output

```
ocp4-cis-node-master-ip-10-0-128-89.ec2.internal-pod.xml.bzip2  ocp4-cis-node-master-ip-10-0-150-5.ec2.internal-pod.xml.bzip2  ocp4-cis-node-master-ip-10-0-163-32.ec2.internal-pod.xml.bzip2
```

Extract the results:

```
$ bunzip2 -c resultsdir/worker-scan/worker-scan-stage-459-tqkg7-compute-0-pod.xml.bzip2 > resultsdir/worker-scan/worker-scan-ip-10-0-170-231.us-east-2.compute.internal-pod.xml
```

View the results:

```
$ ls resultsdir/worker-scan/
```

Example output

```
worker-scan-ip-10-0-170-231.us-east-2.compute.internal-pod.xml
worker-scan-stage-459-tqkg7-compute-0-pod.xml.bzip2
worker-scan-stage-459-tqkg7-compute-1-pod.xml.bzip2
```

5.6.8.3. Re-running scans

Although it is possible to run scans as scheduled jobs, you must often re-run a scan on demand, particularly after remediations are applied or when other changes to the cluster are made.

Procedure

- Rerunning a scan with the Compliance Operator requires use of an annotation on the scan object. However, with the **oc-compliance** plugin you can rerun a scan with a single command. Enter the following command to rerun the scans for the **ScanSettingBinding** object named **my-binding**:

  ```
  $ oc compliance rerun-now scansettingbindings my-binding
  ```

  Example output

  ```
  Rerunning scans from 'my-binding': ocp4-cis
  Re-running scan 'openshift-compliance/ocp4-cis'
  ```

5.6.8.4. Using ScanSettingBinding custom resources

When using the **ScanSetting** and **ScanSettingBinding** custom resources (CRs) that the Compliance Operator provides, it is possible to run scans for multiple profiles while using a common set of scan options, such as **schedule**, **machine roles**, **tolerations**, and so on. While that is easier than working with multiple **ComplianceSuite** or **ComplianceScan** objects, it can confuse new users.

The **oc compliance bind** subcommand helps you create a **ScanSettingBinding** CR.

Procedure

1. Run:

```
$ oc compliance bind [--dry-run] -N <binding name> [-S <scansetting name>]
<objtype/objname> [..<objtype/objname>]
```

- If you omit the **-S** flag, the **default** scan setting provided by the Compliance Operator is used.
- The object type is the Kubernetes object type, which can be **profile** or **tailoredprofile**. More than one object can be provided.
- The object name is the name of the Kubernetes resource, such as **.metadata.name**.
- Add the **--dry-run** option to display the YAML file of the objects that are created.
  For example, given the following profiles and scan settings:

```
$ oc get profile.compliance -n openshift-compliance
```

Example output

```
NAME                   AGE     VERSION
ocp4-cis               3h49m   1.5.0
ocp4-cis-1-4           3h49m   1.4.0
ocp4-cis-1-5           3h49m   1.5.0
ocp4-cis-node          3h49m   1.5.0
ocp4-cis-node-1-4      3h49m   1.4.0
ocp4-cis-node-1-5      3h49m   1.5.0
ocp4-e8                3h49m
ocp4-high              3h49m   Revision 4
ocp4-high-node         3h49m   Revision 4
ocp4-high-node-rev-4   3h49m   Revision 4
ocp4-high-rev-4        3h49m   Revision 4
ocp4-moderate          3h49m   Revision 4
ocp4-moderate-node     3h49m   Revision 4
ocp4-moderate-node-rev-4   3h49m   Revision 4
ocp4-moderate-rev-4    3h49m   Revision 4
ocp4-nerc-cip          3h49m
ocp4-nerc-cip-node     3h49m
ocp4-pci-dss           3h49m   3.2.1
ocp4-pci-dss-3-2       3h49m   3.2.1
ocp4-pci-dss-4-0       3h49m   4.0.0
ocp4-pci-dss-node      3h49m   3.2.1
ocp4-pci-dss-node-3-2  3h49m   3.2.1
ocp4-pci-dss-node-4-0  3h49m   4.0.0
ocp4-stig              3h49m   V2R1
```

```
ocp4-stig-node            3h49m   V2R1
ocp4-stig-node-v1r1       3h49m   V1R1
ocp4-stig-node-v2r1       3h49m   V2R1
ocp4-stig-v1r1            3h49m   V1R1
ocp4-stig-v2r1            3h49m   V2R1
rhcos4-e8                 3h49m
rhcos4-high               3h49m   Revision 4
rhcos4-high-rev-4         3h49m   Revision 4
rhcos4-moderate           3h49m   Revision 4
rhcos4-moderate-rev-4     3h49m   Revision 4
rhcos4-nerc-cip           3h49m
rhcos4-stig               3h49m   V2R1
rhcos4-stig-v1r1          3h49m   V1R1
rhcos4-stig-v2r1          3h49m   V2R1
```

```
$ oc get scansettings -n openshift-compliance
```

Example output

```
NAME                 AGE
default              10m
default-auto-apply   10m
```

2. To apply the **default** settings to the **ocp4-cis** and **ocp4-cis-node** profiles, run:

```
$ oc compliance bind -N my-binding profile/ocp4-cis profile/ocp4-cis-node
```

Example output

```
Creating ScanSettingBinding my-binding
```

After the **ScanSettingBinding** CR is created, the bound profile begins scanning for both profiles with the related settings. Overall, this is the fastest way to begin scanning with the Compliance Operator.

5.6.8.5. Printing controls

Compliance standards are generally organized into a hierarchy as follows:

- A benchmark is the top-level definition of a set of controls for a particular standard. For example, FedRAMP Moderate or Center for Internet Security (CIS) v.1.6.0.
- A control describes a family of requirements that must be met in order to be in compliance with the benchmark. For example, FedRAMP AC-01 (access control policy and procedures).
- A rule is a single check that is specific for the system being brought into compliance, and one or more of these rules map to a control.
- The Compliance Operator handles the grouping of rules into a profile for a single benchmark. It can be difficult to determine which controls that the set of rules in a profile satisfy.

Procedure

- The **oc compliance controls** subcommand provides a report of the standards and controls that a given profile satisfies:

```
$ oc compliance controls profile ocp4-cis-node
```

Example output

```
+-----------+----------+
| FRAMEWORK | CONTROLS |
+-----------+----------+
| CIS-OCP   | 1.1.1    |
+           +----------+
|           | 1.1.10   |
+           +----------+
|           | 1.1.11   |
+           +----------+
...
```

5.6.8.6. Fetching compliance remediation details

The Compliance Operator provides remediation objects that are used to automate the changes required to make the cluster compliant. The **fetch-fixes** subcommand can help you understand exactly which configuration remediations are used. Use the **fetch-fixes** subcommand to extract the remediation objects from a profile, rule, or **ComplianceRemediation** object into a directory to inspect.

Procedure

1. View the remediations for a profile:

```
$ oc compliance fetch-fixes profile ocp4-cis -o /tmp
```

Example output

```
No fixes to persist for rule 'ocp4-api-server-api-priority-flowschema-catch-all' 1
No fixes to persist for rule 'ocp4-api-server-api-priority-gate-enabled'
No fixes to persist for rule 'ocp4-api-server-audit-log-maxbackup'
Persisted rule fix to /tmp/ocp4-api-server-audit-log-maxsize.yaml
No fixes to persist for rule 'ocp4-api-server-audit-log-path'
No fixes to persist for rule 'ocp4-api-server-auth-mode-no-aa'
No fixes to persist for rule 'ocp4-api-server-auth-mode-node'
No fixes to persist for rule 'ocp4-api-server-auth-mode-rbac'
No fixes to persist for rule 'ocp4-api-server-basic-auth'
No fixes to persist for rule 'ocp4-api-server-bind-address'
No fixes to persist for rule 'ocp4-api-server-client-ca'
Persisted rule fix to /tmp/ocp4-api-server-encryption-provider-cipher.yaml
Persisted rule fix to /tmp/ocp4-api-server-encryption-provider-config.yaml
```

1 The **No fixes to persist** warning is expected whenever there are rules in a profile that do not have a corresponding remediation, because either the rule cannot be remediated automatically or a remediation was not provided.

2. You can view a sample of the YAML file. The **head** command will show you the first 10 lines:

```
$ head /tmp/ocp4-api-server-audit-log-maxsize.yaml
```

Example output

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  maximumFileSizeMegabytes: 100
```

3. View the remediation from a **ComplianceRemediation** object created after a scan:

```
$ oc get complianceremediations -n openshift-compliance
```

Example output

```
NAME                                              STATE
ocp4-cis-api-server-encryption-provider-cipher    NotApplied
ocp4-cis-api-server-encryption-provider-config    NotApplied
```

```
$ oc compliance fetch-fixes complianceremediations ocp4-cis-api-server-encryption-provider-cipher -o /tmp
```

Example output

```
Persisted compliance remediation fix to /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

4. You can view a sample of the YAML file. The **head** command will show you the first 10 lines:

```
$ head /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

Example output

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  encryption:
    type: aescbc
```

WARNING

Use caution before applying remediations directly. Some remediations might not be applicable in bulk, such as the usbguard rules in the moderate profile. In these cases, allow the Compliance Operator to apply the rules because it addresses the dependencies and ensures that the cluster remains in a good state.

5.6.8.7. Viewing ComplianceCheckResult object details

When scans are finished running, **ComplianceCheckResult** objects are created for the individual scan rules. The **view-result** subcommand provides a human-readable output of the **ComplianceCheckResult** object details.

Procedure

- Run:

```
$ oc compliance view-result ocp4-cis-scheduler-no-bind-address
```

CHAPTER 6. FILE INTEGRITY OPERATOR

6.1. FILE INTEGRITY OPERATOR OVERVIEW

The File Integrity Operator continually runs file integrity checks on the cluster nodes. It deploys a DaemonSet that initializes and runs privileged Advanced Intrusion Detection Environment (AIDE) containers on each node, providing a log of files that have been modified since the initial run of the DaemonSet pods.

For the latest updates, see the File Integrity Operator release notes.

Installing the File Integrity Operator

Updating the File Integrity Operator

Understanding the File Integrity Operator

Configuring the Custom File Integrity Operator

Performing advanced Custom File Integrity Operator tasks

Troubleshooting the File Integrity Operator

6.2. FILE INTEGRITY OPERATOR RELEASE NOTES

The File Integrity Operator for OpenShift Container Platform continually runs file integrity checks on RHCOS nodes.

These release notes track the development of the File Integrity Operator in the OpenShift Container Platform.

For an overview of the File Integrity Operator, see Understanding the File Integrity Operator.

To access the latest release, see Updating the File Integrity Operator.

6.2.1. OpenShift File Integrity Operator 1.3.5

The following advisory is available for the OpenShift File Integrity Operator 1.3.5:

- RHBA-2024:10366 OpenShift File Integrity Operator Update

This update includes upgraded dependencies in underlying base images.

6.2.2. OpenShift File Integrity Operator 1.3.4

The following advisory is available for the OpenShift File Integrity Operator 1.3.4:

- RHBA-2024:2946 OpenShift File Integrity Operator Bug Fix and Enhancement Update

6.2.2.1. Bug fixes

Previously, File Integrity Operator would issue a **NodeHasIntegrityFailure** alert due to multus certificate rotation. With this release, the alert and failing status are now correctly triggered. (**OCPBUGS-31257**)

6.2.3. OpenShift File Integrity Operator 1.3.3

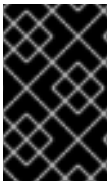
The following advisory is available for the OpenShift File Integrity Operator 1.3.3:

- RHBA-2023:5652 OpenShift File Integrity Operator Bug Fix and Enhancement Update

This update addresses a CVE in an underlying dependency.

6.2.3.1. New features and enhancements

- You can install and use the File Integrity Operator in an OpenShift Container Platform cluster running in FIPS mode.



IMPORTANT

To enable FIPS mode for your cluster, you must run the installation program from a RHEL computer configured to operate in FIPS mode. For more information about configuring FIPS mode on RHEL, see (Installing the system in FIPS mode)

6.2.3.2. Bug fixes

- Previously, some FIO pods with private default mount propagation in combination with **hostPath: path:** / volume mounts would break the CSI driver relying on multipath. This problem has been fixed and the CSI driver works correctly. (Some OpenShift Operator pods blocking unmounting of CSI volumes when multipath is in use)
- This update resolves CVE-2023-39325. (**CVE-2023-39325**)

6.2.4. OpenShift File Integrity Operator 1.3.2

The following advisory is available for the OpenShift File Integrity Operator 1.3.2:

- RHBA-2023:5107 OpenShift File Integrity Operator Bug Fix Update

This update addresses a CVE in an underlying dependency.

6.2.5. OpenShift File Integrity Operator 1.3.1

The following advisory is available for the OpenShift File Integrity Operator 1.3.1:

- RHBA-2023:3600 OpenShift File Integrity Operator Bug Fix Update

6.2.5.1. New features and enhancements

- FIO now includes kubelet certificates as default files, excluding them from issuing warnings when they're managed by OpenShift Container Platform. (**OCPBUGS-14348**)
- FIO now correctly directs email to the address for Red Hat Technical Support. (**OCPBUGS-5023**)

6.2.5.2. Bug fixes

- Previously, FIO would not clean up **FileIntegrityNodeStatus** CRDs when nodes are removed from the cluster. FIO has been updated to correctly clean up node status CRDs on node removal. (OCPBUGS-4321)
- Previously, FIO would also erroneously indicate that new nodes failed integrity checks. FIO has been updated to correctly show node status CRDs when adding new nodes to the cluster. This provides correct node status notifications. (OCPBUGS-8502)
- Previously, when FIO was reconciling **FileIntegrity** CRDs, it would pause scanning until the reconciliation was done. This caused an overly aggressive re-initiatization process on nodes not impacted by the reconciliation. This problem also resulted in unnecessary daemonsets for machine config pools which are unrelated to the **FileIntegrity** being changed. FIO correctly handles these cases and only pauses AIDE scanning for nodes that are affected by file integrity changes. (CMP-1097)

6.2.5.3. Known Issues

In FIO 1.3.1, increasing nodes in IBM Z clusters might result in **Failed** File Integrity node status. For more information, see Adding nodes in IBM Power clusters can result in failed File Integrity node status .

6.2.6. OpenShift File Integrity Operator 1.2.1

The following advisory is available for the OpenShift File Integrity Operator 1.2.1:

- RHBA-2023:1684 OpenShift File Integrity Operator Bug Fix Update
- This release includes updated container dependencies.

6.2.7. OpenShift File Integrity Operator 1.2.0

The following advisory is available for the OpenShift File Integrity Operator 1.2.0:

- RHBA-2023:1273 OpenShift File Integrity Operator Enhancement Update

6.2.7.1. New features and enhancements

- The File Integrity Operator Custom Resource (CR) now contains an **initialDelay** feature that specifies the number of seconds to wait before starting the first AIDE integrity check. For more information, see Creating the FileIntegrity custom resource.
- The File Integrity Operator is now stable and the release channel is upgraded to **stable**. Future releases will follow Semantic Versioning. To access the latest release, see Updating the File Integrity Operator.

6.2.8. OpenShift File Integrity Operator 1.0.0

The following advisory is available for the OpenShift File Integrity Operator 1.0.0:

- RHBA-2023:0037 OpenShift File Integrity Operator Bug Fix Update

6.2.9. OpenShift File Integrity Operator 0.1.32

The following advisory is available for the OpenShift File Integrity Operator 0.1.32:

- RHBA-2022:7095 OpenShift File Integrity Operator Bug Fix Update

6.2.9.1. Bug fixes

- Previously, alerts issued by the File Integrity Operator did not set a namespace, making it difficult to understand from which namespace the alert originated. Now, the Operator sets the appropriate namespace, providing more information about the alert. (BZ#2112394)

- Previously, The File Integrity Operator did not update the metrics service on Operator startup, causing the metrics targets to be unreachable. With this release, the File Integrity Operator now ensures the metrics service is updated on Operator startup. (BZ#2115821)

6.2.10. OpenShift File Integrity Operator 0.1.30

The following advisory is available for the OpenShift File Integrity Operator 0.1.30:

- RHBA-2022:5538 OpenShift File Integrity Operator Bug Fix and Enhancement Update

6.2.10.1. New features and enhancements

- The File Integrity Operator is now supported on the following architectures:
  - IBM Power
  - IBM Z and LinuxONE

6.2.10.2. Bug fixes

- Previously, alerts issued by the File Integrity Operator did not set a namespace, making it difficult to understand where the alert originated. Now, the Operator sets the appropriate namespace, increasing understanding of the alert. (BZ#2101393)

6.2.11. OpenShift File Integrity Operator 0.1.24

The following advisory is available for the OpenShift File Integrity Operator 0.1.24:

- RHBA-2022:1331 OpenShift File Integrity Operator Bug Fix

6.2.11.1. New features and enhancements

- You can now configure the maximum number of backups stored in the **FileIntegrity** Custom Resource (CR) with the **config.maxBackups** attribute. This attribute specifies the number of AIDE database and log backups left over from the **re-init** process to keep on the node. Older backups beyond the configured number are automatically pruned. The default is set to five backups.

6.2.11.2. Bug fixes

- Previously, upgrading the Operator from versions older than 0.1.21 to 0.1.22 could cause the **re-init** feature to fail. This was a result of the Operator failing to update **configMap** resource labels. Now, upgrading to the latest version fixes the resource labels. (BZ#2049206)

- Previously, when enforcing the default **configMap** script contents, the wrong data keys were compared. This resulted in the **aide-reinit** script not being updated properly after an Operator

upgrade, and caused the **re-init** process to fail. Now, **daemonSets** run to completion and the AIDE database **re-init** process executes successfully. (BZ#2072058)

6.2.12. OpenShift File Integrity Operator 0.1.22

The following advisory is available for the OpenShift File Integrity Operator 0.1.22:

- RHBA-2022:0142 OpenShift File Integrity Operator Bug Fix

6.2.12.1. Bug fixes

- Previously, a system with a File Integrity Operator installed might interrupt the OpenShift Container Platform update, due to the **/etc/kubernetes/aide.reinit** file. This occurred if the **/etc/kubernetes/aide.reinit** file was present, but later removed prior to the **ostree** validation. With this update, **/etc/kubernetes/aide.reinit** is moved to the **/run** directory so that it does not conflict with the OpenShift Container Platform update. (BZ#2033311)

6.2.13. OpenShift File Integrity Operator 0.1.21

The following advisory is available for the OpenShift File Integrity Operator 0.1.21:

- RHBA-2021:4631 OpenShift File Integrity Operator Bug Fix and Enhancement Update

6.2.13.1. New features and enhancements

- The metrics related to **FileIntegrity** scan results and processing metrics are displayed on the monitoring dashboard on the web console. The results are labeled with the prefix of **file_integrity_operator_**.
- If a node has an integrity failure for more than 1 second, the default **PrometheusRule** provided in the operator namespace alerts with a warning.
- The following dynamic Machine Config Operator and Cluster Version Operator related filepaths are excluded from the default AIDE policy to help prevent false positives during node updates:
  - /etc/machine-config-daemon/currentconfig
  - /etc/pki/ca-trust/extracted/java/cacerts
  - /etc/cvo/updatepayloads
  - /root/.kube
- The AIDE daemon process has stability improvements over v0.1.16, and is more resilient to errors that might occur when the AIDE database is initialized.

6.2.13.2. Bug fixes

- Previously, when the Operator automatically upgraded, outdated daemon sets were not removed. With this release, outdated daemon sets are removed during the automatic upgrade.

6.2.14. Additional resources

- Understanding the File Integrity Operator

6.3. FILE INTEGRITY OPERATOR SUPPORT

6.3.1. File Integrity Operator lifecycle

The File Integrity Operator is a "Rolling Stream" Operator, meaning updates are available asynchronously of OpenShift Container Platform releases. For more information, see OpenShift Operator Life Cycles on the Red Hat Customer Portal.

6.3.2. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal . From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in OpenShift Cluster Manager Hybrid Cloud Console. Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a Jira issue for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

6.4. INSTALLING THE FILE INTEGRITY OPERATOR

6.4.1. Installing the File Integrity Operator using the web console

Prerequisites

- You must have **admin** privileges.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Search for the File Integrity Operator, then click **Install**.
3. Keep the default selection of **Installation mode** and **namespace** to ensure that the Operator will be installed to the **openshift-file-integrity** namespace.
4. Click **Install**.

Verification

To confirm that the installation is successful:

1. Navigate to the **Operators → Installed Operators** page.
2. Check that the Operator is installed in the **openshift-file-integrity** namespace and its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.
2. Navigate to the **Workloads → Pods** page and check the logs in any pods in the **openshift-file-integrity** project that are reporting issues.

6.4.2. Installing the File Integrity Operator using the CLI

Prerequisites

- You must have **admin** privileges.

Procedure

1. Create a **Namespace** object YAML file by running:

```
$ oc create -f <file-name>.yaml
```

Example output

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-file-integrity
```

(1) In OpenShift Container Platform 4.13, the pod security label must be set to **privileged** at the namespace level.

2. Create the **OperatorGroup** object YAML file:

```
$ oc create -f <file-name>.yaml
```

Example output

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

3. Create the **Subscription** object YAML file:

```
$ oc create -f <file-name>.yaml
```

Example output

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

Verification

1. Verify the installation succeeded by inspecting the CSV file:

```
$ oc get csv -n openshift-file-integrity
```

2. Verify that the File Integrity Operator is up and running:

```
$ oc get deploy -n openshift-file-integrity
```

6.4.3. Additional resources

- The File Integrity Operator is supported in a restricted network environment. For more information, see Using Operator Lifecycle Manager on restricted networks .

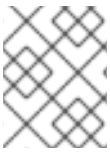
6.5. UPDATING THE FILE INTEGRITY OPERATOR

As a cluster administrator, you can update the File Integrity Operator on your OpenShift Container Platform cluster.

6.5.1. Preparing for an Operator update

The subscription of an installed Operator specifies an update channel that tracks and receives updates for the Operator. You can change the update channel to start tracking and receiving updates from a newer channel.

The names of update channels in a subscription can differ between Operators, but the naming scheme typically follows a common convention within a given Operator. For example, channel names might follow a minor release update stream for the application provided by the Operator (**1.2**, **1.3**) or a release frequency (**stable**, **fast**).



NOTE

You cannot change installed Operators to a channel that is older than the current channel.

Red Hat Customer Portal Labs include the following application that helps administrators prepare to update their Operators:

- Red Hat OpenShift Container Platform Operator Update Information Checker

You can use the application to search for Operator Lifecycle Manager-based Operators and verify the available Operator version per update channel across different versions of OpenShift Container Platform. Cluster Version Operator-based Operators are not included.

6.5.2. Changing the update channel for an Operator

You can change the update channel for an Operator by using the OpenShift Container Platform web console.

TIP

If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new Operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

Prerequisites

- An Operator previously installed using Operator Lifecycle Manager (OLM).

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Operators → Installed Operators**.
2. Click the name of the Operator you want to change the update channel for.
3. Click the **Subscription** tab.
4. Click the name of the update channel under **Update channel**.
5. Click the newer update channel that you want to change to, then click **Save**.
6. For subscriptions with an **Automatic** approval strategy, the update begins automatically. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the update. When complete, the status changes to **Succeeded** and **Up to date**.
   For subscriptions with a **Manual** approval strategy, you can manually approve the update from the **Subscription** tab.

6.5.3. Manually approving a pending Operator update

If an installed Operator has the approval strategy in its subscription set to **Manual**, when new updates are released in its current update channel, the update must be manually approved before installation can begin.

Prerequisites

- An Operator previously installed using Operator Lifecycle Manager (OLM).

Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.

2. Operators that have a pending update display a status with **Upgrade available**. Click the name of the Operator you want to update.

3. Click the **Subscription** tab. Any updates requiring approval are displayed next to **Upgrade status**. For example, it might display **1 requires approval**.

4. Click **1 requires approval**, then click **Preview Install Plan**.

5. Review the resources that are listed as available for update. When satisfied, click **Approve**.

6. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the update. When complete, the status changes to **Succeeded** and **Up to date**.

6.6. UNDERSTANDING THE FILE INTEGRITY OPERATOR

The File Integrity Operator is an OpenShift Container Platform Operator that continually runs file integrity checks on the cluster nodes. It deploys a daemon set that initializes and runs privileged advanced intrusion detection environment (AIDE) containers on each node, providing a status object with a log of files that are modified during the initial run of the daemon set pods.

IMPORTANT

Currently, only Red Hat Enterprise Linux CoreOS (RHCOS) nodes are supported.

6.6.1. Creating the FileIntegrity custom resource

An instance of a **FileIntegrity** custom resource (CR) represents a set of continuous file integrity scans for one or more nodes.

Each **FileIntegrity** CR is backed by a daemon set running AIDE on the nodes matching the **FileIntegrity** CR specification.

Procedure

1. Create the following example **FileIntegrity** CR named **worker-fileintegrity.yaml** to enable scans on worker nodes:

Example FileIntegrity CR

```
apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: worker-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector: 1
    node-role.kubernetes.io/worker: ""
  tolerations: 2
  - key: "myNode"
    operator: "Exists"
    effect: "NoSchedule"
  config: 3
    name: "myconfig"
    namespace: "openshift-file-integrity"
```

```
    key: "config"
    gracePeriod: 20 4
    maxBackups: 5 5
    initialDelay: 60 6
  debug: false
status:
  phase: Active 7
```

1. Defines the selector for scheduling node scans.
2. Specify **tolerations** to schedule on nodes with custom taints. When not specified, a default toleration allowing running on main and infra nodes is applied.
3. Define a **ConfigMap** containing an AIDE configuration to use.
4. The number of seconds to pause in between AIDE integrity checks. Frequent AIDE checks on a node might be resource intensive, so it can be useful to specify a longer interval. Default is 900 seconds (15 minutes).
5. The maximum number of AIDE database and log backups (leftover from the re-init process) to keep on a node. Older backups beyond this number are automatically pruned by the daemon. Default is set to 5.
6. The number of seconds to wait before starting the first AIDE integrity check. Default is set to 0.
7. The running status of the **FileIntegrity** instance. Statuses are **Initializing**, **Pending**, or **Active**.

Initializing	The **FileIntegrity** object is currently initializing or re-initializing the AIDE database.
Pending	The **FileIntegrity** deployment is still being created.
Active	The scans are active and ongoing.

2. Apply the YAML file to the **openshift-file-integrity** namespace:

```
$ oc apply -f worker-fileintegrity.yaml -n openshift-file-integrity
```

Verification

- Confirm the **FileIntegrity** object was created successfully by running the following command:

```
$ oc get fileintegrities -n openshift-file-integrity
```

Example output

```
NAME                AGE
worker-fileintegrity   14s
```

6.6.2. Checking the FileIntegrity custom resource status

The **FileIntegrity** custom resource (CR) reports its status through the . **status.phase** subresource.

Procedure

- To query the **FileIntegrity** CR status, run:

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status.phase }"
```

Example output

```
Active
```

6.6.3. FileIntegrity custom resource phases

- **Pending** - The phase after the custom resource (CR) is created.
- **Active** - The phase when the backing daemon set is up and running.
- **Initializing** - The phase when the AIDE database is being reinitialized.

6.6.4. Understanding the FileIntegrityNodeStatuses object

The scan results of the **FileIntegrity** CR are reported in another object called **FileIntegrityNodeStatuses**.

```
$ oc get fileintegritynodestatuses
```

Example output

```
NAME                                                AGE
worker-fileintegrity-ip-10-0-130-192.ec2.internal   101s
worker-fileintegrity-ip-10-0-147-133.ec2.internal   109s
worker-fileintegrity-ip-10-0-165-160.ec2.internal   102s
```

NOTE

It might take some time for the **FileIntegrityNodeStatus** object results to be available.

There is one result object per node. The **nodeName** attribute of each **FileIntegrityNodeStatus** object corresponds to the node being scanned. The status of the file integrity scan is represented in the **results** array, which holds scan conditions.

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

The **fileintegritynodestatus** object reports the latest status of an AIDE run and exposes the status as **Failed**, **Succeeded**, or **Errored** in a **status** field.

```
$ oc get fileintegritynodestatuses -w
```

Example output

```
NAME                                                                NODE                                                                STATUS
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal   ip-10-0-134-186.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal   ip-10-0-150-230.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-169-137.us-east-2.compute.internal   ip-10-0-169-137.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal   ip-10-0-180-200.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal    ip-10-0-194-66.us-east-2.compute.internal    Failed
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal   ip-10-0-222-188.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal   ip-10-0-134-186.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal   ip-10-0-222-188.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal    ip-10-0-194-66.us-east-2.compute.internal    Failed
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal   ip-10-0-150-230.us-east-2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal   ip-10-0-180-200.us-east-2.compute.internal   Succeeded
```

6.6.5. FileIntegrityNodeStatus CR status types

These conditions are reported in the results array of the corresponding **FileIntegrityNodeStatus** CR status:

- **Succeeded** - The integrity check passed; the files and directories covered by the AIDE check have not been modified since the database was last initialized.
- **Failed** - The integrity check failed; some files or directories covered by the AIDE check have been modified since the database was last initialized.
- **Errored** - The AIDE scanner encountered an internal error.

6.6.5.1. FileIntegrityNodeStatus CR success example

Example output of a condition with a success status

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:57Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:46:03Z"
  }
```

```
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:48Z"
  }
]
```

In this case, all three scans succeeded and so far there are no other conditions.

6.6.5.2. FileIntegrityNodeStatus CR failure status example

To simulate a failure condition, modify one of the files AIDE tracks. For example, modify **/etc/resolv.conf** on one of the worker nodes:

```
$ oc debug node/ip-10-0-130-192.ec2.internal
```

Example output

```
Creating debug namespace/openshift-debug-node-ldfbj ...
Starting pod/ip-10-0-130-192ec2internal-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.0.130.192
If you don't see a command prompt, try pressing enter.
sh-4.2# echo "# integrity test" >> /host/etc/resolv.conf
sh-4.2# exit

Removing debug pod ...
Removing debug namespace/openshift-debug-node-ldfbj ...
```

After some time, the **Failed** condition is reported in the results array of the corresponding **FileIntegrityNodeStatus** object. The previous **Succeeded** condition is retained, which allows you to pinpoint the time the check failed.

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io/worker-fileintegrity-ip-10-0-130-
192.ec2.internal -ojsonpath='{.results}' | jq -r
```

Alternatively, if you are not mentioning the object name, run:

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

Example output

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:54:14Z"
  },
  {
    "condition": "Failed",
    "filesChanged": 1,
    "lastProbeTime": "2020-09-15T12:57:20Z",
    "resultConfigMapName": "aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed",
```

```
    "resultConfigMapNamespace": "openshift-file-integrity"
  }
]
```

The **Failed** condition points to a config map that gives more details about what exactly failed and why:

```
$ oc describe cm aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
```

Example output

```
Name:         aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
Namespace:    openshift-file-integrity
Labels:       file-integrity.openshift.io/node=ip-10-0-130-192.ec2.internal
              file-integrity.openshift.io/owner=worker-fileintegrity
              file-integrity.openshift.io/result-log=
Annotations:  file-integrity.openshift.io/files-added: 0
              file-integrity.openshift.io/files-changed: 1
              file-integrity.openshift.io/files-removed: 0

Data

integritylog:
------
AIDE 0.15.1 found differences between database and filesystem!!
Start timestamp: 2020-09-15 12:58:15

Summary:
  Total number of files:  31553
  Added files:            0
  Removed files:          0
  Changed files:          1


---------------------------------------------------
Changed files:
---------------------------------------------------

changed: /hostroot/etc/resolv.conf

---------------------------------------------------
Detailed information about changes:
---------------------------------------------------


File: /hostroot/etc/resolv.conf
 SHA512   : sTQYpB/AL7FeoGtu/1g7opv6C+KT1CBJ , qAeM+a8yTgHPnIHMaRlS+so61EN8VOpg

Events:  <none>
```

Due to the config map data size limit, AIDE logs over 1 MB are added to the failure config map as a base64-encoded gzip archive. Use the following command to extract the log:

```
$ oc get cm <failure-cm-name> -o json | jq -r '.data.integritylog' | base64 -d | gunzip
```

NOTE

Compressed logs are indicated by the presence of a **file-integrity.openshift.io/compressed** annotation key in the config map.

6.6.6. Understanding events

Transitions in the status of the **FileIntegrity** and **FileIntegrityNodeStatus** objects are logged by *events*. The creation time of the event reflects the latest transition, such as **Initializing** to **Active**, and not necessarily the latest scan result. However, the newest event always reflects the most recent status.

```
$ oc get events --field-selector reason=FileIntegrityStatus
```

Example output

```
LAST SEEN   TYPE     REASON              OBJECT                                MESSAGE
97s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Pending
67s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Initializing
37s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Active
```

When a node scan fails, an event is created with the **add/changed/removed** and config map information.

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

Example output

```
LAST SEEN   TYPE      REASON              OBJECT                                MESSAGE
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-134-173.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-168-238.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-169-175.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-152-92.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-158-144.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-131-30.ec2.internal
87m         Warning   NodeIntegrityStatus   fileintegrity/example-fileintegrity   node ip-10-0-152-92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

Changes to the number of added, changed, or removed files results in a new event, even if the status of the node has not transitioned.

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

Example output

```
LAST SEEN   TYPE      REASON              OBJECT                                MESSAGE
```

```
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-134-173.ec2.internal
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-168-238.ec2.internal
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-169-175.ec2.internal
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-152-92.ec2.internal
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-158-144.ec2.internal
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-131-30.ec2.internal
87m Warning NodeIntegrityStatus fileintegrity/example-fileintegrity node ip-10-0-152-92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
40m Warning NodeIntegrityStatus fileintegrity/example-fileintegrity node ip-10-0-152-92.ec2.internal has changed! a:3,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

6.7. CONFIGURING THE CUSTOM FILE INTEGRITY OPERATOR

6.7.1. Viewing FileIntegrity object attributes

As with any Kubernetes custom resources (CRs), you can run **oc explain fileintegrity**, and then look at the individual attributes using:

```
$ oc explain fileintegrity.spec
```

```
$ oc explain fileintegrity.spec.config
```

6.7.2. Important attributes

Table 6.1. Important spec and spec.config attributes

Attribute	Description
spec.nodeSelector	A map of key-values pairs that must match with node's labels in order for the AIDE pods to be schedulable on that node. The typical use is to set only a single key-value pair where **node-role.kubernetes.io/worker: ""** schedules AIDE on all worker nodes, **node.openshift.io/os_id: "rhcos"** schedules on all Red Hat Enterprise Linux CoreOS (RHCOS) nodes.
spec.debug	A boolean attribute. If set to **true**, the daemon running in the AIDE deamon set's pods would output extra information.

Attribute	Description
spec.tolerations	Specify tolerations to schedule on nodes with custom taints. When not specified, a default toleration is applied, which allows tolerations to run on control plane nodes.
spec.config.gracePeriod	The number of seconds to pause in between AIDE integrity checks. Frequent AIDE checks on a node can be resource intensive, so it can be useful to specify a longer interval. Defaults to **900**, or 15 minutes.
maxBackups	The maximum number of AIDE database and log backups leftover from the **re-init** process to keep on a node. Older backups beyond this number are automatically pruned by the daemon.
spec.config.name	Name of a configMap that contains custom AIDE configuration. If omitted, a default configuration is created.
spec.config.namespace	Namespace of a configMap that contains custom AIDE configuration. If unset, the FIO generates a default configuration suitable for RHCOS systems.
spec.config.key	Key that contains actual AIDE configuration in a config map specified by **name** and **namespace**. The default value is **aide.conf**.
spec.config.initialDelay	The number of seconds to wait before starting the first AIDE integrity check. Default is set to 0. This attribute is optional.

6.7.3. Examine the default configuration

The default File Integrity Operator configuration is stored in a config map with the same name as the **FileIntegrity** CR.

Procedure

- To examine the default config, run:

```
$ oc describe cm/worker-fileintegrity
```

6.7.4. Understanding the default File Integrity Operator configuration

Below is an excerpt from the **aide.conf** key of the config map:

```
@@define DBDIR /hostroot/etc/kubernetes
```

```
@@define LOGDIR /hostroot/etc/kubernetes
database=file:@@{DBDIR}/aide.db.gz
database_out=file:@@{DBDIR}/aide.db.gz
gzip_dbout=yes
verbose=5
report_url=file:@@{LOGDIR}/aide.log
report_url=stdout
PERMS = p+u+g+acl+selinux+xattrs
CONTENT_EX = sha512+ftype+p+u+g+n+acl+selinux+xattrs

/hostroot/boot/     CONTENT_EX
/hostroot/root/\..* PERMS
/hostroot/root/   CONTENT_EX
```

The default configuration for a **FileIntegrity** instance provides coverage for files under the following directories:

- **/root**
- **/boot**
- **/usr**
- **/etc**

The following directories are not covered:

- **/var**
- **/opt**
- Some OpenShift Container Platform-specific excludes under **/etc/**

6.7.5. Supplying a custom AIDE configuration

Any entries that configure AIDE internal behavior such as **DBDIR**, **LOGDIR**, **database**, and **database_out** are overwritten by the Operator. The Operator would add a prefix to **/hostroot/** before all paths to be watched for integrity changes. This makes reusing existing AIDE configs that might often not be tailored for a containerized environment and start from the root directory easier.

NOTE

/hostroot is the directory where the pods running AIDE mount the host's file system. Changing the configuration triggers a reinitializing of the database.

6.7.6. Defining a custom File Integrity Operator configuration

This example focuses on defining a custom configuration for a scanner that runs on the control plane nodes based on the default configuration provided for the **worker-fileintegrity** CR. This workflow might be useful if you are planning to deploy a custom software running as a daemon set and storing its data under **/opt/mydaemon** on the control plane nodes.

Procedure

1. Make a copy of the default configuration.
2. Edit the default configuration with the files that must be watched or excluded.
3. Store the edited contents in a new config map.
4. Point the **FileIntegrity** object to the new config map through the attributes in **spec.config**.
5. Extract the default configuration:

```
$ oc extract cm/worker-fileintegrity --keys=aide.conf
```

This creates a file named **aide.conf** that you can edit. To illustrate how the Operator post-processes the paths, this example adds an exclude directory without the prefix:

```
$ vim aide.conf
```

Example output

```
/hostroot/etc/kubernetes/static-pod-resources
!/hostroot/etc/kubernetes/aide.*
!/hostroot/etc/kubernetes/manifests
!/hostroot/etc/docker/certs.d
!/hostroot/etc/selinux/targeted
!/hostroot/etc/openvswitch/conf.db
```

Exclude a path specific to control plane nodes:

```
!/opt/mydaemon/
```

Store the other content in **/etc**:

```
/hostroot/etc/ CONTENT_EX
```

6. Create a config map based on this file:

```
$ oc create cm master-aide-conf --from-file=aide.conf
```

7. Define a **FileIntegrity** CR manifest that references the config map:

```
apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: master-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector:
     node-role.kubernetes.io/master: ""
  config:
     name: master-aide-conf
     namespace: openshift-file-integrity
```

The Operator processes the provided config map file and stores the result in a config map with the same name as the **FileIntegrity** object:

```
$ oc describe cm/master-fileintegrity | grep /opt/mydaemon
```

Example output

```
!/hostroot/opt/mydaemon
```

6.7.7. Changing the custom File Integrity configuration

To change the File Integrity configuration, never change the generated config map. Instead, change the config map that is linked to the **FileIntegrity** object through the **spec.name**, **namespace**, and **key** attributes.

6.8. PERFORMING ADVANCED CUSTOM FILE INTEGRITY OPERATOR TASKS

6.8.1. Reinitializing the database

If the File Integrity Operator detects a change that was planned, it might be required to reinitialize the database.

Procedure

- Annotate the **FileIntegrity** custom resource (CR) with **file-integrity.openshift.io/re-init**:

```
$ oc annotate fileintegrities/worker-fileintegrity file-integrity.openshift.io/re-init=
```

The old database and log files are backed up and a new database is initialized. The old database and logs are retained on the nodes under **/etc/kubernetes**, as seen in the following output from a pod spawned using **oc debug**:

Example output

```
 ls -lR /host/etc/kubernetes/aide.*
-rw-------. 1 root root 1839782 Sep 17 15:08 /host/etc/kubernetes/aide.db.gz
-rw-------. 1 root root 1839783 Sep 17 14:30 /host/etc/kubernetes/aide.db.gz.backup-
20200917T15_07_38
-rw-------. 1 root root   73728 Sep 17 15:07 /host/etc/kubernetes/aide.db.gz.backup-
20200917T15_07_55
-rw-r--r--. 1 root root       0 Sep 17 15:08 /host/etc/kubernetes/aide.log
-rw-------. 1 root root     613 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_38
-rw-r--r--. 1 root root       0 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_55
```

To provide some permanence of record, the resulting config maps are not owned by the **FileIntegrity** object, so manual cleanup is necessary. As a result, any previous integrity failures would still be visible in the **FileIntegrityNodeStatus** object.

6.8.2. Machine config integration

In OpenShift Container Platform 4, the cluster node configuration is delivered through **MachineConfig** objects. You can assume that the changes to files that are caused by a **MachineConfig** object are expected and should not cause the file integrity scan to fail. To suppress changes to files caused by **MachineConfig** object updates, the File Integrity Operator watches the node objects; when a node is being updated, the AIDE scans are suspended for the duration of the update. When the update finishes, the database is reinitialized and the scans resume.

This pause and resume logic only applies to updates through the **MachineConfig** API, as they are reflected in the node object annotations.

6.8.3. Exploring the daemon sets

Each **FileIntegrity** object represents a scan on a number of nodes. The scan itself is performed by pods managed by a daemon set.

To find the daemon set that represents a **FileIntegrity** object, run:

```
$ oc -n openshift-file-integrity get ds/aide-worker-fileintegrity
```

To list the pods in that daemon set, run:

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

To view logs of a single AIDE pod, call **oc logs** on one of the pods.

```
$ oc -n openshift-file-integrity logs pod/aide-worker-fileintegrity-mr8x6
```

Example output

```
Starting the AIDE runner daemon
initializing AIDE db
initialization finished
running aide check
...
```

The config maps created by the AIDE daemon are not retained and are deleted after the File Integrity Operator processes them. However, on failure and error, the contents of these config maps are copied to the config map that the **FileIntegrityNodeStatus** object points to.

6.9. TROUBLESHOOTING THE FILE INTEGRITY OPERATOR

6.9.1. General troubleshooting

Issue

You want to generally troubleshoot issues with the File Integrity Operator.

Resolution

Enable the debug flag in the **FileIntegrity** object. The **debug** flag increases the verbosity of the daemons that run in the **DaemonSet** pods and run the AIDE checks.

6.9.2. Checking the AIDE configuration

Issue

You want to check the AIDE configuration.

Resolution

The AIDE configuration is stored in a config map with the same name as the **FileIntegrity** object. All AIDE configuration config maps are labeled with **file-integrity.openshift.io/aide-conf**.

6.9.3. Determining the FileIntegrity object's phase

Issue

You want to determine if the **FileIntegrity** object exists and see its current status.

Resolution

To see the **FileIntegrity** object's current status, run:

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status }"
```

Once the **FileIntegrity** object and the backing daemon set are created, the status should switch to **Active**. If it does not, check the Operator pod logs.

6.9.4. Determining that the daemon set's pods are running on the expected nodes

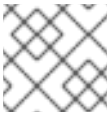
Issue

You want to confirm that the daemon set exists and that its pods are running on the nodes you expect them to run on.

Resolution

Run:

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```



NOTE

Adding **-owide** includes the IP address of the node that the pod is running on.

To check the logs of the daemon pods, run **oc logs**.

Check the return value of the AIDE command to see if the check passed or failed.

6.10. UNINSTALLING THE FILE INTEGRITY OPERATOR

You can remove the File Integrity Operator from your cluster by using the OpenShift Container Platform web console.

6.10.1. Uninstall the File Integrity Operator using the web console

To remove the File Integrity Operator, you must first delete the **FileIntegrity** objects in all namespaces. After the objects are removed, you can then remove the Operator and its namespace.

Prerequisites

- You have access to an OpenShift Container Platform cluster that uses an account with **cluster-admin** permissions.
- The File Integrity Operator is installed.

Procedure

1. Navigate to the **Operators → Installed Operators → File Integrity Operator** page.
2. From the **File Integrity** tab, ensure the **Show operands in: All namespaces** default option is selected to list all **FileIntegrity** objects in all namespaces.
3. Click the Options menu  and then click **Delete FileIntegrity** to delete a **FileIntegrity** object. Ensure all **FileIntegrity** objects are deleted.
4. Go to the **Administration → Operators → Installed Operators** page.
5. Click the Options menu  on the **File Integrity Operator** entry and select **Uninstall Operator**.
6. Go to the **Home → Projects** page.
7. Search for **openshift-file-integrity**.
8. Click the Options menu  for the **openshift-file-integrity** project entry, and then click **Delete Project**. A **Delete Project** dialog box opens on the web console.

Verification

- Type **openshift-file-integrity** in the **Delete Project** dialog box and then click the **Delete** button.

CHAPTER 7. SECURITY PROFILES OPERATOR

7.1. SECURITY PROFILES OPERATOR OVERVIEW

OpenShift Container Platform Security Profiles Operator (SPO) provides a way to define secure computing (seccomp) profiles and SELinux profiles as custom resources, synchronizing profiles to every node in a given namespace. For the latest updates, see the release notes.

The SPO can distribute custom resources to each node while a reconciliation loop ensures that the profiles stay up-to-date. See Understanding the Security Profiles Operator.

The SPO manages SELinux policies and seccomp profiles for namespaced workloads. For more information, see Enabling the Security Profiles Operator.

You can create seccomp and SELinux profiles, bind policies to pods, record workloads, and synchronize all worker nodes in a namespace.

Use Advanced Security Profile Operator tasks to enable the log enricher, configure webhooks and metrics, or restrict profiles to a single namespace.

Troubleshoot the Security Profiles Operator as needed, or engage Red Hat support.

You can Uninstall the Security Profiles Operator by removing the profiles before removing the Operator.

7.2. SECURITY PROFILES OPERATOR RELEASE NOTES

The Security Profiles Operator provides a way to define secure computing (seccomp) and SELinux profiles as custom resources, synchronizing profiles to every node in a given namespace.

These release notes track the development of the Security Profiles Operator in OpenShift Container Platform.

For an overview of the Security Profiles Operator, see Security Profiles Operator Overview.

7.2.1. Security Profiles Operator 0.8.6

The following advisory is available for the Security Profiles Operator 0.8.6:

- RHBA-2024:10380 - OpenShift Security Profiles Operator update

This update includes upgraded dependencies in underlying base images.

7.2.2. Security Profiles Operator 0.8.5

The following advisory is available for the Security Profiles Operator 0.8.5:

- RHBA-2024:5016 - OpenShift Security Profiles Operator bug fix update

7.2.2.1. Bug fixes

- When attempting to install the Security Profile Operator from the web console, the option to enable Operator-recommended cluster monitoring was unavailable for the namespace. With this update, you can now enabled Operator-recommend cluster monitoring in the namespace.

(OCPBUGS-37794)

- Previously, the Security Profiles Operator would intermittently be not visible in the OperatorHub, which caused limited access to install the Operator via the web console. With this update, the Security Profiles Operator is present in the OperatorHub.

7.2.3. Security Profiles Operator 0.8.4

The following advisory is available for the Security Profiles Operator 0.8.4:

- RHBA-2024:4781 - OpenShift Security Profiles Operator bug fix update

This update addresses CVEs in underlying dependencies.

7.2.3.1. New features and enhancements

- You can now specify a default security profile in the **image** attribute of a **ProfileBinding** object by setting a wildcard. For more information, see Binding workloads to profiles with ProfileBindings (SELinux) and Binding workloads to profiles with ProfileBindings (Seccomp) .

7.2.4. Security Profiles Operator 0.8.2

The following advisory is available for the Security Profiles Operator 0.8.2:

- RHBA-2023:5958 - OpenShift Security Profiles Operator bug fix update

7.2.4.1. Bug fixes

- Previously, **SELinuxProfile** objects did not inherit custom attributes from the same namespace. With this update, the issue has now been resolved and **SELinuxProfile** object attributes are inherited from the same namespace as expected. (OCPBUGS-17164)

- Previously, RawSELinuxProfiles would hang during the creation process and would not reach an **Installed** state. With this update, the issue has been resolved and RawSELinuxProfiles are created successfully. (OCPBUGS-19744)

- Previously, patching the **enableLogEnricher** to **true** would cause the **seccompProfile log-enricher-trace** pods to be stuck in a **Pending** state. With this update, **log-enricher-trace** pods reach an **Installed** state as expected. (OCPBUGS-22182)

- Previously, the Security Profiles Operator generated high cardinality metrics, causing Prometheus pods using high amounts of memory. With this update, the following metrics will no longer apply in the Security Profiles Operator namespace:
  - **rest_client_request_duration_seconds**
  - **rest_client_request_size_bytes**
  - **rest_client_response_size_bytes**
    (OCPBUGS-22406)

7.2.5. Security Profiles Operator 0.8.0

The following advisory is available for the Security Profiles Operator 0.8.0:

- RHBA-2023:4689 - OpenShift Security Profiles Operator bug fix update

7.2.5.1. Bug fixes

- Previously, while trying to install Security Profiles Operator in a disconnected cluster, the secure hashes provided were incorrect due to a SHA relabeling issue. With this update, the SHAs provided work consistently with disconnected environments. (OCPBUGS-14404)

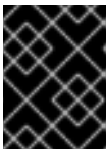
7.2.6. Security Profiles Operator 0.7.1

The following advisory is available for the Security Profiles Operator 0.7.1:

- RHSA-2023:2029 - OpenShift Security Profiles Operator bug fix update

7.2.6.1. New features and enhancements

- Security Profiles Operator (SPO) now automatically selects the appropriate **selinuxd** image for RHEL 8- and 9-based RHCOS systems.



IMPORTANT

Users that mirror images for disconnected environments must mirror both **selinuxd** images provided by the Security Profiles Operator.

- You can now enable memory optimization inside of an **spod** daemon. For more information, see Enabling memory optimization in the spod daemon .

NOTE

SPO memory optimization is not enabled by default.

- The daemon resource requirements are now configurable. For more information, see Customizing daemon resource requirements.

- The priority class name is now configurable in the **spod** configuration. For more information, see Setting a custom priority class name for the spod daemon pod .

7.2.6.2. Deprecated and removed features

- The default **nginx-1.19.1** seccomp profile is now removed from the Security Profiles Operator deployment.

7.2.6.3. Bug fixes

- Previously, a Security Profiles Operator (SPO) SELinux policy did not inherit low-level policy definitions from the container template. If you selected another template, such as net_container, the policy would not work because it required low-level policy definitions that only existed in the container template. This issue occurred when the SPO SELinux policy attempted to translate SELinux policies from the SPO custom format to the Common Intermediate Language (CIL) format. With this update, the container template appends to any SELinux policies that require translation from SPO to CIL. Additionally, the SPO SELinux policy can inherit low-level policy definitions from any supported policy template. (OCPBUGS-12879)

Known issue

- When uninstalling the Security Profiles Operator, the **MutatingWebhookConfiguration** object is not deleted and must be manually removed. As a workaround, delete the **MutatingWebhookConfiguration** object after uninstalling the Security Profiles Operator. These steps are defined in Uninstalling the Security Profiles Operator. (**OCPBUGS-4687**)

7.2.7. Security Profiles Operator 0.5.2

The following advisory is available for the Security Profiles Operator 0.5.2:

- RHBA-2023:0788 - OpenShift Security Profiles Operator bug fix update

This update addresses a CVE in an underlying dependency.

Known issue

- When uninstalling the Security Profiles Operator, the **MutatingWebhookConfiguration** object is not deleted and must be manually removed. As a workaround, delete the **MutatingWebhookConfiguration** object after uninstalling the Security Profiles Operator. These steps are defined in Uninstalling the Security Profiles Operator. (**OCPBUGS-4687**)

7.2.8. Security Profiles Operator 0.5.0

The following advisory is available for the Security Profiles Operator 0.5.0:

- RHBA-2022:8762 - OpenShift Security Profiles Operator bug fix update

Known issue

- When uninstalling the Security Profiles Operator, the **MutatingWebhookConfiguration** object is not deleted and must be manually removed. As a workaround, delete the **MutatingWebhookConfiguration** object after uninstalling the Security Profiles Operator. These steps are defined in Uninstalling the Security Profiles Operator. (**OCPBUGS-4687**)

7.3. SECURITY PROFILES OPERATOR SUPPORT

7.3.1. Security Profiles Operator lifecycle

The Security Profiles Operator is a "Rolling Stream" Operator, meaning updates are available asynchronously of OpenShift Container Platform releases. For more information, see OpenShift Operator Life Cycles on the Red Hat Customer Portal.

7.3.2. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal . From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in OpenShift Cluster Manager Hybrid Cloud Console. Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a Jira issue for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

7.4. UNDERSTANDING THE SECURITY PROFILES OPERATOR

OpenShift Container Platform administrators can use the Security Profiles Operator to define increased security measures in clusters.

IMPORTANT

The Security Profiles Operator supports only Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes. Red Hat Enterprise Linux (RHEL) nodes are not supported.

7.4.1. About Security Profiles

Security profiles can increase security at the container level in your cluster.

Seccomp security profiles list the syscalls a process can make. Permissions are broader than SELinux, enabling users to restrict operations system-wide, such as **write**.

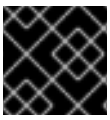
SELinux security profiles provide a label-based system that restricts the access and usage of processes, applications, or files in a system. All files in an environment have labels that define permissions. SELinux profiles can define access within a given structure, such as directories.

7.5. ENABLING THE SECURITY PROFILES OPERATOR

Before you can use the Security Profiles Operator, you must ensure the Operator is deployed in the cluster.

IMPORTANT

The Security Profiles Operator supports only Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes. Red Hat Enterprise Linux (RHEL) nodes are not supported.



IMPORTANT

The Security Profiles Operator only supports **x86_64** architecture.

7.5.1. Installing the Security Profiles Operator

Prerequisites

- You must have **admin** privileges.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Search for the Security Profiles Operator, then click **Install**.

3. Keep the default selection of **Installation mode** and **namespace** to ensure that the Operator will be installed to the **openshift-security-profiles** namespace.

4. Click **Install**.

Verification

To confirm that the installation is successful:

1. Navigate to the **Operators → Installed Operators** page.

2. Check that the Security Profiles Operator is installed in the **openshift-security-profiles** namespace and its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.

2. Navigate to the **Workloads → Pods** page and check the logs in any pods in the **openshift-security-profiles** project that are reporting issues.

7.5.2. Installing the Security Profiles Operator using the CLI

Prerequisites

- You must have **admin** privileges.

Procedure

1. Define a **Namespace** object:

   Example namespace-object.yaml

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

2. Create the **Namespace** object:

```
$ oc create -f namespace-object.yaml
```

3. Define an **OperatorGroup** object:

   Example operator-group-object.yaml

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: security-profiles-operator
  namespace: openshift-security-profiles
```

4. Create the **OperatorGroup** object:

```
$ oc create -f operator-group-object.yaml
```

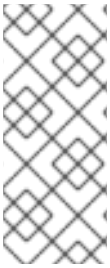
5. Define a **Subscription** object:

Example subscription-object.yaml

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: security-profiles-operator-sub
  namespace: openshift-security-profiles
spec:
  channel: release-alpha-rhel-8
  installPlanApproval: Automatic
  name: security-profiles-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

6. Create the **Subscription** object:

```
$ oc create -f subscription-object.yaml
```



NOTE

If you are setting the global scheduler feature and enable **defaultNodeSelector**, you must create the namespace manually and update the annotations of the **openshift-security-profiles** namespace, or the namespace where the Security Profiles Operator was installed, with **openshift.io/node-selector: “”**. This removes the default node selector and prevents deployment failures.

Verification

1. Verify the installation succeeded by inspecting the following CSV file:

```
$ oc get csv -n openshift-security-profiles
```

2. Verify that the Security Profiles Operator is operational by running the following command:

```
$ oc get deploy -n openshift-security-profiles
```

7.5.3. Configuring logging verbosity

The Security Profiles Operator supports the default logging verbosity of **0** and an enhanced verbosity of **1**.

Procedure

- To enable enhanced logging verbosity, patch the **spod** configuration and adjust the value by running the following command:

```
$ oc -n openshift-security-profiles patch spod \
    spod --type=merge -p '{"spec":{"verbosity":1}}'
```

Example output

```
securityprofilesoperatordaemon.security-profiles-operator.x-k8s.io/spod patched
```

7.6. MANAGING SECCOMP PROFILES

Create and manage seccomp profiles and bind them to workloads.

IMPORTANT

The Security Profiles Operator supports only Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes. Red Hat Enterprise Linux (RHEL) nodes are not supported.

7.6.1. Creating seccomp profiles

Use the **SeccompProfile** object to create profiles.

SeccompProfile objects can restrict syscalls within a container, limiting the access of your application.

Procedure

1. Create a project by running the following command:

```
$ oc new-project my-namespace
```

2. Create the **SeccompProfile** object:

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  namespace: my-namespace
  name: profile1
spec:
  defaultAction: SCMP_ACT_LOG
```

The seccomp profile will be saved in **/var/lib/kubelet/seccomp/operator/<namespace>/<name>.json**.

An **init** container creates the root directory of the Security Profiles Operator to run the Operator without **root** group or user ID privileges. A symbolic link is created from the rootless profile storage **/var/lib/openshift-security-profiles** to the default **seccomp** root path inside of the kubelet root **/var/lib/kubelet/seccomp/operator**.

7.6.2. Applying seccomp profiles to a pod

Create a pod to apply one of the created profiles.

Procedure

1. Create a pod object that defines a **securityContext**:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: operator/my-namespace/profile1.json
  containers:
    - name: test-container
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

2. View the profile path of the **seccompProfile.localhostProfile** attribute by running the following command:

```
$ oc -n my-namespace get seccompprofile profile1 --output wide
```

Example output

```
NAME       STATUS     AGE   SECCOMPPROFILE.LOCALHOSTPROFILE
profile1   Installed  14s   operator/my-namespace/profile1.json
```

3. View the path to the localhost profile by running the following command:

```
$ oc get sp profile1 --output=jsonpath='{.status.localhostProfile}'
```

Example output

```
operator/my-namespace/profile1.json
```

4. Apply the **localhostProfile** output to the patch file:

```
spec:
  template:
    spec:
      securityContext:
        seccompProfile:
          type: Localhost
          localhostProfile: operator/my-namespace/profile1.json
```

5. Apply the profile to any other workload, such as a **Deployment** object, by running the following command:

```
$ oc -n my-namespace patch deployment myapp --patch-file patch.yaml --type=merge
```

Example output

```
deployment.apps/myapp patched
```

Verification

- Confirm the profile was applied correctly by running the following command:

```
$ oc -n my-namespace get deployment myapp --
output=jsonpath='{.spec.template.spec.securityContext}' | jq .
```

Example output

```
{
  "seccompProfile": {
    "localhostProfile": "operator/my-namespace/profile1.json",
    "type": "localhost"
  }
}
```

7.6.2.1. Binding workloads to profiles with ProfileBindings

You can use the **ProfileBinding** resource to bind a security profile to the **SecurityContext** of a container.

Procedure

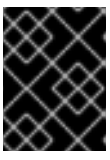
1. To bind a pod that uses a **quay.io/security-profiles-operator/test-nginx-unprivileged:1.21** image to the example **SeccompProfile** profile, create a **ProfileBinding** object in the same namespace with the pod and the **SeccompProfile** objects:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileBinding
metadata:
  namespace: my-namespace
  name: nginx-binding
spec:
  profileRef:
    kind: SeccompProfile (1)
    name: profile (2)
  image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21 (3)
```

(1) The **kind:** variable refers to the kind of the profile.

(2) The **name:** variable refers to the name of the profile.

(3) You can enable a default security profile by using a wildcard in the image attribute: **image: "*"**



IMPORTANT

Using the **image: "*"** wildcard attribute binds all new pods with a default security profile in a given namespace.

2. Label the namespace with **enable-binding=true** by running the following command:

```
$ oc label ns my-namespace spo.x-k8s.io/enable-binding=true
```

3. Define a pod named **test-pod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: test-container
    image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

4. Create the pod:

```
$ oc create -f test-pod.yaml
```

NOTE

If the pod already exists, you must re-create the pod for the binding to work properly.

Verification

- Confirm the pod inherits the **ProfileBinding** by running the following command:

```
$ oc get pod test-pod -o jsonpath='{.spec.containers[*].securityContext.seccompProfile}'
```

Example output

```
{"localhostProfile":"operator/my-namespace/profile.json","type":"Localhost"}
```

7.6.3. Recording profiles from workloads

The Security Profiles Operator can record system calls with **ProfileRecording** objects, making it easier to create baseline profiles for applications.

When using the log enricher for recording seccomp profiles, verify the log enricher feature is enabled. See *Additional resources* for more information.

NOTE

A container with **privileged: true** security context restraints prevents log-based recording. Privileged containers are not subject to seccomp policies, and log-based recording makes use of a special seccomp profile to record events.

Procedure

1. Create a project by running the following command:

```
$ oc new-project my-namespace
```

2. Label the namespace with **enable-recording=true** by running the following command:

```
$ oc label ns my-namespace spo.x-k8s.io/enable-recording=true
```

3. Create a **ProfileRecording** object containing a **recorder: logs** variable:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  namespace: my-namespace
  name: test-recording
spec:
  kind: SeccompProfile
  recorder: logs
  podSelector:
    matchLabels:
      app: my-app
```

4. Create a workload to record:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: my-namespace
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
    - name: nginx
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      ports:
        - containerPort: 8080
    - name: redis
      image: quay.io/security-profiles-operator/redis:6.2.1
```

5. Confirm the pod is in a **Running** state by entering the following command:

```
$ oc -n my-namespace get pods
```

Example output

```
NAME    READY   STATUS    RESTARTS   AGE
my-pod   2/2     Running   0          18s
```

6. Confirm the enricher indicates that it receives audit logs for those containers:

```
$ oc -n openshift-security-profiles logs --since=1m --selector name=spod -c log-enricher
```

Example output

```
I0523 14:19:08.747313  430694 enricher.go:445] log-enricher "msg"="audit"
"container"="redis" "executable"="/usr/local/bin/redis-server" "namespace"="my-namespace"
```

```
"node"="xiyuan-23-5g2q9-worker-eastus2-6rpgf" "pid"=656802 "pod"="my-pod" "syscallID"=0
"syscallName"="read" "timestamp"="1684851548.745:207179" "type"="seccomp"
```

Verification

1. Remove the pod:

```
$ oc -n my-namepace delete pod my-pod
```

2. Confirm the Security Profiles Operator reconciles the two seccomp profiles:

```
$ oc get seccompprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

Example output for seccompprofile

```
NAME                  STATUS      AGE
test-recording-nginx   Installed   2m48s
test-recording-redis   Installed   2m48s
```

7.6.3.1. Merging per-container profile instances

By default, each container instance records into a separate profile. The Security Profiles Operator can merge the per-container profiles into a single profile. Merging profiles is useful when deploying applications using **ReplicaSet** or **Deployment** objects.

Procedure

1. Edit a **ProfileRecording** object to include a **mergeStrategy: containers** variable:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  # The name of the Recording is the same as the resulting SeccompProfile CRD
  # after reconciliation.
  name: test-recording
  namespace: my-namespace
spec:
  kind: SeccompProfile
  recorder: logs
  mergeStrategy: containers
  podSelector:
    matchLabels:
      app: sp-record
```

2. Label the namespace by running the following command:

```
$ oc label ns my-namespace security.openshift.io/scc.podSecurityLabelSync=false pod-
security.kubernetes.io/enforce=privileged pod-security.kubernetes.io/audit=privileged pod-
security.kubernetes.io/warn=privileged --overwrite=true
```

3. Create the workload with the following YAML:

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: nginx-deploy
  namespace: my-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sp-record
  template:
    metadata:
      labels:
        app: sp-record
    spec:
      serviceAccountName: spo-record-sa
      containers:
      - name: nginx-record
        image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
        ports:
        - containerPort: 8080
```

4. To record the individual profiles, delete the deployment by running the following command:

```
$ oc delete deployment nginx-deploy -n my-namespace
```

5. To merge the profiles, delete the profile recording by running the following command:

```
$ oc delete profilerecording test-recording -n my-namespace
```

6. To start the merge operation and generate the results profile, run the following command:

```
$ oc get seccompprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

Example output for seccompprofiles

```
NAME                         STATUS      AGE
test-recording-nginx-record  Installed   55s
```

7. To view the permissions used by any of the containers, run the following command:

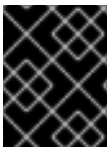
```
$ oc get seccompprofiles test-recording-nginx-record -o yaml
```

Additional resources

- Managing security context constraints
- Managing SCCs in OpenShift
- Using the log enricher
- About security profiles

7.7. MANAGING SELINUX PROFILES

Create and manage SELinux profiles and bind them to workloads.



IMPORTANT

The Security Profiles Operator supports only Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes. Red Hat Enterprise Linux (RHEL) nodes are not supported.

7.7.1. Creating SELinux profiles

Use the **SelinuxProfile** object to create profiles.

The **SelinuxProfile** object has several features that allow for better security hardening and readability:

- Restricts the profiles to inherit from to the current namespace or a system-wide profile. Because there are typically many profiles installed on the system, but only a subset should be used by cluster workloads, the inheritable system profiles are listed in the **spod** instance in **spec.selinuxOptions.allowedSystemProfiles**.
- Performs basic validation of the permissions, classes and labels.
- Adds a new keyword **@self** that describes the process using the policy. This allows reusing a policy between workloads and namespaces easily, as the usage of the policy is based on the name and namespace.
- Adds features for better security hardening and readability compared to writing a profile directly in the SELinux CIL language.

Procedure

1. Create a project by running the following command:

```
$ oc new-project nginx-deploy
```

2. Create a policy that can be used with a non-privileged workload by creating the following **SelinuxProfile** object:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha2
kind: SelinuxProfile
metadata:
  name: nginx-secure
  namespace: nginx-deploy
spec:
  allow:
    '@self':
      tcp_socket:
      - listen
    http_cache_port_t:
      tcp_socket:
      - name_bind
    node_t:
      tcp_socket:
      - node_bind
  inherit:
  - kind: System
    name: container
```

3. Wait for **selinuxd** to install the policy by running the following command:

```
$ oc wait --for=condition=ready -n nginx-deploy selinuxprofile nginx-secure
```

Example output

```
selinuxprofile.security-profiles-operator.x-k8s.io/nginx-secure condition met
```

The policies are placed into an **emptyDir** in the container owned by the Security Profiles Operator. The policies are saved in Common Intermediate Language (CIL) format in **/etc/selinux.d/<name>_<namespace>.cil**.

4. Access the pod by running the following command:

```
$ oc -n openshift-security-profiles rsh -c selinuxd ds/spod
```

Verification

1. View the file contents with **cat** by running the following command:

```
$ cat /etc/selinux.d/nginx-secure_nginx-deploy.cil
```

Example output

```
(block nginx-secure_nginx-deploy
(blockinherit container)
(allow process nginx-secure_nginx-deploy.process ( tcp_socket ( listen )))
(allow process http_cache_port_t ( tcp_socket ( name_bind )))
(allow process node_t ( tcp_socket ( node_bind )))
)
```

2. Verify that a policy has been installed by running the following command:

```
$ semodule -l | grep nginx-secure
```

Example output

```
nginx-secure_nginx-deploy
```

7.7.2. Applying SELinux profiles to a pod

Create a pod to apply one of the created profiles.

For SELinux profiles, the namespace must be labelled to allow privileged workloads.

Procedure

1. Apply the **scc.podSecurityLabelSync=false** label to the **nginx-deploy** namespace by running the following command:

```
$ oc label ns nginx-deploy security.openshift.io/scc.podSecurityLabelSync=false
```

2. Apply the **privileged** label to the **nginx-deploy** namespace by running the following command:

```
$ oc label ns nginx-deploy --overwrite=true pod-security.kubernetes.io/enforce=privileged
```

3. Obtain the SELinux profile usage string by running the following command:

```
$ oc get selinuxprofile.security-profiles-operator.x-k8s.io/nginx-secure -n nginx-deploy -
ojsonpath='{.status.usage}'
```

Example output

```
nginx-secure_nginx-deploy.process
```

4. Apply the output string in the workload manifest in the **.spec.containers[].securityContext.seLinuxOptions** attribute:

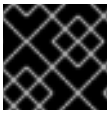
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-secure
  namespace: nginx-deploy
spec:
  containers:
    - image: nginxinc/nginx-unprivileged:1.21
      name: nginx
      securityContext:
        seLinuxOptions:
          # NOTE: This uses an appropriate SELinux type
          type: nginx-secure_nginx-deploy.process
```

IMPORTANT

The SELinux **type** must exist before creating the workload.

7.7.2.1. Applying SELinux log policies

To log policy violations or AVC denials, set the **SElinuxProfile** profile to **permissive**.



IMPORTANT

This procedure defines logging policies. It does not set enforcement policies.

Procedure

- Add **permissive: true** to an **SElinuxProfile**:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha2
kind: SelinuxProfile
metadata:
  name: nginx-secure
```

```
  namespace: nginx-deploy
spec:
  permissive: true
```

7.7.2.2. Binding workloads to profiles with ProfileBindings

You can use the **ProfileBinding** resource to bind a security profile to the **SecurityContext** of a container.

Procedure

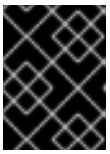
1. To bind a pod that uses a **quay.io/security-profiles-operator/test-nginx-unprivileged:1.21** image to the example **SelinuxProfile** profile, create a **ProfileBinding** object in the same namespace with the pod and the **SelinuxProfile** objects:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileBinding
metadata:
  namespace: my-namespace
  name: nginx-binding
spec:
  profileRef:
    kind: SelinuxProfile (1)
    name: profile (2)
  image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21 (3)
```

(1) The **kind:** variable refers to the kind of the profile.

(2) The **name:** variable refers to the name of the profile.

(3) You can enable a default security profile by using a wildcard in the image attribute: **image: "*"**



IMPORTANT

Using the **image: "*"** wildcard attribute binds all new pods with a default security profile in a given namespace.

2. Label the namespace with **enable-binding=true** by running the following command:

```
$ oc label ns my-namespace spo.x-k8s.io/enable-binding=true
```

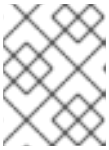
3. Define a pod named **test-pod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: test-container
    image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

-

4. Create the pod:

```
$ oc create -f test-pod.yaml
```



NOTE

If the pod already exists, you must re-create the pod for the binding to work properly.

Verification

- Confirm the pod inherits the **ProfileBinding** by running the following command:

```
$ oc get pod test-pod -o jsonpath='{.spec.containers[*].securityContext.seLinuxOptions.type}'
```

Example output

```
profile_nginx-binding.process
```

7.7.2.3. Replicating controllers and SecurityContextConstraints

When you deploy SELinux policies for replicating controllers, such as deployments or daemon sets, note that the **Pod** objects spawned by the controllers are not running with the identity of the user who creates the workload. Unless a **ServiceAccount** is selected, the pods might revert to using a restricted **SecurityContextConstraints** (SCC) which does not allow use of custom security policies.

Procedure

1. Create a project by running the following command:

```
$ oc new-project nginx-secure
```

2. Create the following **RoleBinding** object to allow SELinux policies to be used in the **nginx-secure** namespace:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: spo-nginx
  namespace: nginx-secure
subjects:
- kind: ServiceAccount
  name: spo-deploy-test
roleRef:
  kind: Role
  name: spo-nginx
  apiGroup: rbac.authorization.k8s.io
```

3. Create the **Role** object:

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: Role
metadata:
  creationTimestamp: null
  name: spo-nginx
  namespace: nginx-secure
rules:
- apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints
  resourceNames:
  - privileged
  verbs:
  - use
```

4. Create the **ServiceAccount** object:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: null
  name: spo-deploy-test
  namespace: nginx-secure
```

5. Create the **Deployment** object:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: selinux-test
  namespace: nginx-secure
  metadata:
    labels:
      app: selinux-test
spec:
  replicas: 3
  selector:
    matchLabels:
      app: selinux-test
  template:
    metadata:
      labels:
        app: selinux-test
    spec:
      serviceAccountName: spo-deploy-test
      securityContext:
        seLinuxOptions:
          type: nginx-secure_nginx-secure.process (1)
      containers:
      - name: nginx-unpriv
        image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
        ports:
        - containerPort: 8080
```

(1) The **.seLinuxOptions.type** must exist before the Deployment is created.

NOTE

The SELinux type is not specified in the workload and is handled by the SCC. When the pods are created by the deployment and the **ReplicaSet**, the pods will run with the appropriate profile.

Ensure that your SCC is usable by only the correct service account. Refer to *Additional resources* for more information.

7.7.3. Recording profiles from workloads

The Security Profiles Operator can record system calls with **ProfileRecording** objects, making it easier to create baseline profiles for applications.

When using the log enricher for recording SELinux profiles, verify the log enricher feature is enabled. See *Additional resources* for more information.

NOTE

A container with **privileged: true** security context restraints prevents log-based recording. Privileged containers are not subject to SELinux policies, and log-based recording makes use of a special SELinux profile to record events.

Procedure

1. Create a project by running the following command:

```
$ oc new-project my-namespace
```

2. Label the namespace with **enable-recording=true** by running the following command:

```
$ oc label ns my-namespace spo.x-k8s.io/enable-recording=true
```

3. Create a **ProfileRecording** object containing a **recorder: logs** variable:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  namespace: my-namespace
  name: test-recording
spec:
  kind: SelinuxProfile
  recorder: logs
  podSelector:
    matchLabels:
      app: my-app
```

4. Create a workload to record:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: my-namespace
```

```
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
    - name: nginx
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      ports:
        - containerPort: 8080
    - name: redis
      image: quay.io/security-profiles-operator/redis:6.2.1
```

5. Confirm the pod is in a **Running** state by entering the following command:

```
$ oc -n my-namespace get pods
```

Example output

```
NAME    READY   STATUS    RESTARTS   AGE
my-pod   2/2     Running   0          18s
```

6. Confirm the enricher indicates that it receives audit logs for those containers:

```
$ oc -n openshift-security-profiles logs --since=1m --selector name=spod -c log-enricher
```

Example output

```
I0517 13:55:36.383187  348295 enricher.go:376] log-enricher "msg"="audit"
"container"="redis" "namespace"="my-namespace" "node"="ip-10-0-189-53.us-east-
2.compute.internal" "perm"="name_bind" "pod"="my-pod" "profile"="test-
recording_redis_6kmrb_1684331729"
"scontext"="system_u:system_r:selinuxrecording.process:s0:c4,c27" "tclass"="tcp_socket"
"tcontext"="system_u:object_r:redis_port_t:s0" "timestamp"="1684331735.105:273965"
"type"="selinux"
```

Verification

1. Remove the pod:

```
$ oc -n my-namepace delete pod my-pod
```

2. Confirm the Security Profiles Operator reconciles the two SELinux profiles:

```
$ oc get selinuxprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

Example output for selinuxprofile

```
NAME                  USAGE                                  STATE
test-recording-nginx   test-recording-nginx_my-namespace.process   Installed
test-recording-redis   test-recording-redis_my-namespace.process   Installed
```

7.7.3.1. Merging per-container profile instances

By default, each container instance records into a separate profile. The Security Profiles Operator can merge the per-container profiles into a single profile. Merging profiles is useful when deploying applications using **ReplicaSet** or **Deployment** objects.

Procedure

1. Edit a **ProfileRecording** object to include a **mergeStrategy: containers** variable:

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  # The name of the Recording is the same as the resulting SelinuxProfile CRD
  # after reconciliation.
  name: test-recording
  namespace: my-namespace
spec:
  kind: SelinuxProfile
  recorder: logs
  mergeStrategy: containers
  podSelector:
    matchLabels:
      app: sp-record
```

2. Label the namespace by running the following command:

```
$ oc label ns my-namespace security.openshift.io/scc.podSecurityLabelSync=false pod-security.kubernetes.io/enforce=privileged pod-security.kubernetes.io/audit=privileged pod-security.kubernetes.io/warn=privileged --overwrite=true
```

3. Create the workload with the following YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  namespace: my-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sp-record
  template:
    metadata:
      labels:
        app: sp-record
    spec:
      serviceAccountName: spo-record-sa
      containers:
      - name: nginx-record
        image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
        ports:
        - containerPort: 8080
```

4. To record the individual profiles, delete the deployment by running the following command:

```
$ oc delete deployment nginx-deploy -n my-namespace
```

5. To merge the profiles, delete the profile recording by running the following command:

```
$ oc delete profilerecording test-recording -n my-namespace
```

6. To start the merge operation and generate the results profile, run the following command:

```
$ oc get selinuxprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

Example output for selinuxprofiles

```
NAME                       USAGE                                      STATE
test-recording-nginx-record   test-recording-nginx-record_my-namespace.process   Installed
```

7. To view the permissions used by any of the containers, run the following command:

```
$ oc get selinuxprofiles test-recording-nginx-record -o yaml
```

7.7.3.2. About seLinuxContext: RunAsAny

Recording of SELinux policies is implemented with a webhook that injects a special SELinux type to the pods being recorded. The SELinux type makes the pod run in **permissive** mode, logging all the AVC denials into **audit.log**. By default, a workload is not allowed to run with a custom SELinux policy, but uses an auto-generated type.

To record a workload, the workload must use a service account that has permissions to use an SCC that allows the webhook to inject the permissive SELinux type. The **privileged** SCC contains **seLinuxContext: RunAsAny**.

In addition, the namespace must be labeled with **pod-security.kubernetes.io/enforce: privileged** if your cluster enables the Pod Security Admission because only the **privileged** Pod Security Standard allows using a custom SELinux policy.

Additional resources

- Managing security context constraints
- Managing SCCs in OpenShift
- Using the log enricher
- About security profiles

7.8. ADVANCED SECURITY PROFILES OPERATOR TASKS

Use advanced tasks to enable metrics, configure webhooks, or restrict syscalls.

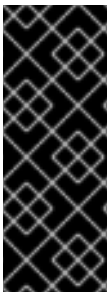
7.8.1. Restrict the allowed syscalls in seccomp profiles

The Security Profiles Operator does not restrict **syscalls** in **seccomp** profiles by default. You can define the list of allowed **syscalls** in the **spod** configuration.

Procedure

- To define the list of **allowedSyscalls**, adjust the **spec** parameter by running the following command:

```
$ oc -n openshift-security-profiles patch spod spod --type merge \
    -p '{"spec":{"allowedSyscalls": ["exit", "exit_group", "futex", "nanosleep"]}}'
```



IMPORTANT

The Operator will install only the **seccomp** profiles, which have a subset of **syscalls** defined into the allowed list. All profiles not complying with this ruleset are rejected.

When the list of allowed **syscalls** is modified in the **spod** configuration, the Operator will identify the already installed profiles which are non-compliant and remove them automatically.

7.8.2. Base syscalls for a container runtime

You can use the **baseProfileName** attribute to establish the minimum required **syscalls** for a given runtime to start a container.

Procedure

- Edit the **SeccompProfile** kind object and add **baseProfileName: runc-v1.0.0** to the **spec** field:

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  namespace: my-namespace
  name: example-name
spec:
  defaultAction: SCMP_ACT_ERRNO
  baseProfileName: runc-v1.0.0
  syscalls:
    - action: SCMP_ACT_ALLOW
      names:
        - exit_group
```

7.8.3. Enabling memory optimization in the spod daemon

The controller running inside of **spod** daemon process watches all pods available in the cluster when profile recording is enabled. This can lead to very high memory usage in large clusters, resulting in the **spod** daemon running out of memory or crashing.

To prevent crashes, the **spod** daemon can be configured to only load the pods labeled for profile recording into the cache memory.

+

NOTE

SPO memory optimization is not enabled by default.

Procedure

1. Enable memory optimization by running the following command:

```
$ oc -n openshift-security-profiles patch spod spod --type=merge -p '{"spec":
{"enableMemoryOptimization":true}}'
```

2. To record a security profile for a pod, the pod must be labeled with **spo.x-k8s.io/enable-recording: "true"**:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-recording-pod
  labels:
    spo.x-k8s.io/enable-recording: "true"
```

7.8.4. Customizing daemon resource requirements

The default resource requirements of the daemon container can be adjusted by using the field **daemonResourceRequirements** from the **spod** configuration.

Procedure

- To specify the memory and cpu requests and limits of the daemon container, run the following command:

```
$ oc -n openshift-security-profiles patch spod spod --type merge -p \
    '{"spec":{"daemonResourceRequirements": { \
    "requests": {"memory": "256Mi", "cpu": "250m"}, \
    "limits": {"memory": "512Mi", "cpu": "500m"}}}}'
```

7.8.5. Setting a custom priority class name for the spod daemon pod

The default priority class name of the **spod** daemon pod is set to **system-node-critical**. A custom priority class name can be configured in the **spod** configuration by setting a value in the **priorityClassName** field.

Procedure

- Configure the priority class name by running the following command:

```
$ oc -n openshift-security-profiles patch spod spod --type=merge -p '{"spec":
{"priorityClassName":"my-priority-class"}}'
```

Example output

```
securityprofilesoperatordaemon.openshift-security-profiles.x-k8s.io/spod patched
```

7.8.6. Using metrics

The **openshift-security-profiles** namespace provides metrics endpoints, which are secured by the kube-rbac-proxy container. All metrics are exposed by the **metrics** service within the **openshift-security-profiles** namespace.

The Security Profiles Operator includes a cluster role and corresponding binding **spo-metrics-client** to retrieve the metrics from within the cluster. There are two metrics paths available:

- **metrics.openshift-security-profiles/metrics**: for controller runtime metrics
- **metrics.openshift-security-profiles/metrics-spod**: for the Operator daemon metrics

Procedure

1. To view the status of the metrics service, run the following command:

```
$ oc get svc/metrics -n openshift-security-profiles
```

Example output

```
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
metrics   ClusterIP   10.0.0.228   <none>        443/TCP   43s
```

2. To retrieve the metrics, query the service endpoint using the default **ServiceAccount** token in the **openshift-security-profiles** namespace by running the following command:

```
$ oc run --rm -i --restart=Never --image=registry.fedoraproject.org/fedora-minimal:latest \
    -n openshift-security-profiles metrics-test -- bash -c \
    'curl -ks -H "Authorization: Bearer $(cat
/var/run/secrets/kubernetes.io/serviceaccount/token)" https://metrics.openshift-security-
profiles/metrics-spod'
```

Example output

```
# HELP security_profiles_operator_seccomp_profile_total Counter about seccomp profile
operations.
# TYPE security_profiles_operator_seccomp_profile_total counter
security_profiles_operator_seccomp_profile_total{operation="delete"} 1
security_profiles_operator_seccomp_profile_total{operation="update"} 2
```

3. To retrieve metrics from a different namespace, link the **ServiceAccount** to the **spo-metrics-client ClusterRoleBinding** by running the following command:

```
$ oc get clusterrolebinding spo-metrics-client -o wide
```

Example output

```
NAME                ROLE                           AGE   USERS   GROUPS   SERVICEACCOUNTS
spo-metrics-client  ClusterRole/spo-metrics-client  35m                    openshift-security-
profiles/default
```

7.8.6.1. controller-runtime metrics

The controller-runtime **metrics** and the DaemonSet endpoint **metrics-spod** provide a set of default metrics. Additional metrics are provided by the daemon, which are always prefixed with **security_profiles_operator_**.

Table 7.1. Available controller-runtime metrics

Metric key	Possible labels	Type	Purpose
seccomp_profile_total	**operation={delete,update}**	Counter	Amount of seccomp profile operations.
seccomp_profile_audit_total	**node**, **namespace**, **pod**, **container**, **executable**, **syscall**	Counter	Amount of seccomp profile audit operations. Requires the log enricher to be enabled.
seccomp_profile_bpf_total	**node**, **mount_namespace**, **profile**	Counter	Amount of seccomp profile bpf operations. Requires the bpf recorder to be enabled.
seccomp_profile_error_total	**reason={ SeccompNotSupportedOnNode, InvalidSeccompProfile, CannotSaveSeccompProfile, CannotRemoveSeccompProfile, CannotUpdateSeccompProfile, CannotUpdateNodeStatus }**	Counter	Amount of seccomp profile errors.
selinux_profile_total	**operation={delete,update}**	Counter	Amount of SELinux profile operations.
selinux_profile_audit_total	**node**, **namespace**, **pod**, **container**, **executable**, **scontext**,**tcontext**	Counter	Amount of SELinux profile audit operations. Requires the log enricher to be enabled.

Metric key	Possible labels	Type	Purpose
selinux_profile_error_total	**reason={ CannotSaveSelinuxPolicy, CannotUpdatePolicyStatus, CannotRemoveSelinuxPolicy, CannotContactSelinuxd, CannotWritePolicyFile, CannotGetPolicyStatus }**	Counter	Amount of SELinux profile errors.

7.8.7. Using the log enricher

The Security Profiles Operator contains a log enrichment feature, which is disabled by default. The log enricher container runs with **privileged** permissions to read the audit logs from the local node. The log enricher runs within the host PID namespace, **hostPID**.

IMPORTANT

The log enricher must have permissions to read the host processes.

Procedure

1. Patch the **spod** configuration to enable the log enricher by running the following command:

```
$ oc -n openshift-security-profiles patch spod spod \
    --type=merge -p '{"spec":{"enableLogEnricher":true}}'
```

Example output

```
securityprofilesoperatordaemon.security-profiles-operator.x-k8s.io/spod patched
```

NOTE

The Security Profiles Operator will re-deploy the **spod** daemon set automatically.

2. View the audit logs by running the following command:

```
$ oc -n openshift-security-profiles logs -f ds/spod log-enricher
```

Example output

```
I0623 12:51:04.257814 1854764 deleg.go:130] setup "msg"="starting component: log-
```

```
enricher"  "buildDate"="1980-01-01T00:00:00Z" "compiler"="gc" "gitCommit"="unknown"
"gitTreeState"="clean" "goVersion"="go1.16.2" "platform"="linux/amd64" "version"="0.4.0-
dev"
I0623 12:51:04.257890 1854764 enricher.go:44] log-enricher "msg"="Starting log-enricher on
node: 127.0.0.1"
I0623 12:51:04.257898 1854764 enricher.go:46] log-enricher "msg"="Connecting to local
GRPC server"
I0623 12:51:04.258061 1854764 enricher.go:69] log-enricher "msg"="Reading from file
/var/log/audit/audit.log"
2021/06/23 12:51:04 Seeked /var/log/audit/audit.log - &{Offset:0 Whence:2}
```

7.8.7.1. Using the log enricher to trace an application

You can use the Security Profiles Operator log enricher to trace an application.

Procedure

1. To trace an application, create a **SeccompProfile** logging profile:

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  name: log
  namespace: default
spec:
  defaultAction: SCMP_ACT_LOG
```

2. Create a pod object to use the profile:

```
apiVersion: v1
kind: Pod
metadata:
  name: log-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: operator/default/log.json
  containers:
    - name: log-container
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

3. Examine the log enricher output by running the following command:

```
$ oc -n openshift-security-profiles logs -f ds/spod log-enricher
```

Example 7.1. Example output

```
…
I0623 12:59:11.479869 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/" "namespace"="default" "node"="127.0.0.1"
"pid"=1905792 "pod"="log-pod" "syscallID"=3 "syscallName"="close"
"timestamp"="1624453150.205:1061" "type"="seccomp"
I0623 12:59:11.487323 1854764 enricher.go:111] log-enricher "msg"="audit"
```

```
"container"="log-container" "executable"="/" "namespace"="default" "node"="127.0.0.1"
"pid"=1905792 "pod"="log-pod" "syscallID"=157 "syscallName"="prctl"
"timestamp"="1624453150.205:1062" "type"="seccomp"
I0623 12:59:11.492157 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/" "namespace"="default" "node"="127.0.0.1"
"pid"=1905792 "pod"="log-pod" "syscallID"=157 "syscallName"="prctl"
"timestamp"="1624453150.205:1063" "type"="seccomp"
…
I0623 12:59:20.258523 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=12 "syscallName"="brk"
"timestamp"="1624453150.235:2873" "type"="seccomp"
I0623 12:59:20.263349 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=21
"syscallName"="access" "timestamp"="1624453150.235:2874" "type"="seccomp"
I0623 12:59:20.354091 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=257
"syscallName"="openat" "timestamp"="1624453150.235:2875" "type"="seccomp"
I0623 12:59:20.358844 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=5 "syscallName"="fstat"
"timestamp"="1624453150.235:2876" "type"="seccomp"
I0623 12:59:20.363510 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=9 "syscallName"="mmap"
"timestamp"="1624453150.235:2877" "type"="seccomp"
I0623 12:59:20.454127 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=3 "syscallName"="close"
"timestamp"="1624453150.235:2878" "type"="seccomp"
I0623 12:59:20.458654 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=257
"syscallName"="openat" "timestamp"="1624453150.235:2879" "type"="seccomp"
…
```

7.8.8. Configuring webhooks

Profile binding and profile recording objects can use webhooks. Profile binding and recording object configurations are **MutatingWebhookConfiguration** CRs, managed by the Security Profiles Operator.

To change the webhook configuration, the **spod** CR exposes a **webhookOptions** field that allows modification of the **failurePolicy**, **namespaceSelector**, and **objectSelector** variables. This allows you to set the webhooks to "soft-fail" or restrict them to a subset of a namespaces so that even if the webhooks failed, other namespaces or resources are not affected.

Procedure

1. Set the **recording.spo.io** webhook configuration to record only pods labeled with **spo-record=true** by creating the following patch file:

```
spec:
```

```
webhookOptions:
  - name: recording.spo.io
    objectSelector:
      matchExpressions:
        - key: spo-record
          operator: In
          values:
            - "true"
```

2. Patch the **spod**/**spod** instance by running the following command:

```
$ oc -n openshift-security-profiles patch spod \
    spod -p $(cat /tmp/spod-wh.patch) --type=merge
```

3. To view the resulting **MutatingWebhookConfiguration** object, run the following command:

```
$ oc get MutatingWebhookConfiguration \
    spo-mutating-webhook-configuration -oyaml
```

7.9. TROUBLESHOOTING THE SECURITY PROFILES OPERATOR

Troubleshoot the Security Profiles Operator to diagnose a problem or provide information in a bug report.

7.9.1. Inspecting seccomp profiles

Corrupted **seccomp** profiles can disrupt your workloads. Ensure that the user cannot abuse the system by not allowing other workloads to map any part of the path **/var/lib/kubelet/seccomp/operator**.

Procedure

1. Confirm that the profile is reconciled by running the following command:

```
$ oc -n openshift-security-profiles logs openshift-security-profiles-<id>
```

Example 7.2. Example output

```
I1019 19:34:14.942464       1 main.go:90] setup "msg"="starting openshift-security-
profiles"  "buildDate"="2020-10-19T19:31:24Z" "compiler"="gc"
"gitCommit"="a3ef0e1ea6405092268c18f240b62015c247dd9d" "gitTreeState"="dirty"
"goVersion"="go1.15.1" "platform"="linux/amd64" "version"="0.2.0-dev"
I1019 19:34:15.348389       1 listener.go:44] controller-runtime/metrics "msg"="metrics
server is starting to listen"  "addr"=":8080"
I1019 19:34:15.349076       1 main.go:126] setup "msg"="starting manager"
I1019 19:34:15.349449       1 internal.go:391] controller-runtime/manager "msg"="starting
metrics server"  "path"="/metrics"
I1019 19:34:15.350201       1 controller.go:142] controller "msg"="Starting EventSource"
"controller"="profile" "reconcilerGroup"="security-profiles-operator.x-k8s.io"
"reconcilerKind"="SeccompProfile" "source"={"Type":{"metadata":
{"creationTimestamp":null},"spec":{"defaultAction":""}}}
I1019 19:34:15.450674       1 controller.go:149] controller "msg"="Starting Controller"
"controller"="profile" "reconcilerGroup"="security-profiles-operator.x-k8s.io"
"reconcilerKind"="SeccompProfile"
```

```
I1019 19:34:15.450757       1 controller.go:176] controller "msg"="Starting workers"
"controller"="profile" "reconcilerGroup"="security-profiles-operator.x-k8s.io"
"reconcilerKind"="SeccompProfile" "worker count"=1
I1019 19:34:15.453102       1 profile.go:148] profile "msg"="Reconciled profile from
SeccompProfile" "namespace"="openshift-security-profiles" "profile"="nginx-1.19.1"
"name"="nginx-1.19.1" "resource version"="728"
I1019 19:34:15.453618       1 profile.go:148] profile "msg"="Reconciled profile from
SeccompProfile" "namespace"="openshift-security-profiles" "profile"="openshift-security-
profiles" "name"="openshift-security-profiles" "resource version"="729"
```

2. Confirm that the **seccomp** profiles are saved into the correct path by running the following command:

```
$ oc exec -t -n openshift-security-profiles openshift-security-profiles-<id> \
    -- ls /var/lib/kubelet/seccomp/operator/my-namespace/my-workload
```

Example output

```
profile-block.json
profile-complain.json
```

7.10. UNINSTALLING THE SECURITY PROFILES OPERATOR

You can remove the Security Profiles Operator from your cluster by using the OpenShift Container Platform web console.

7.10.1. Uninstall the Security Profiles Operator using the web console

To remove the Security Profiles Operator, you must first delete the **seccomp** and SELinux profiles. After the profiles are removed, you can then remove the Operator and its namespace by deleting the **openshift-security-profiles** project.

Prerequisites

- Access to an OpenShift Container Platform cluster that uses an account with **cluster-admin** permissions.
- The Security Profiles Operator is installed.

Procedure

To remove the Security Profiles Operator by using the OpenShift Container Platform web console:

1. Navigate to the **Operators → Installed Operators** page.
2. Delete all **seccomp** profiles, SELinux profiles, and webhook configurations.
3. Switch to the **Administration → Operators → Installed Operators** page.
4. Click the Options menu on the **Security Profiles Operator** entry and select **Uninstall Operator**.

5. Switch to the **Home → Projects** page.
6. Search for **security profiles**.
7. Click the Options menu  next to the **openshift-security-profiles** project, and select **Delete Project**.
    a. Confirm the deletion by typing **openshift-security-profiles** in the dialog box, and click **Delete**.
8. Delete the **MutatingWebhookConfiguration** object by running the following command:

```
$ oc delete MutatingWebhookConfiguration spo-mutating-webhook-configuration
```

CHAPTER 8. NBDE TANG SERVER OPERATOR

8.1. NBDE TANG SERVER OPERATOR OVERVIEW

Network-bound Disk Encryption (NBDE) provides an automated unlocking of LUKS-encrypted volumes using one or more dedicated network-binding servers. The client side of NBDE is called the Clevis decryption policy framework and the server side is represented by Tang.

The NBDE Tang Server Operator allows the automation of deployments of one or several Tang servers in the OpenShift Container Platform (OCP) environment.

8.2. NBDE TANG SERVER OPERATOR RELEASE NOTES

The following release notes track the development of the NBDE Tang Server Operator in OpenShift Container Platform.

RHEA-2023:7491

The NBDE Tang Server Operator 1.0 has been released in the Red Hat OpenShift Enterprise 4 catalog.

RHEA-2024:0854

The NBDE Tang Server Operator 1.0.1 has been moved from the **alpha** channel to the **stable** channel.

RHBA-2024:8681

The 1.0.2 update contains fixes that increase the Container Health Index of containers deployed with the NBDE Tang Server Operator to the highest grade.

RHEA-2024:10970

The 1.0.3 update contains changes that re-increase the Container Health Index to the highest grade.

RHBA-2025:0663

With the NBDE Tang Server Operator 1.1, the **golang** package is provided in version 1.23.2 and the **golang.org/x/net/html** package has been updated to version 0.33.0. The updates increase the Container Health Index.

8.3. UNDERSTANDING THE NBDE TANG SERVER OPERATOR

You can use the NBDE Tang Server Operator to automate the deployment of a Tang server in an OpenShift Container Platform cluster that requires Network Bound Disk Encryption (NBDE) internally, leveraging the tools that OpenShift Container Platform provides to achieve this automation.

The NBDE Tang Server Operator simplifies the installation process and uses native features provided by the OpenShift Container Platform environment, such as multi-replica deployment, scaling, traffic load balancing, and so on. The Operator also provides automation of certain operations that are error-prone when you perform them manually, for example:

- server deployment and configuration
- key rotation
- hidden keys deletion

The NBDE Tang Server Operator is implemented using the Operator SDK and allows the deployment of one or more Tang servers in OpenShift through custom resource definitions (CRDs).

8.3.1. Additional resources

- Tang-Operator: Providing NBDE in OpenShift Red Hat Hybrid Cloud blog article
- tang-operator Github project
- Configuring automated unlocking of encrypted volumes using policy-based decryption chapter in the RHEL 9 Security hardening document

8.4. INSTALLING THE NBDE TANG SERVER OPERATOR

You can install the NBDE Tang Operator either by using the web console or through the **oc** command from CLI.

8.4.1. Installing the NBDE Tang Server Operator using the web console

You can install the NBDE Tang Server Operator from the OperatorHub using the web console.

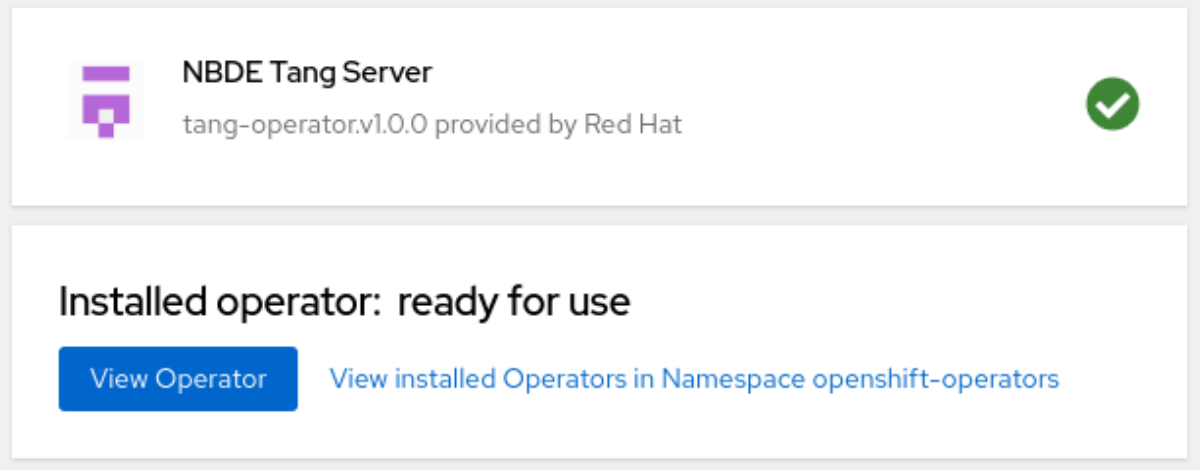
Prerequisites

- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.

Procedure

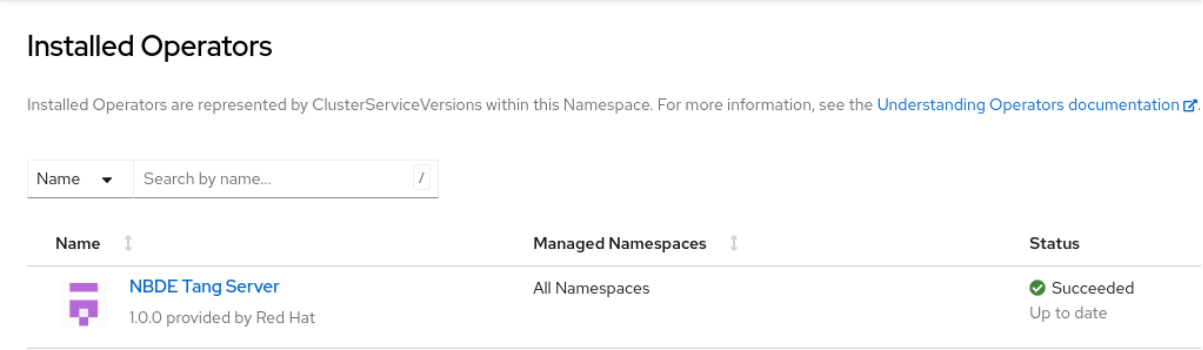
1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Search for the NBDE Tang Server Operator:

3. Click **Install**.
4. On the **Operator Installation** screen, keep the **Update channel**, **Version**, **Installation mode**, **Installed Namespace**, and **Update approval** fields on the default values.
5. After you confirm the installation options by clicking **Install**, the console displays the installation confirmation.



Verification

1. Navigate to the **Operators → Installed Operators** page.
2. Check that the NBDE Tang Server Operator is installed and its status is **Succeeded**.



8.4.2. Installing the NBDE Tang Server Operator using CLI

You can install the NBDE Tang Server Operator from the OperatorHub using the CLI.

Prerequisites

- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Use the following command to list available Operators on OperatorHub, and limit the output to Tang-related results:

```
$ oc get packagemanifests -n openshift-marketplace | grep tang
```

Example output

```
tang-operator        Red Hat
```

In this case, the corresponding packagemanifest name is **tang-operator**.

2. Create a **Subscription** object YAML file to subscribe a namespace to the NBDE Tang Server Operator, for example, **tang-operator.yaml**:

   Example subscription YAML for tang-operator

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: tang-operator
  namespace: openshift-operators
spec:
  channel: stable (1)
  installPlanApproval: Automatic
  name: tang-operator (2)
  source: redhat-operators (3)
  sourceNamespace: openshift-marketplace (4)
```

   (1) Specify the channel name from where you want to subscribe the Operator.

   (2) Specify the name of the Operator to subscribe to.

   (3) Specify the name of the CatalogSource that provides the Operator.

   (4) The namespace of the CatalogSource. Use **openshift-marketplace** for the default OperatorHub CatalogSources.

3. Apply the **Subscription** to the cluster:

```
$ oc apply -f tang-operator.yaml
```

Verification

- Check that the NBDE Tang Server Operator controller runs in the **openshift-operators** namespace:

```
$ oc -n openshift-operators get pods
```

  Example output

```
NAME                                        READY   STATUS    RESTARTS   AGE
tang-operator-controller-manager-694b754bd6-4zk7x   2/2     Running   0          12s
```

8.5. CONFIGURING AND MANAGING TANG SERVERS USING THE NBDE TANG SERVER OPERATOR

With the NBDE Tang Server Operator, you can deploy and quickly configure Tang servers. On the deployed Tang servers, you can list existing keys and rotate them.

8.5.1. Deploying a Tang server using the NBDE Tang Server Operator

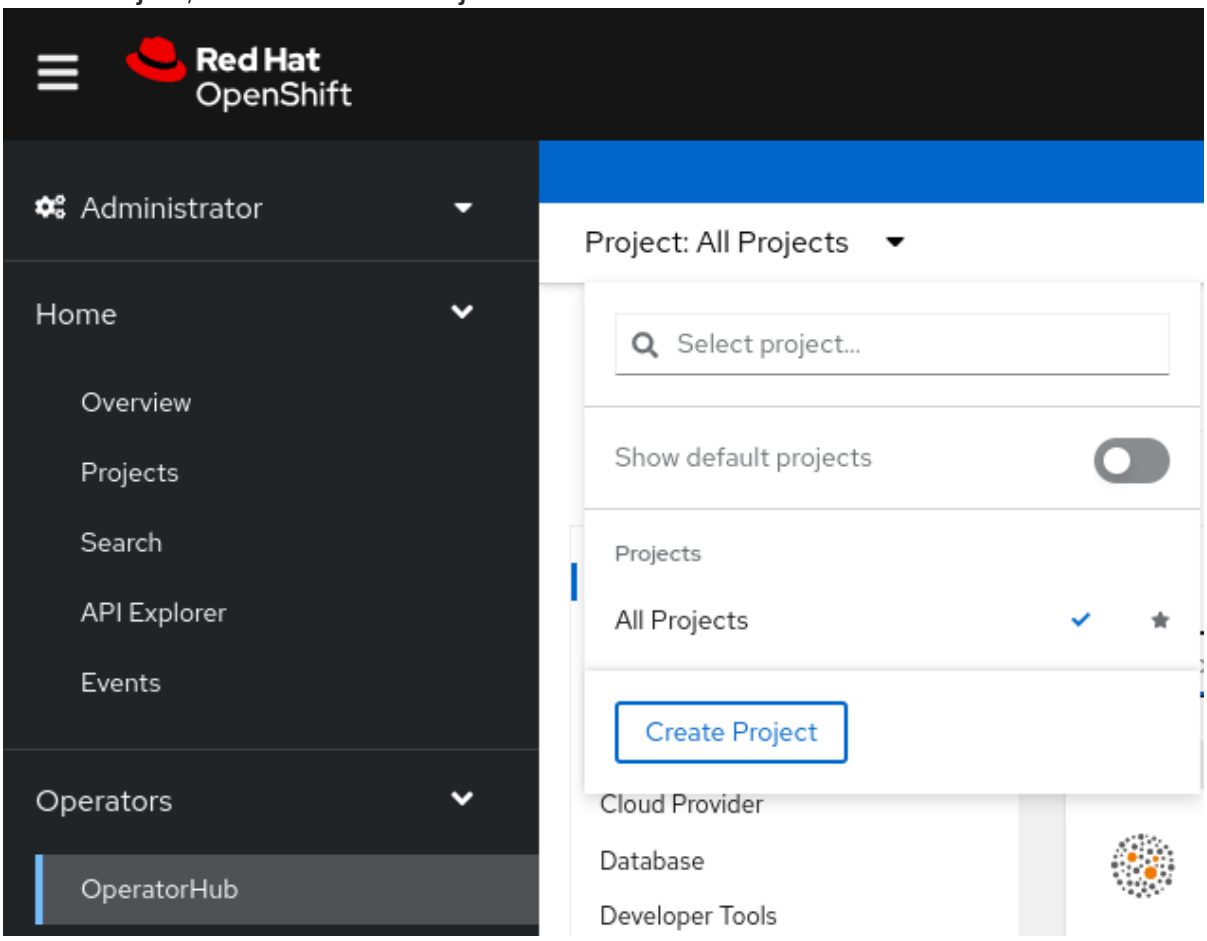
You can deploy and quickly configure one or more Tang servers using the NBDE Tang Server Operator in the web console.

Prerequisites

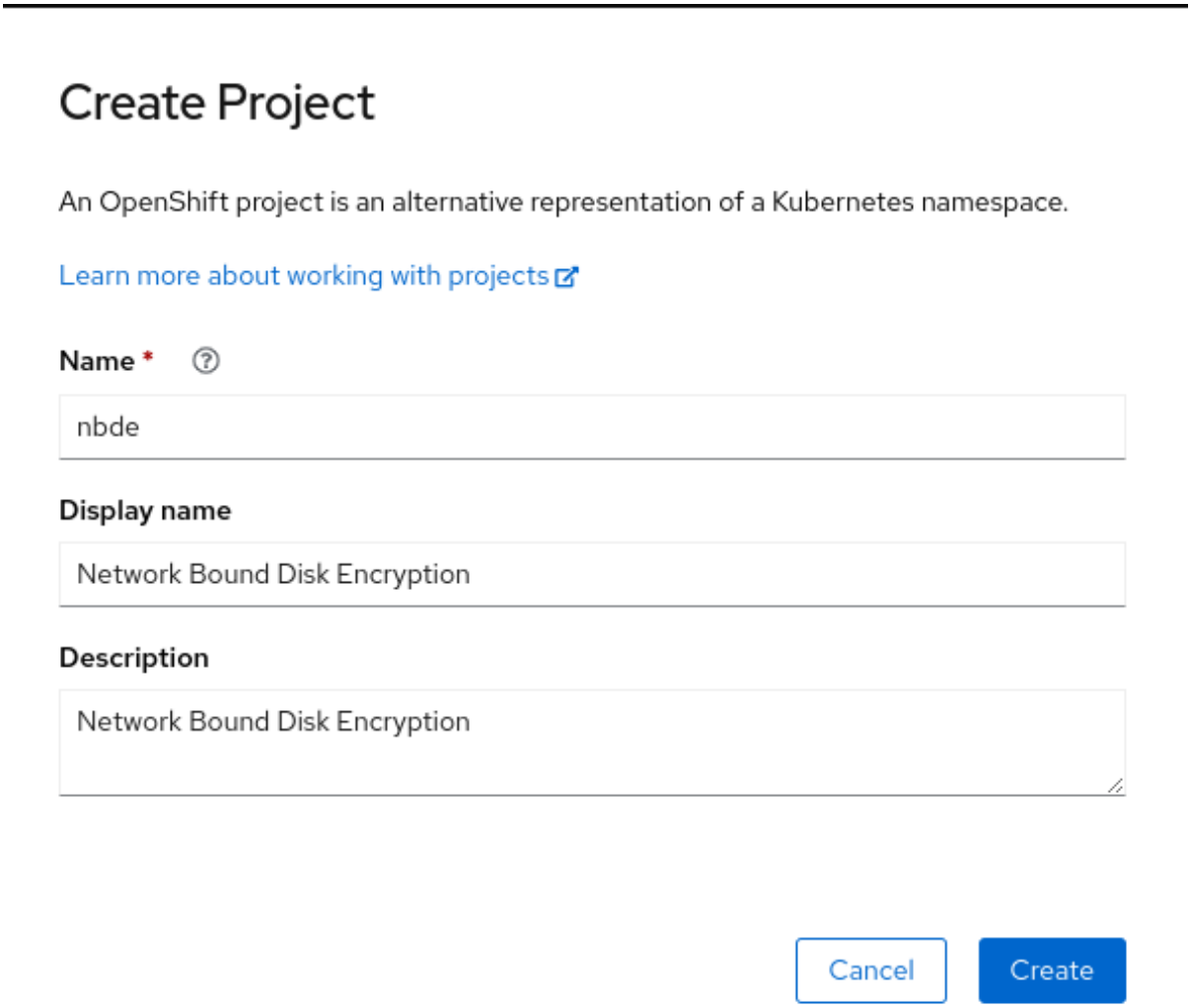
- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.
- You have installed the NBDE Tang Server Operator on your OCP cluster.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Select **Project**, and click **Create Project**:

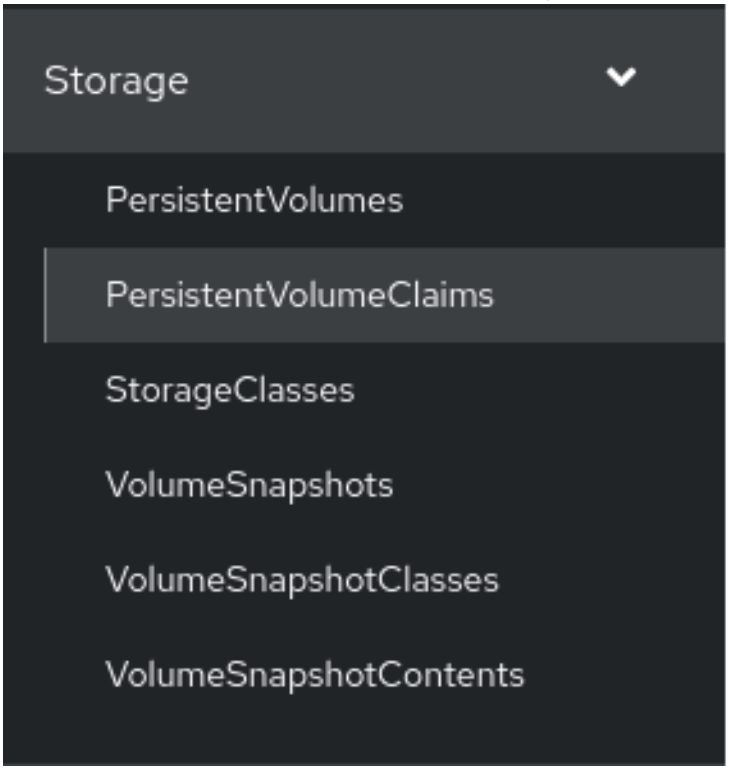


3. On the **Create Project** page, fill in the required information, for example:



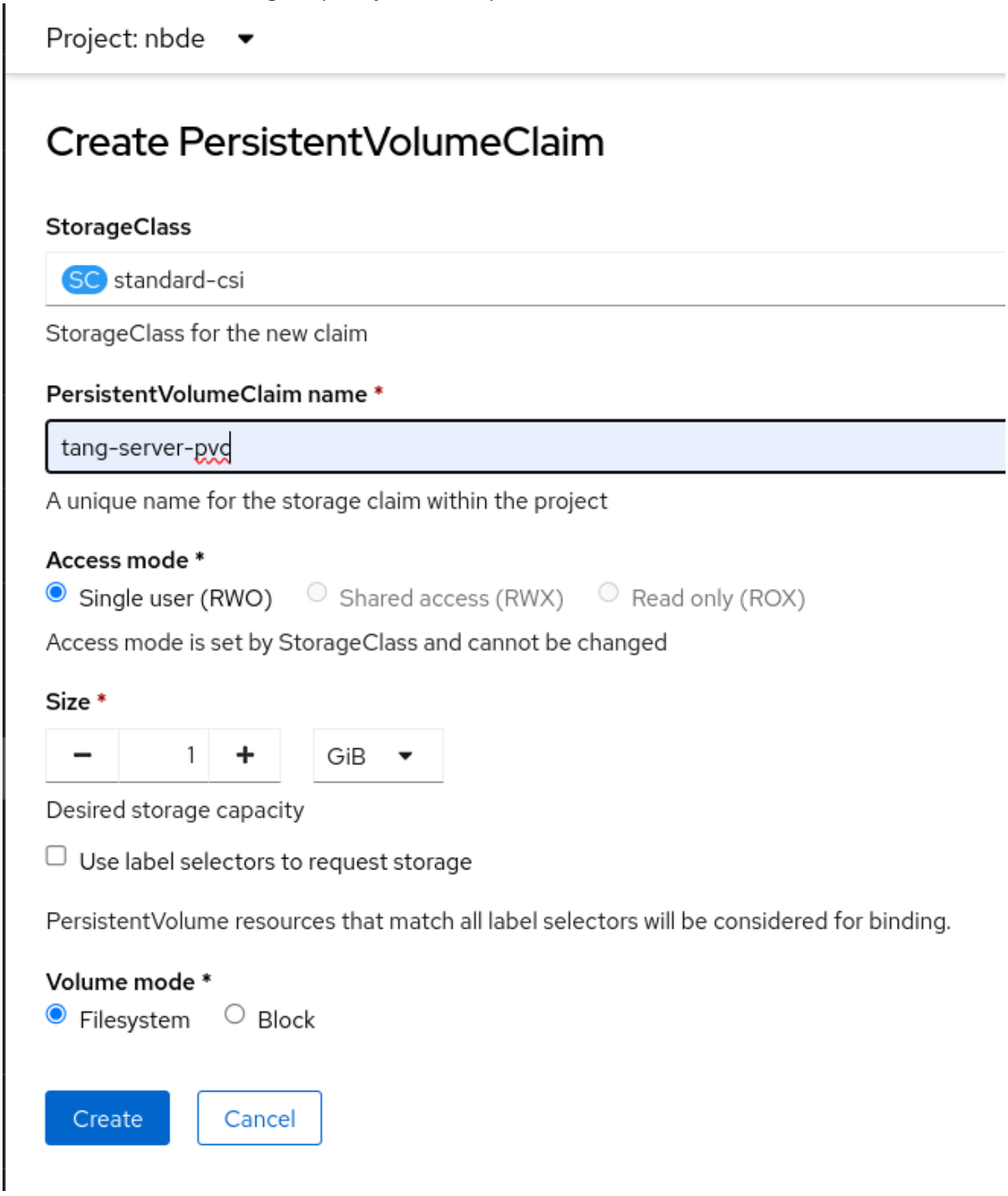
4. Click **Create**.

5. NBDE Tang Server replicas require a Persistent Volume Claim (PVC) for storing encryption keys. In the web console, navigate to **Storage → PersistentVolumeClaims**:



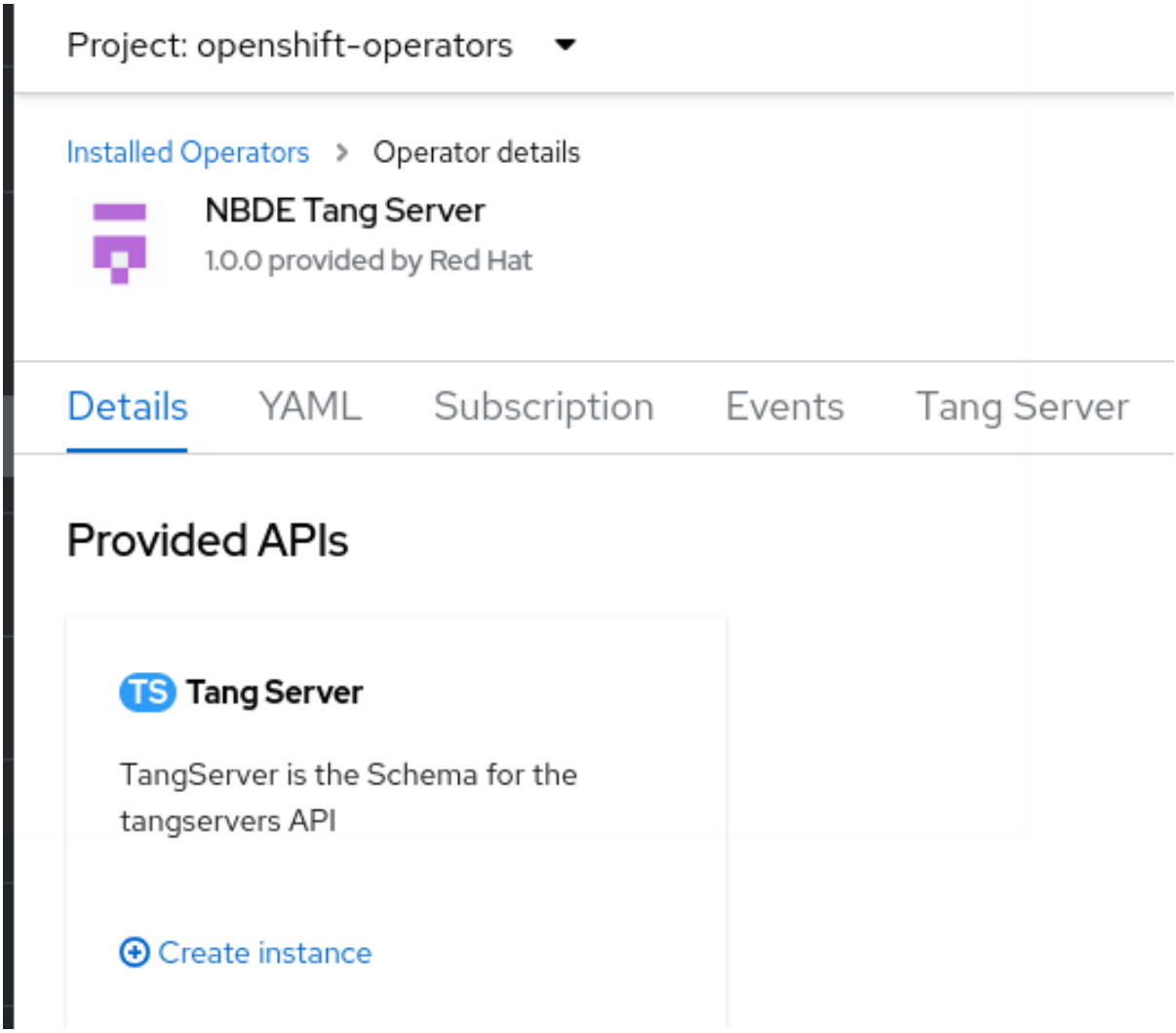
6. On the following **PersistentVolumeClaims** screen, click **Create PersistentVolumeClaim**.

7. On the **Create PersistentVolumeClaim** page, select a storage that fits your deployment scenario. Consider how often you want to rotate the encryption keys. Name your PVC and choose the claimed storage capacity, for example:

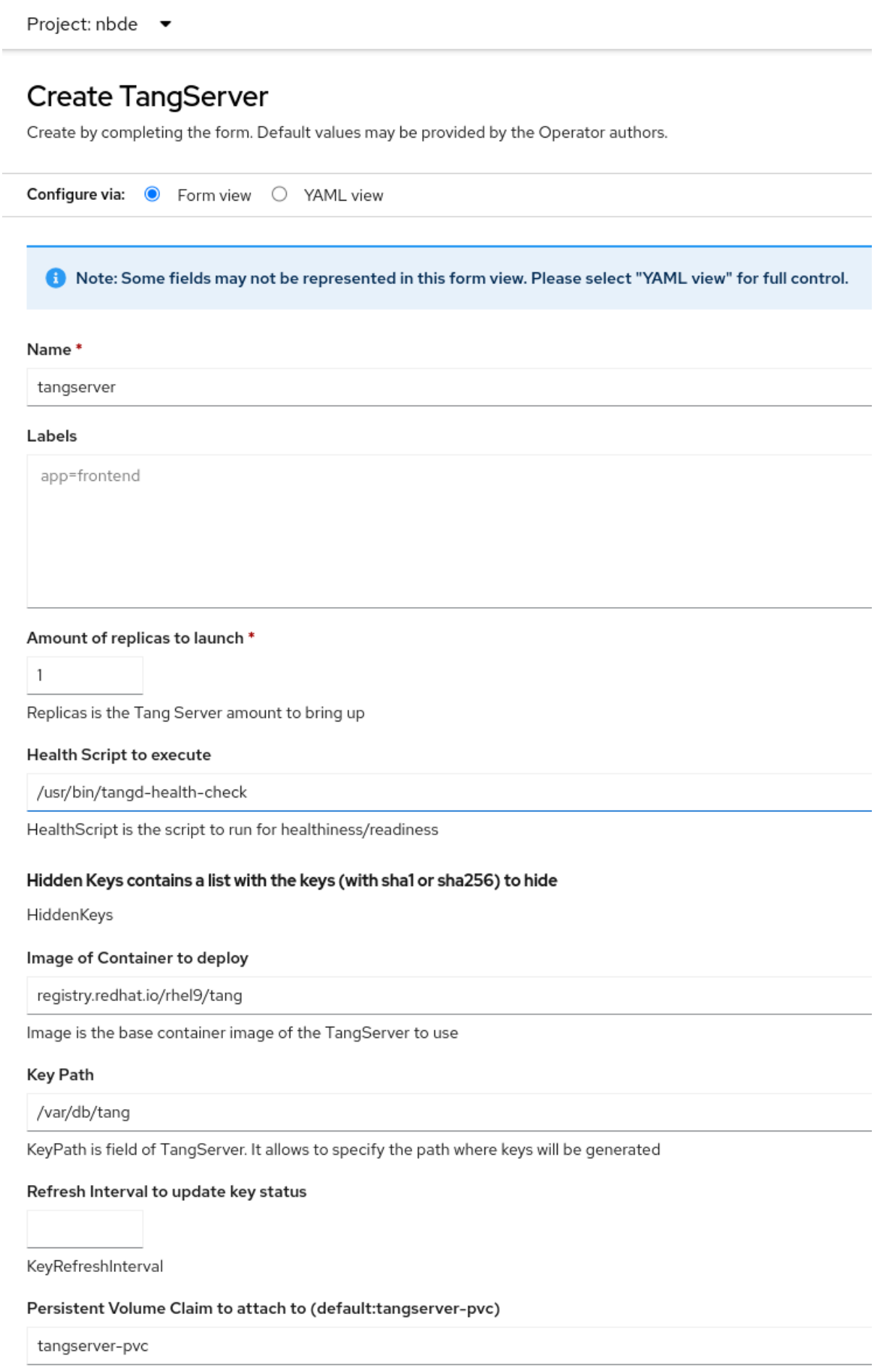


8. Navigate to **Operators → Installed Operators**, and click **NBDE Tang Server**.

9. Click **Create instance**.



10. On the **Create TangServer** page, choose the name of the Tang Server instance, amount of replicas, and specify the name of the previously created Persistent Volume Claim, for example:



11. After you enter the required values a change settings that differ from the default values in your scenario, click **Create**.

8.5.2. Rotating keys using the NBDE Tang Server Operator

With the NBDE Tang Server Operator, you also can rotate your Tang server keys. The precise interval at which you should rotate them depends on your application, key sizes, and institutional policy.

Prerequisites

- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.
- You deployed a Tang server using the NBDE Tang Server Operator on your OpenShift cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. List the existing keys on your Tang server, for example:

```
$ oc -n nbde describe tangserver
```

Example output

```
…
Status:
  Active Keys:
 File Name:   QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
 Generated:   2022-02-08 15:44:17.030090484 +0000
 sha1:        PvYQKtrTuYsMV2AomUeHrUWkCGg
 sha256:      QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

2. Create a YAML file for moving your active keys to hidden keys, for example, **minimal-keyretrieve-rotate-tangserver.yaml**:

Example key-rotation YAML for tang-operator

```
apiVersion: daemons.redhat.com/v1alpha1
kind: TangServer
metadata:
  name: tangserver
  namespace: nbde
  finalizers:
    - finalizer.daemons.tangserver.redhat.com
spec:
  replicas: 1
  hiddenKeys:
    - sha1: "PvYQKtrTuYsMV2AomUeHrUWkCGg" (1)
```

(1) Specify the SHA-1 thumbprint of your active key to rotate it.

3. Apply the YAML file:

```
$ oc apply -f minimal-keyretrieve-rotate-tangserver.yaml
```

Verification

1. After a certain amount of time depending on your configuration, check that the previous **activeKey** value is the new **hiddenKey** value and the **activeKey** key file is newly generated, for example:

```
$ oc -n nbde describe tangserver
```

Example output

```
…
Spec:
  Hidden Keys:
    sha1:    PvYQKtrTuYsMV2AomUeHrUWkCGg
  Replicas:  1
Status:
  Active Keys:
    File Name:  T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY.jwk
    Generated:  2023-10-25 15:38:18.134939752 +0000
    sha1:       vVxkNCNq7gygeeA9zrHrbc3_NZ4
    sha256:     T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY
  Hidden Keys:
    File Name:          .QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg.jwk
    Generated:          2023-10-25 15:37:29.126928965 +0000
    Hidden:             2023-10-25 15:38:13.515467436 +0000
    sha1:               PvYQKtrTuYsMV2AomUeHrUWkCGg
    sha256:             QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

8.5.3. Deleting hidden keys with the NBDE Tang Server Operator

After you rotate your Tang server keys, the previously active keys become hidden and are no longer advertised by the Tang instance. You can use the NBDE Tang Server Operator to remove encryption keys no longer used.

WARNING

Do not remove any hidden keys unless you are sure that all bound Clevis clients already use new keys.

Prerequisites

- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.
- You deployed a Tang server using the NBDE Tang Server Operator on your OpenShift cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. List the existing keys on your Tang server, for example:

```
$ oc -n nbde describe tangserver
```

Example output

-

```
…
Status:
  Active Keys:
 File Name:   PvYQKtrTuYsMV2AomUeHrUWkCGg.jwk
 Generated:   2022-02-08 15:44:17.030090484 +0000
 sha1:      PvYQKtrTuYsMV2AomUeHrUWkCGg
 sha256:      QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

2. Create a YAML file for removing all hidden keys, for example, **hidden-keys-deletion-tangserver.yaml**:

   Example hidden-keys-deletion YAML for tang-operator

```
apiVersion: daemons.redhat.com/v1alpha1
kind: TangServer
metadata:
  name: tangserver
  namespace: nbde
  finalizers:
    - finalizer.daemons.tangserver.redhat.com
spec:
  replicas: 1
  hiddenKeys: [] 1
```

   1 The empty array as the value of the **hiddenKeys** entry indicates you want to preserve no hidden keys on your Tang server.

3. Apply the YAML file:

```
$ oc apply -f hidden-keys-deletion-tangserver.yaml
```

Verification

1. After a certain amount of time depending on your configuration, check that the previous active key still exists, but no hidden key is available, for example:

```
$ oc -n nbde describe tangserver
```

   Example output

```
…
Spec:
  Hidden Keys:
    sha1:    PvYQKtrTuYsMV2AomUeHrUWkCGg
  Replicas:  1
Status:
  Active Keys:
    File Name:  T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY.jwk
    Generated:  2023-10-25 15:38:18.134939752 +0000
    sha1:       vVxkNCNq7gygeeA9zrHrbc3_NZ4
    sha256:     T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY
Status:
```

```
  Ready:                1
  Running:              1
  Service External URL:  http://35.222.247.84:7500/adv
  Tang Server Error:     No
Events:
…
```

8.6. IDENTIFYING URL OF A TANG SERVER DEPLOYED WITH THE NBDE TANG SERVER OPERATOR

Before you can configure your Clevis clients to use encryption keys advertised by your Tang servers, you must identify the URLs of the servers.

8.6.1. Identifying URL of the NBDE Tang Server Operator using the web console

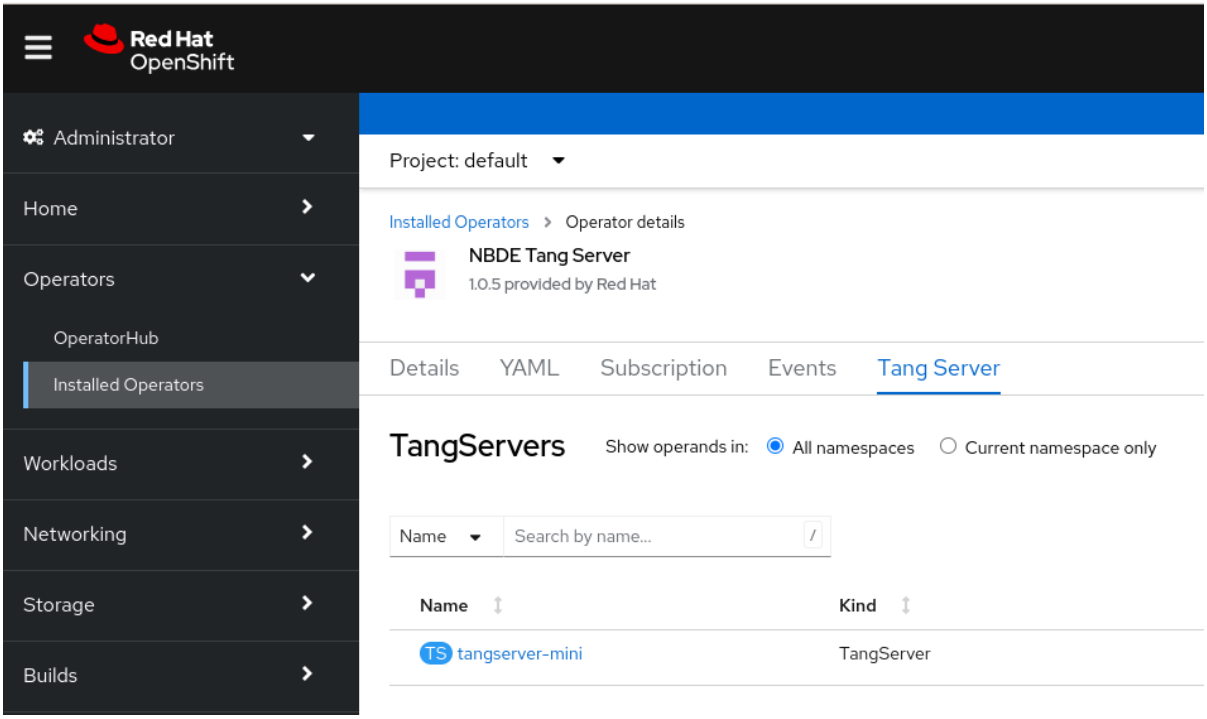
You can identify the URLs of Tang servers deployed with the NBDE Tang Server Operator from the OperatorHub by using the OpenShift Container Platform web console. After you identify the URLs, you use the **clevis luks bind** command on your clients containing LUKS-encrypted volumes that you want to unlock automatically by using keys advertised by the Tang servers. See the Configuring manual enrollment of LUKS-encrypted volumes section in the RHEL 9 Security hardening document for detailed steps describing the configuration of clients with Clevis.

Prerequisites

- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.
- You deployed a Tang server by using the NBDE Tang Server Operator on your OpenShift cluster.

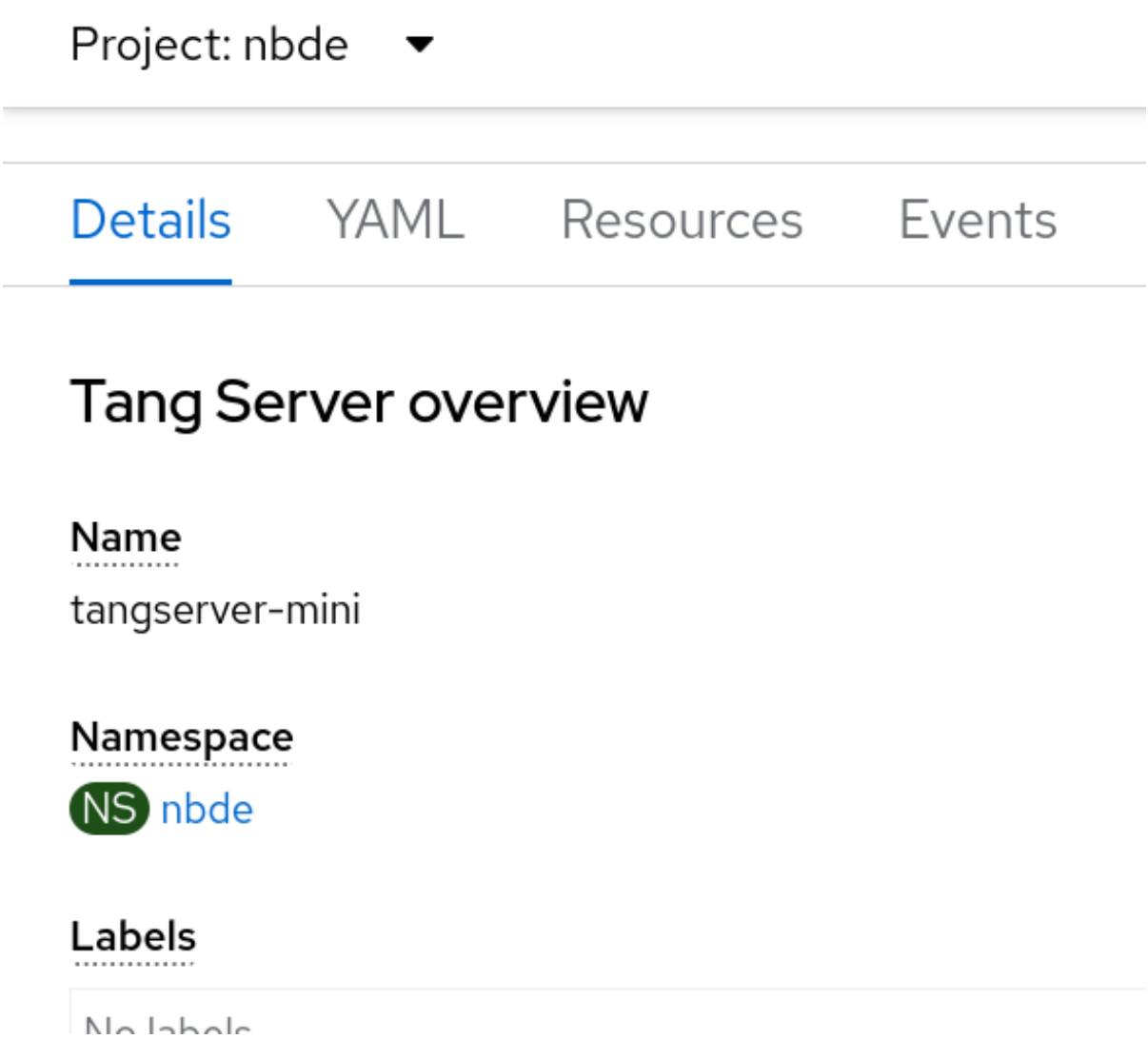
Procedure

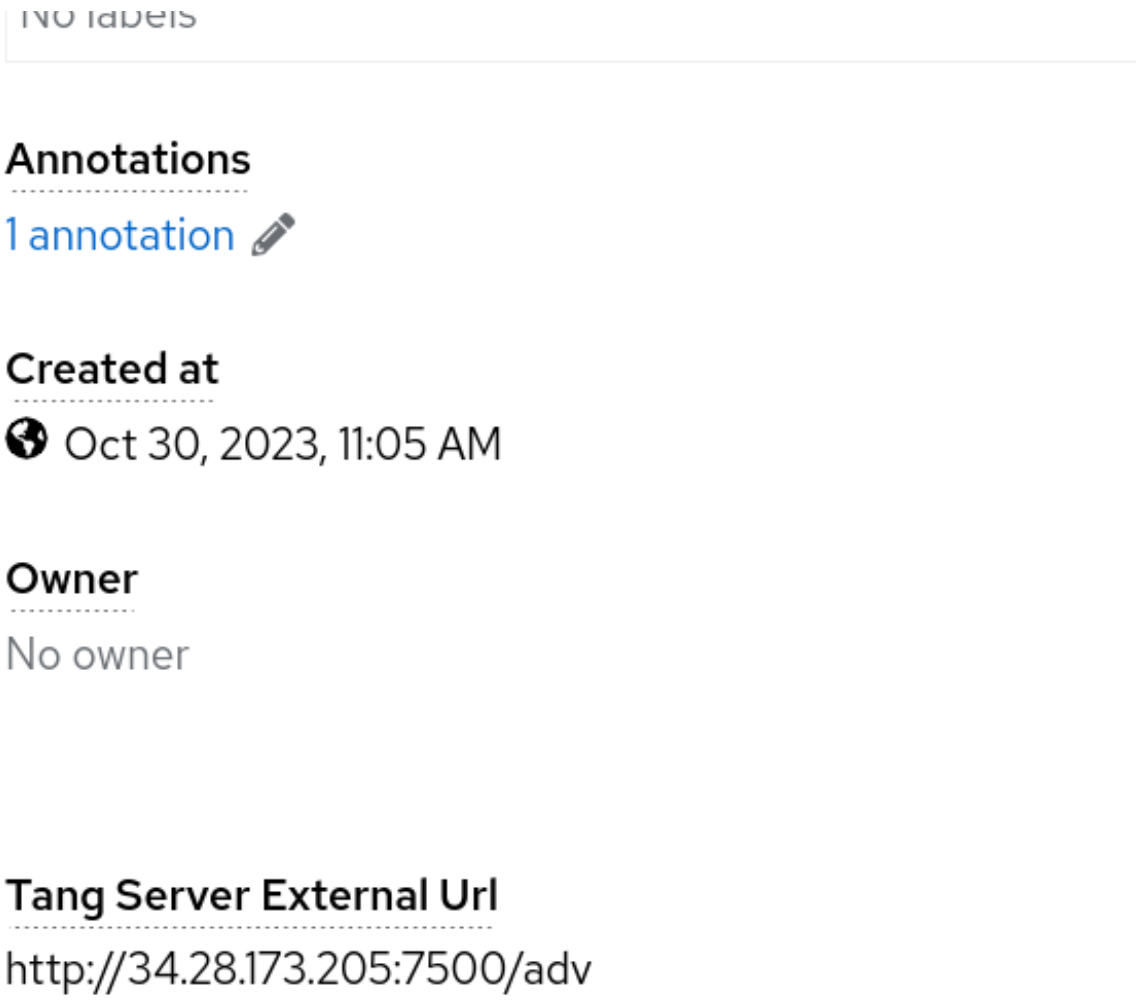
1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators → Tang Server**.
2. On the NBDE Tang Server Operator details page, select **Tang Server**.



3. The list of Tang servers deployed and available for your cluster appears. Click the name of the Tang server you want to bind with a Clevis client.

4. The web console displays an overview of the selected Tang server. You can find the URL of your Tang server in the **Tang Server External Url** section of the screen:





In this example, the URL of the Tang server is **http://34.28.173.205:7500**.

Verification

- You can check that the Tang server is advertising by using **curl**, **wget**, or similar tools, for example:

```
$ curl 2> /dev/null http://34.28.173.205:7500/adv  | jq
```

Example output

```
{
  "payload": "eyJrZXlzIj…eSJdfV19",
  "protected": "eyJhbGciOiJFUzUxMiIsImN0eSI6Imp3ay1zZXQranNvbiJ9",
  "signature": "AUB0qSFx0FJLeTU…aV_GYWlDx50vCXKNyMMCRx"
}
```

8.6.2. Identifying URL of the NBDE Tang Server Operator using CLI

You can identify the URLs of Tang servers deployed with the NBDE Tang Server Operator from the OperatorHub by using the CLI. After you identify the URLs, you use the **clevis luks bind** command on your clients containing LUKS-encrypted volumes that you want to unlock automatically by using keys

advertised by the Tang servers. See the Configuring manual enrollment of LUKS-encrypted volumes section in the RHEL 9 Security hardening document for detailed steps describing the configuration of clients with Clevis.

Prerequisites

- You must have **cluster-admin** privileges on an OpenShift Container Platform cluster.
- You have installed the OpenShift CLI (**oc**).
- You deployed a Tang server by using the NBDE Tang Server Operator on your OpenShift cluster.

Procedure

1. List details about your Tang server, for example:

```
$ oc -n nbde describe tangserver
```

Example output

```
…
Spec:
…
Status:
  Ready:               1
  Running:             1
  Service External URL:  http://34.28.173.205:7500/adv
  Tang Server Error:     No
Events:
…
```

2. Use the value of the **Service External URL:** item without the **/adv** part. In this example, the URL of the Tang server is **http://34.28.173.205:7500**.

Verification

- You can check that the Tang server is advertising by using **curl**, **wget**, or similar tools, for example:

```
$ curl 2> /dev/null http://34.28.173.205:7500/adv  | jq
```

Example output

```
{
  "payload": "eyJrZXlzIj…eSJdfV19",
  "protected": "eyJhbGciOiJFUzUxMiIsImN0eSI6Imp3ay1zZXQranNvbiJ9",
  "signature": "AUB0qSFx0FJLeTU…aV_GYWlDx50vCXKNyMMCRx"
}
```

8.6.3. Additional resources

- Configuring manual enrollment of LUKS-encrypted volumes section in the RHEL 9 Security hardening document.

CHAPTER 9. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

9.1. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT OVERVIEW

The cert-manager Operator for Red Hat OpenShift is a cluster-wide service that provides application certificate lifecycle management. The cert-manager Operator for Red Hat OpenShift allows you to integrate with external certificate authorities and provides certificate provisioning, renewal, and retirement.

9.1.1. About the cert-manager Operator for Red Hat OpenShift

The cert-manager project introduces certificate authorities and certificates as resource types in the Kubernetes API, which makes it possible to provide certificates on demand to developers working within your cluster. The cert-manager Operator for Red Hat OpenShift provides a supported way to integrate cert-manager into your OpenShift Container Platform cluster.

The cert-manager Operator for Red Hat OpenShift provides the following features:

- Support for integrating with external certificate authorities
- Tools to manage certificates
- Ability for developers to self-serve certificates
- Automatic certificate renewal

IMPORTANT

Do not attempt to use both cert-manager Operator for Red Hat OpenShift for OpenShift Container Platform and the community cert-manager Operator at the same time in your cluster.

Also, you should not install cert-manager Operator for Red Hat OpenShift for OpenShift Container Platform in multiple namespaces within a single OpenShift cluster.

9.1.2. Supported issuer types

The cert-manager Operator for Red Hat OpenShift supports the following issuer types:

- Automated Certificate Management Environment (ACME)
- Certificate authority (CA)
- Self-signed
- Vault
- Venafi

9.1.3. Certificate request methods

There are two ways to request a certificate using the cert-manager Operator for Red Hat OpenShift:

Using the cert-manager.io/CertificateRequest object

With this method a service developer creates a **CertificateRequest** object with a valid **issuerRef** pointing to a configured issuer (configured by a service infrastructure administrator). A service infrastructure administrator then accepts or denies the certificate request. Only accepted certificate requests create a corresponding certificate.

Using the cert-manager.io/Certificate object

With this method, a service developer creates a **Certificate** object with a valid **issuerRef** and obtains a certificate from a secret that they pointed to the **Certificate** object.

9.1.4. Additional resources

- cert-manager project documentation

9.2. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT RELEASE NOTES

The cert-manager Operator for Red Hat OpenShift is a cluster-wide service that provides application certificate lifecycle management.

These release notes track the development of cert-manager Operator for Red Hat OpenShift.

For more information, see About the cert-manager Operator for Red Hat OpenShift .

9.2.1. cert-manager Operator for Red Hat OpenShift 1.13.1

Issued: 2024-05-15

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.13.1:

- RHEA-2024:2849

Version **1.13.1** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.13.6**. For more information, see the cert-manager project release notes for v1.13.6 .

9.2.1.1. CVEs

- CVE-2023-45288
- CVE-2023-48795
- CVE-2024-24783

9.2.2. cert-manager Operator for Red Hat OpenShift 1.13.0

Issued: 2024-01-16

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.13.0:

- RHEA-2024:0260

Version **1.13.0** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.13.3**. For more information, see the cert-manager project release notes for v1.13.0 .

9.2.2.1. New features and enhancements

- You can now manage certificates for API Server and Ingress Controller by using the cert-manager Operator for Red Hat OpenShift. For more information, see Configuring certificates with an issuer.
- With this release, the scope of the cert-manager Operator for Red Hat OpenShift, which was previously limited to the OpenShift Container Platform on AMD64 architecture, has now been expanded to include support for managing certificates on OpenShift Container Platform running on IBM Z (**s390x**), IBM Power (**ppc64le**), and ARM64 architectures.
- With this release, you can use DNS over HTTPS (DoH) for performing the self-checks during the ACME DNS-01 challenge verification. The DNS self-check method can be controlled by using the command-line flags, **--dns01-recursive-nameservers-only** and **--dns01-recursive-nameservers**. For more information, see Customizing cert-manager by overriding arguments from the cert-manager Operator API.

9.2.2.2. CVEs

- CVE-2023-39615
- CVE-2023-3978
- CVE-2023-37788
- CVE-2023-29406

9.2.3. Release notes for cert-manager Operator for Red Hat OpenShift 1.12.1

Issued: 2023-11-15

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.12.1:

- RHSA-2023:6269-02

Version **1.12.1** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.12.5**. For more information, see the cert-manager project release notes for v1.12.5 .

9.2.3.1. Bug fixes

- Previously, in a multi-architecture environment, the cert-manager Operator pods were prone to failures because of the invalid node affinity configuration. With this fix, the cert-manager Operator pods run without any failures. (**OCPBUGS-19446**)

9.2.3.2. CVEs

- CVE-2023-44487
- CVE-2023-39325
- CVE-2023-4527

- CVE-2023-4806
- CVE-2023-4813
- CVE-2023-4911
- CVE-2023-38545
- CVE-2023-38546

9.2.4. Release notes for cert-manager Operator for Red Hat OpenShift 1.12.0

Issued: 2023-10-02

The following advisories are available for the cert-manager Operator for Red Hat OpenShift 1.12.0:

- RHEA-2023:5339
- RHBA-2023:5412

Version **1.12.0** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.12.4**. For more information, see the cert-manager project release notes for v1.12.4 .

9.2.4.1. Bug fixes

- Previously, you could not configure the CPU and memory requests and limits for the cert-manager components such as cert-manager controller, CA injector, and Webhook. Now, you can configure the CPU and memory requests and limits for the cert-manager components by using the command-line interface (CLI). For more information, see Overriding CPU and memory limits for the cert-manager components. (**OCPBUGS-13830**)
- Previously, if you updated the **ClusterIssuer** object, the cert-manager Operator for Red Hat OpenShift could not verify and update the change in the cluster issuer. Now, if you modify the **ClusterIssuer** object, the cert-manager Operator for Red Hat OpenShift verifies the ACME account registration and updates the change. (**OCPBUGS-8210**)
- Previously, the cert-manager Operator for Red Hat OpenShift did not support enabling the **--enable-certificate-owner-ref** flag. Now, the cert-manager Operator for Red Hat OpenShift supports enabling the **--enable-certificate-owner-ref** flag by adding the **spec.controllerConfig.overrideArgs** field in the **cluster** object. After enabling the **--enable-certificate-owner-ref** flag, cert-manager can automatically delete the secret when the **Certificate** resource is removed from the cluster. For more information on enabling the **--enable-certificate-owner-ref** flag and deleting the TLS secret automatically, see Deleting a TLS secret automatically upon Certificate removal (**CM-98**)
- Previously, the cert-manager Operator for Red Hat OpenShift could not pull the **jetstack-cert-manager-container-v1.12.4-1** image. The cert-manager controller, CA injector, and Webhook pods were stuck in the **ImagePullBackOff** state. Now, the cert-manager Operator for Red Hat OpenShift pulls the **jetstack-cert-manager-container-v1.12.4-1** image to run the cert-manager controller, CA injector, and Webhook pods successfully. (**OCPBUGS-19986**)

9.2.5. Release notes for cert-manager Operator for Red Hat OpenShift 1.11.5

Issued: 2023-11-15

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.11.5:

- RHSA-2023:6279-03

The golang version is updated to the version **1.20.10** to fix Common Vulnerabilities and Exposures (CVEs). Version **1.11.5** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.11.5**. For more information, see the cert-manager project release notes for v1.11.5.

9.2.5.1. Bug fixes

- Previously, in a multi-architecture environment, the cert-manager Operator pods were prone to failures because of the invalid node affinity configuration. With this fix, the cert-manager Operator pods run without any failures. (**OCPBUGS-19446**)

9.2.5.2. CVEs

- CVE-2023-44487
- CVE-2023-39325
- CVE-2023-29409
- CVE-2023-2602
- CVE-2023-2603
- CVE-2023-4527
- CVE-2023-4806
- CVE-2023-4813
- CVE-2023-4911
- CVE-2023-28484
- CVE-2023-29469
- CVE-2023-38545
- CVE-2023-38546

9.2.6. Release notes for cert-manager Operator for Red Hat OpenShift 1.11.4

Issued: 2023-07-26

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.11.4:

- RHEA-2023:4081

The golang version is updated to the version **1.19.10** to fix Common Vulnerabilities and Exposures (CVEs). Version **1.11.4** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.11.4**. For more information, see the cert-manager project release notes for v1.11.4.

9.2.6.1. Bug fixes

- Previously, the cert-manager Operator for Red Hat OpenShift did not allow you to install older versions of the cert-manager Operator for Red Hat OpenShift. Now, you can install older versions of the cert-manager Operator for Red Hat OpenShift using the web console or the command-line interface (CLI). For more information on how to use the web console to install older versions, see Installing the cert-manager Operator for Red Hat OpenShift . (**OCPBUGS-16393**)

9.2.7. Release notes for cert-manager Operator for Red Hat OpenShift 1.11.1

Issued: 2023-06-21

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.11.1:

- RHEA-2023:3439

Version **1.11.1** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.11.1**. For more information, see the cert-manager project release notes for v1.11.1 .

9.2.7.1. New features and enhancements

This is the general availability (GA) release of the cert-manager Operator for Red Hat OpenShift.

9.2.7.1.1. Setting log levels for cert-manager and the cert-manager Operator for Red Hat OpenShift

- To troubleshoot issues with cert-manager and the cert-manager Operator for Red Hat OpenShift, you can now configure the log level verbosity by setting a log level for cert-manager and the cert-manager Operator for Red Hat OpenShift. For more information, see Configuring log levels for cert-manager and the cert-manager Operator for Red Hat OpenShift.

9.2.7.1.2. Authenticating the cert-manager Operator for Red Hat OpenShift with AWS

- You can now configure cloud credentials for the cert-manager Operator for Red Hat OpenShift on AWS clusters with Security Token Service (STS) and without STS. For more information, see Authenticating the cert-manager Operator for Red Hat OpenShift on AWS Security Token Service and Authenticating the cert-manager Operator for Red Hat OpenShift on AWS .

9.2.7.1.3. Authenticating the cert-manager Operator for Red Hat OpenShift with GCP

- You can now configure cloud credentials for the cert-manager Operator for Red Hat OpenShift on GCP clusters with Workload Identity and without Workload Identity. For more information, see Authenticating the cert-manager Operator for Red Hat OpenShift with GCP Workload Identity and Authenticating the cert-manager Operator for Red Hat OpenShift with GCP

9.2.7.2. Bug fixes

- Previously, the **cm-acme-http-solver** pod did not use the latest published Red Hat image **registry.redhat.io/cert-manager/jetstack-cert-manager-acmesolver-rhel9**. With this release, the **cm-acme-http-solver** pod uses the latest published Red Hat image **registry.redhat.io/cert-manager/jetstack-cert-manager-acmesolver-rhel9**. (OCPBUGS-10821)

- Previously, the cert-manager Operator for Red Hat OpenShift did not support changing labels for cert-manager pods such as controller, CA injector, and Webhook pods. With this release, you can add labels to cert-manager pods. (**OCPBUGS-8466**)

- Previously, you could not update the log verbosity level in the cert-manager Operator for Red Hat OpenShift. You can now update the log verbosity level by using an environmental variable **OPERATOR_LOG_LEVEL** in its subscription resource. (OCPBUGS-9994)

- Previously, when uninstalling the cert-manager Operator for Red Hat OpenShift, if you select the **Delete all operand instances for this operator**checkbox in the OpenShift Container Platform web console, the Operator was not uninstalled properly. The cert-manager Operator for Red Hat OpenShift is now properly uninstalled. (OCPBUGS-9960)

- Previously, the cert-manager Operator for Red Hat OpenShift did not support using Google workload identity federation. The cert-manager Operator for Red Hat OpenShift now supports using Google workload identity federation. (OCPBUGS-9998)

9.2.7.3. Known issues

- After installing the cert-manager Operator for Red Hat OpenShift, if you navigate to **Operators → Installed Operators** and select **Operator details** in the OpenShift Container Platform web console, you cannot see the cert-manager resources that are created across all namespaces. As a workaround, you can navigate to **Home → API Explorer** to see the cert-manager resources. (OCPBUGS-11647)

- After uninstalling the cert-manager Operator for Red Hat OpenShift by using the web console, the cert-manager Operator for Red Hat OpenShift does not remove the cert-manager controller, CA injector, and Webhook pods automatically from the **cert-manager** namespace. As a workaround, you can manually delete the cert-manager controller, CA injector, and Webhook pod deployments present in the **cert-manager** namespace. (OCPBUGS-13679)

9.2.8. Release notes for cert-manager Operator for Red Hat OpenShift 1.10.3

Issued: 2023-08-08

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.10.3:

- RHSA-2023:4335

The version **1.10.3** of the cert-manager Operator for Red Hat OpenShift is based on the **cert-manager** upstream version **v1.10.2**. With this release, the version of the cert-manager Operator for Red Hat OpenShift is **1.10.3** but the **cert-manager** operand version is **1.10.2**. For more information, see the cert-manager project release notes for v1.10.2 .

9.2.8.1. CVEs

- CVE-2022-41725
- CVE-2022-41724
- CVE-2023-24536
- CVE-2023-24538
- CVE-2023-24537
- CVE-2023-24534
- CVE-2022-41723

- CVE-2023-29400
- CVE-2023-24540
- CVE-2023-24539

9.2.9. Release notes for cert-manager Operator for Red Hat OpenShift 1.10.2

Issued: 2023-03-23

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.10.2:

- RHEA-2023:1238

Version **1.10.2** of the cert-manager Operator for Red Hat OpenShift is based on the upstream cert-manager version **v1.10.2**. For more information, see the cert-manager project release notes for v1.10.2.

IMPORTANT

If you used the Technology Preview version of the cert-manager Operator for Red Hat OpenShift, you must uninstall it and remove all related resources for the Technology Preview version before installing this version of the cert-manager Operator for Red Hat OpenShift.

For more information, see Uninstalling the cert-manager Operator for Red Hat OpenShift.

9.2.9.1. New features and enhancements

This is the general availability (GA) release of the cert-manager Operator for Red Hat OpenShift.

- The following issuer types are supported:
  - Automated Certificate Management Environment (ACME)
  - Certificate authority (CA)
  - Self-signed
- The following ACME challenge types are supported:
  - DNS-01
  - HTTP-01
- The following DNS-01 providers for ACME issuers are supported:
  - Amazon Route 53
  - Azure DNS
  - Google Cloud DNS
- The cert-manager Operator for Red Hat OpenShift now supports injecting custom CA certificates and propagating cluster-wide egress proxy environment variables.

- You can customize the cert-manager Operator for Red Hat OpenShift API fields by overriding environment variables and arguments. For more information, see Customizing cert-manager Operator API fields

- You can enable monitoring and metrics collection for the cert-manager Operator for Red Hat OpenShift by using a service monitor to perform the custom metrics scraping. After you have enabled monitoring for the cert-manager Operator for Red Hat OpenShift, you can query its metrics by using the OpenShift Container Platform web console. For more information, see Enabling monitoring for the cert-manager Operator for Red Hat OpenShift

9.2.9.2. Bug fixes

- Previously, the **unsupportedConfigOverrides** field replaced user-provided arguments instead of appending them. Now, the **unsupportedConfigOverrides** field properly appends user-provided arguments. (CM-23)

WARNING

Using the **unsupportedConfigOverrides** section to modify the configuration of an Operator is unsupported and might block cluster upgrades.

- Previously, the cert-manager Operator for Red Hat OpenShift was installed as a cluster Operator. With this release, the cert-manager Operator for Red Hat OpenShift is now properly installed as an OLM Operator. (CM-35)

9.2.9.3. Known issues

- Using **Route** objects is not fully supported. Currently, to use cert-manager Operator for Red Hat OpenShift with **Routes**, users must create **Ingress** objects, which are translated to **Route** objects by the Ingress-to-Route Controller. (CM-16)

- The cert-manager Operator for Red Hat OpenShift does not support using Azure Active Directory (Azure AD) pod identities to assign a managed identity to a pod. As a workaround, you can use a service principal to assign a managed identity. (OCPBUGS-8665)

- The cert-manager Operator for Red Hat OpenShift does not support using Google workload identity federation. (OCPBUGS-9998)

- When uninstalling the cert-manager Operator for Red Hat OpenShift, if you select the **Delete all operand instances for this operator** checkbox in the OpenShift Container Platform web console, the Operator is not uninstalled properly. As a workaround, do not select this checkbox when uninstalling the cert-manager Operator for Red Hat OpenShift. (OCPBUGS-9960)

9.3. INSTALLING THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

The cert-manager Operator for Red Hat OpenShift is not installed in OpenShift Container Platform by default. You can install the cert-manager Operator for Red Hat OpenShift by using the web console.

9.3.1. Installing the cert-manager Operator for Red Hat OpenShift using the web console

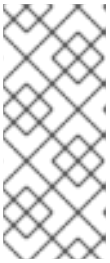
You can use the web console to install the cert-manager Operator for Red Hat OpenShift.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → OperatorHub**.
3. Enter **cert-manager Operator for Red Hat OpenShift**into the filter box.
4. Select the **cert-manager Operator for Red Hat OpenShift**and click **Install**.



NOTE

From the cert-manager Operator for Red Hat OpenShift **1.12.0** and later, the z-stream versions of the upstream cert-manager operands such as cert-manager controller, CA injector, Webhook, and cert-manager Operator for Red Hat OpenShift are decoupled. For example, for the cert-manager Operator for Red Hat OpenShift **1.12.0**, the cert-manager operand version is **v1.12.4**.

5. On the **Install Operator** page:
   a. Update the **Update channel**, if necessary. The channel defaults to **stable-v1**, which installs the latest stable release of the cert-manager Operator for Red Hat OpenShift.
   b. Choose the **Installed Namespace** for the Operator. The default Operator namespace is **cert-manager-operator**.
   If the **cert-manager-operator** namespace does not exist, it is created for you.
   c. Select an **Update approval** strategy.
      - The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
      - The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.
   d. Click **Install**.

Verification

1. Navigate to **Operators → Installed Operators**.
2. Verify that **cert-manager Operator for Red Hat OpenShift**is listed with a **Status** of **Succeeded** in the **cert-manager-operator** namespace.

3. Verify that cert-manager pods are up and running by entering the following command:

```
$ oc get pods -n cert-manager
```

Example output

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt               1/1     Running   0          3m39s
cert-manager-cainjector-56cc5f9868-7g9z7   1/1     Running   0          4m5s
cert-manager-webhook-d4f79d7f7-9dg9w       1/1     Running   0          4m9s
```

You can use the cert-manager Operator for Red Hat OpenShift only after cert-manager pods are up and running.

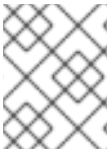
9.3.2. Understanding update channels of the cert-manager Operator for Red Hat OpenShift

Update channels are the mechanism by which you can declare the version of your cert-manager Operator for Red Hat OpenShift in your cluster. The cert-manager Operator for Red Hat OpenShift offers the following update channels:

- **stable-v1**
- **stable-v1.y**

9.3.2.1. stable-v1 channel

The **stable-v1** channel installs and updates the latest release version of the cert-manager Operator for Red Hat OpenShift. Select the **stable-v1** channel if you want to use the latest stable release of the cert-manager Operator for Red Hat OpenShift.



NOTE

The **stable-v1** channel is the default and suggested channel while installing the cert-manager Operator for Red Hat OpenShift.

The **stable-v1** channel offers the following update approval strategies:

Automatic

If you choose automatic updates for an installed cert-manager Operator for Red Hat OpenShift, a new version of the cert-manager Operator for Red Hat OpenShift is available in the **stable-v1** channel. The Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention.

Manual

If you select manual updates, when a newer version of the cert-manager Operator for Red Hat OpenShift is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the cert-manager Operator for Red Hat OpenShift updated to the new version.

9.3.2.2. stable-v1.y channel

The y-stream version of the cert-manager Operator for Red Hat OpenShift installs updates from the

stable-v1.y channels such as **stable-v1.10**, **stable-v1.11**, and **stable-v1.12**. Select the **stable-v1.y** channel if you want to use the y-stream version and stay updated to the z-stream version of the cert-manager Operator for Red Hat OpenShift.

The **stable-v1.y** channel offers the following update approval strategies:

Automatic

If you choose automatic updates for an installed cert-manager Operator for Red Hat OpenShift, a new z-stream version of the cert-manager Operator for Red Hat OpenShift is available in the **stable-v1.y** channel. OLM automatically upgrades the running instance of your Operator without human intervention.

Manual

If you select manual updates, when a newer version of the cert-manager Operator for Red Hat OpenShift is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the cert-manager Operator for Red Hat OpenShift updated to the new version of the z-stream releases.

9.3.3. Additional resources

- Adding Operators to a cluster
- Updating installed Operators

9.4. CONFIGURING AN ACME ISSUER

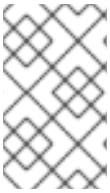
The cert-manager Operator for Red Hat OpenShift supports using Automated Certificate Management Environment (ACME) CA servers, such as *Let's Encrypt*, to issue certificates. Explicit credentials are configured by specifying the secret details in the **Issuer** API object. Ambient credentials are extracted from the environment, metadata services, or local files which are not explicitly configured in the **Issuer** API object.

NOTE

The **Issuer** object is namespace scoped. It can only issue certificates from the same namespace. You can also use the **ClusterIssuer** object to issue certificates across all namespaces in the cluster.

Example YAML file that defines the ClusterIssuer object

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: acme-cluster-issuer
spec:
  acme:
    ...
```



NOTE

By default, you can use the **ClusterIssuer** object with ambient credentials. To use the **Issuer** object with ambient credentials, you must enable the **--issuer-ambient-credentials** setting for the cert-manager controller.

9.4.1. About ACME issuers

The ACME issuer type for the cert-manager Operator for Red Hat OpenShift represents an Automated Certificate Management Environment (ACME) certificate authority (CA) server. ACME CA servers rely on a *challenge* to verify that a client owns the domain names that the certificate is being requested for. If the challenge is successful, the cert-manager Operator for Red Hat OpenShift can issue the certificate. If the challenge fails, the cert-manager Operator for Red Hat OpenShift does not issue the certificate.

NOTE

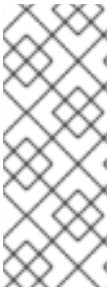
Private DNS zones are not supported with *Let's Encrypt* and internet ACME servers.

9.4.1.1. Supported ACME challenges types

The cert-manager Operator for Red Hat OpenShift supports the following challenge types for ACME issuers:

HTTP-01

With the HTTP-01 challenge type, you provide a computed key at an HTTP URL endpoint in your domain. If the ACME CA server can get the key from the URL, it can validate you as the owner of the domain.
For more information, see HTTP01 in the upstream cert-manager documentation.



NOTE

HTTP-01 requires that the Let's Encrypt servers can access the route of the cluster. If an internal or private cluster is behind a proxy, the HTTP-01 validations for certificate issuance fail.

The HTTP-01 challenge is restricted to port 80. For more information, see HTTP-01 challenge (Let's Encrypt).

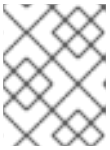
DNS-01

With the DNS-01 challenge type, you provide a computed key at a DNS TXT record. If the ACME CA server can get the key by DNS lookup, it can validate you as the owner of the domain.
For more information, see DNS01 in the upstream cert-manager documentation.

9.4.1.2. Supported DNS-01 providers

The cert-manager Operator for Red Hat OpenShift supports the following DNS-01 providers for ACME issuers:

- Amazon Route 53
- Azure DNS



NOTE

The cert-manager Operator for Red Hat OpenShift does not support using Azure Active Directory (Azure AD) pod identities to assign a managed identity to a pod.

- Google Cloud DNS
- Webhook
  Red Hat tests and supports DNS providers using an external webhook with cert-manager on OpenShift Container Platform. The following DNS providers are tested and supported with OpenShift Container Platform:
  - cert-manager-webhook-ibmcis

NOTE

Using a DNS provider that is not listed might work with OpenShift Container Platform, but the provider was not tested by Red Hat and therefore is not supported by Red Hat.

9.4.2. Configuring an ACME issuer to solve HTTP-01 challenges

You can use cert-manager Operator for Red Hat OpenShift to set up an ACME issuer to solve HTTP-01 challenges. This procedure uses *Let's Encrypt* as the ACME CA server.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a service that you want to expose. In this procedure, the service is named **sample-workload**.

Procedure

1. Create an ACME cluster issuer.
   a. Create a YAML file that defines the **ClusterIssuer** object:

Example acme-cluster-issuer.yaml file

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging                                    (1)
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_for_private_key>                           (2)
    server: https://acme-staging-v02.api.letsencrypt.org/directory (3)
    solvers:
    - http01:
        ingress:
          ingressClassName: openshift-default                  (4)
```

(1) Provide a name for the cluster issuer.

(2) Replace **<secret_private_key>** with the name of secret to store the ACME account private key in.

3 Specify the URL to access the ACME server's **directory** endpoint. This example uses the *Let's Encrypt* staging environment.

4 Specify the Ingress class.

b. Optional: If you create the object without specifying **ingressClassName**, use the following command to patch the existing ingress:

```
$ oc patch ingress/<ingress-name> --type=merge --patch '{"spec":
{"ingressClassName":"openshift-default"}}' -n <namespace>
```

c. Create the **ClusterIssuer** object by running the following command:

```
$ oc create -f acme-cluster-issuer.yaml
```

2. Create an Ingress to expose the service of the user workload.

a. Create a YAML file that defines a **Namespace** object:

Example namespace.yaml file

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-ingress-namespace 1
```

1 Specify the namespace for the Ingress.

b. Create the **Namespace** object by running the following command:

```
$ oc create -f namespace.yaml
```

c. Create a YAML file that defines the **Ingress** object:

Example ingress.yaml file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sample-ingress                                  1
  namespace: my-ingress-namespace                       2
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-staging 3
    acme.cert-manager.io/http01-ingress-class: openshift-default 4
spec:
  ingressClassName: openshift-default                   5
  tls:
  - hosts:
    - <hostname>                                        6
    secretName: sample-tls                              7
  rules:
```

```
- host: <hostname>                                    8
  http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: sample-workload                       9
          port:
            number: 80
```

1. Specify the name of the Ingress.
2. Specify the namespace that you created for the Ingress.
3. Specify the cluster issuer that you created.
4. Specify the Ingress class.
5. Specify the Ingress class.
6. Replace **<hostname>** with the Subject Alternative Name to be associated with the certificate. This name is used to add DNS names to the certificate.
7. Specify the secret to store the created certificate in.
8. Replace **<hostname>** with the hostname. You can use the **<host_name>.<cluster_ingress_domain>** syntax to take advantage of the ***.<cluster_ingress_domain>** wildcard DNS record and serving certificate for the cluster. For example, you might use **apps.<cluster_base_domain>**. Otherwise, you must ensure that a DNS record exists for the chosen hostname.
9. Specify the name of the service to expose. This example uses a service named **sample-workload**.

d. Create the **Ingress** object by running the following command:

```
$ oc create -f ingress.yaml
```

9.4.3. Configuring an ACME issuer by using explicit credentials for AWS Route53

You can use cert-manager Operator for Red Hat OpenShift to set up an Automated Certificate Management Environment (ACME) issuer to solve DNS-01 challenges by using explicit credentials on AWS. This procedure uses *Let's Encrypt* as the ACME certificate authority (CA) server and shows how to solve DNS-01 challenges with Amazon Route 53.

Prerequisites

- You must provide the explicit **accessKeyID** and **secretAccessKey** credentials. For more information, see Route53 in the upstream cert-manager documentation.

NOTE

You can use Amazon Route 53 with explicit credentials in an OpenShift Container Platform cluster that is not running on AWS.

Procedure

1. Optional: Override the nameserver settings for the DNS-01 self check.
This step is required only when the target public-hosted zone overlaps with the cluster's default private-hosted zone.

   a. Edit the **CertManager** resource by running the following command:

   ```
   $ oc edit certmanager cluster
   ```

   b. Add a **spec.controllerConfig** section with the following override arguments:

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: CertManager
   metadata:
     name: cluster
     ...
   spec:
     ...
     controllerConfig:                                  (1)
       overrideArgs:
         - '--dns01-recursive-nameservers-only'         (2)
         - '--dns01-recursive-nameservers=1.1.1.1:53'   (3)
   ```

   (1) Add the **spec.controllerConfig** section.

   (2) Specify to only use recursive nameservers instead of checking the authoritative nameservers associated with that domain.

   (3) Provide a comma-separated list of **<host>:<port>** nameservers to query for the DNS-01 self check. You must use a **1.1.1.1:53** value to avoid the public and private zones overlapping.

   c. Save the file to apply the changes.

2. Optional: Create a namespace for the issuer:

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

3. Create a secret to store your AWS credentials in by running the following command:

   ```
   $ oc create secret generic aws-secret --from-literal=awsSecretAccessKey=
   <aws_secret_access_key> \ (1)
     -n my-issuer-namespace
   ```

   (1) Replace **<aws_secret_access_key>** with your AWS secret access key.

4. Create an issuer:

   a. Create a YAML file that defines the **Issuer** object:

   Example issuer.yaml file

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <letsencrypt_staging>                                  (1)
  namespace: <issuer_namespace>                                (2)
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory  (3)
    email: "<email_address>"                                   (4)
    privateKeySecretRef:
      name: <secret_private_key>                               (5)
    solvers:
    - dns01:
        route53:
          accessKeyID: <aws_key_id>                            (6)
          hostedZoneID: <hosted_zone_id>                       (7)
          region: <region_name>                                (8)
          secretAccessKeySecretRef:
            name: "aws-secret"                                 (9)
            key: "awsSecretAccessKey"                          (10)
```

(1) Provide a name for the issuer.

(2) Specify the namespace that you created for the issuer.

(3) Specify the URL to access the ACME server's **directory** endpoint. This example uses the *Let's Encrypt* staging environment.

(4) Replace **<email_address>** with your email address.

(5) Replace **<secret_private_key>** with the name of the secret to store the ACME account private key in.

(6) Replace **<aws_key_id>** with your AWS key ID.

(7) Replace **<hosted_zone_id>** with your hosted zone ID.

(8) Replace **<region_name>** with the AWS region name. For example, **us-east-1**.

(9) Specify the name of the secret you created.

(10) Specify the key in the secret you created that stores your AWS secret access key.

   b. Create the **Issuer** object by running the following command:

```
$ oc create -f issuer.yaml
```

9.4.4. Configuring an ACME issuer by using ambient credentials on AWS

You can use cert-manager Operator for Red Hat OpenShift to set up an ACME issuer to solve DNS-01 challenges by using ambient credentials on AWS. This procedure uses *Let's Encrypt* as the ACME CA server and shows how to solve DNS-01 challenges with Amazon Route 53.

Prerequisites

- If your cluster is configured to use the AWS Security Token Service (STS), you followed the instructions from the *Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift for the AWS Security Token Service cluster* section.
- If your cluster does not use the AWS STS, you followed the instructions from the *Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift on AWS* section.

Procedure

1. Optional: Override the nameserver settings for the DNS-01 self check.
   This step is required only when the target public-hosted zone overlaps with the cluster's default private-hosted zone.

   a. Edit the **CertManager** resource by running the following command:

   ```
   $ oc edit certmanager cluster
   ```

   b. Add a **spec.controllerConfig** section with the following override arguments:

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: CertManager
   metadata:
     name: cluster
     ...
   spec:
     ...
     controllerConfig:                                  (1)
       overrideArgs:
         - '--dns01-recursive-nameservers-only'         (2)
         - '--dns01-recursive-nameservers=1.1.1.1:53'   (3)
   ```

   (1) Add the **spec.controllerConfig** section.

   (2) Specify to only use recursive nameservers instead of checking the authoritative nameservers associated with that domain.

   (3) Provide a comma-separated list of **<host>:<port>** nameservers to query for the DNS-01 self check. You must use a **1.1.1.1:53** value to avoid the public and private zones overlapping.

   c. Save the file to apply the changes.

2. Optional: Create a namespace for the issuer:

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

3. Modify the **CertManager** resource to add the **--issuer-ambient-credentials** argument:

```
$ oc patch certmanager/cluster \
  --type=merge \
  -p='{"spec":{"controllerConfig":{"overrideArgs":["--issuer-ambient-credentials"]}}}'
```

4. Create an issuer:

   a. Create a YAML file that defines the **Issuer** object:

   Example issuer.yaml file

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <letsencrypt_staging>                                    1
  namespace: <issuer_namespace>                                  2
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory  3
    email: "<email_address>"                                     4
    privateKeySecretRef:
      name: <secret_private_key>                                 5
    solvers:
    - dns01:
        route53:
          hostedZoneID: <hosted_zone_id>                         6
          region: us-east-1
```

   1. Provide a name for the issuer.
   2. Specify the namespace that you created for the issuer.
   3. Specify the URL to access the ACME server's **directory** endpoint. This example uses the *Let's Encrypt* staging environment.
   4. Replace **<email_address>** with your email address.
   5. Replace **<secret_private_key>** with the name of the secret to store the ACME account private key in.
   6. Replace **<hosted_zone_id>** with your hosted zone ID.

   b. Create the **Issuer** object by running the following command:

```
$ oc create -f issuer.yaml
```

9.4.5. Configuring an ACME issuer by using explicit credentials for Google Cloud DNS

You can use the cert-manager Operator for Red Hat OpenShift to set up an ACME issuer to solve DNS-01 challenges by using explicit credentials on Google Cloud. This procedure uses *Let's Encrypt* as the ACME CA server and shows how to solve DNS-01 challenges with Google Cloud DNS.

Prerequisites

- You have set up a Google Cloud service account with a desired role for Google Cloud DNS. For more information, see Google Cloud DNS in the upstream cert-manager documentation.

NOTE

You can use Google Cloud DNS with explicit credentials in an OpenShift Container Platform cluster that is not running on Google Cloud.

Procedure

1. Optional: Override the nameserver settings for the DNS-01 self check.
   This step is required only when the target public-hosted zone overlaps with the cluster's default private-hosted zone.

   a. Edit the **CertManager** resource by running the following command:

   ```
   $ oc edit certmanager cluster
   ```

   b. Add a **spec.controllerConfig** section with the following override arguments:

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: CertManager
   metadata:
     name: cluster
     ...
   spec:
     ...
     controllerConfig:                                  1
       overrideArgs:
         - '--dns01-recursive-nameservers-only'         2
         - '--dns01-recursive-nameservers=1.1.1.1:53'   3
   ```

   1. Add the **spec.controllerConfig** section.
   2. Specify to only use recursive nameservers instead of checking the authoritative nameservers associated with that domain.
   3. Provide a comma-separated list of **<host>:<port>** nameservers to query for the DNS-01 self check. You must use a **1.1.1.1:53** value to avoid the public and private zones overlapping.

   c. Save the file to apply the changes.

2. Optional: Create a namespace for the issuer:

   ```
   $ oc new-project my-issuer-namespace
   ```

3. Create a secret to store your Google Cloud credentials by running the following command:

```
$ oc create secret generic clouddns-dns01-solver-svc-acct --from-file=service_account.json=
<path/to/gcp_service_account.json> -n my-issuer-namespace
```

4. Create an issuer:

   a. Create a YAML file that defines the **Issuer** object:

   Example issuer.yaml file

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <acme_dns01_clouddns_issuer> 1
  namespace: <issuer_namespace> 2
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_private_key> 3
    server: https://acme-staging-v02.api.letsencrypt.org/directory 4
    solvers:
    - dns01:
        cloudDNS:
          project: <project_id> 5
          serviceAccountSecretRef:
            name: clouddns-dns01-solver-svc-acct 6
            key: service_account.json 7
```

1. Provide a name for the issuer.
2. Replace **<issuer_namespace>** with your issuer namespace.
3. Replace **<secret_private_key>** with the name of the secret to store the ACME account private key in.
4. Specify the URL to access the ACME server's **directory** endpoint. This example uses the *Let's Encrypt* staging environment.
5. Replace **<project_id>** with the name of the Google Cloud project that contains the Cloud DNS zone.
6. Specify the name of the secret you created.
7. Specify the key in the secret you created that stores your Google Cloud secret access key.

   b. Create the **Issuer** object by running the following command:

```
$ oc create -f issuer.yaml
```

9.4.6. Configuring an ACME issuer by using ambient credentials on Google Cloud

You can use the cert-manager Operator for Red Hat OpenShift to set up an ACME issuer to solve DNS-01 challenges by using ambient credentials on Google Cloud. This procedure uses *Let's Encrypt* as the ACME CA server and shows how to solve DNS-01 challenges with Google Cloud DNS.

Prerequisites

- If your cluster is configured to use Google Cloud Workload Identity, you followed the instructions from the *Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift with Google Cloud Workload Identity* section.
- If your cluster does not use Google Cloud Workload Identity, you followed the instructions from the *Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift on Google Cloud* section.

Procedure

1. Optional: Override the nameserver settings for the DNS-01 self check.
   This step is required only when the target public-hosted zone overlaps with the cluster's default private-hosted zone.

   a. Edit the **CertManager** resource by running the following command:

   ```
   $ oc edit certmanager cluster
   ```

   b. Add a **spec.controllerConfig** section with the following override arguments:

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: CertManager
   metadata:
     name: cluster
     ...
   spec:
     ...
     controllerConfig:                                  (1)
       overrideArgs:
         - '--dns01-recursive-nameservers-only'         (2)
         - '--dns01-recursive-nameservers=1.1.1.1:53'   (3)
   ```

   (1) Add the **spec.controllerConfig** section.

   (2) Specify to only use recursive nameservers instead of checking the authoritative nameservers associated with that domain.

   (3) Provide a comma-separated list of **<host>:<port>** nameservers to query for the DNS-01 self check. You must use a **1.1.1.1:53** value to avoid the public and private zones overlapping.

   c. Save the file to apply the changes.

2. Optional: Create a namespace for the issuer:

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

3. Modify the **CertManager** resource to add the **--issuer-ambient-credentials** argument:

```
$ oc patch certmanager/cluster \
  --type=merge \
  -p='{"spec":{"controllerConfig":{"overrideArgs":["--issuer-ambient-credentials"]}}}'
```

4. Create an issuer:

   a. Create a YAML file that defines the **Issuer** object:

   Example issuer.yaml file

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <acme_dns01_clouddns_issuer> 1
  namespace: <issuer_namespace>
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_private_key> 2
    server: https://acme-staging-v02.api.letsencrypt.org/directory 3
    solvers:
    - dns01:
        cloudDNS:
          project: <gcp_project_id> 4
```

   1. Provide a name for the issuer.
   2. Replace **<secret_private_key>** with the name of the secret to store the ACME account private key in.
   3. Specify the URL to access the ACME server's **directory** endpoint. This example uses the *Let's Encrypt* staging environment.
   4. Replace **<gcp_project_id>** with the name of the Google Cloud project that contains the Cloud DNS zone.

   b. Create the **Issuer** object by running the following command:

```
$ oc create -f issuer.yaml
```

9.4.7. Configuring an ACME issuer by using explicit credentials for Microsoft Azure DNS

You can use cert-manager Operator for Red Hat OpenShift to set up an ACME issuer to solve DNS-01 challenges by using explicit credentials on Microsoft Azure. This procedure uses *Let's Encrypt* as the ACME CA server and shows how to solve DNS-01 challenges with Azure DNS.

Prerequisites

- You have set up a service principal with desired role for Azure DNS. For more information, see Azure DNS in the upstream cert-manager documentation.

NOTE

You can follow this procedure for an OpenShift Container Platform cluster that is not running on Microsoft Azure.

Procedure

1. Optional: Override the nameserver settings for the DNS-01 self check.
   This step is required only when the target public-hosted zone overlaps with the cluster's default private-hosted zone.

   a. Edit the **CertManager** resource by running the following command:

   ```
   $ oc edit certmanager cluster
   ```

   b. Add a **spec.controllerConfig** section with the following override arguments:

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: CertManager
   metadata:
     name: cluster
     ...
   spec:
     ...
     controllerConfig:                                  (1)
       overrideArgs:
         - '--dns01-recursive-nameservers-only'         (2)
         - '--dns01-recursive-nameservers=1.1.1.1:53'   (3)
   ```

   (1) Add the **spec.controllerConfig** section.

   (2) Specify to only use recursive nameservers instead of checking the authoritative nameservers associated with that domain.

   (3) Provide a comma-separated list of **<host>:<port>** nameservers to query for the DNS-01 self check. You must use a **1.1.1.1:53** value to avoid the public and private zones overlapping.

   c. Save the file to apply the changes.

2. Optional: Create a namespace for the issuer:

   ```
   $ oc new-project my-issuer-namespace
   ```

3. Create a secret to store your Azure credentials in by running the following command:

   ```
   $ oc create secret generic <secret_name> --from-literal=
   <azure_secret_access_key_name>=<azure_secret_access_key_value> \ (1) (2) (3)
     -n my-issuer-namespace
   ```

1. Replace **<secret_name>** with your secret name.
2. Replace **<azure_secret_access_key_name>** with your Azure secret access key name.
3. Replace **<azure_secret_access_key_value>** with your Azure secret key.

4. Create an issuer:

   a. Create a YAML file that defines the **Issuer** object:

   Example issuer.yaml file

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <acme-dns01-azuredns-issuer> 1
  namespace: <issuer_namespace> 2
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_private_key> 3
    server: https://acme-staging-v02.api.letsencrypt.org/directory 4
    solvers:
    - dns01:
        azureDNS:
          clientID: <azure_client_id> 5
          clientSecretSecretRef:
            name: <secret_name> 6
            key: <azure_secret_access_key_name> 7
          subscriptionID: <azure_subscription_id> 8
          tenantID: <azure_tenant_id> 9
          resourceGroupName: <azure_dns_zone_resource_group> 10
          hostedZoneName: <azure_dns_zone> 11
          environment: AzurePublicCloud
```

1. Provide a name for the issuer.
2. Replace **<issuer_namespace>** with your issuer namespace.
3. Replace **<secret_private_key>** with the name of the secret to store the ACME account private key in.
4. Specify the URL to access the ACME server's **directory** endpoint. This example uses the *Let's Encrypt* staging environment.
5. Replace **<azure_client_id>** with your Azure client ID.
6. Replace **<secret_name>** with a name of the client secret.
7. Replace **<azure_secret_access_key_name>** with the client secret key name.
8. Replace **<azure_subscription_id>** with your Azure subscription ID.

9 Replace **<azure_tenant_id>** with your Azure tenant ID.

10 Replace **<azure_dns_zone_resource_group>** with the name of the Azure DNS zone resource group.

11 Replace **<azure_dns_zone>** with the name of Azure DNS zone.

b. Create the **Issuer** object by running the following command:

```
$ oc create -f issuer.yaml
```

9.4.8. Additional resources

- Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift for the AWS Security Token Service cluster
- Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift on AWS
- Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift with Google Cloud Workload Identity
- Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift on Google Cloud

9.5. CONFIGURING CERTIFICATES WITH AN ISSUER

By using the cert-manager Operator for Red Hat OpenShift, you can manage certificates, handling tasks such as renewal and issuance, for workloads within the cluster, as well as components interacting externally to the cluster.

9.5.1. Creating certificates for user workloads

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have installed the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Create an issuer. For more information, see "Configuring an issuer" in the "Additional resources" section.
2. Create a certificate:
   a. Create a YAML file, for example, **certificate.yaml**, that defines the **Certificate** object:

Example certificate.yaml file

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: <tls_cert> 1
```

```
  namespace: <issuer_namespace> 2
spec:
  isCA: false
  commonName: '<common_name>' 3
  secretName: <secret_name> 4
  dnsNames:
  - "<domain_name>" 5
  issuerRef:
    name: <issuer_name> 6
    kind: Issuer
```

1. Provide a name for the certificate.
2. Specify the namespace of the issuer.
3. Specify the common name (CN).
4. Specify the name of the secret to create that contains the certificate.
5. Specify the domain name.
6. Specify the name of the issuer.

b. Create the **Certificate** object by running the following command:

```
$ oc create -f certificate.yaml
```

Verification

- Verify that the certificate is created and ready to use by running the following command:

```
$ oc get certificate -w -n <issuer_namespace>
```

Once certificate is in **Ready** status, workloads on your cluster can start using the generated certificate secret.

9.5.2. Creating certificates for the API server

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have installed version 1.13.0 or later of the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Create an issuer. For more information, see "Configuring an issuer" in the "Additional resources" section.
2. Create a certificate:
   a. Create a YAML file, for example, **certificate.yaml**, that defines the **Certificate** object:

Example certificate.yaml file

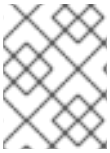
```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: <tls_cert> 1
  namespace: openshift-config
spec:
  isCA: false
  commonName: "api.<cluster_base_domain>" 2
  secretName: <secret_name> 3
  dnsNames:
  - "api.<cluster_base_domain>" 4
  issuerRef:
    name: <issuer_name> 5
    kind: Issuer
```

1. Provide a name for the certificate.
2. Specify the common name (CN).
3. Specify the name of the secret to create that contains the certificate.
4. Specify the DNS name of the API server.
5. Specify the name of the issuer.

b. Create the **Certificate** object by running the following command:

```
$ oc create -f certificate.yaml
```

3. Add the API server named certificate. For more information, see "Adding an API server named certificate" section in the "Additional resources" section.



NOTE

To ensure the certificates are updated, run the **oc login** command again after the certificate is created.

Verification

- Verify that the certificate is created and ready to use by running the following command:

```
$ oc get certificate -w -n openshift-config
```

Once certificate is in **Ready** status, API server on your cluster can start using the generated certificate secret.

9.5.3. Creating certificates for the Ingress Controller

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have installed version 1.13.0 or later of the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Create an issuer. For more information, see "Configuring an issuer" in the "Additional resources" section.
2. Create a certificate:
   a. Create a YAML file, for example, **certificate.yaml**, that defines the **Certificate** object:

Example certificate.yaml file

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: <tls_cert> 1
  namespace: openshift-ingress
spec:
  isCA: false
  commonName: "apps.<cluster_base_domain>" 2
  secretName: <secret_name> 3
  dnsNames:
  - "apps.<cluster_base_domain>" 4
  - "*.apps.<cluster_base_domain>" 5
  issuerRef:
    name: <issuer_name> 6
    kind: Issuer
```

1. Provide a name for the certificate.
2. Specify the common name (CN).
3. Specify the name of the secret to create that contains the certificate.
4. 5 Specify the DNS name of the ingress.
6. Specify the name of the issuer.

   b. Create the **Certificate** object by running the following command:

```
$ oc create -f certificate.yaml
```

3. Replace the default ingress certificate. For more information, see "Replacing the default ingress certificate" section in the "Additional resources" section.

Verification

- Verify that the certificate is created and ready to use by running the following command:

```
$ oc get certificate -w -n openshift-ingress
```

Once certificate is in **Ready** status, Ingress Controller on your cluster can start using the generated certificate secret.

9.5.4. Additional resources

- Configuring an issuer
  - Supported issuer types
  - Configuring an ACME issuer
- Adding an API server named certificate
- Replacing the default ingress certificate

9.6. ENABLING MONITORING FOR THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

You can expose controller metrics for the cert-manager Operator for Red Hat OpenShift in the format provided by the Prometheus Operator.

9.6.1. Enabling monitoring by using a service monitor for the cert-manager Operator for Red Hat OpenShift

You can enable monitoring and metrics collection for the cert-manager Operator for Red Hat OpenShift by using a service monitor to perform the custom metrics scraping.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- The cert-manager Operator for Red Hat OpenShift is installed.

Procedure

1. Add the label to enable cluster monitoring by running the following command:

```
$ oc label namespace cert-manager openshift.io/cluster-monitoring=true
```

2. Create a service monitor:
   a. Create a YAML file that defines the **Role**, **RoleBinding**, and **ServiceMonitor** objects:

Example monitoring.yaml file

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: prometheus-k8s
  namespace: cert-manager
rules:
- apiGroups:
  - ""
  resources:
```

```
  - services
  - endpoints
  - pods
  verbs:
  - get
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: prometheus-k8s
  namespace: cert-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: prometheus-k8s
subjects:
- kind: ServiceAccount
  name: prometheus-k8s
  namespace: openshift-monitoring
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: cert-manager
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: cert-manager
    app.kubernetes.io/name: cert-manager
  name: cert-manager
  namespace: cert-manager
spec:
  endpoints:
  - interval: 30s
    port: tcp-prometheus-servicemonitor
    scheme: http
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/instance: cert-manager
      app.kubernetes.io/name: cert-manager
```

b. Create the **Role**, **RoleBinding**, and **ServiceMonitor** objects by running the following command:

```
$ oc create -f monitoring.yaml
```

Additional resources

- Setting up metrics collection for user-defined projects

9.6.2. Querying metrics for the cert-manager Operator for Red Hat OpenShift

After you have enabled monitoring for the cert-manager Operator for Red Hat OpenShift, you can query its metrics by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the cert-manager Operator for Red Hat OpenShift.
- You have enabled monitoring and metrics collection for the cert-manager Operator for Red Hat OpenShift.

Procedure

1. From the OpenShift Container Platform web console, navigate to **Observe → Metrics**.
2. Add a query by using one of the following formats:
   - Specify the endpoints:

     ```
     {instance="<endpoint>"} 1
     ```

     1 Replace **<endpoint>** with the value of the endpoint for the **cert-manager** service. You can find the endpoint value by running the following command: **oc describe service cert-manager -n cert-manager**.

   - Specify the **tcp-prometheus-servicemonitor** port:

     ```
     {endpoint="tcp-prometheus-servicemonitor"}
     ```

9.7. CONFIGURING THE EGRESS PROXY FOR THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

If a cluster-wide egress proxy is configured in OpenShift Container Platform, Operator Lifecycle Manager (OLM) automatically configures Operators that it manages with the cluster-wide proxy. OLM automatically updates all of the Operator's deployments with the **HTTP_PROXY**, **HTTPS_PROXY**, **NO_PROXY** environment variables.

You can inject any CA certificates that are required for proxying HTTPS connections into the cert-manager Operator for Red Hat OpenShift.

9.7.1. Injecting a custom CA certificate for the cert-manager Operator for Red Hat OpenShift

If your OpenShift Container Platform cluster has the cluster-wide proxy enabled, you can inject any CA certificates that are required for proxying HTTPS connections into the cert-manager Operator for Red Hat OpenShift.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have enabled the cluster-wide proxy for OpenShift Container Platform.

Procedure

1. Create a config map in the **cert-manager** namespace by running the following command:

```
$ oc create configmap trusted-ca -n cert-manager
```

2. Inject the CA bundle that is trusted by OpenShift Container Platform into the config map by running the following command:

```
$ oc label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true -n cert-manager
```

3. Update the deployment for the cert-manager Operator for Red Hat OpenShift to use the config map by running the following command:

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --
type='merge' -p '{"spec":{"config":{"env":
[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}]}}}'
```

Verification

1. Verify that the deployments have finished rolling out by running the following command:

```
$ oc rollout status deployment/cert-manager-operator-controller-manager -n cert-manager-
operator && \
oc rollout status deployment/cert-manager -n cert-manager && \
oc rollout status deployment/cert-manager-webhook -n cert-manager && \
oc rollout status deployment/cert-manager-cainjector -n cert-manager
```

Example output

```
deployment "cert-manager-operator-controller-manager" successfully rolled out
deployment "cert-manager" successfully rolled out
deployment "cert-manager-webhook" successfully rolled out
deployment "cert-manager-cainjector" successfully rolled out
```

2. Verify that the CA bundle was mounted as a volume by running the following command:

```
$ oc get deployment cert-manager -n cert-manager -o=jsonpath=
{.spec.template.spec.'containers[0].volumeMounts'}
```

Example output

```
[{"mountPath":"/etc/pki/tls/certs/cert-manager-tls-ca-bundle.crt","name":"trusted-
ca","subPath":"ca-bundle.crt"}]
```

3. Verify that the source of the CA bundle is the **trusted-ca** config map by running the following command:

```
$ oc get deployment cert-manager -n cert-manager -o=jsonpath=
{.spec.template.spec.volumes}
```

Example output

```
[{"configMap":{"defaultMode":420,"name":"trusted-ca"},"name":"trusted-ca"}]
```

9.7.2. Additional resources

- Configuring proxy support in Operator Lifecycle Manager

9.8. CUSTOMIZING CERT-MANAGER OPERATOR API FIELDS

You can customize the cert-manager Operator for Red Hat OpenShift API fields by overriding environment variables and arguments.

WARNING

To override unsupported arguments, you can add **spec.unsupportedConfigOverrides** section in the **CertManager** resource, but using **spec.unsupportedConfigOverrides** is unsupported.

9.8.1. Customizing cert-manager by overriding environment variables from the cert-manager Operator API

You can override the supported environment variables for the cert-manager Operator for Red Hat OpenShift by adding a **spec.controllerConfig** section in the **CertManager** resource.

Prerequisites

- You have access to the OpenShift Container Platform cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **CertManager** resource by running the following command:

```
$ oc edit certmanager cluster
```

2. Add a **spec.controllerConfig** section with the following override arguments:

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
  ...
spec:
  ...
  controllerConfig:
    overrideEnv:
      - name: HTTP_PROXY
        value: http://<proxy_url> 1
      - name: HTTPS_PROXY
```

```
    value: https://<proxy_url> 2
  - name: NO_PROXY
    value: <ignore_proxy_domains> 3
```

1 2 Replace **<proxy_url>** with the proxy server URL.

3 Replace **<ignore_proxy_domains>** with a comma separated list of domains. These domains are ignored by the proxy server.

3. Save your changes and quit the text editor to apply your changes.

Verification

1. Verify that the cert-manager controller pod is redeployed by running the following command:

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

Example output

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          39s
```

2. Verify that environment variables are updated for the cert-manager pod by running the following command:

```
$ oc get pod <redeployed_cert-manager_controller_pod> -n cert-manager -o yaml
```

Example output

```
env:
...
- name: HTTP_PROXY
  value: http://<PROXY_URL>
- name: HTTPS_PROXY
  value: https://<PROXY_URL>
- name: NO_PROXY
  value: <IGNORE_PROXY_DOMAINS>
```

9.8.2. Customizing cert-manager by overriding arguments from the cert-manager Operator API

You can override the supported arguments for the cert-manager Operator for Red Hat OpenShift by adding a **spec.controllerConfig** section in the **CertManager** resource.

Prerequisites

- You have access to the OpenShift Container Platform cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **CertManager** resource by running the following command:

```
$ oc edit certmanager cluster
```

2. Add a **spec.controllerConfig** section with the following override arguments:

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
  ...
spec:
  ...
  controllerConfig:
    overrideArgs:
      - '--dns01-recursive-nameservers=<host>:<port>' 1
      - '--dns01-recursive-nameservers-only' 2
      - '--acme-http01-solver-nameservers=<host>:<port>' 3
      - '--v=<verbosity_level>' 4
      - '--metrics-listen-address=<host>:<port>' 5
      - '--issuer-ambient-credentials' 6
  webhookConfig:
    overrideArgs:
      - '--v=4' 7
  cainjectorConfig:
    overrideArgs:
      - '--v=2' 8
```

1. Provide a comma-separated list of **<host>:<port>** nameservers to query for the DNS-01 self check. For example, **--dns01-recursive-nameservers=1.1.1.1:53**.
2. Specify to only use recursive nameservers instead of checking the authoritative nameservers associated with that domain.
3. Provide a comma-separated list of **<host>:<port>** nameservers to query for the Automated Certificate Management Environment (ACME) HTTP01 self check. For example, **--acme-http01-solver-nameservers=1.1.1.1:53**.
4. 7. 8. Specify to set the log level verbosity to determine the verbosity of log messages.
5. Specify the host and port for the metrics endpoint. The default value is **--metrics-listen-address=0.0.0.0:9402**.
6. You must use the **--issuer-ambient-credentials** argument when configuring an ACME Issuer to solve DNS-01 challenges by using ambient credentials.

3. Save your changes and quit the text editor to apply your changes.

Verification

- Verify that arguments are updated for cert-manager pods by running the following command:

```
$ oc get pods -n cert-manager -o yaml
```

-

Example output

```
...
  metadata:
    name: cert-manager-6d4b5d4c97-kldwl
    namespace: cert-manager
...
  spec:
    containers:
    - args:
      - --acme-http01-solver-nameservers=1.1.1.1:53
      - --cluster-resource-namespace=$(POD_NAMESPACE)
      - --dns01-recursive-nameservers=1.1.1.1:53
      - --dns01-recursive-nameservers-only
      - --leader-election-namespace=kube-system
      - --max-concurrent-challenges=60
      - --metrics-listen-address=0.0.0.0:9042
      - --v=6
...
  metadata:
    name: cert-manager-cainjector-866c4fd758-ltxxj
    namespace: cert-manager
...
  spec:
    containers:
    - args:
      - --leader-election-namespace=kube-system
      - --v=2
...
  metadata:
    name: cert-manager-webhook-6d48f88495-c88gd
    namespace: cert-manager
...
  spec:
    containers:
    - args:
      ...
      - --v=4
```

9.8.3. Deleting a TLS secret automatically upon Certificate removal

You can enable the **--enable-certificate-owner-ref** flag for the cert-manager Operator for Red Hat OpenShift by adding a **spec.controllerConfig** section in the **CertManager** resource. The **--enable-certificate-owner-ref** flag sets the certificate resource as an owner of the secret where the TLS certificate is stored.

WARNING

If you uninstall the cert-manager Operator for Red Hat OpenShift or delete certificate resources from the cluster, the secret is deleted automatically. This might cause network connectivity issues depending upon where the certificate TLS secret is being used.

Prerequisites

- You have access to the OpenShift Container Platform cluster as a user with the **cluster-admin** role.
- You have installed version 1.12.0 or later of the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Check that the **Certificate** object and its secret are available by running the following command:

```
$ oc get certificate
```

Example output

```
NAME                                          READY   SECRET                                        AGE
certificate-from-clusterissuer-route53-ambient   True    certificate-from-clusterissuer-route53-ambient   8h
```

2. Edit the **CertManager** resource by running the following command:

```
$ oc edit certmanager cluster
```

3. Add a **spec.controllerConfig** section with the following override arguments:

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
# ...
spec:
# ...
  controllerConfig:
    overrideArgs:
      - '--enable-certificate-owner-ref'
```

4. Save your changes and quit the text editor to apply your changes.

Verification

- Verify that the **--enable-certificate-owner-ref** flag is updated for cert-manager controller pod by running the following command:

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager -o yaml
```

Example output

```
# ...
  metadata:
    name: cert-manager-6e4b4d7d97-zmdnb
    namespace: cert-manager
# ...
  spec:
    containers:
    - args:
      - --enable-certificate-owner-ref
```

9.8.4. Overriding CPU and memory limits for the cert-manager components

After installing the cert-manager Operator for Red Hat OpenShift, you can configure the CPU and memory limits from the cert-manager Operator for Red Hat OpenShift API for the cert-manager components such as cert-manager controller, CA injector, and Webhook.

Prerequisites

- You have access to the OpenShift Container Platform cluster as a user with the **cluster-admin** role.
- You have installed version 1.12.0 or later of the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Check that the deployments of the cert-manager controller, CA injector, and Webhook are available by entering the following command:

```
$ oc get deployment -n cert-manager
```

Example output

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager          1/1     1            1           53m
cert-manager-cainjector   1/1     1            1           53m
cert-manager-webhook      1/1     1            1           53m
```

2. Before setting the CPU and memory limit, check the existing configuration for the cert-manager controller, CA injector, and Webhook by entering the following command:

```
$ oc get deployment -n cert-manager -o yaml
```

Example output

```
# ...
  metadata:
    name: cert-manager
    namespace: cert-manager
```

```
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-controller
          resources: {} 1
# ...
  metadata:
    name: cert-manager-cainjector
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-cainjector
          resources: {} 2
# ...
  metadata:
    name: cert-manager-webhook
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-webhook
          resources: {} 3
# ...
```

1 2 3 The **spec.resources** field is empty by default. The cert-manager components do not have CPU and memory limits.

3. To configure the CPU and memory limits for the cert-manager controller, CA injector, and Webhook, enter the following command:

```
$ oc patch certmanager.operator cluster --type=merge -p="
spec:
  controllerConfig:
    overrideResources:
      limits: 1
        cpu: 200m 2
        memory: 64Mi 3
      requests: 4
        cpu: 10m 5
        memory: 16Mi 6
  webhookConfig:
    overrideResources:
      limits: 7
        cpu: 200m 8
        memory: 64Mi 9
      requests: 10
```

```
      cpu: 10m (11)
      memory: 16Mi (12)
  cainjectorConfig:
    overrideResources:
      limits: (13)
        cpu: 200m (14)
        memory: 64Mi (15)
      requests: (16)
        cpu: 10m (17)
        memory: 16Mi (18)
"
```

(1) Defines the maximum amount of CPU and memory that a single container in a cert-manager controller pod can request.

(2) (5) You can specify the CPU limit that a cert-manager controller pod can request. The default value is **10m**.

(3) (6) You can specify the memory limit that a cert-manager controller pod can request. The default value is **32Mi**.

(4) Defines the amount of CPU and memory set by scheduler for the cert-manager controller pod.

(7) Defines the maximum amount of CPU and memory that a single container in a CA injector pod can request.

(8) (11) You can specify the CPU limit that a CA injector pod can request. The default value is **10m**.

(9) (12) You can specify the memory limit that a CA injector pod can request. The default value is **32Mi**.

(10) Defines the amount of CPU and memory set by scheduler for the CA injector pod.

(13) Defines the maximum amount of CPU and memory Defines the maximum amount of CPU and memory that a single container in a Webhook pod can request.

(14) (17) You can specify the CPU limit that a Webhook pod can request. The default value is **10m**.

(15) (18) You can specify the memory limit that a Webhook pod can request. The default value is **32Mi**.

(16) Defines the amount of CPU and memory set by scheduler for the Webhook pod.

Example output

```
certmanager.operator.openshift.io/cluster patched
```

Verification

1. Verify that the CPU and memory limits are updated for the cert-manager components:

```
$ oc get deployment -n cert-manager -o yaml
```

Example output

```
# ...
  metadata:
    name: cert-manager
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-controller
          resources:
            limits:
              cpu: 200m
              memory: 64Mi
            requests:
              cpu: 10m
              memory: 16Mi
# ...
  metadata:
    name: cert-manager-cainjector
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-cainjector
          resources:
            limits:
              cpu: 200m
              memory: 64Mi
            requests:
              cpu: 10m
              memory: 16Mi
# ...
  metadata:
    name: cert-manager-webhook
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-webhook
          resources:
            limits:
              cpu: 200m
              memory: 64Mi
            requests:
```

```
            cpu: 10m
            memory: 16Mi
# ...
```

9.9. AUTHENTICATING THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT WITH AWS SECURITY TOKEN SERVICE

You can authenticate the cert-manager Operator for Red Hat OpenShift on the AWS Security Token Service (STS) cluster. You can configure cloud credentials for the cert-manager Operator for Red Hat OpenShift by using the **ccoctl** binary.

9.9.1. Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift for the AWS Security Token Service cluster

To configure the cloud credentials for the cert-manager Operator for Red Hat OpenShift on the AWS Security Token Service (STS) cluster with the cloud credentials. You must generate the cloud credentials manually, and apply it on the cluster by using the **ccoctl** binary.

Prerequisites

- You have extracted and prepared the **ccoctl** binary.
- You have configured an OpenShift Container Platform cluster with AWS STS by using the Cloud Credential Operator in manual mode.

Procedure

1. Create a directory to store a **CredentialsRequest** resource YAML file by running the following command:

```
$ mkdir credentials-request
```

2. Create a **CredentialsRequest** resource YAML file under the **credentials-request** directory, such as, **sample-credential-request.yaml**, by applying the following yaml:

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
      - "route53:GetChange"
      effect: Allow
      resource: "arn:aws:route53:::change/*"
    - action:
      - "route53:ChangeResourceRecordSets"
      - "route53:ListResourceRecordSets"
      effect: Allow
```

```
      resource: "arn:aws:route53:::hostedzone/*"
    - action:
      - "route53:ListHostedZonesByName"
      effect: Allow
      resource: "*"
  secretRef:
    name: aws-creds
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

3. Use the **ccoctl** tool to process **CredentialsRequest** objects by running the following command:

```
$ ccoctl aws create-iam-roles \
    --name <user_defined_name> --region=<aws_region> \
    --credentials-requests-dir=<path_to_credrequests_dir> \
    --identity-provider-arn <oidc_provider_arn> --output-dir=<path_to_output_dir>
```

Example output

```
2023/05/15 18:10:34 Role arn:aws:iam::XXXXXXXXXXXX:role/<user_defined_name>-cert-manager-aws-creds created
2023/05/15 18:10:34 Saved credentials configuration to: <path_to_output_dir>/manifests/cert-manager-aws-creds-credentials.yaml
2023/05/15 18:10:35 Updated Role policy for Role <user_defined_name>-cert-manager-aws-creds
```

Copy the **<aws_role_arn>** from the output to use in the next step. For example, **"arn:aws:iam::XXXXXXXXXXXX:role/<user_defined_name>-cert-manager-aws-creds"**

4. Add the **eks.amazonaws.com/role-arn="<aws_role_arn>"** annotation to the service account by running the following command:

```
$ oc -n cert-manager annotate serviceaccount cert-manager eks.amazonaws.com/role-arn="<aws_role_arn>"
```

5. To create a new pod, delete the existing cert-manager controller pod by running the following command:

```
$ oc delete pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

The AWS credentials are applied to a new cert-manager controller pod within a minute.

Verification

1. Get the name of the updated cert-manager controller pod by running the following command:

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

Example output

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          39s
```

2. Verify that AWS credentials are updated by running the following command:

```
$ oc set env -n cert-manager po/<cert_manager_controller_pod_name> --list
```

Example output

```
# pods/cert-manager-57f9555c54-vbcpg, container cert-manager-controller
# POD_NAMESPACE from field path metadata.namespace
AWS_ROLE_ARN=XXXXXXXXXXXX
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

9.9.2. Additional resources

- Configuring the Cloud Credential Operator utility

9.10. CONFIGURING LOG LEVELS FOR CERT-MANAGER AND THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

To troubleshoot issues with the cert-manager components and the cert-manager Operator for Red Hat OpenShift, you can configure the log level verbosity.

NOTE

To use different log levels for different cert-manager components, see *Customizing cert-manager Operator API fields*.

9.10.1. Setting a log level for cert-manager

You can set a log level for cert-manager to determine the verbosity of log messages.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have installed version 1.11.1 or later of the cert-manager Operator for Red Hat OpenShift.

Procedure

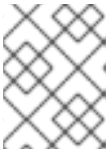
1. Edit the **CertManager** resource by running the following command:

```
$ oc edit certmanager.operator cluster
```

2. Set the log level value by editing the **spec.logLevel** section:

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
...
spec:
  logLevel: Normal (1)
```

1 The default **logLevel** is **Normal**. Replace **Normal** with the desired log level value. The valid log level values for the **CertManager** resource are **Normal**, **Debug**, **Trace**, and **TraceAll**. To



NOTE

TraceAll generates huge amount of logs. After setting **logLevel** to **TraceAll**, you might experience performance issues.

3. Save your changes and quit the text editor to apply your changes.
   After applying the changes, the verbosity level for the cert-manager components controller, CA injector, and webhook is updated.

9.10.2. Setting a log level for the cert-manager Operator for Red Hat OpenShift

You can set a log level for the cert-manager Operator for Red Hat OpenShift to determine the verbosity of the operator log messages.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have installed version 1.11.1 or later of the cert-manager Operator for Red Hat OpenShift.

Procedure

- Update the subscription object for cert-manager Operator for Red Hat OpenShift to provide the verbosity level for the operator logs by running the following command:

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --
type='merge' -p '{"spec":{"config":{"env":[{"name":"OPERATOR_LOG_LEVEL","value":"v"}]}}}'
1
```

1 Replace **v** with the desired log level number. The valid values for **v** can range from **1`to `10**. The default value is **2**.

Verification

1. The cert-manager Operator pod is redeployed. Verify that the log level of the cert-manager Operator for Red Hat OpenShift is updated by running the following command:

```
$ oc set env deploy/cert-manager-operator-controller-manager -n cert-manager-operator --
list | grep -e OPERATOR_LOG_LEVEL -e container
```

Example output

```
# deployments/cert-manager-operator-controller-manager, container kube-rbac-proxy
OPERATOR_LOG_LEVEL=9
# deployments/cert-manager-operator-controller-manager, container cert-manager-operator
OPERATOR_LOG_LEVEL=9
```

2. Verify that the log level of the cert-manager Operator for Red Hat OpenShift is updated by running the **oc logs** command:

```
$ oc logs deploy/cert-manager-operator-controller-manager -n cert-manager-operator
```

9.10.3. Additional resources

- Customizing cert-manager Operator API fields

9.11. AUTHENTICATING THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT WITH GCP WORKLOAD IDENTITY

You can authenticate the cert-manager Operator for Red Hat OpenShift on the GCP Workload Identity cluster by using the cloud credentials. You can configure the cloud credentials by using the **ccoctl** binary.

9.11.1. Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift with Google Cloud Workload Identity

Generate the cloud credentials for the cert-manager Operator for Red Hat OpenShift by using the **ccoctl** binary. Then, apply them to the Google Cloud Workload Identity cluster.

Prerequisites

- You extracted and prepared the **ccoctl** binary.
- You have installed version 1.11.1 or later of the cert-manager Operator for Red Hat OpenShift.
- You have configured an OpenShift Container Platform cluster with Google Cloud Workload Identity by using the Cloud Credential Operator in a manual mode.

Procedure

1. Create a directory to store a **CredentialsRequest** resource YAML file by running the following command:

```
$ mkdir credentials-request
```

2. In the **credentials-request** directory, create a YAML file that contains the following **CredentialsRequest** manifest:

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: GCPProviderSpec
    predefinedRoles:
    - roles/dns.admin
  secretRef:
```

```
    name: gcp-credentials
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

NOTE

The **dns.admin** role provides admin privileges to the service account for managing Google Cloud DNS resources. To ensure that the cert-manager runs with the service account that has the least privilege, you can create a custom role with the following permissions:

- **dns.resourceRecordSets.***
- **dns.changes.***
- **dns.managedZones.list**

3. Use the **ccoctl** tool to process **CredentialsRequest** objects by running the following command:

```
$ ccoctl gcp create-service-accounts \
    --name <user_defined_name> --output-dir=<path_to_output_dir> \
    --credentials-requests-dir=<path_to_credrequests_dir> \
    --workload-identity-pool <workload_identity_pool> \
    --workload-identity-provider <workload_identity_provider> \
    --project <gcp_project_id>
```

Example command

```
$ ccoctl gcp create-service-accounts \
    --name abcde-20230525-4bac2781 --output-dir=/home/outputdir \
    --credentials-requests-dir=/home/credentials-requests \
    --workload-identity-pool abcde-20230525-4bac2781 \
    --workload-identity-provider abcde-20230525-4bac2781 \
    --project openshift-gcp-devel
```

4. Apply the secrets generated in the manifests directory of your cluster by running the following command:

```
$ ls <path_to_output_dir>/manifests/*-credentials.yaml | xargs -I{} oc apply -f {}
```

5. Update the subscription object for cert-manager Operator for Red Hat OpenShift by running the following command:

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --
type=merge -p '{"spec":{"config":{"env":
[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"gcp-credentials"}]}}}'
```

Verification

1. Get the name of the redeployed cert-manager controller pod by running the following command:

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

Example output

```
NAME                           READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt   1/1     Running   0          15m39s
```

2. Verify that the cert-manager controller pod is updated with Google Cloud workload identity credential volumes that are mounted under the path specified in **mountPath** by running the following command:

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

Example output

```
spec:
  containers:
  - args:
    ...
    volumeMounts:
    - mountPath: /var/run/secrets/openshift/serviceaccount
      name: bound-sa-token
      ...
    - mountPath: /.config/gcloud
      name: cloud-credentials
  ...
  volumes:
  - name: bound-sa-token
    projected:
      ...
      sources:
      - serviceAccountToken:
          audience: openshift
          ...
          path: token
  - name: cloud-credentials
    secret:
      ...
      items:
      - key: service_account.json
        path: application_default_credentials.json
      secretName: gcp-credentials
```

9.11.2. Additional resources

- Configuring the Cloud Credential Operator utility
- Configuring an OpenShift Container Platform cluster by using the manual mode with GCP Workload Identity

9.12. AUTHENTICATING THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT ON AWS

You can configure the cloud credentials for the cert-manager Operator for Red Hat OpenShift on the AWS cluster. The cloud credentials are generated by the Cloud Credential Operator.

9.12.1. Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift on AWS

To configure the cloud credentials for the cert-manager Operator for Red Hat OpenShift on the AWS cluster you must generate the cloud credentials secret by creating a **CredentialsRequest** object, and allowing the Cloud Credential Operator.

Prerequisites

- You have installed version 1.11.1 or later of the cert-manager Operator for Red Hat OpenShift.
- You have configured the Cloud Credential Operator to operate in *mint* or *passthrough* mode.

Procedure

1. Create a **CredentialsRequest** resource YAML file, for example, **sample-credential-request.yaml**, as follows:

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
      - "route53:GetChange"
      effect: Allow
      resource: "arn:aws:route53:::change/*"
    - action:
      - "route53:ChangeResourceRecordSets"
      - "route53:ListResourceRecordSets"
      effect: Allow
      resource: "arn:aws:route53:::hostedzone/*"
    - action:
      - "route53:ListHostedZonesByName"
      effect: Allow
      resource: "*"
  secretRef:
    name: aws-creds
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

2. Create a **CredentialsRequest** resource by running the following command:

```
$ oc create -f sample-credential-request.yaml
```

3. Update the subscription object for cert-manager Operator for Red Hat OpenShift by running the following command:

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --
type=merge -p '{"spec":{"config":{"env":
[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"aws-creds"}]}}}'
```

Verification

1. Get the name of the redeployed cert-manager controller pod by running the following command:

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

Example output

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          15m39s
```

2. Verify that the cert-manager controller pod is updated with AWS credential volumes that are mounted under the path specified in **mountPath** by running the following command:

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

Example output

```
...
spec:
  containers:
  - args:
    ...
    - mountPath: /.aws
      name: cloud-credentials
  ...
  volumes:
  ...
  - name: cloud-credentials
    secret:
      ...
      secretName: aws-creds
```

9.13. AUTHENTICATING THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT ON GCP

You can configure cloud credentials for the cert-manager Operator for Red Hat OpenShift on a GCP cluster. The cloud credentials are generated by the Cloud Credential Operator.

9.13.1. Configuring cloud credentials for the cert-manager Operator for Red Hat OpenShift on Google Cloud

To configure the cloud credentials for the cert-manager Operator for Red Hat OpenShift on a Google Cloud cluster you must create a **CredentialsRequest** object, and allow the Cloud Credential Operator to generate the cloud credentials secret.

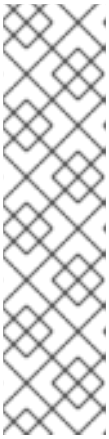
Prerequisites

- You have installed version 1.11.1 or later of the cert-manager Operator for Red Hat OpenShift.
- You have configured the Cloud Credential Operator to operate in *mint* or *passthrough* mode.

Procedure

1. Create a **CredentialsRequest** resource YAML file, such as, **sample-credential-request.yaml** by applying the following yaml:

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: GCPProviderSpec
    predefinedRoles:
    - roles/dns.admin
  secretRef:
    name: gcp-credentials
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```



NOTE

The **dns.admin** role provides admin privileges to the service account for managing Google Cloud DNS resources. To ensure that the cert-manager runs with the service account that has the least privilege, you can create a custom role with the following permissions:

- **dns.resourceRecordSets.***
- **dns.changes.***
- **dns.managedZones.list**

2. Create a **CredentialsRequest** resource by running the following command:

```
$ oc create -f sample-credential-request.yaml
```

3. Update the subscription object for cert-manager Operator for Red Hat OpenShift by running the following command:

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --
type=merge -p '{"spec":{"config":{"env":
[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"gcp-credentials"}]}}}'
```

Verification

1. Get the name of the redeployed cert-manager controller pod by running the following command:

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

Example output

```
NAME                                READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt        1/1     Running   0          15m39s
```

2. Verify that the cert-manager controller pod is updated with Google Cloud credential volumes that are mounted under the path specified in **mountPath** by running the following command:

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

Example output

```
spec:
  containers:
  - args:
    ...
    volumeMounts:
    ...
    - mountPath: /.config/gcloud
      name: cloud-credentials
    ....
  volumes:
  ...
  - name: cloud-credentials
    secret:
      ...
      items:
      - key: service_account.json
        path: application_default_credentials.json
      secretName: gcp-credentials
```

9.14. UNINSTALLING THE CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

You can remove the cert-manager Operator for Red Hat OpenShift from OpenShift Container Platform by uninstalling the Operator and removing its related resources.

9.14.1. Uninstalling the cert-manager Operator for Red Hat OpenShift

You can uninstall the cert-manager Operator for Red Hat OpenShift by using the web console.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- The cert-manager Operator for Red Hat OpenShift is installed.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Uninstall the cert-manager Operator for Red Hat OpenShift Operator.
   a. Navigate to **Operators → Installed Operators**.
   b. Click the Options menu  next to the **cert-manager Operator for Red Hat OpenShift** entry and click **Uninstall Operator**.
   c. In the confirmation dialog, click **Uninstall**.

9.14.2. Removing cert-manager Operator for Red Hat OpenShift resources

Once you have uninstalled the cert-manager Operator for Red Hat OpenShift, you have the option to eliminate its associated resources from your cluster.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Remove the deployments of the cert-manager components, such as **cert-manager**, **cainjector**, and **webhook**, present in the **cert-manager** namespace.
   a. Click the **Project** drop-down menu to see a list of all available projects, and select the **cert-manager** project.
   b. Navigate to **Workloads → Deployments**.
   c. Select the deployment that you want to delete.
   d. Click the **Actions** drop-down menu, and select **Delete Deployment** to see a confirmation dialog box.
   e. Click **Delete** to delete the deployment.
   f. Alternatively, delete deployments of the cert-manager components such as **cert-manager**, **cainjector** and **webhook** present in the **cert-manager** namespace by using the command-line interface (CLI).

```
$ oc delete deployment -n cert-manager -l app.kubernetes.io/instance=cert-manager
```

3. Optional: Remove the custom resource definitions (CRDs) that were installed by the cert-manager Operator for Red Hat OpenShift:
   a. Navigate to **Administration → CustomResourceDefinitions**.
   b. Enter **certmanager** in the **Name** field to filter the CRDs.
   c. Click the Options menu  next to each of the following CRDs, and select **Delete Custom Resource Definition**:
      - **Certificate**
      - **CertificateRequest**
      - **CertManager** (**operator.openshift.io**)
      - **Challenge**
      - **ClusterIssuer**
      - **Issuer**
      - **Order**
4. Optional: Remove the **cert-manager-operator** namespace.
   a. Navigate to **Administration → Namespaces**.
   b. Click the Options menu  next to the **cert-manager-operator** and select **Delete Namespace**.
   c. In the confirmation dialog, enter **cert-manager-operator** in the field and click **Delete**.

CHAPTER 10. VIEWING AUDIT LOGS

OpenShift Container Platform auditing provides a security-relevant chronological set of records documenting the sequence of activities that have affected the system by individual users, administrators, or other components of the system.

10.1. ABOUT THE API AUDIT LOG

Audit works at the API server level, logging all requests coming to the server. Each audit log contains the following information:

Table 10.1. Audit log fields

Field	Description
level	The audit level at which the event was generated.
auditID	A unique audit ID, generated for each request.
stage	The stage of the request handling when this event instance was generated.
requestURI	The request URI as sent by the client to a server.
verb	The Kubernetes verb associated with the request. For non-resource requests, this is the lowercase HTTP method.
user	The authenticated user information.
impersonatedUser	Optional. The impersonated user information, if the request is impersonating another user.
sourceIPs	Optional. The source IPs, from where the request originated and any intermediate proxies.
userAgent	Optional. The user agent string reported by the client. Note that the user agent is provided by the client, and must not be trusted.
objectRef	Optional. The object reference this request is targeted at. This does not apply for **List**-type requests, or non-resource requests.
responseStatus	Optional. The response status, populated even when the **ResponseObject** is not a **Status** type. For successful responses, this will only include the code. For non-status type error responses, this will be auto-populated with the error message.

Field	Description
requestObject	Optional. The API object from the request, in JSON format. The **RequestObject** is recorded as is in the request (possibly re-encoded as JSON), prior to version conversion, defaulting, admission or merging. It is an external versioned object type, and might not be a valid object on its own. This is omitted for non-resource requests and is only logged at request level and higher.
responseObject	Optional. The API object returned in the response, in JSON format. The **ResponseObject** is recorded after conversion to the external type, and serialized as JSON. This is omitted for non-resource requests and is only logged at response level.
requestReceivedTimestamp	The time that the request reached the API server.
stageTimestamp	The time that the request reached the current audit stage.
annotations	Optional. An unstructured key value map stored with an audit event that may be set by plugins invoked in the request serving chain, including authentication, authorization and admission plugins. Note that these annotations are for the audit event, and do not correspond to the **metadata.annotations** of the submitted object. Keys should uniquely identify the informing component to avoid name collisions, for example **podsecuritypolicy.admission.k8s.io/policy**. Values should be short. Annotations are included in the metadata level.

Example output for the Kubernetes API server:


```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"ad209ce1-fec7-4130-8192-c4cc63f1d8cd","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/openshift-kube-controller-manager/configmaps/cert-recovery-controller-lock?timeout=35s","verb":"update","user":{"username":"system:serviceaccount:openshift-kube-controller-manager:localhost-recovery-client","uid":"dd4997e3-d565-4e37-80f8-7fc122ccd785","groups":["system:serviceaccounts","system:serviceaccounts:openshift-kube-controller-manager","system:authenticated"]},"sourceIPs":["::1"],"userAgent":"cluster-kube-controller-manager-operator/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"configmaps","namespace":"openshift-kube-controller-manager","name":"cert-recovery-controller-lock","uid":"5c57190b-6993-425d-8101-8337e48c7548","apiVersion":"v1","resourceVersion":"574307"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2020-04-02T08:27:20.200962Z","stageTimestamp":"2020-04-02T08:27:20.206710Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:openshift:operator:kube-controller-manager-recovery\" of ClusterRole \"cluster-admin\" to ServiceAccount \"localhost-recovery-client/openshift-kube-controller-manager\""}}
```


10.2. VIEWING THE AUDIT LOGS

You can view the logs for the OpenShift API server, Kubernetes API server, OpenShift OAuth API server, and OpenShift OAuth server for each control plane node.

Procedure

To view the audit logs:

- View the OpenShift API server audit logs:
  a. List the OpenShift API server audit logs that are available for each control plane node:

```
$ oc adm node-logs --role=master --path=openshift-apiserver/
```

Example output

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T00-12-19.834.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T00-11-49.835.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T00-13-00.128.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

  b. View a specific OpenShift API server audit log by providing the node name and the log name:

```
$ oc adm node-logs <node_name> --path=openshift-apiserver/<log_name>
```

For example:

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=openshift-apiserver/audit-2021-03-09T00-12-19.834.log
```

Example output

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"381acf6d-5f30-4c7d-8175-c9c317ae5893","stage":"ResponseComplete","requestURI":"/metrics","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","uid":"825b60a0-3976-4861-a342-3b2b561e8f82","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"]},"sourceIPs":["10.129.2.6"],"userAgent":"Prometheus/2.23.0","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T18:02:04.086545Z","stageTimestamp":"2021-03-08T18:02:04.107102Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"prometheus-k8s\" of ClusterRole \"prometheus-k8s\" to ServiceAccount \"prometheus-k8s/openshift-monitoring\""}}
```

- View the Kubernetes API server audit logs:
  a. List the Kubernetes API server audit logs that are available for each control plane node:

```
$ oc adm node-logs --role=master --path=kube-apiserver/
```

Example output

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T14-07-27.129.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T19-24-22.620.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T18-37-07.511.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. View a specific Kubernetes API server audit log by providing the node name and the log name:

```
$ oc adm node-logs <node_name> --path=kube-apiserver/<log_name>
```

For example:

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=kube-apiserver/audit-2021-03-09T14-07-27.129.log
```

Example output

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"cfce8a0b-b5f5-4365-8c9f-79c1227d10f9","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/openshift-kube-scheduler/serviceaccounts/openshift-kube-scheduler-sa","verb":"get","user":{"username":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-kube-scheduler-operator","uid":"2574b041-f3c8-44e6-a057-baef7aa81516","groups":["system:serviceaccounts","system:serviceaccounts:openshift-kube-scheduler-operator","system:authenticated"]},"sourceIPs":["10.128.0.8"],"userAgent":"cluster-kube-scheduler-operator/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"serviceaccounts","namespace":"openshift-kube-scheduler","name":"openshift-kube-scheduler-sa","apiVersion":"v1"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T18:06:42.512619Z","stageTimestamp":"2021-03-08T18:06:42.516145Z","annotations":{"authentication.k8s.io/legacy-token":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-kube-scheduler-operator","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:openshift:operator:cluster-kube-scheduler-operator\" of ClusterRole \"cluster-admin\" to ServiceAccount \"openshift-kube-scheduler-operator/openshift-kube-scheduler-operator\""}}
```

- View the OpenShift OAuth API server audit logs:

a. List the OpenShift OAuth API server audit logs that are available for each control plane node:

```
$ oc adm node-logs --role=master --path=oauth-apiserver/
```

Example output

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T13-06-26.128.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T18-23-21.619.log
```

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T17-36-06.510.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. View a specific OpenShift OAuth API server audit log by providing the node name and the log name:

```
$ oc adm node-logs <node_name> --path=oauth-apiserver/<log_name>
```

For example:

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-apiserver/audit-2021-03-09T13-06-26.128.log
```

Example output

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"dd4c44e2-3ea1-4830-9ab7-c91a5f1388d6","stage":"ResponseComplete","requestURI":"/apis/user.openshift.io/v1/users/~","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"]},"sourceIPs":["10.0.32.4","10.128.0.1"],"userAgent":"dockerregistry/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"users","name":"~","apiGroup":"user.openshift.io","apiVersion":"v1"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T17:47:43.653187Z","stageTimestamp":"2021-03-08T17:47:43.660187Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"basic-users\" of ClusterRole \"basic-user\" to Group \"system:authenticated\""}}
```

- View the OpenShift OAuth server audit logs:

a. List the OpenShift OAuth server audit logs that are available for each control plane node:

```
$ oc adm node-logs --role=master --path=oauth-server/
```

Example output

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2022-05-11T18-57-32.395.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2022-05-11T19-07-07.021.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2022-05-11T19-06-51.844.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. View a specific OpenShift OAuth server audit log by providing the node name and the log name:

```
$ oc adm node-logs <node_name> --path=oauth-server/<log_name>
```

For example:

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-server/audit-2022-05-11T18-57-32.395.log
```

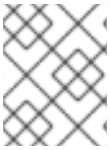
Example output

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"13c20345-f33b-4b7d-b3b6-e7793f805621","stage":"ResponseComplete","requestURI":"/login","verb":"post","user":{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":["10.128.2.6"],"userAgent":"Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0","responseStatus":{"metadata":{},"code":302},"requestReceivedTimestamp":"2022-05-11T17:31:16.280155Z","stageTimestamp":"2022-05-11T17:31:16.297083Z","annotations":{"authentication.openshift.io/decision":"error","authentication.openshift.io/username":"kubeadmin","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
```

The possible values for the **authentication.openshift.io/decision** annotation are **allow**, **deny**, or **error**.

10.3. FILTERING AUDIT LOGS

You can use **jq** or another JSON parsing tool to filter the API server audit logs.



NOTE

The amount of information logged to the API server audit logs is controlled by the audit log policy that is set.

The following procedure provides examples of using **jq** to filter audit logs on control plane node **node-1.example.com**. See the jq Manual for detailed information on using **jq**.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed **jq**.

Procedure

- Filter OpenShift API server audit logs by user:

```
$ oc adm node-logs node-1.example.com  \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.user.username == "myusername")'
```

- Filter OpenShift API server audit logs by user agent:

```
$ oc adm node-logs node-1.example.com  \
  --path=openshift-apiserver/audit.log \
```

```
  | jq 'select(.userAgent == "cluster-version-operator/v0.0.0 (linux/amd64)
kubernetes/$Format")'
```

- Filter Kubernetes API server audit logs by a certain API version and only output the user agent:

```
$ oc adm node-logs node-1.example.com  \
  --path=kube-apiserver/audit.log \
  | jq 'select(.requestURI | startswith("/apis/apiextensions.k8s.io/v1beta1")) | .userAgent'
```

- Filter OpenShift OAuth API server audit logs by excluding a verb:

```
$ oc adm node-logs node-1.example.com  \
  --path=oauth-apiserver/audit.log \
  | jq 'select(.verb != "get")'
```

- Filter OpenShift OAuth server audit logs by events that identified a username and failed with an error:

```
$ oc adm node-logs node-1.example.com  \
  --path=oauth-server/audit.log \
  | jq 'select(.annotations["authentication.openshift.io/username"] != null and
.annotations["authentication.openshift.io/decision"] == "error")'
```

10.4. GATHERING AUDIT LOGS

You can use the must-gather tool to collect the audit logs for debugging your cluster, which you can review or send to Red Hat Support.

Procedure

1. Run the **oc adm must-gather** command with **-- /usr/bin/gather_audit_logs**:

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

2. Create a compressed file from the **must-gather** directory that was just created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather.local.472290403699006248 1
```

1 Replace **must-gather-local.472290403699006248** with the actual directory name.

3. Attach the compressed file to your support case on the the **Customer Support** page of the Red Hat Customer Portal.

10.5. ADDITIONAL RESOURCES

- Must-gather tool
- API audit log event structure

- Configuring the audit log policy
- About log forwarding

CHAPTER 11. CONFIGURING THE AUDIT LOG POLICY

You can control the amount of information that is logged to the API server audit logs by choosing the audit log policy profile to use.

11.1. ABOUT AUDIT LOG POLICY PROFILES

Audit log profiles define how to log requests that come to the OpenShift API server, Kubernetes API server, OpenShift OAuth API server, and OpenShift OAuth server.

OpenShift Container Platform provides the following predefined audit policy profiles:

Profile	Description
Default	Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. This is the default policy.
WriteRequestBodies	In addition to logging metadata for all requests, logs request bodies for every write request to the API servers (**create**, **update**, **patch**, **delete**, **deletecollection**). This profile has more resource overhead than the **Default** profile. [1]
AllRequestBodies	In addition to logging metadata for all requests, logs request bodies for every read and write request to the API servers (**get**, **list**, **create**, **update**, **patch**). This profile has the most resource overhead.[1]
None	No requests are logged; even OAuth access token requests and OAuth authorize token requests are not logged. Custom rules are ignored when this profile is set. **WARNING** It is not recommended to disable audit logging by using the **None** profile unless you are fully aware of the risks of not logging data that can be beneficial when troubleshooting issues. If you disable audit logging and a support situation arises, you might need to enable audit logging and reproduce the issue in order to troubleshoot properly.

1. Sensitive resources, such as **Secret**, **Route**, and **OAuthClient** objects, are only ever logged at the metadata level. OpenShift OAuth server events are only ever logged at the metadata level.

By default, OpenShift Container Platform uses the **Default** audit log profile. You can use another audit policy profile that also logs request bodies, but be aware of the increased resource usage (CPU, memory, and I/O).

11.2. CONFIGURING THE AUDIT LOG POLICY

You can configure the audit log policy to use when logging requests that come to the API servers.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **APIServer** resource:

```
$ oc edit apiserver cluster
```

2. Update the **spec.audit.profile** field:

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: WriteRequestBodies 1
```

1 Set to **Default**, **WriteRequestBodies**, **AllRequestBodies**, or **None**. The default profile is **Default**.

WARNING

It is not recommended to disable audit logging by using the **None** profile unless you are fully aware of the risks of not logging data that can be beneficial when troubleshooting issues. If you disable audit logging and a support situation arises, you might need to enable audit logging and reproduce the issue in order to troubleshoot properly.

3. Save the file to apply the changes.

Verification

- Verify that a new revision of the Kubernetes API server pods is rolled out. It can take several minutes for all nodes to update to the new revision.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Review the **NodeInstallerProgressing** status condition for the Kubernetes API server to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 12 (1)
```

(1) In this example, the latest revision number is **12**.

If the output shows a message similar to one of the following messages, the update is still in progress. Wait a few minutes and try again.

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

11.3. CONFIGURING THE AUDIT LOG POLICY WITH CUSTOM RULES

You can configure an audit log policy that defines custom rules. You can specify multiple groups and define which profile to use for that group.

These custom rules take precedence over the top-level profile field. The custom rules are evaluated from top to bottom, and the first that matches is applied.

IMPORTANT

If you set the top-level profile field to **None**, an API server, such as the Kubernetes API server, ignores custom rules and disables audit logging.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **APIServer** resource:

```
$ oc edit apiserver cluster
```

2. Add the **spec.audit.customRules** field:

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    customRules:                                  (1)
    - group: system:authenticated:oauth
      profile: WriteRequestBodies
```

```
    - group: system:authenticated
      profile: AllRequestBodies
  profile: Default                    2
```

1. Add one or more groups and specify the profile to use for that group. These custom rules take precedence over the top-level profile field. The custom rules are evaluated from top to bottom, and the first that matches is applied.

2. Set to **Default**, **WriteRequestBodies**, or **AllRequestBodies**. If you do not set this top-level profile field, it defaults to the **Default** profile.

3. Save the file to apply the changes.

Verification

- Verify that a new revision of the Kubernetes API server pods is rolled out. It can take several minutes for all nodes to update to the new revision.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Review the **NodeInstallerProgressing** status condition for the Kubernetes API server to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 12 1
```

1. In this example, the latest revision number is **12**.

If the output shows a message similar to one of the following messages, the update is still in progress. Wait a few minutes and try again.

  - **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
  - **2 nodes are at revision 11; 1 nodes are at revision 12**

11.4. DISABLING AUDIT LOGGING

You can disable audit logging for OpenShift Container Platform. When you disable audit logging, even OAuth access token requests and OAuth authorize token requests are not logged.

WARNING

It is not recommended to disable audit logging by using the **None** profile unless you are fully aware of the risks of not logging data that can be beneficial when troubleshooting issues. If you disable audit logging and a support situation arises, you might need to enable audit logging and reproduce the issue in order to troubleshoot properly.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **APIServer** resource:

```
$ oc edit apiserver cluster
```

2. Set the **spec.audit.profile** field to **None**:

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: None
```

NOTE

You can also disable audit logging only for specific groups by specifying custom rules in the **spec.audit.customRules** field.

3. Save the file to apply the changes.

Verification

- Verify that a new revision of the Kubernetes API server pods is rolled out. It can take several minutes for all nodes to update to the new revision.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Review the **NodeInstallerProgressing** status condition for the Kubernetes API server to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 12 (1)
```

(1) In this example, the latest revision number is **12**.

If the output shows a message similar to one of the following messages, the update is still in progress. Wait a few minutes and try again.

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

CHAPTER 12. CONFIGURING TLS SECURITY PROFILES

TLS security profiles provide a way for servers to regulate which ciphers a client can use when connecting to the server. This ensures that OpenShift Container Platform components use cryptographic libraries that do not allow known insecure protocols, ciphers, or algorithms.

Cluster administrators can choose which TLS security profile to use for each of the following components:

- the Ingress Controller
- the control plane
  This includes the Kubernetes API server, Kubernetes controller manager, Kubernetes scheduler, OpenShift API server, OpenShift OAuth API server, OpenShift OAuth server, and etcd.
- the kubelet, when it acts as an HTTP server for the Kubernetes API server

12.1. UNDERSTANDING TLS SECURITY PROFILES

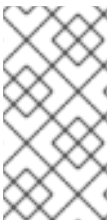
You can use a TLS (Transport Layer Security) security profile to define which TLS ciphers are required by various OpenShift Container Platform components. The OpenShift Container Platform TLS security profiles are based on Mozilla recommended configurations.

You can specify one of the following TLS security profiles for each component:

Table 12.1. TLS security profiles

Profile	Description
Old	This profile is intended for use with legacy clients or libraries. The profile is based on the Old backward compatibility recommended configuration. The **Old** profile requires a minimum TLS version of 1.0.  **NOTE** For the Ingress Controller, the minimum TLS version is converted from 1.0 to 1.1.
Intermediate	This profile is the default TLS security profile for the Ingress Controller, kubelet, and control plane. The profile is based on the Intermediate compatibility recommended configuration. The **Intermediate** profile requires a minimum TLS version of 1.2.  **NOTE** This profile is the recommended configuration for the majority of clients.

Profile	Description
Modern	This profile is intended for use with modern clients that have no need for backwards compatibility. This profile is based on the Modern compatibility recommended configuration. The **Modern** profile requires a minimum TLS version of 1.3.
Custom	This profile allows you to define the TLS version and ciphers to use. **WARNING** Use caution when using a **Custom** profile, because invalid configurations can cause problems.



NOTE

When using one of the predefined profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the Intermediate profile deployed on release X.Y.Z, an upgrade to release X.Y.Z+1 might cause a new profile configuration to be applied, resulting in a rollout.

12.2. VIEWING TLS SECURITY PROFILE DETAILS

You can view the minimum TLS version and ciphers for the predefined TLS security profiles for each of the following components: Ingress Controller, control plane, and kubelet.

IMPORTANT

The effective configuration of minimum TLS version and list of ciphers for a profile might differ between components.

Procedure

- View details for a specific TLS security profile:

```
$ oc explain <component>.spec.tlsSecurityProfile.<profile> 1
```

1 For **<component>**, specify **ingresscontroller**, **apiserver**, or **kubeletconfig**. For **<profile>**, specify **old**, **intermediate**, or **custom**.

For example, to check the ciphers included for the **intermediate** profile for the control plane:

```
$ oc explain apiserver.spec.tlsSecurityProfile.intermediate
```

Example output

```
KIND:     APIServer
VERSION:  config.openshift.io/v1

DESCRIPTION:
    intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended
.29
    and looks like this (yaml):
    ciphers: - TLS_AES_128_GCM_SHA256 - TLS_AES_256_GCM_SHA384 -
    TLS_CHACHA20_POLY1305_SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256 -
    ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES256-GCM-SHA384 -
    ECDHE-RSA-AES256-GCM-SHA384 - ECDHE-ECDSA-CHACHA20-POLY1305 -
    ECDHE-RSA-CHACHA20-POLY1305 - DHE-RSA-AES128-GCM-SHA256 -
    DHE-RSA-AES256-GCM-SHA384 minTLSVersion: TLSv1.2
```

- View all details for the **tlsSecurityProfile** field of a component:

```
$ oc explain <component>.spec.tlsSecurityProfile (1)
```

(1) For **<component>**, specify **ingresscontroller**, **apiserver**, or **kubeletconfig**.

For example, to check all details for the **tlsSecurityProfile** field for the Ingress Controller:

```
$ oc explain ingresscontroller.spec.tlsSecurityProfile
```

Example output

```
KIND:     IngressController
VERSION:  operator.openshift.io/v1

RESOURCE: tlsSecurityProfile <Object>

DESCRIPTION:
     ...

FIELDS:
   custom <>
     custom is a user-defined TLS security profile. Be extremely careful using a
     custom profile as invalid configurations can be catastrophic. An example
     custom profile looks like this:
     ciphers: - ECDHE-ECDSA-CHACHA20-POLY1305 - ECDHE-RSA-CHACHA20-
POLY1305 -
     ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256
minTLSVersion:
     TLSv1.1

   intermediate <>
     intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended
.29
     and looks like this (yaml):
```

```
    ... 1

  modern <>
    modern is a TLS security profile based on:
    https://wiki.mozilla.org/Security/Server_Side_TLS#Modern_compatibility and
    looks like this (yaml):
    ... 2
    NOTE: Currently unsupported.

  old <>
    old is a TLS security profile based on:
    https://wiki.mozilla.org/Security/Server_Side_TLS#Old_backward_compatibility
    and looks like this (yaml):
    ... 3

  type <string>
    ...
```

1. Lists ciphers and minimum version for the **intermediate** profile here.
2. Lists ciphers and minimum version for the **modern** profile here.
3. Lists ciphers and minimum version for the **old** profile here.

12.3. CONFIGURING THE TLS SECURITY PROFILE FOR THE INGRESS CONTROLLER

To configure a TLS security profile for an Ingress Controller, edit the **IngressController** custom resource (CR) to specify a predefined or custom TLS security profile. If a TLS security profile is not configured, the default value is based on the TLS security profile set for the API server.

Sample IngressController CR that configures the Old TLS security profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

The TLS security profile defines the minimum TLS version and the TLS ciphers for TLS connections for Ingress Controllers.

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the **IngressController** custom resource (CR) under **Status.Tls Profile** and the configured TLS security profile under **Spec.Tls Security Profile**. For the **Custom** TLS security profile, the specific ciphers and minimum TLS version are listed under both parameters.

NOTE

The HAProxy Ingress Controller image supports TLS **1.3** and the **Modern** profile.

The Ingress Operator also converts the TLS **1.0** of an **Old** or **Custom** profile to **1.1**.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **IngressController** CR in the **openshift-ingress-operator** project to configure the TLS security profile:

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. Add the **spec.tlsSecurityProfile** field:

Sample IngressController CR for a Custom profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
...
```

(1) Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.

(2) Specify the appropriate field for the selected type:

- **old: {}**
- **intermediate: {}**
- **custom:**

(3) For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.

3. Save the file to apply the changes.

Verification

- Verify that the profile is set in the **IngressController** CR:

```
$ oc describe IngressController default -n openshift-ingress-operator
```

Example output

```
Name:         default
Namespace:    openshift-ingress-operator
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

12.4. CONFIGURING THE TLS SECURITY PROFILE FOR THE CONTROL PLANE

To configure a TLS security profile for the control plane, edit the **APIServer** custom resource (CR) to specify a predefined or custom TLS security profile. Setting the TLS security profile in the **APIServer** CR propagates the setting to the following control plane components:

- Kubernetes API server
- Kubernetes controller manager
- Kubernetes scheduler
- OpenShift API server
- OpenShift OAuth API server
- OpenShift OAuth server
- etcd

If a TLS security profile is not configured, the default TLS security profile is **Intermediate**.

NOTE

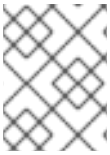
The default TLS security profile for the Ingress Controller is based on the TLS security profile set for the API server.

Sample APIServer CR that configures the Old TLS security profile

```
apiVersion: config.openshift.io/v1
kind: APIServer
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

The TLS security profile defines the minimum TLS version and the TLS ciphers required to communicate with the control plane components.

You can see the configured TLS security profile in the **APIServer** custom resource (CR) under **Spec.Tls Security Profile**. For the **Custom** TLS security profile, the specific ciphers and minimum TLS version are listed.



NOTE

The control plane does not support TLS **1.3** as the minimum TLS version; the **Modern** profile is not supported because it requires TLS **1.3**.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the default **APIServer** CR to configure the TLS security profile:

```
$ oc edit APIServer cluster
```

2. Add the **spec.tlsSecurityProfile** field:

Sample APIServer CR for a Custom profile

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
```

1. Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.
2. Specify the appropriate field for the selected type:
   - **old: {}**
   - **intermediate: {}**
   - **custom:**
3. For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.

3. Save the file to apply the changes.

Verification

- Verify that the TLS security profile is set in the **APIServer** CR:

```
$ oc describe apiserver cluster
```

Example output

```
Name:         cluster
Namespace:
 ...
API Version:  config.openshift.io/v1
Kind:         APIServer
 ...
Spec:
  Audit:
    Profile:  Default
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

- Verify that the TLS security profile is set in the **etcd** CR:

```
$ oc describe etcd cluster
```

Example output

```
Name:         cluster
Namespace:
 ...
API Version:  operator.openshift.io/v1
```

```
Kind:         Etcd
 ...
Spec:
  Log Level:         Normal
  Management State:  Managed
  Observed Config:
    Serving Info:
      Cipher Suites:
        TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
        TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
      Min TLS Version:           VersionTLS12
 ...
```

12.5. CONFIGURING THE TLS SECURITY PROFILE FOR THE KUBELET

To configure a TLS security profile for the kubelet when it is acting as an HTTP server, create a **KubeletConfig** custom resource (CR) to specify a predefined or custom TLS security profile for specific nodes. If a TLS security profile is not configured, the default TLS security profile is **Intermediate**.

The kubelet uses its HTTP/GRPC server to communicate with the Kubernetes API server, which sends commands to pods, gathers logs, and run exec commands on pods through the kubelet.

Sample KubeletConfig CR that configures the Old TLS security profile on worker nodes

```
apiVersion: config.openshift.io/v1
kind: KubeletConfig
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
#...
```

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the **kubelet.conf** file on a configured node.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.

Procedure

1. Create a **KubeletConfig** CR to configure the TLS security profile:

   ### Sample KubeletConfig CR for a Custom profile

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   ```

```
kind: KubeletConfig
metadata:
  name: set-kubelet-tls-security-profile
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (4)
#...
```

1. Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.
2. Specify the appropriate field for the selected type:
   - **old: {}**
   - **intermediate: {}**
   - **custom:**
3. For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.
4. Optional: Specify the machine config pool label for the nodes you want to apply the TLS security profile.

2. Create the **KubeletConfig** object:

```
$ oc create -f <filename>
```

Depending on the number of worker nodes in the cluster, wait for the configured nodes to be rebooted one by one.

Verification

To verify that the profile is set, perform the following steps after the nodes are in the **Ready** state:

1. Start a debug session for a configured node:

```
$ oc debug node/<node_name>
```

2. Set /**host** as the root directory within the debug shell:

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

3. View the **kubelet.conf** file:

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

Example output

```
  "kind": "KubeletConfiguration",
  "apiVersion": "kubelet.config.k8s.io/v1beta1",
#...
  "tlsCipherSuites": [
    "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
    "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
  ],
  "tlsMinVersion": "VersionTLS12",
#...
```

CHAPTER 13. CONFIGURING SECCOMP PROFILES

An OpenShift Container Platform container or a pod runs a single application that performs one or more well-defined tasks. The application usually requires only a small subset of the underlying operating system kernel APIs. Secure computing mode, seccomp, is a Linux kernel feature that can be used to limit the process running in a container to only using a subset of the available system calls.

The **restricted-v2** SCC applies to all newly created pods in 4.13. The default seccomp profile **runtime/default** is applied to these pods.

Seccomp profiles are stored as JSON files on the disk.

IMPORTANT

Seccomp profiles cannot be applied to privileged containers.

13.1. VERIFYING THE DEFAULT SECCOMP PROFILE APPLIED TO A POD

OpenShift Container Platform ships with a default seccomp profile that is referenced as **runtime/default**. In 4.13, newly created pods have the Security Context Constraint (SCC) set to **restricted-v2** and the default seccomp profile applies to the pod.

Procedure

1. You can verify the Security Context Constraint (SCC) and the default seccomp profile set on a pod by running the following commands:

   a. Verify what pods are running in the namespace:

   ```
   $ oc get pods -n <namespace>
   ```

   For example, to verify what pods are running in the **workshop** namespace run the following:

   ```
   $ oc get pods -n workshop
   ```

   Example output

   ```
   NAME                READY   STATUS      RESTARTS   AGE
   parksmap-1-4xkwf    1/1     Running     0          2m17s
   parksmap-1-deploy   0/1     Completed   0          2m22s
   ```

   b. Inspect the pods:

   ```
   $ oc get pod parksmap-1-4xkwf -n workshop -o yaml
   ```

   Example output

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     annotations:
   ```

```
k8s.v1.cni.cncf.io/network-status: |-
  [{
      "name": "openshift-sdn",
      "interface": "eth0",
      "ips": [
          "10.131.0.18"
      ],
      "default": true,
      "dns": {}
  }]
k8s.v1.cni.cncf.io/network-status: |-
  [{
      "name": "openshift-sdn",
      "interface": "eth0",
      "ips": [
          "10.131.0.18"
      ],
      "default": true,
      "dns": {}
  }]
openshift.io/deployment-config.latest-version: "1"
openshift.io/deployment-config.name: parksmap
openshift.io/deployment.name: parksmap-1
openshift.io/generated-by: OpenShiftWebConsole
openshift.io/scc: restricted-v2 (1)
seccomp.security.alpha.kubernetes.io/pod: runtime/default (2)
```

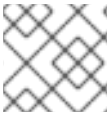
(1) The **restricted-v2** SCC is added by default if your workload does not have access to a different SCC.

(2) Newly created pods in 4.13 will have the seccomp profile configured to **runtime/default** as mandated by the SCC.

13.1.1. Upgraded cluster

In clusters upgraded to 4.13 all authenticated users have access to the **restricted** and **restricted-v2** SCC.

A workload admitted by the SCC **restricted** for example, on a OpenShift Container Platform v4.10 cluster when upgraded may get admitted by **restricted-v2**. This is because **restricted-v2** is the more restrictive SCC between **restricted** and **restricted-v2**.



NOTE

The workload must be able to run with **retricted-v2**.

Conversely with a workload that requires **privilegeEscalation: true** this workload will continue to have the **restricted** SCC available for any authenticated user. This is because **restricted-v2** does not allow **privilegeEscalation**.

13.1.2. Newly installed cluster

For newly installed OpenShift Container Platform 4.11 or later clusters, the **restricted-v2** replaces the

restricted SCC as an SCC that is available to be used by any authenticated user. A workload with **privilegeEscalation: true**, is not admitted into the cluster since **restricted-v2** is the only SCC available for authenticated users by default.

The feature **privilegeEscalation** is allowed by **restricted** but not by **restricted-v2**. More features are denied by **restricted-v2** than were allowed by **restricted** SCC.

A workload with **privilegeEscalation: true** may be admitted into a newly installed OpenShift Container Platform 4.11 or later cluster. To give access to the **restricted** SCC to the ServiceAccount running the workload (or any other SCC that can admit this workload) using a RoleBinding run the following command:

```
$ oc -n <workload-namespace> adm policy add-scc-to-user <scc-name> -z <serviceaccount_name>
```

In OpenShift Container Platform 4.13 the ability to add the pod annotations **seccomp.security.alpha.kubernetes.io/pod: runtime/default** and **container.seccomp.security.alpha.kubernetes.io/<container_name>: runtime/default** is deprecated.

13.2. CONFIGURING A CUSTOM SECCOMP PROFILE

You can configure a custom seccomp profile, which allows you to update the filters based on the application requirements. This allows cluster administrators to have greater control over the security of workloads running in OpenShift Container Platform.

Seccomp security profiles list the system calls (syscalls) a process can make. Permissions are broader than SELinux, which restrict operations, such as **write**, system-wide.

13.2.1. Creating seccomp profiles

You can use the **MachineConfig** object to create profiles.

Seccomp can restrict system calls (syscalls) within a container, limiting the access of your application.

Prerequisites

- You have cluster admin permissions.
- You have created a custom security context constraints (SCC). For more information, see *Additional resources*.

Procedure

- Create the **MachineConfig** object:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: custom-seccomp
spec:
  config:
    ignition:
      version: 3.2.0
```

```
storage:
  files:
  - contents:
      source: data:text/plain;charset=utf-8;base64,<hash>
    filesystem: root
    mode: 0644
    path: /var/lib/kubelet/seccomp/seccomp-nostat.json
```

13.2.2. Setting up the custom seccomp profile

Prerequisite

- You have cluster administrator permissions.
- You have created a custom security context constraints (SCC). For more information, see "Additional resources".
- You have created a custom seccomp profile.

Procedure

1. Upload your custom seccomp profile to **/var/lib/kubelet/seccomp/<custom-name>.json** by using the Machine Config. See "Additional resources" for detailed steps.
2. Update the custom SCC by providing reference to the created custom seccomp profile:

```
seccompProfiles:
- localhost/<custom-name>.json 1
```

1 Provide the name of your custom seccomp profile.

13.2.3. Applying the custom seccomp profile to the workload

Prerequisite

- The cluster administrator has set up the custom seccomp profile. For more details, see "Setting up the custom seccomp profile".

Procedure

- Apply the seccomp profile to the workload by setting the **securityContext.seccompProfile.type** field as following:

  Example

```
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: <custom-name>.json 1
```

1 Provide the name of your custom seccomp profile.

Alternatively, you can use the pod annotations **seccomp.security.alpha.kubernetes.io/pod: localhost/<custom-name>.json**. However, this method is deprecated in OpenShift Container Platform 4.13.

During deployment, the admission controller validates the following:

- The annotations against the current SCCs allowed by the user role.
- The SCC, which includes the seccomp profile, is allowed for the pod.

If the SCC is allowed for the pod, the kubelet runs the pod with the specified seccomp profile.

IMPORTANT

Ensure that the seccomp profile is deployed to all worker nodes.

NOTE

The custom SCC must have the appropriate priority to be automatically assigned to the pod or meet other conditions required by the pod, such as allowing CAP_NET_ADMIN.

13.3. ADDITIONAL RESOURCES

- Managing security context constraints
- Post-installation machine configuration tasks

CHAPTER 14. ALLOWING JAVASCRIPT-BASED ACCESS TO THE API SERVER FROM ADDITIONAL HOSTS

14.1. ALLOWING JAVASCRIPT-BASED ACCESS TO THE API SERVER FROM ADDITIONAL HOSTS

The default OpenShift Container Platform configuration only allows the web console to send requests to the API server.

If you need to access the API server or OAuth server from a JavaScript application using a different hostname, you can configure additional hostnames to allow.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **APIServer** resource:

```
$ oc edit apiserver.config.openshift.io cluster
```

2. Add the **additionalCORSAllowedOrigins** field under the **spec** section and specify one or more additional hostnames:

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-07-11T17:35:37Z"
  generation: 1
  name: cluster
  resourceVersion: "907"
  selfLink: /apis/config.openshift.io/v1/apiservers/cluster
  uid: 4b45a8dd-a402-11e9-91ec-0219944e0696
spec:
  additionalCORSAllowedOrigins:
  - (?i)//my\.subdomain\.domain\.com(:|\z) 1
```

1 The hostname is specified as a Golang regular expression that matches against CORS headers from HTTP requests against the API server and OAuth server.

NOTE

This example uses the following syntax:

- The **(?i)** makes it case-insensitive.
- The // pins to the beginning of the domain and matches the double slash following **http:** or **https:**.
- The **\.** escapes dots in the domain name.
- The **(:|\z)** matches the end of the domain name **(\z)** or a port separator **(:)**.

3. Save the file to apply the changes.

CHAPTER 15. ENCRYPTING ETCD DATA

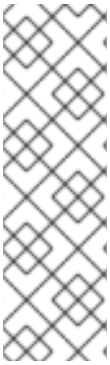
15.1. ABOUT ETCD ENCRYPTION

By default, etcd data is not encrypted in OpenShift Container Platform. You can enable etcd encryption for your cluster to provide an additional layer of data security. For example, it can help protect the loss of sensitive data if an etcd backup is exposed to the incorrect parties.

When you enable etcd encryption, the following OpenShift API server and Kubernetes API server resources are encrypted:

- Secrets
- Config maps
- Routes
- OAuth access tokens
- OAuth authorize tokens

When you enable etcd encryption, encryption keys are created. You must have these keys to restore from an etcd backup.



NOTE

Etcd encryption only encrypts values, not keys. Resource types, namespaces, and object names are unencrypted.

If etcd encryption is enabled during a backup, the ***static_kuberesources_<datetimestamp>.tar.gz*** file contains the encryption keys for the etcd snapshot. For security reasons, store this file separately from the etcd snapshot. However, this file is required to restore a previous state of etcd from the respective etcd snapshot.

15.2. SUPPORTED ENCRYPTION TYPES

The following encryption types are supported for encrypting etcd data in OpenShift Container Platform:

AES-CBC

Uses AES-CBC with PKCS#7 padding and a 32 byte key to perform the encryption. The encryption keys are rotated weekly.

AES-GCM

Uses AES-GCM with a random nonce and a 32 byte key to perform the encryption. The encryption keys are rotated weekly.

15.3. ENABLING ETCD ENCRYPTION

You can enable etcd encryption to encrypt sensitive resources in your cluster.

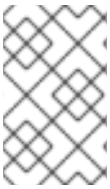
WARNING

Do not back up etcd resources until the initial encryption process is completed. If the encryption process is not completed, the backup might be only partially encrypted.

After you enable etcd encryption, several changes can occur:

- The etcd encryption might affect the memory consumption of a few resources.
- You might notice a transient affect on backup performance because the leader must serve the backup.
- A disk I/O can affect the node that receives the backup state.

You can encrypt the etcd database in either AES-GCM or AES-CBC encryption.



NOTE

To migrate your etcd database from one encryption type to the other, you can modify the API server's **spec.encryption.type** field. Migration of the etcd data to the new encryption type occurs automatically.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Modify the **APIServer** object:

```
$ oc edit apiserver
```

2. Set the **spec.encryption.type** field to **aesgcm** or **aescbc**:

```
spec:
  encryption:
    type: aesgcm 1
```

1 Set to **aesgcm** for AES-GCM encryption or **aescbc** for AES-CBC encryption.

3. Save the file to apply the changes.
The encryption process starts. It can take 20 minutes or longer for this process to complete, depending on the size of the etcd database.

4. Verify that etcd encryption was successful.

   a. Review the **Encrypted** status condition for the OpenShift API server to verify that its

resources were successfully encrypted:

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **EncryptionCompleted** upon successful encryption:

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

b. Review the **Encrypted** status condition for the Kubernetes API server to verify that its resources were successfully encrypted:

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **EncryptionCompleted** upon successful encryption:

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

c. Review the **Encrypted** status condition for the OpenShift OAuth API server to verify that its resources were successfully encrypted:

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **EncryptionCompleted** upon successful encryption:

```
EncryptionCompleted
All resources encrypted: oauthaccesstokens.oauth.openshift.io,
oauthauthorizetokens.oauth.openshift.io
```

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

15.4. DISABLING ETCD ENCRYPTION

You can disable encryption of etcd data in your cluster.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Modify the **APIServer** object:

```
$ oc edit apiserver
```

2. Set the **encryption** field type to **identity**:

```
spec:
  encryption:
    type: identity (1)
```

(1) The **identity** type is the default value and means that no encryption is performed.

3. Save the file to apply the changes.
The decryption process starts. It can take 20 minutes or longer for this process to complete, depending on the size of your cluster.

4. Verify that etcd decryption was successful.

   a. Review the **Encrypted** status condition for the OpenShift API server to verify that its resources were successfully decrypted:

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

   If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

   b. Review the **Encrypted** status condition for the Kubernetes API server to verify that its resources were successfully decrypted:

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

   If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

   c. Review the **Encrypted** status condition for the OpenShift OAuth API server to verify that its resources were successfully decrypted:

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

CHAPTER 16. SCANNING PODS FOR VULNERABILITIES

IMPORTANT

The Red Hat Quay Container Security Operator has been deprecated and is planned for removal in a future release of OpenShift Container Platform. The official replacement product of the Red Hat Quay Container Security Operator is Red Hat Advanced Cluster Security for Kubernetes.

Using the Red Hat Quay Container Security Operator, you can access vulnerability scan results from the OpenShift Container Platform web console for container images used in active pods on the cluster. The Red Hat Quay Container Security Operator:

- Watches containers associated with pods on all or specified namespaces
- Queries the container registry where the containers came from for vulnerability information, provided an image's registry is running image scanning (such as Quay.io or a Red Hat Quay registry with Clair scanning)
- Exposes vulnerabilities via the **ImageManifestVuln** object in the Kubernetes API

Using the instructions here, the Red Hat Quay Container Security Operator is installed in the **openshift-operators** namespace, so it is available to all namespaces on your OpenShift Container Platform cluster.

16.1. INSTALLING THE RED HAT QUAY CONTAINER SECURITY OPERATOR

You can install the Red Hat Quay Container Security Operator from the OpenShift Container Platform web console Operator Hub, or by using the CLI.

Prerequisites

- You have installed the **oc** CLI.
- You have administrator privileges to the OpenShift Container Platform cluster.
- You have containers that come from a Red Hat Quay or Quay.io registry running on your cluster.

Procedure

1. You can install the Red Hat Quay Container Security Operator by using the OpenShift Container Platform web console:
    a. On the web console, navigate to **Operators → OperatorHub** and select **Security**.
    b. Select the **Red Hat Quay Container Security Operator**Operator, and then select **Install**.
    c. On the **Red Hat Quay Container Security Operator**page, select **Install**. **Update channel**, **Installation mode**, and **Update approval** are selected automatically. The **Installed Namespace** field defaults to **openshift-operators**. You can adjust these settings as needed.
    d. Select **Install**. The **Red Hat Quay Container Security Operator**appears after a few moments on the **Installed Operators** page.

e. Optional: You can add custom certificates to the Red Hat Quay Container Security Operator. For example, create a certificate named **quay.crt** in the current directory. Then, run the following command to add the custom certificate to the Red Hat Quay Container Security Operator:

```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n
openshift-operators
```

f. Optional: If you added a custom certificate, restart the Red Hat Quay Container Security Operator pod for the new certificates to take effect.

2. Alternatively, you can install the Red Hat Quay Container Security Operator by using the CLI:

a. Retrieve the latest version of the Container Security Operator and its channel by entering the following command:

```
$ oc get packagemanifests container-security-operator \
  -o jsonpath='{range .status.channels[*]}{@.currentCSV} {@.name}{"\n"}{end}' \
  | awk '{print "STARTING_CSV=" $1 " CHANNEL=" $2 }' \
  | sort -Vr \
  | head -1
```

Example output

```
STARTING_CSV=container-security-operator.v3.8.9 CHANNEL=stable-3.8
```

b. Using the output from the previous command, create a **Subscription** custom resource for the Red Hat Quay Container Security Operator and save it as **container-security-operator.yaml**. For example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: container-security-operator
  namespace: openshift-operators
spec:
  channel: ${CHANNEL} 1
  installPlanApproval: Automatic
  name: container-security-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: ${STARTING_CSV} 2
```

1 Specify the value you obtained in the previous step for the **spec.channel** parameter.

2 Specify the value you obtained in the previous step for the **spec.startingCSV** parameter.

c. Enter the following command to apply the configuration:

```
$ oc apply -f container-security-operator.yaml
```

Example output

```
subscription.operators.coreos.com/container-security-operator created
```

16.2. USING THE RED HAT QUAY CONTAINER SECURITY OPERATOR

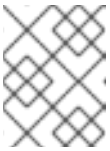
The following procedure shows you how to use the Red Hat Quay Container Security Operator.

Prerequisites

- You have installed the Red Hat Quay Container Security Operator.

Procedure

1. On the OpenShift Container Platform web console, navigate to **Home → Overview**. Under the **Status** section, **Image Vulnerabilities** provides the number of vulnerabilities found.
2. Click **Image Vulnerabilities** to reveal the **Image Vulnerabilities breakdown** tab, which details the severity of the vulnerabilities, whether the vulnerabilities can be fixed, and the total number of vulnerabilities.
3. You can address detected vulnerabilities in one of two ways:
   a. Select a link under the **Vulnerabilities** section. This takes you to the container registry that the container came from, where you can see information about the vulnerability.
   b. Select the **namespace** link. This takes you to the **Image Manifest Vulnerabilities** page, where you can see the name of the selected image and all of the namespaces where that image is running.
4. After you have learned what images are vulnerable, how to fix those vulnerabilities, and the namespaces that the images are being run in, you can improve security by performing the following actions:
   a. Alert anyone in your organization who is running the image and request that they correct the vulnerability.
   b. Stop the images from running by deleting the deployment or other object that started the pod that the image is in.



NOTE

If you delete the pod, it might take several minutes for the vulnerability information to reset on the dashboard.

16.3. QUERYING IMAGE VULNERABILITIES FROM THE CLI

Using the **oc** command, you can display information about vulnerabilities detected by the Red Hat Quay Container Security Operator.

Prerequisites

- You have installed the Red Hat Quay Container Security Operator on your OpenShift Container Platform instance.

Procedure

1. Enter the following command to query for detected container image vulnerabilities:

```
$ oc get vuln --all-namespaces
```

Example output

```
NAMESPACE     NAME              AGE
default       sha256.ca90...    6m56s
skynet        sha256.ca90...    9m37s
```

2. To display details for a particular vulnerability, append the vulnerability name and its namespace to the **oc describe** command. The following example shows an active container whose image includes an RPM package with a vulnerability:

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
```

Example output

```
Name:         sha256.ac50e3752...
Namespace:    quay-enterprise
...
Spec:
  Features:
    Name:           nss-util
    Namespace Name:  centos:7
    Version:        3.44.0-3.el7
    Versionformat:   rpm
    Vulnerabilities:
      Description: Network Security Services (NSS) is a set of libraries...
```

16.4. UNINSTALLING THE RED HAT QUAY CONTAINER SECURITY OPERATOR

To uninstall the Container Security Operator, you must uninstall the Operator and delete the **imagemanifestvulns.secscan.quay.redhat.com** custom resource definition (CRD).

Procedure

1. On the OpenShift Container Platform web console, click **Operators → Installed Operators**.
2. Click the menu  of the Container Security Operator.
3. Click **Uninstall Operator**.
4. Confirm your decision by clicking **Uninstall** in the popup window.
5. Use the CLI to delete the **imagemanifestvulns.secscan.quay.redhat.com** CRD.
    a. Remove the **imagemanifestvulns.secscan.quay.redhat.com** custom resource definition by entering the following command:

```
$ oc delete customresourcedefinition imagemanifestvulns.secscan.quay.redhat.com
```

Example output

```
customresourcedefinition.apiextensions.k8s.io
"imagemanifestvulns.secscan.quay.redhat.com" deleted
```

CHAPTER 17. NETWORK-BOUND DISK ENCRYPTION (NBDE)

17.1. ABOUT DISK ENCRYPTION TECHNOLOGY

Network-Bound Disk Encryption (NBDE) allows you to encrypt root volumes of hard drives on physical and virtual machines without having to manually enter a password when restarting machines.

17.1.1. Disk encryption technology comparison

To understand the merits of Network-Bound Disk Encryption (NBDE) for securing data at rest on edge servers, compare key escrow and TPM disk encryption without Clevis to NBDE on systems running Red Hat Enterprise Linux (RHEL).

The following table presents some tradeoffs to consider around the threat model and the complexity of each encryption solution.

Scenario	Key escrow	TPM disk encryption (without Clevis)	NBDE
Protects against single-disk theft	X	X	X
Protects against entire-server theft	X		X
Systems can reboot independently from the network		X	
No periodic rekeying		X	
Key is never transmitted over a network		X	X
Supported by OpenShift		X	X

17.1.1.1. Key escrow

Key escrow is the traditional system for storing cryptographic keys. The key server on the network stores the encryption key for a node with an encrypted boot disk and returns it when queried. The complexities around key management, transport encryption, and authentication do not make this a reasonable choice for boot disk encryption.

Although available in Red Hat Enterprise Linux (RHEL), key escrow-based disk encryption setup and management is a manual process and not suited to OpenShift Container Platform automation operations, including automated addition of nodes, and currently not supported by OpenShift Container Platform.

17.1.1.2. TPM encryption

Trusted Platform Module (TPM) disk encryption is best suited for data centers or installations in remote

protected locations. Full disk encryption utilities such as dm-crypt and BitLocker encrypt disks with a TPM bind key, and then store the TPM bind key in the TPM, which is attached to the motherboard of the node. The main benefit of this method is that there is no external dependency, and the node is able to decrypt its own disks at boot time without any external interaction.

TPM disk encryption protects against decryption of data if the disk is stolen from the node and analyzed externally. However, for insecure locations this may not be sufficient. For example, if an attacker steals the entire node, the attacker can intercept the data when powering on the node, because the node decrypts its own disks. This applies to nodes with physical TPM2 chips as well as virtual machines with Virtual Trusted Platform Module (VTPM) access.

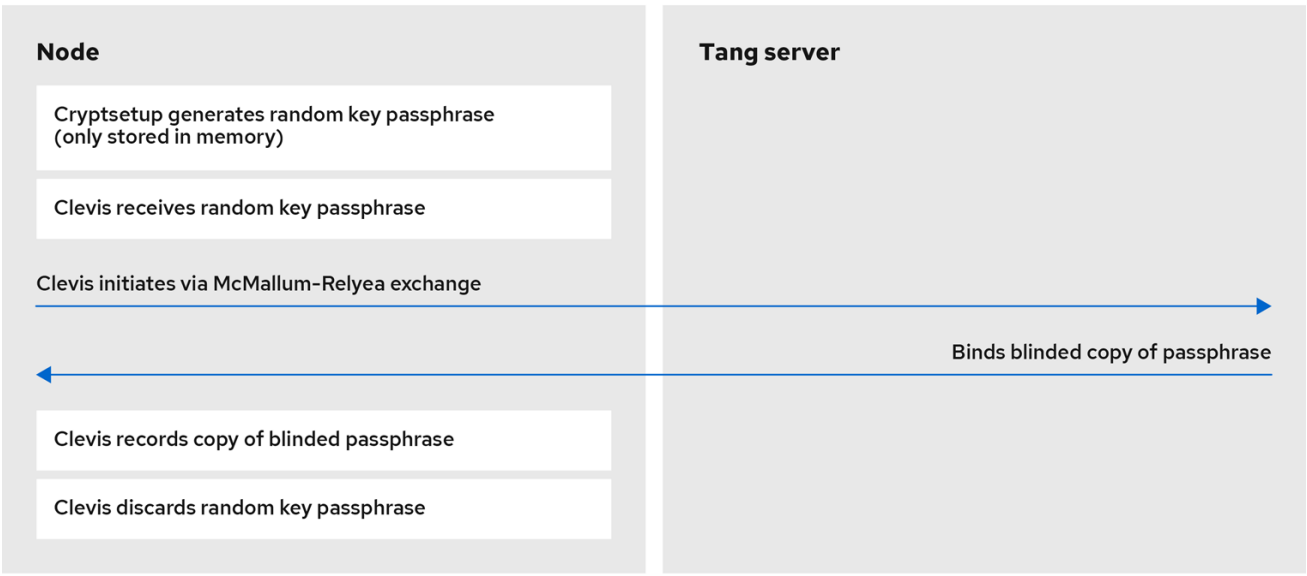
17.1.1.3. Network-Bound Disk Encryption (NBDE)

Network-Bound Disk Encryption (NBDE) effectively ties the encryption key to an external server or set of servers in a secure and anonymous way across the network. This is not a key escrow, in that the nodes do not store the encryption key or transfer it over the network, but otherwise behaves in a similar fashion.

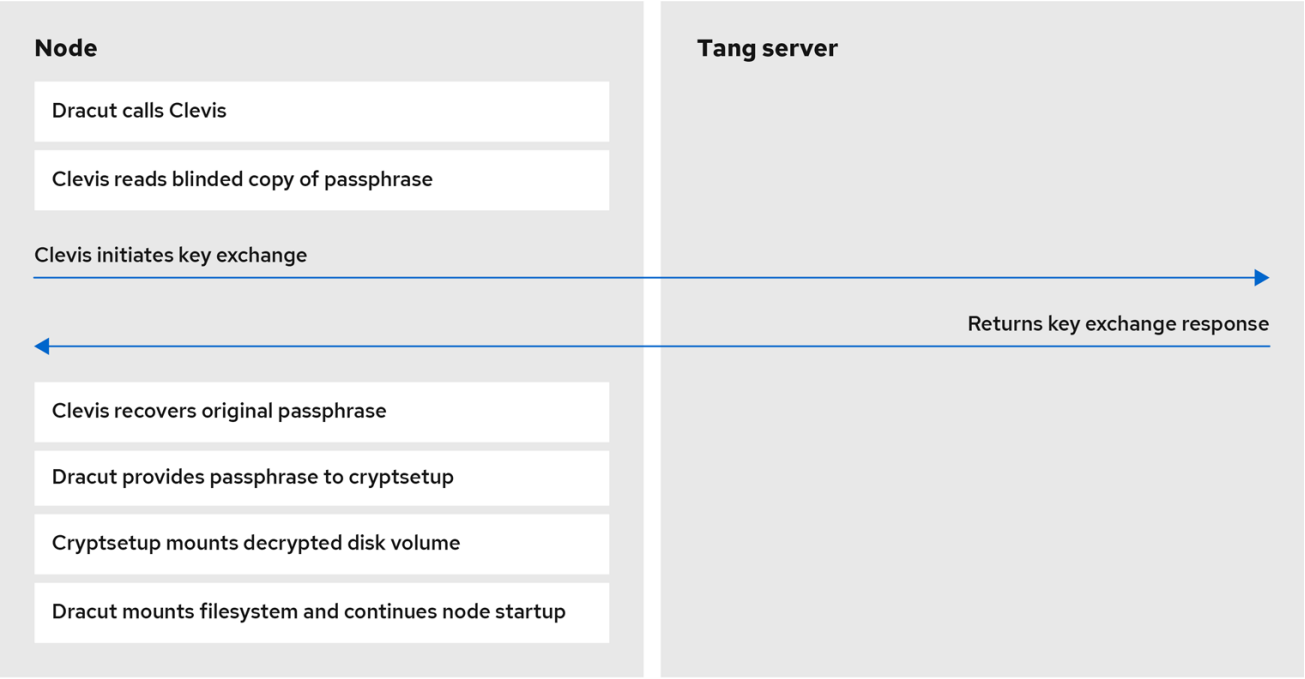
Clevis and Tang are generic client and server components that provide network-bound encryption. Red Hat Enterprise Linux CoreOS (RHCOS) uses these components in conjunction with Linux Unified Key Setup-on-disk-format (LUKS) to encrypt and decrypt root and non-root storage volumes to accomplish Network-Bound Disk Encryption.

When a node starts, it attempts to contact a predefined set of Tang servers by performing a cryptographic handshake. If it can reach the required number of Tang servers, the node can construct its disk decryption key and unlock the disks to continue booting. If the node cannot access a Tang server due to a network outage or server unavailability, the node cannot boot and continues retrying indefinitely until the Tang servers become available again. Because the key is effectively tied to the node's presence in a network, an attacker attempting to gain access to the data at rest would need to obtain both the disks on the node, and network access to the Tang server as well.

The following figure illustrates the deployment model for NBDE.



The following figure illustrates NBDE behavior during a reboot.



17.1.1.4. Secret sharing encryption

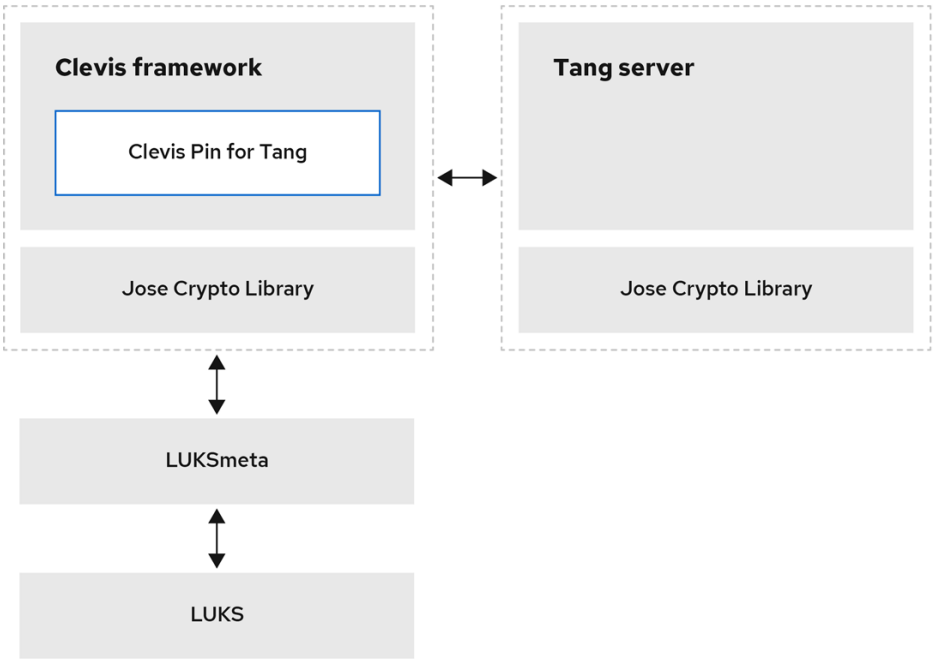
Shamir's secret sharing (sss) is a cryptographic algorithm to securely divide up, distribute, and re-assemble keys. Using this algorithm, OpenShift Container Platform can support more complicated mixtures of key protection.

When you configure a cluster node to use multiple Tang servers, OpenShift Container Platform uses sss to set up a decryption policy that will succeed if at least one of the specified servers is available. You can create layers for additional security. For example, you can define a policy where OpenShift Container Platform requires both the TPM and one of the given list of Tang servers to decrypt the disk.

17.1.2. Tang server disk encryption

The following components and technologies implement Network-Bound Disk Encryption (NBDE).

Figure 17.1. NBDE scheme when using a LUKS1-encrypted volume. The luksmeta package is not used for LUKS2 volumes.



Tang is a server for binding data to network presence. It makes a node containing the data available when the node is bound to a certain secure network. Tang is stateless and does not require Transport Layer Security (TLS) or authentication. Unlike escrow-based solutions, where the key server stores all encryption keys and has knowledge of every encryption key, Tang never interacts with any node keys, so it never gains any identifying information from the node.

Clevis is a pluggable framework for automated decryption that provides automated unlocking of Linux Unified Key Setup-on-disk-format (LUKS) volumes. The Clevis package runs on the node and provides the client side of the feature.

A *Clevis pin* is a plugin into the Clevis framework. There are three pin types:

TPM2

Binds the disk encryption to the TPM2.

Tang

Binds the disk encryption to a Tang server to enable NBDE.

Shamir's secret sharing (sss)

Allows more complex combinations of other pins. It allows more nuanced policies such as the following:

- Must be able to reach one of these three Tang servers
- Must be able to reach three of these five Tang servers
- Must be able to reach the TPM2 AND at least one of these three Tang servers

17.1.3. Tang server location planning

When planning your Tang server environment, consider the physical and network locations of the Tang servers.

Physical location

The geographic location of the Tang servers is relatively unimportant, as long as they are suitably secured from unauthorized access or theft and offer the required availability and accessibility to run a critical service.
Nodes with Clevis clients do not require local Tang servers as long as the Tang servers are available at all times. Disaster recovery requires both redundant power and redundant network connectivity to Tang servers regardless of their location.

Network location

Any node with network access to the Tang servers can decrypt their own disk partitions, or any other disks encrypted by the same Tang servers.
Select network locations for the Tang servers that ensure the presence or absence of network connectivity from a given host allows for permission to decrypt. For example, firewall protections might be in place to prohibit access from any type of guest or public network, or any network jack located in an unsecured area of the building.

Additionally, maintain network segregation between production and development networks. This assists in defining appropriate network locations and adds an additional layer of security.

Do not deploy Tang servers on the same resource, for example, the same **rolebindings.rbac.authorization.k8s.io** cluster, that they are responsible for unlocking. However, a cluster of Tang servers and other security resources can be a useful configuration to enable support of multiple additional clusters and cluster resources.

17.1.4. Tang server sizing requirements

The requirements around availability, network, and physical location drive the decision of how many Tang servers to use, rather than any concern over server capacity.

Tang servers do not maintain the state of data encrypted using Tang resources. Tang servers are either fully independent or share only their key material, which enables them to scale well.

There are two ways Tang servers handle key material:

- Multiple Tang servers share key material:
    - You must load balance Tang servers sharing keys behind the same URL. The configuration can be as simple as round-robin DNS, or you can use physical load balancers.
    - You can scale from a single Tang server to multiple Tang servers. Scaling Tang servers does not require rekeying or client reconfiguration on the node when the Tang servers share key material and the same URL.
    - Client node setup and key rotation only requires one Tang server.
- Multiple Tang servers generate their own key material:
    - You can configure multiple Tang servers at installation time.
    - You can scale an individual Tang server behind a load balancer.
    - All Tang servers must be available during client node setup or key rotation.

- When a client node boots using the default configuration, the Clevis client contacts all Tang servers. Only *n* Tang servers must be online to proceed with decryption. The default value for *n* is 1.
- Red Hat does not support postinstallation configuration that changes the behavior of the Tang servers.

17.1.5. Logging considerations

Centralized logging of Tang traffic is advantageous because it might allow you to detect such things as unexpected decryption requests. For example:

- A node requesting decryption of a passphrase that does not correspond to its boot sequence
- A node requesting decryption outside of a known maintenance activity, such as cycling keys

17.2. TANG SERVER INSTALLATION CONSIDERATIONS

Network-Bound Disk Encryption (NBDE) must be enabled when a cluster node is installed. However, you can change the disk encryption policy at any time after it was initialized at installation.

17.2.1. Installation scenarios

Consider the following recommendations when planning Tang server installations:

- Small environments can use a single set of key material, even when using multiple Tang servers:
  - Key rotations are easier.
  - Tang servers can scale easily to permit high availability.
- Large environments can benefit from multiple sets of key material:
  - Physically diverse installations do not require the copying and synchronizing of key material between geographic regions.
  - Key rotations are more complex in large environments.
  - Node installation and rekeying require network connectivity to all Tang servers.
  - A small increase in network traffic can occur due to a booting node querying all Tang servers during decryption. Note that while only one Clevis client query must succeed, Clevis queries all Tang servers.
- Further complexity:
  - Additional manual reconfiguration can permit the Shamir's secret sharing (sss) of **any N of M servers online** in order to decrypt the disk partition. Decrypting disks in this scenario requires multiple sets of key material, and manual management of Tang servers and nodes with Clevis clients after the initial installation.
- High level recommendations:
  - For a single RAN deployment, a limited set of Tang servers can run in the corresponding domain controller (DC).

- For multiple RAN deployments, you must decide whether to run Tang servers in each corresponding DC or whether a global Tang environment better suits the other needs and requirements of the system.

17.2.2. Installing a Tang server

To deploy one or more Tang servers, you can choose from the following options depending on your scenario:

1. Deploying a Tang server using the NBDE Tang Server Operator
2. Deploying a Tang server with SELinux in enforcing mode on RHEL systems
3. Configuring a Tang server in the RHEL web console
4. Deploying Tang as a container
5. Using the nbde_server System Role for setting up multiple Tang servers

17.2.2.1. Compute requirements

The computational requirements for the Tang server are very low. Any typical server grade configuration that you would use to deploy a server into production can provision sufficient compute capacity.

High availability considerations are solely for availability and not additional compute power to satisfy client demands.

17.2.2.2. Automatic start at boot

Due to the sensitive nature of the key material the Tang server uses, you should keep in mind that the overhead of manual intervention during the Tang server's boot sequence can be beneficial.

By default, if a Tang server starts and does not have key material present in the expected local volume, it will create fresh material and serve it. You can avoid this default behavior by either starting with pre-existing key material or aborting the startup and waiting for manual intervention.

17.2.2.3. HTTP versus HTTPS

Traffic to the Tang server can be encrypted (HTTPS) or plaintext (HTTP). There are no significant security advantages of encrypting this traffic, and leaving it decrypted removes any complexity or failure conditions related to Transport Layer Security (TLS) certificate checking in the node running a Clevis client.

While it is possible to perform passive monitoring of unencrypted traffic between the node's Clevis client and the Tang server, the ability to use this traffic to determine the key material is at best a future theoretical concern. Any such traffic analysis would require large quantities of captured data. Key rotation would immediately invalidate it. Finally, any threat actor able to perform passive monitoring has already obtained the necessary network access to perform manual connections to the Tang server and can perform the simpler manual decryption of captured Clevis headers.

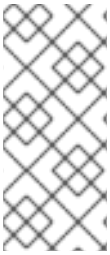
However, because other network policies in place at the installation site might require traffic encryption regardless of application, consider leaving this decision to the cluster administrator.

Additional resources

- Configuring automated unlocking of encrypted volumes using policy-based decryption in the RHEL 8 Security hardening document
- Official Tang server container
- Encrypting and mirroring disks during installation

17.3. TANG SERVER ENCRYPTION KEY MANAGEMENT

The cryptographic mechanism to recreate the encryption key is based on the *blinded key* stored on the node and the private key of the involved Tang servers.



NOTE

To protect against the possibility of an attacker who has obtained both the Tang server private key and the node's encrypted disk, periodic rekeying is advisable.

You must perform the rekeying operation for every node before you can delete the old key from the Tang server.

The following sections provide procedures for rekeying and deleting old keys.

17.3.1. Backing up keys for a Tang server

The Tang server uses **/usr/libexec/tangd-keygen** to generate new keys and stores them in the **/var/db/tang** directory by default. To recover the Tang server in the event of a failure, back up this directory. The keys are sensitive and because they are able to perform the boot disk decryption of all hosts that have used them, the keys must be protected accordingly.

Procedure

- Copy the backup key from the **/var/db/tang** directory to the temp directory from which you can restore the key.

17.3.2. Recovering keys for a Tang server

You can recover the keys for a Tang server by accessing the keys from a backup.

Procedure

- Restore the key from your backup folder to the **/var/db/tang/** directory.
  When the Tang server starts up, it advertises and uses these restored keys.

17.3.3. Rekeying Tang servers

This procedure uses a set of three Tang servers, each with unique keys, as an example.

Using redundant Tang servers reduces the chances of nodes failing to boot automatically.

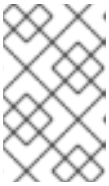
Rekeying a Tang server, and all associated NBDE-encrypted nodes, is a three-step procedure.

Prerequisites

- A working Network-Bound Disk Encryption (NBDE) installation on one or more nodes.

Procedure

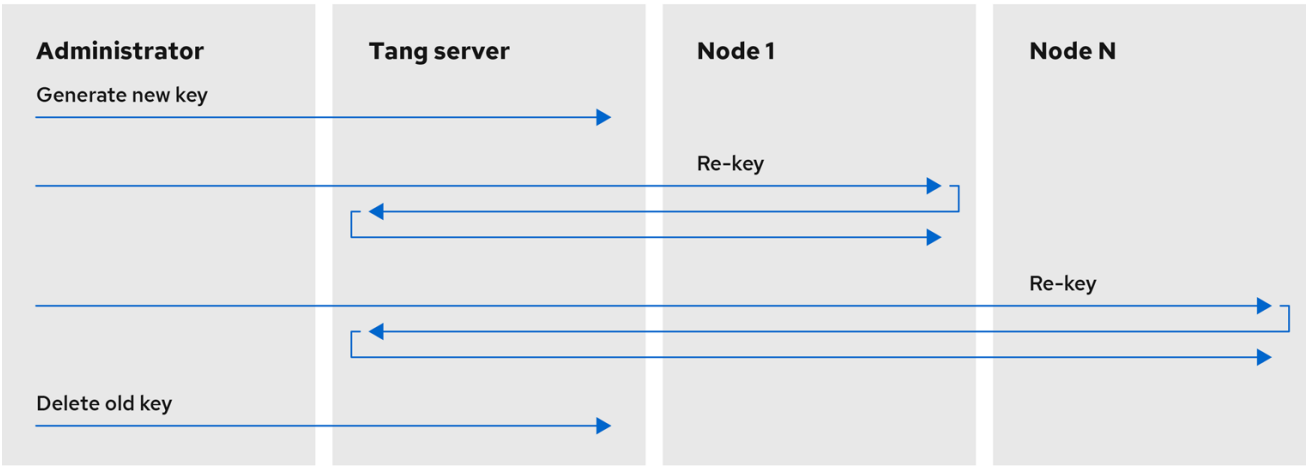
1. Generate a new Tang server key.
2. Rekey all NBDE-encrypted nodes so they use the new key.
3. Delete the old Tang server key.



NOTE

Deleting the old key before all NBDE-encrypted nodes have completed their rekeying causes those nodes to become overly dependent on any other configured Tang servers.

Figure 17.2. Example workflow for rekeying a Tang server



17.3.3.1. Generating a new Tang server key

Prerequisites

- A root shell on the Linux machine running the Tang server.
- To facilitate verification of the Tang server key rotation, encrypt a small test file with the old key:

```
# echo plaintext | clevis encrypt tang '{"url":"http://localhost:7500”}' -y >/tmp/encrypted.oldkey
```

- Verify that the encryption succeeded and the file can be decrypted to produce the same string **plaintext**:

```
# clevis decrypt </tmp/encrypted.oldkey
```

Procedure

1. Locate and access the directory that stores the Tang server key. This is usually the **/var/db/tang** directory. Check the currently advertised key thumbprint:

```
# tang-show-keys 7500
```

Example output

```
36AHjNH3NZDSnlONLz1-V4ie6t8
```

2. Enter the Tang server key directory:

```
# cd /var/db/tang/
```

3. List the current Tang server keys:

```
# ls -A1
```

Example output

```
36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
```

During normal Tang server operations, there are two **.jwk** files in this directory: one for signing and verification, and another for key derivation.

4. Disable advertisement of the old keys:

```
# for key in *.jwk; do \
  mv -- "$key" ".$key"; \
done
```

New clients setting up Network-Bound Disk Encryption (NBDE) or requesting keys will no longer see the old keys. Existing clients can still access and use the old keys until they are deleted. The Tang server reads but does not advertise keys stored in UNIX hidden files, which start with the **.** character.

5. Generate a new key:

```
# /usr/libexec/tangd-keygen /var/db/tang
```

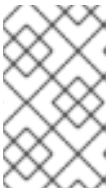
6. List the current Tang server keys to verify the old keys are no longer advertised, as they are now hidden files, and new keys are present:

```
# ls -A1
```

Example output

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

Tang automatically advertises the new keys.



NOTE

More recent Tang server installations include a helper **/usr/libexec/tangd-rotate-keys** directory that takes care of disabling advertisement and generating the new keys simultaneously.

7. If you are running multiple Tang servers behind a load balancer that share the same key material, ensure the changes made here are properly synchronized across the entire set of servers before proceeding.

Verification

1. Verify that the Tang server is advertising the new key, and not advertising the old key:

```
# tang-show-keys 7500
```

Example output

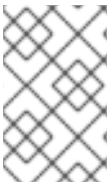
```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. Verify that the old key, while not advertised, is still available to decryption requests:

```
# clevis decrypt </tmp/encrypted.oldkey
```

17.3.3.2. Rekeying all NBDE nodes

You can rekey all of the nodes on a remote cluster by using a **DaemonSet** object without incurring any downtime to the remote cluster.



NOTE

If a node loses power during the rekeying, it is possible that it might become unbootable, and must be redeployed via Red Hat Advanced Cluster Management (RHACM) or a GitOps pipeline.

Prerequisites

- **cluster-admin** access to all clusters with Network-Bound Disk Encryption (NBDE) nodes.
- All Tang servers must be accessible to every NBDE node undergoing rekeying, even if the keys of a Tang server have not changed.
- Obtain the Tang server URL and key thumbprint for every Tang server.

Procedure

1. Create a **DaemonSet** object based on the following template. This template sets up three redundant Tang servers, but can be easily adapted to other situations. Change the Tang server URLs and thumbprints in the **NEW_TANG_PIN** environment to suit your environment:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: tang-rekey
  namespace: openshift-machine-config-operator
spec:
  selector:
    matchLabels:
      name: tang-rekey
  template:
    metadata:
      labels:
        name: tang-rekey
    spec:
      containers:
      - name: tang-rekey
        image: registry.access.redhat.com/ubi9/ubi-minimal:latest
        imagePullPolicy: IfNotPresent
        command:
        - "/sbin/chroot"
        - "/host"
        - "/bin/bash"
        - "-ec"
        args:
        - |
          rm -f /tmp/rekey-complete || true
          echo "Current tang pin:"
          clevis-luks-list -d $ROOT_DEV -s 1
          echo "Applying new tang pin: $NEW_TANG_PIN"
          clevis-luks-edit -f -d $ROOT_DEV -s 1 -c "$NEW_TANG_PIN"
          echo "Pin applied successfully"
          touch /tmp/rekey-complete
          sleep infinity
        readinessProbe:
          exec:
            command:
            - cat
            - /host/tmp/rekey-complete
          initialDelaySeconds: 30
          periodSeconds: 10
        env:
        - name: ROOT_DEV
          value: /dev/disk/by-partlabel/root
        - name: NEW_TANG_PIN
          value: >-
            {"t":1,"pins":{"tang":[
              {"url":"http://tangserver01:7500","thp":"WOjQYkyK7DxY_T5pMncMO5w0f6E"},
              {"url":"http://tangserver02:7500","thp":"I5Ynh2JefoAO3tNH9TgI4obIaXI"},
              {"url":"http://tangserver03:7500","thp":"38qWZVeDKzCPG9pHLqKzs6k1ons"}
            ]}}
        volumeMounts:
        - name: hostroot
          mountPath: /host
        securityContext:
          privileged: true
      volumes:
```

```
      - name: hostroot
        hostPath:
          path: /
      nodeSelector:
        kubernetes.io/os: linux
      priorityClassName: system-node-critical
      restartPolicy: Always
      serviceAccount: machine-config-daemon
      serviceAccountName: machine-config-daemon
```

In this case, even though you are rekeying **tangserver01**, you must specify not only the new thumbprint for **tangserver01**, but also the current thumbprints for all other Tang servers. Failure to specify all thumbprints for a rekeying operation opens up the opportunity for a man-in-the-middle attack.

2. To distribute the daemon set to every cluster that must be rekeyed, run the following command:

```
$ oc apply -f tang-rekey.yaml
```

However, to run at scale, wrap the daemon set in an ACM policy. This ACM configuration must contain one policy to deploy the daemon set, a second policy to check that all the daemon set pods are READY, and a placement rule to apply it to the appropriate set of clusters.

NOTE

After validating that the daemon set has successfully rekeyed all servers, delete the daemon set. If you do not delete the daemon set, it must be deleted before the next rekeying operation.

Verification

After you distribute the daemon set, monitor the daemon sets to ensure that the rekeying has completed successfully. The script in the example daemon set terminates with an error if the rekeying failed, and remains in the **CURRENT** state if successful. There is also a readiness probe that marks the pod as **READY** when the rekeying has completed successfully.

- This is an example of the output listing for the daemon set before the rekeying has completed:

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

Example output

```
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR           AGE
tang-rekey   1         1         0       1            0           kubernetes.io/os=linux   11s
```

- This is an example of the output listing for the daemon set after the rekeying has completed successfully:

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

Example output

```
NAME        DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE
SELECTOR                  AGE
tang-rekey  1        1        1      1           1          kubernetes.io/os=linux  13h
```

Rekeying usually takes a few minutes to complete.

NOTE

If you use ACM policies to distribute the daemon sets to multiple clusters, you must include a compliance policy that checks every daemon set's READY count is equal to the DESIRED count. In this way, compliance to such a policy demonstrates that all daemon set pods are READY and the rekeying has completed successfully. You could also use an ACM search to query all of the daemon sets' states.

17.3.3.3. Troubleshooting temporary rekeying errors for Tang servers

To determine if the error condition from rekeying the Tang servers is temporary, perform the following procedure. Temporary error conditions might include:

- Temporary network outages
- Tang server maintenance

Generally, when these types of temporary error conditions occur, you can wait until the daemon set succeeds in resolving the error or you can delete the daemon set and not try again until the temporary error condition has been resolved.

Procedure

1. Restart the pod that performs the rekeying operation using the normal Kubernetes pod restart policy.
2. If any of the associated Tang servers are unavailable, try rekeying until all the servers are back online.

17.3.3.4. Troubleshooting permanent rekeying errors for Tang servers

If, after rekeying the Tang servers, the **READY** count does not equal the **DESIRED** count after an extended period of time, it might indicate a permanent failure condition. In this case, the following conditions might apply:

- A typographical error in the Tang server URL or thumbprint in the **NEW_TANG_PIN** definition.
- The Tang server is decommissioned or the keys are permanently lost.

Prerequisites

- The commands shown in this procedure can be run on the Tang server or on any Linux system that has network access to the Tang server.

Procedure

1. Validate the Tang server configuration by performing a simple encrypt and decrypt operation on each Tang server's configuration as defined in the daemon set.

This is an example of an encryption and decryption attempt with a bad thumbprint:

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver02:7500","thp":"badthumbprint"}' | \
  clevis decrypt
```

Example output

```
Unable to fetch advertisement: 'http://tangserver02:7500/adv/badthumbprint'!
```

This is an example of an encryption and decryption attempt with a good thumbprint:

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver03:7500","thp":"goodthumbprint"}' | \
  clevis decrypt
```

Example output

```
okay
```

2. After you identify the root cause, remedy the underlying situation:
   a. Delete the non-working daemon set.
   b. Edit the daemon set definition to fix the underlying issue. This might include any of the following actions:
      - Edit a Tang server entry to correct the URL and thumbprint.
      - Remove a Tang server that is no longer in service.
      - Add a new Tang server that is a replacement for a decommissioned server.
3. Distribute the updated daemon set again.

NOTE

When replacing, removing, or adding a Tang server from a configuration, the rekeying operation will succeed as long as at least one original server is still functional, including the server currently being rekeyed. If none of the original Tang servers are functional or can be recovered, recovery of the system is impossible and you must redeploy the affected nodes.

Verification

Check the logs from each pod in the daemon set to determine whether the rekeying completed successfully. If the rekeying is not successful, the logs might indicate the failure condition.

1. Locate the name of the container that was created by the daemon set:

```
$ oc get pods -A | grep tang-rekey
```

Example output

```
-
```

```
openshift-machine-config-operator  tang-rekey-7ks6h  1/1  Running   20 (8m39s ago)  89m
```

2. Print the logs from the container. The following log is from a completed successful rekeying operation:

```
$ oc logs tang-rekey-7ks6h
```

Example output

```
Current tang pin:
1: sss '{"t":1,"pins":{"tang":[{"url":"http://10.46.55.192:7500"},{"url":"http://10.46.55.192:7501"},
{"url":"http://10.46.55.192:7502"}]}}'
Applying new tang pin: {"t":1,"pins":{"tang":[
  {"url":"http://tangserver01:7500","thp":"WOjQYkyK7DxY_T5pMncMO5w0f6E"},
  {"url":"http://tangserver02:7500","thp":"I5Ynh2JefoAO3tNH9TgI4obIaXI"},
  {"url":"http://tangserver03:7500","thp":"38qWZVeDKzCPG9pHLqKzs6k1ons"}
]}}
Updating binding...
Binding edited successfully
Pin applied successfully
```

17.3.4. Deleting old Tang server keys

Prerequisites

- A root shell on the Linux machine running the Tang server.

Procedure

1. Locate and access the directory where the Tang server key is stored. This is usually the **/var/db/tang** directory:

```
# cd /var/db/tang/
```

2. List the current Tang server keys, showing the advertised and unadvertised keys:

```
# ls -A1
```

Example output

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

3. Delete the old keys:

```
# rm .*.jwk
```

4. List the current Tang server keys to verify the unadvertised keys are no longer present:

```
# ls -A1
```

-

Example output

```
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

Verification

At this point, the server still advertises the new keys, but an attempt to decrypt based on the old key will fail.

1. Query the Tang server for the current advertised key thumbprints:

```
# tang-show-keys 7500
```

Example output

```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. Decrypt the test file created earlier to verify decryption against the old keys fails:

```
# clevis decrypt </tmp/encryptValidation
```

Example output

```
Error communicating with the server!
```

If you are running multiple Tang servers behind a load balancer that share the same key material, ensure the changes made are properly synchronized across the entire set of servers before proceeding.

17.4. DISASTER RECOVERY CONSIDERATIONS

This section describes several potential disaster situations and the procedures to respond to each of them. Additional situations will be added here as they are discovered or presumed likely to be possible.

17.4.1. Loss of a client machine

The loss of a cluster node that uses the Tang server to decrypt its disk partition is *not* a disaster. Whether the machine was stolen, suffered hardware failure, or another loss scenario is not important: the disks are encrypted and considered unrecoverable.

However, in the event of theft, a precautionary rotation of the Tang server's keys and rekeying of all remaining nodes would be prudent to ensure the disks remain unrecoverable even in the event the thieves subsequently gain access to the Tang servers.

To recover from this situation, either reinstall or replace the node.

17.4.2. Planning for a loss of client network connectivity

The loss of network connectivity to an individual node will cause it to become unable to boot in an unattended fashion.

If you are planning work that might cause a loss of network connectivity, you can reveal the passphrase for an onsite technician to use manually, and then rotate the keys afterwards to invalidate it:

Procedure

1. Before the network becomes unavailable, show the password used in the first slot **-s 1** of device /**dev**/**vda2** with this command:

```
$ sudo clevis luks pass -d /dev/vda2 -s 1
```

2. Invalidate that value and regenerate a new random boot-time passphrase with this command:

```
$ sudo clevis luks regen -d /dev/vda2 -s 1
```

17.4.3. Unexpected loss of network connectivity

If the network disruption is unexpected and a node reboots, consider the following scenarios:

- If any nodes are still online, ensure that they do not reboot until network connectivity is restored. This is not applicable for single-node clusters.
- The node will remain offline until such time that either network connectivity is restored, or a pre-established passphrase is entered manually at the console. In exceptional circumstances, network administrators might be able to reconfigure network segments to reestablish access, but this is counter to the intent of NBDE, which is that lack of network access means lack of ability to boot.
- The lack of network access at the node can reasonably be expected to impact that node's ability to function as well as its ability to boot. Even if the node were to boot via manual intervention, the lack of network access would make it effectively useless.

17.4.4. Recovering network connectivity manually

A somewhat complex and manually intensive process is also available to the onsite technician for network recovery.

Procedure

1. The onsite technician extracts the Clevis header from the hard disks. Depending on BIOS lockdown, this might involve removing the disks and installing them in a lab machine.
2. The onsite technician transmits the Clevis headers to a colleague with legitimate access to the Tang network who then performs the decryption.
3. Due to the necessity of limited access to the Tang network, the technician should not be able to access that network via VPN or other remote connectivity. Similarly, the technician cannot patch the remote server through to this network in order to decrypt the disks automatically.
4. The technician reinstalls the disk and manually enters the plain text passphrase provided by their colleague.
5. The machine successfully starts even without direct access to the Tang servers. Note that the transmission of the key material from the install site to another site with network access must be done carefully.

6. When network connectivity is restored, the technician rotates the encryption keys.

17.4.5. Emergency recovery of network connectivity

If you are unable to recover network connectivity manually, consider the following steps. Be aware that these steps are discouraged if other methods to recover network connectivity are available.

- This method must only be performed by a highly trusted technician.
- Taking the Tang server's key material to the remote site is considered to be a breach of the key material and all servers must be rekeyed and re-encrypted.
- This method must be used in extreme cases only, or as a proof of concept recovery method to demonstrate its viability.
- Equally extreme, but theoretically possible, is to power the server in question with an Uninterruptible Power Supply (UPS), transport the server to a location with network connectivity to boot and decrypt the disks, and then restore the server at the original location on battery power to continue operation.
- If you want to use a backup manual passphrase, you must create it before the failure situation occurs.
- Just as attack scenarios become more complex with TPM and Tang compared to a stand-alone Tang installation, so emergency disaster recovery processes are also made more complex if leveraging the same method.

17.4.6. Loss of a network segment

The loss of a network segment, making a Tang server temporarily unavailable, has the following consequences:

- OpenShift Container Platform nodes continue to boot as normal, provided other servers are available.
- New nodes cannot establish their encryption keys until the network segment is restored. In this case, ensure connectivity to remote geographic locations for the purposes of high availability and redundancy. This is because when you are installing a new node or rekeying an existing node, all of the Tang servers you are referencing in that operation must be available.

A hybrid model for a vastly diverse network, such as five geographic regions in which each client is connected to the closest three clients is worth investigating.

In this scenario, new clients are able to establish their encryption keys with the subset of servers that are reachable. For example, in the set of **tang1**, **tang2** and **tang3** servers, if **tang2** becomes unreachable clients can still establish their encryption keys with **tang1** and **tang3**, and at a later time re-establish with the full set. This can involve either a manual intervention or a more complex automation to be available.

17.4.7. Loss of a Tang server

The loss of an individual Tang server within a load balanced set of servers with identical key material is completely transparent to the clients.

The temporary failure of all Tang servers associated with the same URL, that is, the entire load balanced set, can be considered the same as the loss of a network segment. Existing clients have the ability to decrypt their disk partitions so long as another preconfigured Tang server is available. New clients

cannot enroll until at least one of these servers comes back online.

You can mitigate the physical loss of a Tang server by either reinstalling the server or restoring the server from backups. Ensure that the backup and restore processes of the key material is adequately protected from unauthorized access.

17.4.8. Rekeying compromised key material

If key material is potentially exposed to unauthorized third parties, such as through the physical theft of a Tang server or associated data, immediately rotate the keys.

Procedure

1. Rekey any Tang server holding the affected material.
2. Rekey all clients using the Tang server.
3. Destroy the original key material.
4. Scrutinize any incidents that result in unintended exposure of the master encryption key. If possible, take compromised nodes offline and re-encrypt their disks.

TIP

Reformatting and reinstalling on the same physical hardware, although slow, is easy to automate and test.