OpenShift Container Platform 4.16

安全性与合规性

了解和管理 OpenShift Container Platform 的安全性

Last Updated: 2024-07-03

OpenShift Container Platform 4.16 安全性与合规性

了解和管理 OpenShift Container Platform 的安全性

法律通告

摘要

本文档讨论容器安全性、配置证书和启用加密以便帮助保护集群的安全。

目录

第 1 章 OPENSHIFT CONTAINER PLATFORM 安全和合规性

1.1. 安全概述

了解如何正确保护 OpenShift Container Platform 集群的各个方面非常重要。

容器安全性
了解 OpenShift Container Platform 安全性的良好起点是回顾了解容器安全性中的概念。本节和接下来的章节介绍了 OpenShift Container Platform 中提供的容器安全措施，包括主机层、容器和编配层以及构建和应用程序层的解决方案。这些部分还包括有关以下主题的信息：

- 为什么容器安全性很重要，以及它与现有安全标准相比较的情况。
- 哪些容器安全措施是由主机（RHCOS 和 RHEL）层提供的，哪些是由 OpenShift Container Platform 提供的。
- 如何评估您的容器内容和漏洞来源。
- 如何设计您的构建和部署过程，以主动检查容器的内容。
- 如何通过身份验证和授权控制对容器的访问。
- 如何在 OpenShift Container Platform 中保护网络和附加存储。
- 用于 API 管理和 SSO 的容器化解决方案。

Auditing
OpenShift Container Platform auditing（审计）提供一组安全相关的按时间排序的记录，记录各个用户、管理员或其他系统组件影响系统的一系列活动。管理员可以配置审计日志策略，并查看审计日志。

证书
证书供各种组件用于验证对集群的访问。管理员可以替换默认入口证书、添加 API 服务器证书或添加服务证书。

您还可以查看集群使用的证书类型的更多详情：

- API 服务器的用户提供的证书
- 代理证书
- 服务 CA 证书
- 节点证书
- Bootstrap 证书
- etcd 证书
- olm 证书
- 聚合的 API 客户端证书
- Machine Config Operator 证书
- 用户提供的默认入口证书

- 入口证书（Ingress certificate）
- 监控和集群日志记录 Operator 组件证书
- Control plane 证书

加密数据
您可以为集群启用 etcd 加密以提供额外的数据安全层。例如，如果 etcd 备份暴露给不应该获得这个数据的人员，它会帮助保护敏感数据。

漏洞扫描
管理员可以使用 Red Hat Quay Container Security Operator 运行 vulnerability scans 并查看有关检测到的漏洞的信息。

1.2. 合规性概述

对于许多 OpenShift Container Platform 客户，在将任何系统投入生产前需要达到一定级别的法规就绪状态或合规性。这种法规就绪性可通过国家标准、行业标准或组织的企业管理框架来实施。

合规性检查
管理员可以使用 Compliance Operator 运行合规性扫描，并为发现的问题推荐补救。**oc-compliance 插件** 是一个 OpenShift CLI (**oc**) 插件，提供一组实用程序来轻松地与 Compliance Operator 交互。

文件完整性检查
管理员可以使用 File Integrity Operator 在集群节点上持续运行文件完整性检查，并提供已修改的文件日志。

1.3. 其他资源

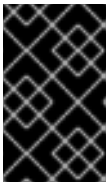
- 了解身份验证
- 配置内部 OAuth 服务器
- 了解身份提供程序配置
- 使用 RBAC 定义和应用权限
- 管理安全性上下文约束

第 2 章 容器安全性

2.1. 了解容器安全性

保护容器化应用程序需要依赖于多个级别的安全性：

- 容器安全性从可信基础容器镜像开始，一直到经过您的 CI/CD 管道的容器构建过程。



重要

默认情况下，镜像流不会自动更新。这个默认行为可能会造成安全问题，因为镜像流引用的镜像的安全更新不会自动进行。有关如何覆盖此默认行为的详情，请参阅配置定期导入 imagestreamtag。

- 部署容器时，其安全性取决于它运行在安全的操作系统和网络上，并在容器本身和与之交互的用户和主机之间建立明确界限。
- 持续安全性取决于能够扫描容器镜像以获取漏洞，并具有高效的方法来更正和替换有漏洞的镜像。

除了 OpenShift Container Platform 等平台提供的开箱即用的功能外，您的机构可能会有自己的安全需求。在将 OpenShift Container Platform 放入数据中心之前，可能需要进行一定程度的合规性验证。

同样，您可能需要将自己的代理、专用硬件驱动程序或加密功能添加到 OpenShift Container Platform 中，才能满足您机构的安全标准。

本指南全面介绍了 OpenShift Container Platform 中提供的容器安全措施，包括主机层、容器和编配层以及构建和应用程序层的解决方案。然后，它会指引您参考特定的 OpenShift Container Platform 文档以帮助您实现这些安全措施。

本指南包含以下信息：

- 为什么容器安全性很重要，以及它与现有安全标准相比较的情况。
- 哪些容器安全措施是由主机（RHCOS 和 RHEL）层提供的，哪些是由 OpenShift Container Platform 提供的。
- 如何评估您的容器内容和漏洞来源。
- 如何设计您的构建和部署过程，以主动检查容器的内容。
- 如何通过身份验证和授权控制对容器的访问。
- 如何在 OpenShift Container Platform 中保护网络和附加存储。
- 用于 API 管理和 SSO 的容器化解决方案。

本指南的目的是了解将 OpenShift Container Platform 用于容器化工作负载的极大安全优势，以及整个红帽生态系统在提供和保持容器安全方面发挥的作用。它还将帮助您了解如何与 OpenShift Container Platform 互动以实现您机构的安全目标。

2.1.1. 什么是容器？

容器将一个应用程序及其所有依赖项打包成单一镜像，可在不发生改变的情况下从开发环境提升到测试环境，再提升到生产环境。容器可能是大型应用程序的一部分，与其他容器紧密合作。

容器提供不同环境间的一致性和多个部署目标：物理服务器、虚拟机 (VM) 和私有或公有云。

使用容器的一些好处包括：

基础架构	应用程序
在共享的 Linux 操作系统内核上将应用程序进程沙盒化	将我的应用程序及其所有依赖项打包
与虚拟机相比更简单、更轻便且密度更高	部署到任意环境只需几秒并启用 CI/CD
可在不同环境间移植	轻松访问和共享容器化组件

请参阅红帽客户门户中的了解 Linux 容器以查找更多有关 Linux 容器的信息。如需了解有关 RHEL 容器工具的信息，请参阅 RHEL 产品文档中的构建、运行和管理容器。

2.1.2. OpenShift Container Platform 是什么？

OpenShift Container Platform 等平台的任务是自动化部署、运行和管理容器化应用程序。作为核心功能，OpenShift Container Platform 依赖于 Kubernetes 项目来提供在可扩展数据中心跨许多节点编配容器的引擎。

Kubernetes 是一个项目，可使用不同的操作系统和附加组件来运行，它们不提供项目的支持性保证。因此，不同 Kubernetes 平台的安全性可能会有所不同。

OpenShift Container Platform 旨在锁定 Kubernetes 安全性，并将平台与各种扩展组件集成。为实现这一目标，OpenShift Container Platform 利用了广泛的红帽开源技术生态系统，包括操作系统、身份验证、存储、网络、开发工具、基础容器镜像和其他许多组件。

OpenShift Container Platform 可以利用红帽的经验，发现平台本身以及在平台上运行的容器化应用程序中存在的漏洞并快速部署修复程序。红帽的经验还涉及到在新组件可用后高效地将它们与 OpenShift Container Platform 集成，以及根据各个客户的需求对技术进行调整。

其他资源

- OpenShift Container Platform 架构
- OpenShift 安全性指南

2.2. 了解主机和虚拟机安全性

容器和虚拟机都提供了将主机上运行的应用程序与操作系统分开的方法。了解 OpenShift Container Platform 使用的 RHCOS 操作系统将帮助您理解主机系统如何保护容器和主机不受彼此影响。

2.2.1. 保护 Red Hat Enterprise Linux CoreOS (RHCOS) 上的容器

容器简化了将许多应用程序部署在同一主机上运行的操作，每个容器启动都使用相同的内核和容器运行时。应用程序可以归很多用户所有，并且由于是保持独立的，他们可以毫无问题地同时运行这些应用程序的不同版本甚至不兼容的版本。

在 Linux 中，容器只是一种特殊的进程，因此在很多方面，保护容器与保护任何其他运行的进程类似。运行容器的环境始于操作系统，它可以保护主机内核不受容器和主机上运行的其他进程的影响，同时还可以保护容器不受彼此的影响。

由于 OpenShift Container Platform 4.16 在 RHCOS 主机上运行，并可以选择使用 Red Hat Enterprise Linux (RHEL) 作为 worker 节点，因此以下概念将默认应用于任何已部署的 OpenShift Container Platform 集群。这些 RHEL 安全功能是确保在 OpenShift Container Platform 中运行容器的核心所在：

- *Linux 命名空间*支持创建特定全局系统资源的抽象集，使其显示为一个实例，独立于命名空间中的进程。因此，几个容器可以同时使用相同的计算资源，而不会产生冲突。默认情况下独立于主机的容器命名空间包括挂载表、进程表、网络接口、用户、控制组、UTS 和 IPC 命名空间。需要直接访问主机命名空间的容器需要具有升级权限才能请求该访问权限。如需了解有关命名空间类型的详细信息，请参阅 RHEL 8 容器文档中的红帽系统中的容器概述。

- *SELinux* 提供了额外一层安全性，可以使容器相互隔离并与主机隔离。SELinux 允许管理员为每个用户、应用程序、进程和文件实行强制访问控制 (MAC)。

警告

不支持在 RHCOS 上禁用 SELinux。

- *CGroups*（控制组）限制、说明和隔离一组进程的资源用量（CPU、内存、磁盘 I/O、网络等等）。CGroups 用于确保同一主机上的容器不会相互影响。

- *安全计算模式 (seccomp)* 配置集可以与容器关联来限制可用的系统调用。有关 seccomp 的详情，请参阅 OpenShift 安全性指南的第 94 页。

- 使用 *RHCOS* 部署容器可最大程度缩小主机环境并根据容器进行调整，从而减少攻击面。CRI-O 容器引擎只会实现 Kubernetes 和 OpenShift Container Platform 运行和管理容器所需的功能，而不像其他容器引擎一样实现面向桌面的独立功能，因此进一步减少了这一攻击面。

RHCOS 是 Red Hat Enterprise Linux (RHEL) 的一个版本，它经过特别配置，可用作 OpenShift Container Platform 集群上的 control plane (master) 和 worker 节点。因此，RHCOS 被调优为高效地运行容器工作负载，以及 Kubernetes 和 OpenShift Container Platform 服务。

为了进一步保护 OpenShift Container Platform 集群中的 RHCOS 系统，大多数容器（除了管理或监控主机系统本身的容器外）都应以非 root 用户身份运行。要保护您自己的 OpenShift Container Platform 集群，推荐的最佳实践是降低权限级别或创建包含最少权限的容器。

其他资源

- 节点如何强制实施资源限制
- 管理安全性上下文约束
- OpenShift 集群支持的平台
- 使用用户置备基础架构的集群的要求
- 选择如何配置 RHCOS
- Ignition

- 内核参数
- 内核模块
- 磁盘加密
- Chrony 时间服务
- 关于 OpenShift Update 服务
- FIPS 加密

2.2.2. 虚拟化和容器比较

传统虚拟化提供了另一种在相同物理主机上将应用程序环境保持独立的方法。但是，虚拟机的工作方式与容器不同。虚拟化依赖于虚拟机监控程序启动虚拟客户机 (VM)，每个虚拟机都有自己的操作系统 (OS)，具体表现为运行的内核、正在运行的应用程序及其依赖项。

使用 VM，虚拟机监控程序会将虚拟客户机相互隔离并与主机内核隔离。这样可减少可访问虚拟机监控程序的个人和进程，进而缩小物理服务器上的攻击面。尽管如此，仍然必须对安全性进行监控：一个虚拟客户机可能利用虚拟机监控程序的错误来获取对另一个虚拟机或主机内核的访问权限。当需要修补操作系统时，必须对所有使用该操作系统的虚拟客户机进行修补。

容器可在虚拟客户机中运行，这种方式在有些用例中可能是可取的。例如，您可能在容器中部署传统应用程序，以便将某个应用程序转移到云端。

但是，在单一主机上进行容器分离提供了一种更加轻便、灵活且易于部署的解决方案。这种部署模型特别适合云原生应用程序。容器通常比 VM 小得多，消耗的内存和 CPU 也更少。

请参阅 RHEL 7 容器文档中的 Linux 容器与 KVM 虚拟化的比较，以了解容器和虚拟机之间的差别。

2.2.3. 保护 OpenShift Container Platform

在部署 OpenShift Container Platform 时，您可以选择安装程序置备的基础架构（有多个可用的平台）或您自己的用户置备的基础架构。用户置备的基础架构可能有益于一些低级别的与安全相关的配置，如启用 FIPS 合规性或在第一次引导时添加内核模块。同样，用户置备的基础架构适合用于断开连接的 OpenShift Container Platform 部署。

请记住，在为 OpenShift Container Platform 进行安全增强和其他配置更改方面，应包括以下目标：

- 尽可能保持底层节点的通用性。您需要能够快速、轻松地以指定的方式丢弃和启动类似的节点。
- 尽可能通过 OpenShift Container Platform 管理对节点的修改，而不是对节点进行直接的一次性更改。

为实现这些目标，应该在安装过程中使用 Machine Config Operator 应用到各组节点的 MachineConfig，通过 Ignition 或更高版本进行大多数节点更改。您可以通过这种方式进行的与安全性相关的配置更改示例包括：

- 添加内核参数
- 添加内核模块
- 启用 FIPS 加密支持
- 配置磁盘加密

- 配置 chrony 时间服务

除了 Machine Config Operator 外，还有一些其他的 Operator 可用来配置由 Cluster Version Operator (CVO) 管理的 OpenShift Container Platform 基础架构。CVO 可以为 OpenShift Container Platform 集群更新的很多方面实现自动化。

其他资源

- FIPS 加密

2.3. 强化 RHCOS

RHCOS 在创建后经过调整以部署到 OpenShift Container Platform 中，只需对 RHCOS 节点进行很少更改甚至无需更改。每个采用 OpenShift Container Platform 的机构都对系统加强有自己的要求。作为一个 RHEL 系统，RHCOS 中添加了针对 OpenShift 的修改和功能（如 Ignition、ostree 和一个只读 **/usr**，用来提供有限的不可变性），像任何 RHEL 系统一样，可以对它进行强化。不同之处在于您管理强化的方式。

OpenShift Container Platform 及其 Kubernetes 引擎的一个主要功能就是根据需要迅速缩放应用程序和基础架构。除非不可避免，否则您不需要通过登录到主机并添加软件或更改设置来直接更改 RHCOS。您需要让 OpenShift Container Platform 安装程序和 control plane 管理对 RHCOS 的更改，以便可以在不手动干预的情况下启动新节点。

因此，如果您准备在 OpenShift Container Platform 中强化 RHCOS 节点来满足安全需求，您应该同时考虑要强化的功能以及如何着手进行这种强化。

2.3.1. 在 RHCOS 中选择要强化的功能

RHEL 8 安全强化 指南介绍了如何处理任何 RHEL 系统的安全问题。

使用本指南来学习如何处理加密、评估漏洞以及评估不同服务受到的威胁。同样，您可以了解如何扫描合规标准、检查文件完整性、执行审核以及对存储设备进行加密。

了解了您要强化的功能后，您可以决定如何在 RHCOS 中强化它们。

2.3.2. 选择如何强化 RHCOS

不建议在 OpenShift Container Platform 中直接修改 RHCOS 系统。取而代之，您应该考虑在节点池中修改系统，如 worker 节点和 control plane 节点。在非裸机安装中，当需要一个新节点时，您可以请求一个您所需的类型的新节点，并且它将从 RHCOS 镜像创建，再加上之前创建的修改。

在安装前、在安装过程中以及集群启动和运行后，您有机会修改 RHCOS。

2.3.2.1. 安装前强化

对于裸机安装，您可以在开始 OpenShift Container Platform 安装前为 RHCOS 添加强化功能。例如，您可以在引导 RHCOS 安装程序时添加内核选项来开启或关闭安全功能，如各种 SELinux 布尔值或低级别设置，如对称多线程。

警告

不支持在 RHCOS 节点上禁用 SELinux。

虽然裸机 RHCOS 安装难度更大，但有机会在开始 OpenShift Container Platform 安装前完成操作系统更改。如果您需要确保尽早设置某些功能，比如磁盘加密或者特殊联网设置，这一点就很重要。

2.3.2.2. 安装过程中强化

您可以中断 OpenShift Container Platform 安装过程并更改 Ignition 配置。通过 Ignition 配置，您可以将自己的文件和 systemd 服务添加到 RHCOS 节点中。您还可以对用于安装的 **install-config.yaml** 文件进行一些基本的安全相关更改。以这种方式添加的内容将在每个节点第一次引导时可用。

2.3.2.3. 集群运行后强化

在 OpenShift Container Platform 集群启动并运行后，有几种方法可用来将强化功能应用到 RHCOS：

- 守护进程集：如果您需要在每个节点上运行某个服务，您可以使用 Kubernetes 的 **DaemonSet** 对象添加该服务。
- 机器配置：**MachineConfig** 对象包含 Ignition 配置的子集，其格式相同。通过将机器配置应用到所有 worker 或 control plane 节点，您可以确保添加到集群中的同类型的下一个节点会应用相同的更改。

这里提到的所有功能在 OpenShift Container Platform 产品文档中都有介绍。

其他资源

- OpenShift 安全性指南
- 选择如何配置 RHCOS
- 修改节点
- 手动创建安装配置文件
- 创建 Kubernetes 清单和 Ignition 配置文件
- 使用 ISO 镜像安装 RHCOS
- 自定义节点
- 为节点添加内核参数
- 可选的配置参数
- 支持 FIPS 加密
- RHEL 核心加密组件

2.4. 容器镜像签名

红帽为 Red Hat Container Registries 中的镜像提供签名。在使用 Machine Config Operator（MCO）拉取到 OpenShift Container Platform 4 集群时，会自动验证这些签名。

Quay.io 提供了组成 OpenShift Container Platform 的大多数镜像，只有发行镜像会被签名。发行镜像指的是批准的 OpenShift Container Platform 镜像，它可以对供应链攻击提供一定程度的保护。但是，OpenShift Container Platform 的一些扩展（如日志记录、监控和服务网格）会作为 Operator Lifecycle Manager（OLM）的 Operator 提供。这些镜像来自 红帽生态系统目录容器镜像 registry。

要验证这些镜像在红帽 registry 和您的基础架构间的完整性，启用签名验证。

2.4.1. 为 Red Hat Container registry 启用签名验证

为 Red Hat Container Registries 启用容器签名验证需要编写签名验证策略文件，指定从这些 registry 中验证镜像的密钥。对于 RHEL8 节点，默认已在 **/etc/containers/registries.d** 中定义 registry。

流程

1. 创建 Butane 配置文件 **51-worker-rh-registry-trust.bu**，其中包含 worker 节点的必要配置。

注意

如需有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

```
variant: openshift
version: 4.16.0
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. 使用 Butane 生成机器配置 YAML 文件 **51-worker-rh-registry-trust.yaml**，其中包含要写入 worker 节点上的磁盘的文件：

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

3. 应用创建的机器配置：

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

4. 检查 worker 机器配置池已使用新机器配置推出：

   a. 检查是否创建了新机器配置：

```
$ oc get mc
```

输**出示例**

```
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) 新机器配置

(2) 新的渲染机器配置

b. 检查 worker 机器配置池是否使用新机器配置更新：

```
$ oc get mcp
```

输**出示例**

```
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) 当 **UPDATING** 字段为 **True** 时，机器配置池会使用新机器配置进行更新。当字段变为 **False** 时，代表 worker 机器配置池已应用到新机器配置。

5. 如果您的集群使用任何 RHEL7 worker 节点，当 worker 机器配置池被更新时，在 **/etc/containers/registries.d** 目录中在这些节点上创建 YAML 文件，用于指定给定 registry 服务器的分离签名的位置。以下示例只适用于托管在 **registry.access.redhat.com** 和 **registry.redhat.io** 中的镜像。

   a. 为每个 RHEL7 worker 节点启动一个 debug 会话：

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

   b. 将您的根目录改为 **/host** ：

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

   c. 创建一个包含以下内容的 **/etc/containers/registries.d/registry.redhat.io.yaml** 文件：

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

d. 创建一个包含以下内容的 **/etc/containers/registries.d/registry.access.redhat.com.yaml** 文件：

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

e. 退出 debug 会话。

2.4.2. 验证签名验证配置

将机器配置应用到集群后，Machine Config Controller 会检测到新的 **MachineConfig** 对象，并生成新的 **rendered-worker-<hash>** 版本。

先决条件

- 您可以使用机器配置文件启用签名验证。

流程

1. 在命令行中运行以下命令显示所需 worker 的信息：

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

初始 worker 监控的输出示例

```
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. 再次运行 **oc describe** 命令：

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

worker 更新后的输出示例

```
...
    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
...
```

注意

Observed Generation 参数显示基于控制器生成的配置的生成数量的增加数。此控制器即使没有处理规格并生成修订，也会更新这个值。**Configuration Source** 值指向 **51-worker-rh-registry-trust** 配置。

3. 使用以下命令确认 **policy.json** 文件已存在：

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

输**出示例**

```
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. 使用以下命令确认 **registry.redhat.io.yaml** 文件已存在：

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

输出示例

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

5. 使用以下命令确认 **registry.access.redhat.com.yaml** 文件已存在：

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

输出示例

```
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. 了解缺少可验证的容器镜像的验证

每个 OpenShift Container Platform 发行镜像都是不可变的，并使用红帽产品密钥签名。在 OpenShift Container Platform 更新或安装过程中，发行镜像可能会部署没有可验证签名的容器镜像。每个签名的发行镜像摘要都是不可变的。发行镜像中的每个引用都是另一个镜像的不可变摘要，因此内容可以被信任。换句话说，发行镜像中的签名会验证所有发行内容。

例如，没有可验证的签名的镜像引用包含在签名的 OpenShift Container Platform 发行镜像中：

发行信息输出示例

```
$ 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. 签名的发行镜像 SHA。
2. 容器镜像缺少发行版本中包含的可验证签名。

2.4.3.1. 更新过程中自动验证

签名验证是自动的。OpenShift Cluster Version Operator (CVO) 在 OpenShift Container Platform 更新过程中会在发行镜像上验证签名。这是一个内部过程。如果自动验证失败，OpenShift Container Platform 安装会失败。

也可以使用 **skopeo** 命令行工具手动验证签名。

其他资源

- OpenShift 更新简介

2.4.3.2. 使用 skopeo 验证红帽容器镜像的签名

您可以通过从 OCP 发行镜像站点拉取这些签名来验证 OpenShift Container Platform 发行镜像中包含的容器镜像的签名。因为镜像站点上的签名不是 Podman 或 CRI-O 理解的格式，所以您可以使用 **skopeo standalone-verify** 命令来验证您的发行镜像是否由红帽签名。

先决条件

- 已安装 **skopeo** 命令行工具。

流程

1. 运行以下命令，获取您的发行版本的完整 SHA：

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

1 将 <release_version> 替换为您的版本号，如 **4.14.3**。

输出片断示例

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

2. 运行以下命令关闭红帽发行密钥：

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

3. 运行以下命令，获取您要验证的特定发行版本的签名文件：

```
$ 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 将 **<sha_from_version>** 替换为与发行版本 SHA 匹配的镜像站点的完整链接中的 SHA 值。例如，到 4.12.23 发行版本的签名链接为 **https://mirror.openshift.com/pub/openshift-v4/signatures/openshift-release-dev/ocp-release/sha256%e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55/signature-1**，SHA 值为 **e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55**。

4. 运行以下命令，获取发行镜像的清单：

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

1 将 **<quay_link_to_release>** 替换为 **oc adm release info** 命令的输出。例如：**quay.io/openshift-release-dev/ocp-release@sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18**

ec55。

5. 使用 skopeo 验证签名：

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

其中：

<release_number>

指定发行号，如 **4.14.3**。

<arch>

指定架构，如 **x86_64**。

输**出示例**

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

2.4.4. 其他资源

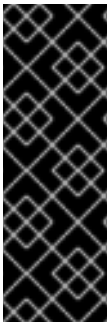
- 机器配置概述

2.5. 了解合规性

对于许多 OpenShift Container Platform 客户，在将任何系统投入生产前需要达到一定级别的法规就绪状态或合规性。这种法规就绪状态可通过国家标准、行业标准或机构的企业监管框架来施加。

2.5.1. 了解合规性及风险管理

FIPS 合规性是高安全性环境中所需的最重要的组件之一，可确保节点上只允许使用支持的加密技术。



重要

要为集群启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL) 计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。

当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 x86_64、ppc64le 和 s390x 架构上提交到 NIST FIPS 140-2/140-3 Validation。

要了解红帽对 OpenShift Container Platform 合规框架的观点，请参阅 OpenShift 安全性指南手册中的“风险管理和法规就绪状态”一章。

其他资源

- 在 FIPS 模式下安装集群

2.6. 保护容器内容

要确保容器内所含内容的安全性，需要以可信的基础镜像（如红帽通用基础镜像）开始，并添加可信软件。为了检查容器镜像的持续安全性，红帽及第三方都有可用于扫描镜像的工具。

2.6.1. 确保容器内安全

应用程序和基础架构由随时可用的组件组成，许多组件都是开源软件包，如 Linux 操作系统、JBoss Web Server、PostgreSQL 和 Node.js。

这些软件包也有容器化版本可用。然而，您需要知道软件包最初来自哪里，使用什么版本，是谁构建的，以及软件包内是否有恶意代码。

需要回答的一些问题包括：

- 容器内的内容是否会破坏您的基础架构？
- 应用程序层是否存在已知的漏洞？
- 运行时和操作系统层是不是最新的？

通过从红帽通用基础镜像 (UBI) 构建容器，您可以保证您的容器镜像基础由 Red Hat Enterprise Linux 中包含的同一 RPM 打包软件组成。使用或重新分发 UBI 镜像不需要订阅。

为确保容器本身持续安全，安全扫描功能（直接从 RHEL 使用或添加到 OpenShift Container Platform）可在您使用的镜像有漏洞时发出警告。RHEL 中提供了 OpenSCAP 镜像扫描，并且可添加 Red Hat Quay Container Security Operator 来检查 OpenShift Container Platform 中使用的容器镜像。

2.6.2. 使用 UBI 创建可重新分发的镜像

要创建容器化应用程序，您通常以可信基础镜像开始，该镜像提供的组件通常由操作系统提供。这些组件包括库、实用程序以及应用程序在操作系统文件系统中应该看到的其他功能。

创建红帽通用基础镜像 (UBI) 是为了鼓励任何人在构建其自己的容器时都先使用一个完全由 Red Hat Enterprise Linux rpm 软件包及其他内容组成的容器镜像。这些 UBI 镜像会定期更新，以应用最新的安全补丁，并可自由地与构建用来包含您自己的软件的容器镜像一起使用和重新分发。

搜索红帽生态系统目录，以便查找和检查不同 UBI 镜像的健康状态。作为创建安全容器镜像的人员，您可能对两种通用 UBI 镜像类型感兴趣：

- **UBI** ：RHEL 7、8 和 9 有标准的 UBI 镜像(**ubi7/ubi**、**ubi8/ubi** 和 **ubi9/ubi**)，以及基于这些系统 (**ubi7/ubi-minimal**、**ubi8/ubi-mimimal**) 和 ubi9/ubi-minimal 的镜像的最小镜像。所有这些镜像已预先配置，以指向您可以使用标准 **yum** 和 **dnf** 命令添加到构建的容器镜像中的免费 RHEL 软件存储库。红帽鼓励人们在其他发行版（如 Fedora 和 Ubuntu）上使用这些镜像。
- **Red Hat Software Collections**：在红帽生态系统目录中搜索 **rhscl/** 以查找为用作特定应用程序类型的基础镜像而创建的镜像。例如，有 Apache httpd (**rhscl/httpd-***)、Python (**rhscl/python-***), Ruby (**rhscl/ruby-***)、Node.js (**rhscl/nodejs-***) 和 Perl (**rhscl/perl-***) rhscl 镜像。

请记住，虽然 UBI 镜像可自由使用且可重新发布，但红帽对这些镜像的支持只能通过 Red Hat 产品订阅获得。

请参阅 Red Hat Enterprise Linux 文档中 的使用红帽通用基础镜像来获得有关如何使用标准、最小和 init UBI 镜像作为构建基础的信息。

2.6.3. RHEL 中的安全扫描

对于 Red Hat Enterprise Linux (RHEL) 系统，可从 **openscap-utils** 软件包中获得 OpenSCAP 扫描功能。在 RHEL 中，您可以使用 **openscap-podman** 命令扫描针镜像中的漏洞。请参阅 Red Hat Enterprise Linux 文档中的扫描容器和容器镜像中的漏洞。

OpenShift Container Platform 可让您在 CI/CD 过程中利用 RHEL 扫描程序。例如，您可以集成静态代码分析工具来测试源代码中的安全漏洞，并集成软件组成分析工具来标识开源库，以提供关于这些库的元数据，如已知漏洞。

2.6.3.1. 扫描 OpenShift 镜像

对于在 OpenShift Container Platform 中运行并且从 Red Hat Quay registry 中拉取的容器镜像，您可以使用 Operator 来列出这些镜像的漏洞。Red Hat Quay Container Security Operator 可以添加到 OpenShift Container Platform 中，为添加到所选命名空间的镜像提供漏洞报告。

Red Hat Quay 的容器镜像扫描由 Clair 执行。在 Red Hat Quay 中，Clair 可以搜索和报告从 RHEL、CentOS、Oracle、Alpine、Debian 和 Ubuntu 操作系统软件构建的镜像中的漏洞。

2.6.4. 集成外部扫描

OpenShift Container Platform 使用对象注解来扩展功能。外部工具（如漏洞扫描程序）可以使用元数据为镜像对象添加注解，以汇总结果和控制 Pod 执行。本节描述了该注解的公认格式，以便在控制台中可靠使用它来为用户显示有用的数据。

2.6.4.1. 镜像元数据

镜像质量数据有多种不同的类型，包括软件包漏洞和开源软件 (OSS) 许可证合规性。另外，该元数据的供应商可能不止一个。为此，保留了以下注解格式：

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

表 2.1. 注解键格式

组件	描述	可接受值
qualityType	元数据类型	**漏洞** **许可证** **操作** **策略**
providerId	供应商 ID 字符串	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

2.6.4.1.1. 注解键示例

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

镜像质量注解的值是必须遵循以下格式的结构化数据：

表 2.2. 注解值格式

字段	必需？	描述	类型
name	是	供应商显示名称	字符串
timestamp	是	扫描时间戳	字符串
description	否	简短描述	字符串
reference	是	信息来源的 URL 或更多详细信息。必需，以便用户可以验证数据。	字符串
scannerVersion	否	扫描程序版本	字符串
compliant	否	合规性通过或未通过	布尔值
summary	否	找到的问题摘要	列表（请参阅下表）

summary 字段必须遵循以下格式：

表 2.3. Summary 字段值格式

字段	描述	类型
label	显示组件标签（例如："critical"、"important"、"moderate"、"low" 或 "health"）	字符串
data	此组件的数据（例如：发现的漏洞计数或分数）	字符串
severityIndex	组件索引，允许对图形表示进行排序和分配。该值范围为 **0..3**，其中 **0** = low。	整数
reference	信息来源的 URL 或更多详细信息。可选。	字符串

2.6.4.1.2. 注解值示例

本示例显示了一个镜像的 OpenSCAP 注解，带有漏洞概述数据以及一个合规性布尔值：

OpenSCAP 注解

```
{
  "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 }
  ]
}
```

本例演示了红帽生态系统目录注解中镜像的容器镜像部分，包含健康索引数据以及获取更多详细信息的外部 URL:

红帽生态系统目录注解

```
{
  "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. 为镜像对象添加注解

虽然 OpenShift Container Platform 最终用户操作针对的是镜像流对象，但会使用安全元数据为镜像对象添加注解。镜像对象是集群范围的，指向可能由多个镜像流和标签引用的单一镜像。

2.6.4.2.1. 注解 CLI 命令示例

将 **<image>** 替换为镜像摘要，如 **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. 控制 Pod 执行

使用 **images.openshift.io/deny-execution** 镜像策略，以编程方式控制镜像是否可以运行。

2.6.4.3.1. 注解示例

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

2.6.4.4. 集成参考

在大多数情况下，漏洞扫描程序等外部工具会开发一个脚本或插件来监视镜像更新，执行扫描，并使用结果为相关的镜像对象添加注解。通常，这个自动化过程会调用 OpenShift Container Platform 4.16 REST API 来编写注解。有关 REST API 的常规信息，请参阅 OpenShift Container Platform REST API。

2.6.4.4.1. REST API 调用示例

以下使用 **curl** 的示例调用会覆盖注解值。请务必替换 **<token>**、**<openshift_server>**、**<image_id>** 和 **<image_annotation>** 的值。

修补 API 调用

```
$ 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> }'
```

以下是 **PATCH** 有效负载数据的示例：

修补调用数据

```
{
"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}] }"
    }
  }
}
```

其他资源

- 镜像流对象

2.7. 安全地使用容器 REGISTRY

容器 registry 存储容器镜像以便：

- 使镜像可供其他用户访问
- 将镜像组织到可包含多个镜像版本的存储库中
- 选择性地根据不同的身份验证方法限制对镜像的访问，或者将其设为可公开使用

有一些公共容器 registry（如 Quay.io 和 Docker Hub）可供很多个人和机构共享其镜像。红帽 Registry 提供支持的红帽和合作伙伴镜像，而红帽生态系统目录为这些镜像提供了详细的描述和健康状态检查。要管理自己的 registry，您可以购买一个容器 registry，如 Red Hat Quay。

从安全角度来说，有些 registry 提供了特殊的功能来检查并改进容器的健康状态。例如，Red Hat Quay 通过 Clair 安全扫描程序提供容器漏洞扫描功能，提供构建触发器以在 GitHub 和其他位置的源代码发生更改时自动重建镜像，并支持使用基于角色的访问控制 (RBAC) 来保护对镜像的访问。

2.7.1. 知道容器来自哪里？

您可以使用一些工具来扫描和跟踪您下载和部署的容器镜像的内容。但是，容器镜像有很多公共来源。在使用公共容器 registry 时，您可以使用可信源添加一层保护。

2.7.2. 不可变和已认证的容器

在管理*不可变容器*时，消耗安全更新尤其重要。不可变容器是在运行时永远不会更改的容器。当您部署不可变容器时，您不会介入正在运行的容器来替换一个或多个二进制文件。从操作角度来说，您可以重建并重新部署更新的容器镜像以替换某个容器，而不是更改该容器。

红帽的已认证镜像：

- 在平台组件或层中没有已知漏洞
- 在 RHEL 平台间兼容，从裸机到云端
- 受红帽支持

已知漏洞列表不断扩展，因此您必须一直跟踪部署的容器镜像的内容以及新下载的镜像。您可以使用红帽安全公告 (RHSA) 来提醒您红帽的已认证容器镜像中出现的任何新问题，并指引您找到更新的镜像。另外，您还可以访问红帽生态系统目录查找每个红帽镜像的各种安全相关问题。

2.7.3. 从红帽 Registry 和生态系统目录获取容器

红帽在红帽生态系统目录的容器镜像部分列出了适用于红帽产品和合作伙伴产品的已认证容器镜像。在该目录中，您可以查看每个镜像的详情，包括 CVE、软件包列表和健康状态分数。

红帽镜像实际存储在所谓的 *红帽 Registry* 中，其具体代表为公共容器 registry (**registry.access.redhat.com**) 和经过身份验证的 registry (**registry.redhat.io**)。这两者基本包括同一组容器镜像，其中 **registry.redhat.io** 包括了一些需要使用红帽订阅凭证进行身份验证的额外镜像。

红帽会监控容器内容以了解漏洞，并定期进行更新。当红帽发布安全更新（如 *glibc*、DROWN 或 Dirty Cow 的修复程序）时，任何受影响的容器镜像也会被重建并推送到 Red Hat Registry。

红帽使用 **health index** 来反映通过红帽生态系统目录提供的每个容器的安全风险。由于容器消耗红帽提供的软件和勘误表流程，旧的、过时的容器不安全，而全新容器则更安全。

为了说明容器的年龄，红帽生态系统目录使用一个等级系统。新鲜度等级是一个镜像可用的最旧、最严重的安全公告衡量标准。“A”比“F”状态更新。如需了解这个等级系统的更多详情，请参阅 Red Hat Ecosystem Catalog 内部使用的容器健康状态索引等级。

如需了解有关红帽软件安全更新和漏洞的详细信息，请参阅红帽产品安全中心。查看红帽安全公告以搜索具体公告和 CVE。

2.7.4. OpenShift Container Registry

OpenShift Container Platform 包括 *OpenShift Container Registry*，它是作为平台集成组件运行的私有 registry，可用于管理容器镜像。OpenShift Container Registry 提供基于角色的访问控制，供您管理谁可以拉取和推送哪些容器镜像。

OpenShift Container Platform 还支持与其他您可能已经使用的私有 registry 集成，如 Red Hat Quay。

其他资源

- 集成的 OpenShift 镜像 registry

2.7.5. 使用 Red Hat Quay 存储容器

Red Hat Quay 是红帽的一个企业级容器 registry 产品。Red Hat Quay 的开发是通过上游 Project Quay 完成的。Red Hat Quay 可用于在内部部署，或通过 Red Hat Quay 在 Quay.io 的托管版本部署。

与安全性相关的 Red Hat Quay 功能包括：

- **时间机器**：允许带有旧标签的镜像在一段设定时间后或基于用户选择的过期时间过期。
- **存储库镜像**：让您出于安全原因为其他 registry 创建镜像，如在公司防火墙后面的 Red Hat Quay 上托管公共存储库，或出于性能原因让 registry 更接近使用的位置。
- **操作日志存储**：将 Red Hat Quay 日志输出保存到 Elasticsearch 存储或 Splunk，以便稍后进行搜索和分析。
- **Clair**：根据每个容器镜像的来源，针对各种 Linux 漏洞数据库扫描镜像。
- **内部身份验证**：使用默认本地数据库处理面向 Red Hat Quay 的 RBAC 身份验证，或者从 LDAP、Keystone (OpenStack)、JWT 自定义身份验证或 External Application Token 身份验证中选择。
- **外部授权 (OAuth)**：允许通过 GitHub、GitHub Enterprise 或 Google 身份验证对 Red Hat Quay 进行授权。
- **访问设置**：生成令牌，允许从 docker、rkt、匿名访问、用户创建的帐户、加密客户端密码或前缀用户名自动完成访问 Red Hat Quay。

Red Hat Quay 不断与 OpenShift Container Platform 持续集成，包含几个特别值得关注的 OpenShift Container Platform Operator。Quay Bridge Operator 可让您将内部 OpenShift 镜像 registry 替换为 Red Hat Quay。Red Hat Quay Container Security Operator 可让您检查从 Red Hat Quay registry 中拉取的 OpenShift Container Platform 中运行的镜像的漏洞。

2.8. 保护构建过程

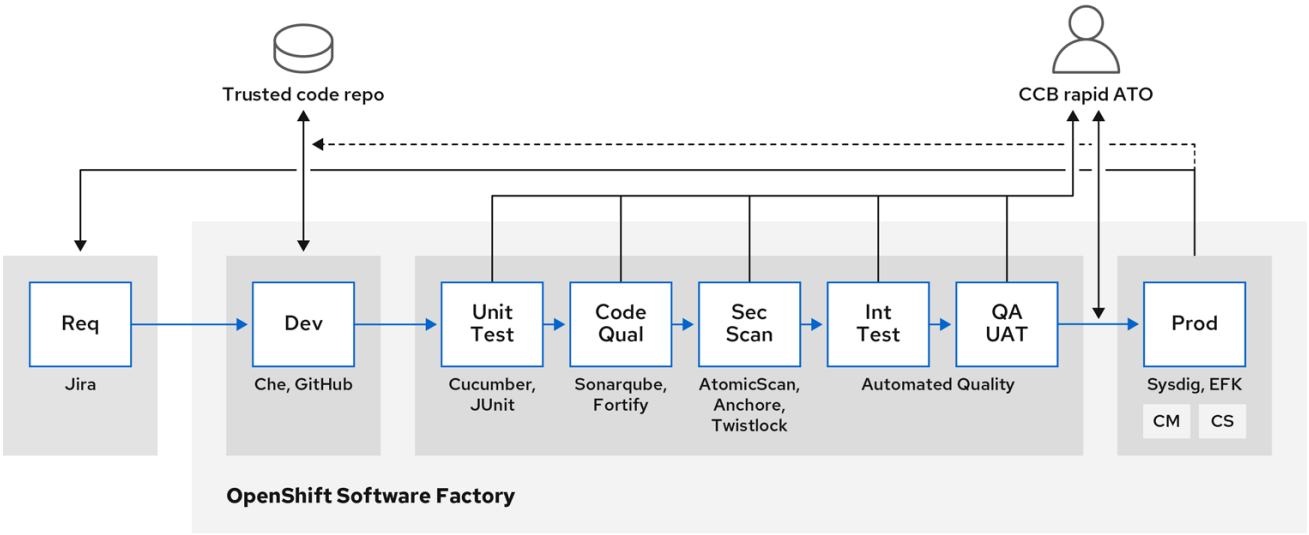
在容器环境中，软件构建过程是生命周期中应用程序代码与所需的运行时库集成的阶段。管理此构建过程是保护软件堆栈的关键。

2.8.1. 一次构建，随处部署

使用 OpenShift Container Platform 作为容器构建的标准平台可保证构建环境的安全。遵循"一次构建，随处部署"的原则可确保构建过程的产品就是在生产环境中部署的产品。

保持容器的不可变性也是很重要的。您不应该修补运行中的容器，而应该重建并重新部署这些容器。

随着您的软件逐步进入构建、测试和生产阶段，组成软件供给链的工具必须是可信的。下图演示了可整合到容器化软件的可信软件供应链中的流程和工具：



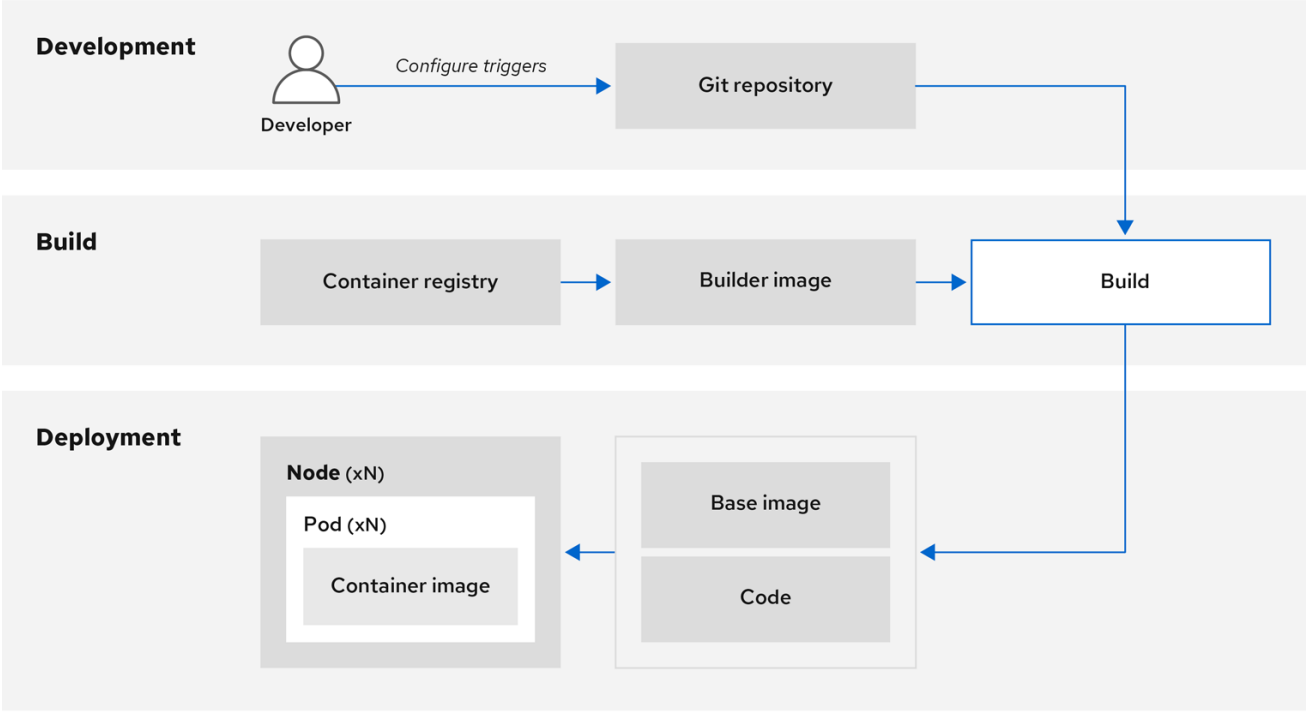
OpenShift Container Platform 可以与可信代码存储库（如 GitHub）和开发平台（如 Che）集成，用于创建和管理安全代码。单元测试可以依赖 Cucumber 和 JUnit。您可以通过 Anchore 或 Twistlock 检查容器中的漏洞和合规问题，并使用镜像扫描工具，如 AtomicScan 或 Clair。Sysdig 等工具可以提供对容器化应用程序的持续监控。

2.8.2. 管理构建

您可以使用 Source-to-Image (S2I) 将源代码和基础镜像组合起来。*构建器镜像*利用 S2I 使您的开发和运维团队能够就可重复生成的构建环境展开合作。对于可作为通用基础镜像 (UBI) 镜像的 Red Hat S2I 镜像，您现在可以使用从真实 RHEL RPM 软件包构建的基础镜像自由重新分发您的软件。红帽取消了订阅限制以允许这一操作。

当开发人员使用构建镜像通过 Git 提交某个应用程序的代码时，OpenShift Container Platform 可以执行以下功能：

- 通过使用代码存储库上的 Webhook 或其他自动持续集成 (CI) 过程进行触发，以从可用的工件、S2I 构建器镜像和新提交的代码中自动编译新镜像。
- 自动部署新构建的镜像以进行测试。
- 将测试镜像提升到生产环境中，以使用 CI 过程自动进行部署。



您可以使用集成的 OpenShift Container Registry 来管理对最终镜像的访问。S2I 和原生构建镜像会自动推送到 OpenShift Container Registry。

除了包含的用于 CI 的 Jenkins 外，您还可以使用 RESTful API 将您自己的构建和 CI 环境与 OpenShift Container Platform 集成，并使用与 API 兼容的镜像 registry。

2.8.3. 在构建期间保护输入

在某些情况下，构建操作需要凭证才能访问依赖的资源，但这些凭证最好不要在通过构建生成的最终应用程序镜像中可用。您可以定义输入 secret 以实现这一目的。

例如，在构建 Node.js 应用程序时，您可以为 Node.js 模块设置私有镜像。要从该私有镜像下载模块，您必须为包含 URL、用户名和密码的构建提供自定义的 **.npmrc** 文件。为安全起见，不应在应用程序镜像中公开您的凭证。

通过使用此示例场景，您可以在新 **BuildConfig** 对象中添加输入 secret：

1. 如果 secret 不存在，则进行创建：

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

这会创建一个名为 **secret-npmrc** 的新 secret，其包含 **~/.npmrc** 文件的 base64 编码内容。

2. 将该 secret 添加到现有 **BuildConfig** 对象的 **source** 部分中：

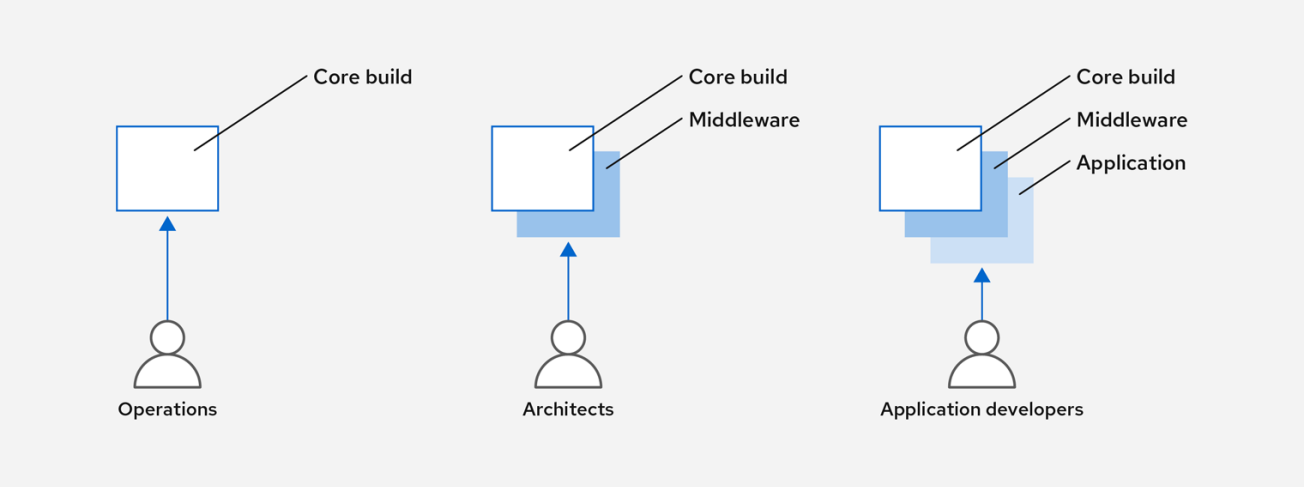
```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
  - destinationDir: .
    secret:
      name: secret-npmrc
```

3. 要在新 **BuildConfig** 对象中包含该 secret，请运行以下命令：

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

2.8.4. 设计构建过程

您可以对容器镜像管理和构建过程进行设计以使用容器层，以便您可以分开控制。



例如，一个运维团队负责管理基础镜像，而架构师则负责管理中间件、运行时、数据库和其他解决方案。然后开发人员可以专注于应用程序层并专注于编写代码。

由于每天都会识别出新的漏洞，因此您需要一直主动检查容器内容。要做到这一点，您应该将自动安全测试集成到构建或 CI 过程中。例如：

- SAST / DAST – 静态和动态安全测试工具。
- 根据已知漏洞进行实时检查的扫描程序。这类工具会为您的容器中的开源软件包编目，就任何已知漏洞通知您，并在之前扫描的软件包中发现新漏洞时为您提供最新信息。

您的 CI 过程应该包含相应的策略，为构建标记出通过安全扫描发现的问题，以便您的团队能够采取适当行动来解决这些问题。您应该为自定义构建容器签名，以确保在构建和部署之间不会修改任何内容。

利用 GitOps 方法，您不仅可以使用相同的 CI/CD 机制来管理应用程序配置，还可以管理 OpenShift Container Platform 基础架构。

2.8.5. 构建 Knative 无服务器应用程序

通过使用 Kubernetes 和 Kourier，您可以在 OpenShift Container Platform 中使用 OpenShift Serverless 来构建、部署和管理无服务器应用程序。

和其他构建一样，您可以使用 S2I 镜像来构建容器，然后使用 Knative 服务提供它们。通过 OpenShift Container Platform Web 控制台的 **Topology** 视图 查看 Knative 应用程序构建。

2.8.6. 其他资源

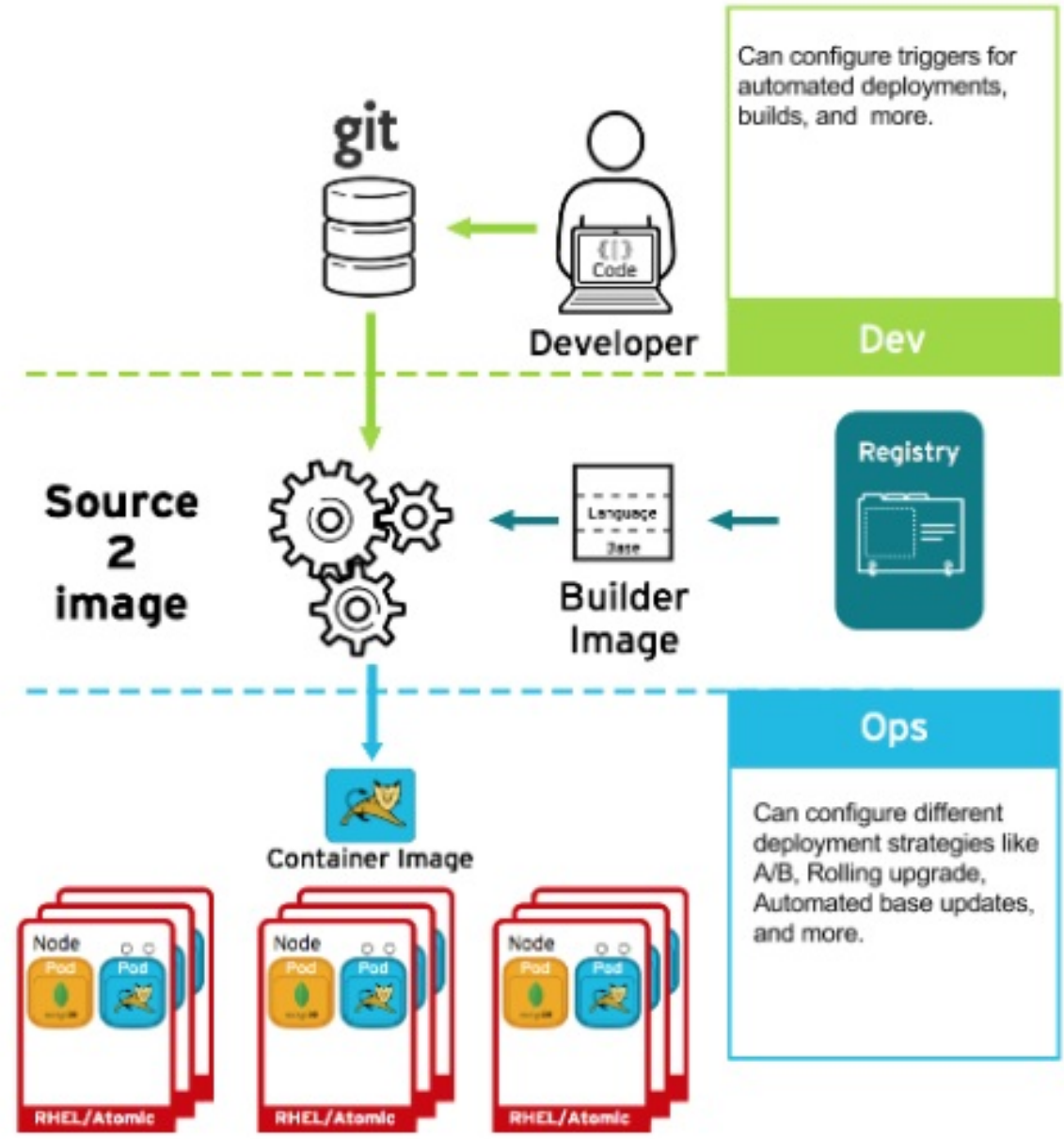
- 理解镜像构建
- 触发和修改构建
- 创建构建输入
- 输入 secret 和配置映射
- OpenShift Serverless 概述
- 使用 Topology 视图查看应用程序组成情况

2.9. 部署容器

您可以使用各种技术来确保所部署的容器包含最新的生产级内容，并确保这些容器没有被修改。这些技术包括设置构建触发器以纳入最新的代码，以及使用签名来确保容器来自可信源且未修改。

2.9.1. 使用触发器控制容器部署

如果在构建过程中发生某种情况，或者部署了镜像后发现一个漏洞，您可以使用基于策略的自动化部署工具进行修复。您可以使用触发器来重建和替换镜像，确保不可变的容器进程，而不是修补正在运行的容器，这种做法是不推荐的。



例如，您使用三个容器镜像层构建了一个应用程序：核心、中间件和应用程序。由于在核心镜像中发现了一个问题，该镜像被重建。构建完成后，该镜像被推送到 OpenShift Container Registry。OpenShift Container Platform 检测到镜像已更改，并根据定义的触发器自动重建并部署应用程序镜像。这一更改包含了固定的库，并确保产品代码与最新镜像是一致的。

您可以使用 **oc set triggers** 命令来设置部署触发器。例如，要为名为 deployment-example 的部署设置触发器：

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

2.9.2. 控制可以部署的镜像源

务必要确保实际部署了所需的镜像，还要确保包括容器内容的镜像来自可信源，且尚未更改。加密签名提供了这一保证。OpenShift Container Platform 可让集群管理员应用广泛或狭窄的安全策略，以反应部署环境和安全要求。该策略由两个参数定义：

- 一个或多个带有可选项目命名空间的 registry

- 信任类型，如接受、拒绝或要求公钥

您可以使用这些策略参数来允许、拒绝整个 registry、部分 registry 或单独的镜像，或者要求具有信任关系。使用可信公钥，您可以确保以加密的方式验证源。该策略规则应用于节点。策略可以在所有节点中统一应用，或针对不同的节点工作负载（例如：构建、区域或环境）加以应用。

镜像签名策略文件示例

```
{
    "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"}]
        }
    }
}
```

该策略可以在节点上保存为 **/etc/containers/policy.json**。最好使用新的 **MachineConfig** 对象将此文件保存到节点。这个示例强制执行以下规则：

- 要求 Red Hat Registry (**registry.access.redhat.com**) 中的镜像由红帽公钥签名。
- 要求 **openshift** 命名空间中的 OpenShift Container Registry 中的镜像由 Red Hat 公钥签名。
- 要求 **production** 命名空间中的 OpenShift Container Registry 中的镜像由 **example.com** 的公钥签名。
- 拒绝未由全局 **默认** 定义指定的所有其他 registry。

2.9.3. 使用签名传输

签名传输是一种存储和检索二进制签名 blob 的方法。签名传输有两种类型。

- **Atomic**：由 OpenShift Container Platform API 管理。
- **Docker**：作为本地文件或通过 Web 服务器提供。

OpenShift Container Platform API 负责管理使用 **atomic** 传输类型的签名。您必须将使用此签名类型的镜像存储在 OpenShift Container Registry 中。由于 docker/distribution **extensions** API 会自动发现镜像签名端点，因此不需要额外的配置。

使用 **docker** 传输类型的签名由本地文件或者 Web 服务器提供。这些签名更为灵活，您可以提供来自任何容器镜像 registry 的镜像，并使用独立的服务器来提供二进制签名。

但是，**docker** 传输类型需要进行额外的配置。您必须为节点配置签名服务器的 URI，方法是将随机命名的 YAML 文件放在主机系统上的目录中，默认为 **/etc/containers/registries.d**。YAML 配置文件包含 registry URI 和签名服务器 URI，或 *sigstore*：

Registries.d 文件示例

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

在这个示例中，Red Hat Registry **access.redhat.com** 是为 **docker** 传输类型提供签名的签名服务器。其 URI 在 **sigstore** 参数中定义。您可以将此文件命名为 **/etc/containers/registries.d/redhat.com.yaml**，并使用 Machine Config Operator 自动将文件放在集群中的每个节点上。由于策略和 **registry.d** 文件由容器运行时动态加载，因此不需要重启服务。

2.9.4. 创建 secret 和配置映射

Secret 对象类型提供了一种机制来保存敏感信息，如密码、OpenShift Container Platform 客户端配置文件、**dockercfg** 文件和私有源存储库凭证。Secret 将敏感内容与 Pod 分离。您可以使用卷插件将 secret 信息挂载到容器中，系统也可以使用 secret 代表 Pod 执行操作。

例如，要在部署配置中添加 secret，以便它可以访问私有镜像存储库，请执行以下操作：

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 创建新项目。
3. 导航到 **Resources → Secrets** 并创建新 secret。将 **Secret Type** 设为 **Image Secret**，并将 **Authentication Type** 设为 **Image Registry Credentials**，以输入用于访问私有镜像存储库的凭证。
4. 在创建部署配置时（例如，从 **Add to Project → Deploy Image** 页面），将 **Pull Secret** 设置为您的新 secret。

配置映射与 secret 类似，但设计为能支持与不含敏感信息的字符串配合使用。**ConfigMap** 对象包含配置数据的键值对，这些数据可在 Pod 中消耗或用于存储控制器等系统组件的配置数据。

2.9.5. 自动化持续部署

您可以将自己的持续部署 (CD) 工具与 OpenShift Container Platform 集成。

利用 CI/CD 和 OpenShift Container Platform，您可以自动执行重建应用程序的过程，以纳入最新的修复、测试，并确保它在环境中随处部署。

其他资源

- 输入 secret 和配置映射

2.10. 保护容器平台

OpenShift Container Platform 和 Kubernetes API 是大规模自动化容器管理的关键。API 用于：

- 验证并配置 Pod、服务和复制控制器的数据。
- 在收到传入请求时执行项目验证，并对其他主要系统组件调用触发器。

OpenShift Container Platform 中基于 Kubernetes 的安全相关功能包括：

- 多租户，将基于角色的访问控制和网络策略组合起来，以在多个级别上隔离容器。
- 准入插件，在 API 和向 API 发出请求的各方之间形成界限。

OpenShift Container Platform 使用 Operator 来自动化和简化 Kubernetes 级别安全功能的管理。

2.10.1. 使用多租户隔离容器

多租户允许 OpenShift Container Platform 集群上由多个用户拥有并在多个主机和命名空间中运行的应用程序保持相互隔离并与外部攻击隔离。要获取多租户，您可以将基于角色的访问控制 (RBAC) 应用到 Kubernetes 命名空间。

在 Kubernetes 中，*命名空间*是应用程序可以独立于其他应用程序运行的区域。OpenShift Container Platform 使用和扩展命名空间的方式是添加额外的注解，包括在 SELinux 中的 MCS 标签，并将这些扩展命名空间标识为*项目*。在项目范围内，用户可以维护自己的集群资源，包括服务帐户、策略、限制和各种其他对象。

可将 RBAC 对象分配给项目，以便授权所选用户访问这些项目。该授权采用规则、角色和绑定的形式：

- 规则会定义用户可在项目中创建或访问的内容。
- 角色是您可以绑定到所选用户或组的规则集合。
- 绑定会定义用户或组与角色之间的关联。

本地 RBAC 角色和绑定将用户或组附加到特定项目。集群 RBAC 可将集群范围的角色和绑定附加到集群中的所有项目。有默认的集群角色可以分配，用来提供 **admin**、**basic-user**、**cluster-admin** 和 **cluster-status** 访问。

2.10.2. 使用准入插件保护 control plane

RBAC 可控制用户和组与可用项目之间的访问规则，而*准入插件*可定义对 OpenShift Container Platform 主 API 的访问。准入插件形成一个由以下部分组成的规则链：

- 默认准入插件：这些插件实现了一组默认策略和资源限制以应用于 OpenShift Container Platform control plane 的组件。
- 变异准入插件：这些插件会动态地扩展准入链。它们调用 Webhook 服务器，不仅可对请求进行身份验证，还可修改所选资源。

- 验证准入插件：这些插件验证所选资源的请求，不仅可验证请求，还可确保资源不会再次更改。

API 请求经过一个链中的准入插件，沿途的任何失败都会导致请求遭到拒绝。每个准入插件都与特定资源关联，且只响应这些资源的请求。

2.10.2.1. 安全性上下文约束(SCC)

您可以使用*安全性上下文约束* (SCC) 定义 Pod 运行必须满足的一组条件，以便其能被系统接受。

可由 SCC 管理的一些方面包括：

- 运行特权容器
- 容器可请求添加的功能
- 将主机目录用作卷。
- 容器的 SELinux 上下文。
- 容器用户 ID。

如果具有所需的权限，您可以根据需要将默认 SCC 策略调整为更宽松。

2.10.2.2. 为服务帐户授予角色

就像为用户分配基于角色的访问一样，您可以为服务帐户分配角色。为每个项目创建的默认服务帐户有三个。服务帐户：

- 范围限制为特定项目
- 名称来自其项目
- 会被自动分配一个 API 令牌和凭证来访问 OpenShift Container Registry

与平台组件关联的服务帐户自动使其密钥轮转。

2.10.3. 认证和授权

2.10.3.1. 使用 OAuth 控制访问

您可以通过身份验证和授权使用 API 访问控制来保护容器平台。OpenShift Container Platform master 包含内置的 OAuth 服务器。用户可以获取 OAuth 访问令牌来对自身进行 API 身份验证。

作为管理员，您可以使用*用户身份供应商*（如 LDAP、GitHub 或 Google）配置 OAuth 以进行身份验证。用户身份供应商默认用于新的 OpenShift Container Platform 部署，但您可以在初始安装时或安装后进行此配置。

2.10.3.2. API 访问控制和管理

应用程序可以拥有多个独立的 API 服务，这些服务具有不同的端点需要管理。OpenShift Container Platform 包含 3scale API 网关的容器化版本，以便您管理 API 并控制访问。

3scale 为您提供用于 API 身份验证和安全性的各种标准选项，它们可单独或组合起来用于发布凭证和控制访问：标准 API 密钥、应用程序 ID 和密钥对以及 OAuth 2.0。

您可以限制对特定端点、方法和服务的访问，并为用户组应用访问策略。您可以通过应用程序计划来为各组开发人员设置 API 使用率限制并控制流量。

有关使用容器化 3scale API 网关 APIcast v2 的教程，请参阅 3scale 文档中的在 Red Hat OpenShift 上运行 APIcast。

2.10.3.3. 红帽单点登录

通过红帽单点登录服务器，您可以提供基于标准的 Web 单点登录功能，包括 SAML 2.0、OpenID Connect 和 OAuth 2.0，从而保护应用程序。该服务器可充当基于 SAML 或 OpenID Connect 的用户身份供应商 (IdP)，使用基于标准的令牌在您的企业用户目录或用于身份信息的第三方用户身份供应商与您的应用程序之间进行调和。您可以将红帽单点登录与基于 LDAP 的目录服务集成，包括 Microsoft Active Directory 和 Red Hat Enterprise Linux Identity Management。

2.10.3.4. 安全自助服务 Web 控制台

OpenShift Container Platform 提供了一个自助服务 Web 控制台，以确保团队在没有授权的情况下无法访问其他环境。OpenShift Container Platform 通过提供以下功能来确保安全多租户 master：

- 使用传输层安全 (TLS) 访问 master
- 使用 X.509 证书或 OAuth 访问令牌访问 API 服务器
- 通过项目配额限制异常令牌可以造成的破坏
- Etcd 服务不直接向集群公开

2.10.4. 为平台管理证书

OpenShift Container Platform 的框架中有多个组件，它们使用基于 REST 的 HTTPS 通信，通过 TLS 证书利用加密功能。OpenShift Container Platform 的安装程序会在安装过程中配置这些证书。生成此流量的一些主要组件如下：

- master（API 服务器和控制器）
- etcd
- 节点
- registry
- 路由器

2.10.4.1. 配置自定义证书

您可以在初始安装过程中或在重新部署证书时为 API 服务器和 Web 控制台的公共主机名配置自定义服务证书。您还可以使用自定义 CA。

其他资源

- OpenShift Container Platform 简介
- 使用 RBAC 定义和应用权限
- 关于准入插件

- 管理安全性上下文约束
- SCC 参考命令
- 为服务帐户授予角色的示例
- 配置内部 OAuth 服务器
- 了解身份提供程序配置
- 证书类型和描述
- 代理证书

2.11. 保护网络

可以在多个级别管理网络安全性。在 Pod 级别上，网络命名空间可以通过限制网络访问来防止容器查看其他 Pod 或主机系统。网络策略可让您控制允许或拒绝连接。您可以管理容器化应用程序的入口和出口流量。

2.11.1. 使用网络命名空间

OpenShift Container Platform 使用软件定义网络 (SDN) 来提供一个统一的集群网络，它允许集群中的不同容器相互间进行通信。

默认情况下，网络策略模式使项目中的所有 Pod 都可被其他 Pod 和网络端点访问。要在一个项目中隔离一个或多个 Pod，您可以在该项目中创建 **NetworkPolicy** 对象来指示允许的入站连接。使用多租户模式，您可以为 Pod 和服务提供项目级别的隔离。

2.11.2. 使用网络策略隔离 Pod

使用*网络策略*，您可以在同一项目中将 Pod 相互隔离。网络策略可以拒绝对 Pod 的所有网络访问，只允许入口控制器的连接，拒绝其他项目中的 Pod 的连接，或为网络的行为方式设置类似的规则。

其他资源

- 关于网络策略

2.11.3. 使用多个 Pod 网络

默认情况下，每个运行中的容器只有一个网络接口。Multus CNI 插件可让您创建多个 CNI 网络，然后将任何这些网络附加到您的 Pod。这样，您可以执行一些操作，例如将私有数据单独放在更为受限的网络上，并在每个节点上使用多个网络接口。

其他资源

- 使用多网络

2.11.4. 隔离应用程序

OpenShift Container Platform 允许您为单个集群上的网络流量分段以创建多租户集群，使用户、团队、应用程序和环境与非全局资源隔离。

其他资源

- 使用 OpenShift SDN 配置网络隔离

2.11.5. 保护入口流量

如何配置从 OpenShift Container Platform 集群外对 Kubernetes 服务的访问会产生很多安全影响。除了公开 HTTP 和 HTTPS 路由外，入口路由还允许您设置 NodePort 或 LoadBalancer 入口类型。NodePort 从每个集群 worker 中公开应用程序的服务 API 对象。借助 LoadBalancer，您可以将外部负载均衡器分配给 OpenShift Container Platform 集群中关联的服务 API 对象。

其他资源

- 配置集群入口流量

2.11.6. 保护出口流量

OpenShift Container Platform 提供了使用路由器或防火墙方法控制出口流量的功能。例如，您可以使用 IP 白名单来控制对数据库的访问。集群管理员可以为 OpenShift Container Platform SDN 网络供应商中的项目分配一个或多个出口 IP 地址。同样，集群管理员可以使用出口防火墙防止出口流量传到 OpenShift Container Platform 集群之外。

通过分配固定出口 IP 地址，您可以将特定项目的所有出站流量分配到该 IP 地址。使用出口防火墙时，您可以防止 Pod 连接到外部网络，防止 Pod 连接到内部网络，或限制 Pod 对特定内部子网的访问。

其他资源

- 配置出口防火墙来控制对外部 IP 地址的访问
- 为项目配置出口 IP

2.12. 保护附加存储

OpenShift Container Platform 支持多种存储类型，包括内部存储和云供应商。特别是，OpenShift Container Platform 可以使用支持 Container Storage Interface 的存储类型。

2.12.1. 持久性卷插件

容器对于无状态和有状态的应用程序都很有用。保护附加存储是保护有状态服务的一个关键元素。通过使用 Container Storage Interface (CSI)，OpenShift Container Platform 可以包含支持 CSI 接口的任何存储后端中的存储。

OpenShift Container Platform 为多种存储提供插件，包括：

- 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 *
- 网络文件系统 (NFS)
- FlexVolume
- Fibre Channel
- iSCSI

具有动态置备的存储类型的插件标记为星号 (*)。对于相互通信的所有 OpenShift Container Platform 组件，传输中的数据都通过 HTTPS 来加密。

您可以以任何方式在主机上挂载持久性卷（PV）。不同的存储类型具有不同的功能，每个 PV 的访问模式可以被设置为特定卷支持的特定模式。

例如：NFS 可以支持多个读/写客户端，但一个特定的 NFS PV 可能会以只读方式导出。每个 PV 都有自己的一组访问模式来描述特定 PV 的功能，如 **ReadWriteOnce**、**ReadOnlyMany** 和 **ReadWriteMany**。

2.12.2. 共享存储

对于 NFS 等共享存储供应商，PV 将其组 ID (GID) 注册为 PV 资源上的注解。然后，当 Pod 声明 PV 时，注解的 GID 会添加到 Pod 的补充组中，为该 pod 授予共享存储内容的访问权限。

2.12.3. 块存储

对于 AWS Elastic Block Store (EBS)、GCE Persistent Disk 和 iSCSI 等块存储供应商，OpenShift Container Platform 使用 SELinux 功能为非特权 Pod 保护挂载卷的根目录，从而使所挂载的卷由与其关联的容器所有并只对该容器可见。

其他资源

- 了解持久性存储
- 配置 CSI 卷
- 动态置备
- 使用 NFS 的持久性存储
- 使用 AWS Elastic Block Store 的持久性存储
- 使用 GCE Persistent Disk 的持久性存储

2.13. 监控集群事件和日志

监控和审核 OpenShift Container Platform 集群的功能是防止集群及其用户遭到不当使用的重要措施。

有两个主要的集群级别信息来源可用来实现这一目的：事件和日志记录。

2.13.1. 监视集群事件

建议集群管理员熟悉 **Event** 资源类型，并查看系统事件列表以确定值得关注的事件。事件与命名空间关联，可以是与它们相关的资源的命名空间，对于集群事件，也可以是 **default** 命名空间。default 命名空间包含与监控或审核集群相关的事件，如节点事件和与基础架构组件相关的资源事件。

Master API 和 **oc** 命令不通过提供参数来将事件列表的范围限定为与节点相关的事件。一个简单的方法是使用 **grep**：

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

输**出示例**

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

更灵活的方法是以其他工具可以处理的形式输出事件。例如，以下示例针对 JSON 输出使用 **jq** 工具以仅提取 **NodeHasDiskPressure** 事件：

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

输**出示例**

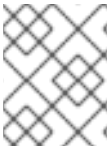
```
{
  "apiVersion": "v1",
  "count": 3,
  "involvedObject": {
    "kind": "Node",
    "name": "origin-node-1.example.local",
    "uid": "origin-node-1.example.local"
  },
  "kind": "Event",
  "reason": "NodeHasDiskPressure",
  ...
}
```

与资源创建、修改或删除相关的事件也很适合用来检测到集群误用情况。例如，以下查询可以用来查找过度拉取镜像：

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

输**出示例**

```
4
```



注意

删除命名空间时，也会被删除其事件。也可以让事件过期并将其删除以防止占用 etcd 存储。事件不作为持久记录存储，且需要持续进行频繁的轮询来捕获统计数据。

2.13.2. 日志记录

使用 **oc log** 命令，您可以实时查看容器日志、构建配置和部署。不同的用户可对日志具有不同的访问权限：

- 有权访问项目的用户默认可以查看该项目的日志。

- 具有 admin 角色的用户可以访问所有容器日志。

要保存日志以供进一步审核和分析，您可以启用 **cluster-logging** 附加功能来收集、管理和查看系统、容器和审计日志。您可以通过 OpenShift Elasticsearch Operator 和 Red Hat OpenShift Logging Operator 部署、管理和升级 OpenShift Logging。

2.13.3. 审计日志

使用*审计日志*，您可以跟踪与用户、管理员或其他 OpenShift Container Platform 组件的行为方式相关的一系列活动。API 审计日志记录在每个服务器上完成。

其他资源

- 系统事件列表
- 了解 OpenShift Logging
- 查看审计日志

第 3 章 配置证书

3.1. 替换默认入口证书

3.1.1. 了解默认入口证书

默认情况下，OpenShift Container Platform 使用 Ingress Operator 创建内部 CA 并发布对 **.apps** 子域下应用程序有效的通配符证书。web 控制台和 CLI 也使用此证书。

内部基础架构 CA 证书是自签名的。虽然这种流程被某些安全或 PKI 团队认为是不当做法，但这里的风险非常小。隐式信任这些证书的客户端仅是集群中的其他组件。将默认通配符证书替换为由 CA bundle 中已包括的公共 CA 发布的证书，该证书由容器用户空间提供，允许外部客户端安全地连接到 **.apps** 子域下运行的应用程序。

3.1.2. 替换默认入口证书

您可以替换 **.apps** 子域下所有应用程序的默认入口证书。替换了证书后，包括 web 控制台和 CLI 在内的所有应用程序都会具有指定证书提供的加密。

先决条件

- 您必须有用于完全限定 **.apps** 子域及其对应私钥的通配符证书。每个文件都应该采用单独的 PEM 格式。
- 私钥必须取消加密。如果您的密钥是加密的，请在将其导入到 OpenShift Container Platform 前对其进行解密。
- 证书必须包含显示 ***.apps.<clustername>.<domain>** 的 **subjectAltName** 扩展。
- 证书文件可以包含链中的一个或者多个证书。通配符证书必须是文件中的第一个证书。然后可以跟随所有中间证书，文件以 root CA 证书结尾。
- 将 root CA 证书复制到额外的 PEM 格式文件中。
- 验证所有包含 **-----END CERTIFICATE-----** 的证书在该行后有一个回车。

流程

1. 创建仅包含用于为通配符证书签名的 root CA 证书的配置映射：

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

1 **</path/to/cert.crt>** 是 root CA 证书文件在本地文件系统中的路径。

2. 使用新创建的配置映射更新集群范围的代理配置：

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

3. 创建包含通配符证书链和密钥的 secret ：

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

1 **<secret>** 是将要包含证书链和私钥的 secret 的名称。

2 **</path/to/cert.crt>** 是证书链在本地文件系统中的路径。

3 **</path/to/cert.key>** 是与此证书关联的私钥的路径。

4. 使用新创建的 secret 更新 Ingress Controller 配置：

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

1 将 **<secret>** 替换为上一步中用于 secret 的名称。

其他资源

- 替换 CA Bundle 证书
- 代理证书自定义

3.2. 添加 API 服务器证书

默认 API 服务器证书由内部 OpenShift Container Platform 集群 CA 发布。默认情况下，位于集群外的客户端无法验证 API 服务器的证书。此证书可以替换为由客户端信任的 CA 发布的证书。

3.2.1. 向 API 服务器添加指定名称的证书

默认 API 服务器证书由内部 OpenShift Container Platform 集群 CA 发布。您可以添加一个或多个 API 服务器将根据客户端请求的完全限定域名（FQDN）返回的证书，例如使用反向代理或负载均衡器时。

先决条件

- 您必须有 FQDN 及其对应私钥的证书。每个文件都应该采用单独的 PEM 格式。
- 私钥必须取消加密。如果您的密钥是加密的，请在将其导入到 OpenShift Container Platform 前对其进行解密。
- 证书必须包含显示 FQDN 的 **subjectAltName** 扩展。
- 证书文件可以包含链中的一个或者多个证书。API 服务器 FQDN 的证书必须是文件中的第一个证书。然后可以跟随所有中间证书，文件以 root CA 证书结尾。

警告

不要为内部负载均衡器（主机名 **api-int.<cluster_name>.<base_domain>**）提供指定了名称的证书。这样可让集群处于降级状态。

流程

1. 以 **kubeadmin** 用户身份登录新的 API。

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

2. 获取 **kubeconfig** 文件。

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

3. 创建一个包含 **openshift-config** 命名空间中证书链和密钥的 secret。

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

1. **<secret>** 是将要包含证书链和私钥的 secret 的名称。
2. **</path/to/cert.crt>** 是证书链在本地文件系统中的路径。
3. **</path/to/cert.key>** 是与此证书关联的私钥的路径。

4. 更新 API 服务器以引用所创建的 secret。

```
$ oc patch apiserver cluster \
    --type=merge -p \
    '{"spec":{"servingCerts": {"namedCertificates":
    [{"names": ["<FQDN>"], 1
    "servingCertificate": {"name": "<secret>"}}]}}' 2
```

1. 将 **<FQDN>** 替换为 API 服务器应为其提供证书的 FQDN。
2. 将 **<secret>** 替换为上一步中用于 secret 的名称。

5. 检查 **apiserver/cluster** 对象并确认该 secret 现已被引用。

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

输出示例

```
...
```

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

6. 检查 **kube-apiserver** operator，并验证 Kubernetes API 服务器的新修订版本是否已推出。可能需要一分钟时间，Operator 才会检测配置更改并触发新部署。当新修订版本被推出时，**PROGRESSING** 会报告 **True**。

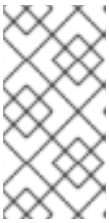
```
$ oc get clusteroperators kube-apiserver
```

在 **PROGRESSING** 列为 **False** 前不要继续进入下一步，如下所示：

输**出示例**

```
NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
kube-apiserver   4.16.0    True        False         False      145m
```

如果 **PROGRESSING** 显示为 **True**，请等待几分钟后再试一次。



注意

只有在第一次添加了名为 certificate 的 API 服务器时，Kubernetes API 服务器才会推出新的修订版本。当名为 certificate 的 API 服务器被续订时，Kubernetes API 服务器的新修订版本不会推出，因为 **kube-apiserver** pod 会动态重新载入更新的证书。

3.3. 使用服务提供的证书 SECRET 保护服务流量

3.3.1. 了解服务用证书

服务用证书旨在为需要加密的复杂中间件应用程序提供支持。这些证书是作为 TLS web 服务器证书发布的。

service-ca 控制器使用 **x509.SHA256WithRSA** 签名算法来生成服务证书。

生成的证书和密钥采用 PEM 格式，分别存储在所创建 secret 的 **tls.crt** 和 **tls.key** 中。证书和密钥在接近到期时自动替换。

用于发布服务证书的服务 CA 证书在 26 个月内有效，并在有效期少于 13 个月时进行自动轮转。轮转后，以前的服务 CA 配置仍会被信任直到其过期为止。这将为所有受影响的服务建立一个宽限期，以在过期前刷新其密钥内容。如果没有在这个宽限期内对集群进行升级（升级会重启服务并刷新其密钥），您可能需要手动重启服务以避免在上一个服务 CA 过期后出现故障。

注意

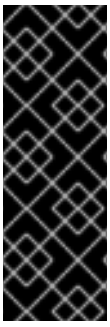
您可以使用以下命令来手动重启集群中的所有 pod。此命令会导致服务中断，因为它将删除每个命名空间中运行的所有 pod。这些 Pod 会在删除后自动重启。

```
$ 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. 添加服务证书

要保证与服务的通信的安全，请在与服务相同的命名空间中将签名的服务证书和密钥对生成 secret。

生成的证书仅对内部服务 DNS 名称 **<service.name>.<service.namespace>.svc** 有效，并且只适用于内部通信。如果您的服务是一个无头服务（未设置 **clusterIP** 值），则生成的证书还包含通配符主题，格式为 ***.<service.name>.<service.namespace>.svc**。



重要

因为生成的证书包含无头服务的通配符主题，因此如果您的客户端必须区分不同的 pod，则不得使用服务 CA。在这种情况下：

- 使用其他 CA 生成各个 TLS 证书。
- 对于定向到单个 pod 且不得被其他 pod 模拟的连接，不接受服务 CA 作为可信 CA。这些连接必须配置为信任用于生成单个 TLS 证书的 CA。

先决条件

- 必须定义了服务。

流程

1. 使用 **service.beta.openshift.io/serving-cert-secret-name** 注解该服务：

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

1 将 **<service_name>** 替换为要保护的服务的名称。

2 **<secret_name>** 是生成的 secret 的名称，该 secret 包含证书和密钥对。为方便起见，建议您使用与 **<service_name>** 相同的名称。

例如，使用以下命令来注解服务 **test1**:

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

2. 检查服务以确认是否存在注解：

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

输出示例

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

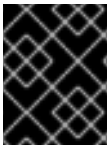
3. 在集群为服务生成 secret 后，**Pod** spec 可以挂载它，pod 将在可用后运行。

其他资源

- 您可以使用服务证书来配置使用重新加密 TLS 终止的安全路由。如需更多信息，请参阅使用自定义证书创建重新加密路由。

3.3.3. 将服务 CA 捆绑包添加到配置映射中

Pod 可通过挂载使用 **service.beta.openshift.io/inject-cabundle=true** 注解的 **ConfigMap** 对象来访问服务 CA 证书。注解后，集群会自动将服务 CA 证书注入配置映射上的 **service-ca.crt** 键。访问此 CA 证书可允许 TLS 客户端使用服务用证书验证服务连接。



重要

将这个注解添加到配置映射后，会删除其中的所有现有数据。建议您使用单独的配置映射来包含 **service-ca.crt**，而不是使用存储您的 Pod 配置的同一配置映射。

流程

1. 使用 **service.beta.openshift.io/inject-cabundle=true** 注解配置映射：

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

1 将 **<config_map_name>** 替换为配置映射的名称。

注意

在卷挂载中明确引用 **service-ca.crt** 键可防止 pod 启动，直到配置映射使用 CA 捆绑包注入为止。可通过为卷的 serving 证书将 **optional**字段设置为 **true** 来覆盖此行为。

例如，使用以下命令来注解配置映射 **test1**:

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

2. 查看配置映射，以确保注入了服务 CA 捆绑包：

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

CA 捆绑包在 YAML 输出中作为 **service-ca.crt** 键的值显示：

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

3.3.4. 将服务 CA 捆绑包添加到 API 服务

您可以使用 **service.beta.openshift.io/inject-cabundle=true** 注解 **APIService** 对象，使其 **spec.caBundle** 字段由服务 CA 捆绑包填充。这可让 Kubernetes API 服务器验证用于保护目标端点的安全的服务 CA 证书。

流程

1. 使用 **service.beta.openshift.io/inject-cabundle=true** 注解 API 服务：

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

(1) 将 **<api_service_name>** 替换为要注解的 API 服务的名称。

例如，使用以下命令来注解 API 服务 **test1**:

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

2. 查看 API 服务，以确保注入了服务 CA 捆绑包：

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

CA 捆绑包在 YAML 输出中的 **spec.caBundle** 字段中显示：

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

3.3.5. 将服务 CA 捆绑包添加到自定义资源定义中

您可以使用 **service.beta.openshift.io/inject-cabundle=true** 注解 **CustomResourceDefinition**（CRD）对象，使其 **spec.conversion.webhook.clientConfig.caBundle 字段** 由服务 CA 捆绑包填充。这可让 Kubernetes API 服务器验证用于保护目标端点的安全的服务 CA 证书。

注意

只有在 CRD 被配置为使用 webhook 进行转换，才会将服务 CA 捆绑包注入 CRD。只有在 CRD 的 webhook 需要使用服务 CA 证书时，注入服务 CA 捆绑包才有意义。

流程

1. 使用 **service.beta.openshift.io/inject-cabundle=true** 注解 CRD：

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

1 将 **<crd_name>** 替换为要注解的 CRD 的名称。

例如，使用以下命令来注解 CRD **test1**：

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

2. 查看 CRD，以确保注入了服务 CA 捆绑包：

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

CA 捆绑包在 YAML 输出中的 **spec.conversion.webhook.clientConfig.caBundle** 字段中显示：

```
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. 将服务 CA 捆绑包添加到变异的 webhook 配置中

您可以使用 **service.beta.openshift.io/inject-cabundle=true** 注解 **MutatingWebhookConfiguration** 对象，使每个 webhook 的 **clientConfig.caBundle** 字段由服务 CA 捆绑包填充。这可让 Kubernetes API 服务器验证用于保护目标端点的安全的服务 CA 证书。

注意

不要为 admission webhook 配置设置此注解，不同的 webhook 需要指定不同的 CA 捆绑包。如果您这样做了，则会为所有 webhook 注入这个服务 CA 捆绑包。

流程

1. 使用 **service.beta.openshift.io/inject-cabundle=true** 注解变异 Webhook 配置：

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

1 将 **<mutating_webhook_name>** 替换为要注解的变异 Webhook 配置的名称。

例如，使用以下命令来注解变异 Webhook 配置 **test1**：

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

2. 查看变异 Webhook 配置，以确保注入了服务 CA 捆绑包：

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

CA 捆包在 YAML 输出中所有 webhook 的 **clientConfig.caBundle** 字段中显示：

```
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. 将服务 CA 捆绑包添加到验证 webhook 配置中

您可以使用 **service.beta.openshift.io/inject-cabundle=true** 注解 **ValidatingWebhookConfiguration** 对象，使每个 webhook 的 **clientConfig.caBundle 字段** 由服务 CA 捆绑包填充。这可让 Kubernetes API 服务器验证用于保护目标端点的安全的服务 CA 证书。

注意

不要为 admission webhook 配置设置此注解，不同的 webhook 需要指定不同的 CA 捆绑包。如果您这样做了，则会为所有 webhook 注入这个服务 CA 捆绑包。

流程

1. 使用 **service.beta.openshift.io/inject-cabundle=true** 注解验证 Webhook 配置：

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

1 将 **<validating_webhook_name>** 替换为要注解的验证 webhook 配置的名称。

例如，使用以下命令来注解验证 webhook 配置 **test1**：

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

2. 查看验证 Webhook 配置，以确保注入了服务 CA 捆绑包：

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

CA 捆包在 YAML 输出中所有 webhook 的 **clientConfig.caBundle** 字段中显示：

```
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. 手动轮转生成的服务证书

您可以通过删除关联的 secret 来轮换服务证书。删除 secret 会导致自动创建新 secret，进而生成新的证书。

先决条件

- 必须为服务生成了包含证书和密钥对的 secret。

流程

1. 检查该服务以确定包含证书的 secret。这可以在 **service-cert-secret-name** 注解中找到，如下所示。

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

输**出示例**

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

2. 删除为服务生成的 secret。此过程将自动重新创建 secret。

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

1 将 **<secret>** 替换为前一步中的 secret 名称。

3. 通过获取新 secret 并检查 **AGE** 来确认已经重新创建了证书。

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

输**出示例**

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

3.3.9. 手动轮转服务 CA 证书

服务 CA 在 26 个月内有效，并在有效期少于 13 个月时进行刷新。

如果需要，您可以按照以下步骤手动刷新服务 CA。

警告

手动轮换的服务 CA 不会保留对上一个服务 CA 的信任。在集群中的 pod 重启完成前，您的服务可能会临时中断。pod 重启可以确保 Pod 使用由新服务 CA 发布的证书服务。

先决条件

- 必须以集群管理员身份登录。

流程

1. 使用以下命令，查看当前服务 CA 证书的到期日期。

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

2. 手动轮转服务 CA。此过程会生成一个新的服务 CA，用来为新服务证书签名。

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

3. 要将新证书应用到所有服务，请重启集群中的所有 pod。此命令确保所有服务都使用更新的证书。

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

警告

此命令会导致服务中断，因为它将遍历并删除每个命名空间中运行的 Pod。这些 Pod 会在删除后自动重启。

3.4. 更新 CA 捆绑包

3.4.1. 了解 CA 捆绑包证书

通过代理证书，用户可以指定，在平台组件创建出口连接时使用的一个或多个自定义证书颁发机构 (CA)。

Proxy 对象的 **trustedCA** 字段是对包含用户提供的可信证书颁发机构 (CA) 捆绑包的配置映射的引用。这个捆绑包与 Red Hat Enterprise Linux CoreOS (RHCOS) 信任捆绑包合并，并注入到生成出口 HTTPS 调用的平台组件的信任存储中。例如，**image-registry-operator** 调用外部镜像 registry 来下载镜像。如果没有指定 **trustedCA**，则只有 RHCOS 信任的捆绑包用于代理 HTTPS 连接。如果您想要使用自己的证书基础架构，请向 RHCOS 信任捆绑包提供自定义 CA 证书。

trustedCA 字段应当仅由代理验证器使用。验证程序负责从所需的键 **ca-bundle.crt** 中读取证书捆绑包，并将其复制到 **openshift-config-managed** 命名空间中名为 **trusted-ca-bundle** 的配置映射中。被 **trustedCA** 引用的配置映射的命名空间是 **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. 替换 CA Bundle 证书

流程

1. 创建包含用于为通配符证书签名的 root CA 证书的配置映射：

```
$ 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>** 是 CA 证书捆绑包在本地文件系统中的路径。

2. 使用新创建的配置映射更新集群范围的代理配置：

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

其他资源

- 替换默认入口证书
- 启用集群范围代理
- 代理证书自定义

第 4 章 证书类型和描述

4.1. API 服务器的用户提供的证书

4.1.1. 用途

集群以外的客户端通过 **api.<cluster_name>.<base_domain>** 可访问 API 服务器。您可能希望客户端使用不同主机名访问 API 服务器，无需向客户端发布集群管理的证书颁发机构 (CA) 证书。管理员必须在提供内容时设置 API 服务器使用的自定义默认证书。

4.1.2. 位置

用户提供的证书必须在 **openshift-config** 命名空间中的 **kubernetes.io/tls** 类型 **Secret** 中提供。更新 API 服务器集群配置(**apiserver/cluster** 资源），以启用用户提供的证书。

4.1.3. 管理

用户提供的证书由用户管理。

4.1.4. 过期

API 服务器客户端证书过期时间少于五分钟。

用户提供的证书由用户管理。

4.1.5. 自定义

根据需要，更新包含用户管理的证书的 secret。

其他资源

- 添加 API 服务器证书

4.2. 代理证书

4.2.1. 用途

通过代理证书，用户可以指定，在平台组件创建出口连接时使用的一个或多个自定义证书颁发机构 (CA) 证书。

Proxy 对象的 **trustedCA** 字段是对包含用户提供的可信证书颁发机构 (CA) 捆绑包的配置映射的引用。这个捆绑包与 Red Hat Enterprise Linux CoreOS (RHCOS) 信任捆绑包合并，并注入到生成出口 HTTPS 调用的平台组件的信任存储中。例如，**image-registry-operator** 调用外部镜像 registry 来下载镜像。如果没有指定 **trustedCA**，则只有 RHCOS 信任的捆绑包用于代理 HTTPS 连接。如果您想要使用自己的证书基础架构，请向 RHCOS 信任捆绑包提供自定义 CA 证书。

trustedCA 字段应当仅由代理验证器使用。验证程序负责从所需的键 **ca-bundle.crt** 中读取证书捆绑包，并将其复制到 **openshift-config-managed** 命名空间中名为 **trusted-ca-bundle** 的配置映射中。被 **trustedCA** 引用的配置映射的命名空间是 **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-----
```

其他资源

- 配置集群范围代理

4.2.2. 在安装过程中管理代理证书

安装程序配置的 **additionalTrustBundle** 值用于在安装过程中指定任何代理信任的 CA 证书。例如：

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

输**出示例**

```
...
proxy:
  httpProxy: http://<https://username:password@proxy.example.com:123/>
  httpsProxy: https://<https://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. 位置

用户提供的信任捆绑包以配置映射表示。配置映射挂载到进行 HTTPS 调用的平台组件的文件系统中。通常，Operator 会将配置映射挂载到 **/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem**，但代理并不要求这样做。代理可以修改或检查 HTTPS 连接。在这两种情况下，代理都必须为连接生成新证书并为新证书签名。

完整的代理支持意味着连接到指定的代理服务器并信任它所生成的所有签名。因此，需要让用户指定一个信任的根用户，以便任何连接到该可信根的证书链都被信任。

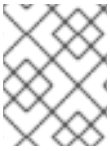
如果使用 RHCOS 信任捆绑包，请将 CA 证书放在 **/etc/pki/ca-trust/source/anchors** 中。

详情请参阅 Red Hat Enterprise Linxu 文档的使用共享系统证书。

4.2.4. 过期

用户为用户提供的信任捆绑包设定了过期期限。

默认到期条件由 CA 证书本身定义。在 OpenShift Container Platform 或 RHCOS 使用前，由 CA 管理员为证书配置这个证书。



注意

红帽不会监控 CA 何时到期。但是，由于 CA 的长生命周期，这通常不是个问题。但是，您可能需要周期性更新信任捆绑包。

4.2.5. 服务

默认情况下，所有使用出口 HTTPS 调用的平台组件将使用 RHCOS 信任捆绑包。如果定义了 **trustedCA**，它将会被使用。

任何在 RHCOS 节点上运行的服务都可以使用该节点的信任捆绑包。

4.2.6. 管理

这些证书由系统而不是用户管理。

4.2.7. 自定义

更新用户提供的信任捆绑包包括：

- 在 **trustedCA** 引用的配置映射中更新 PEM 编码的证书，或
- 在命名空间 **openshift-config** 中创建配置映射，其中包含新的信任捆绑包并更新 **trustedCA** 来引用新配置映射的名称。

将 CA 证书写入 RHCOS 信任捆绑包的机制与将其它文件写入 RHCOS（使用机器配置）完全相同。当 Machine Config Operator (MCO)应用包含新 CA 证书的新机器配置时，它会稍后运行 program **update-ca-trust**，并重启 RHCOS 节点上的 CRI-O 服务。在这个版本中，不需要节点重启。重启 CRI-O 服务会自动使用新的 CA 证书更新信任捆绑包。例如：

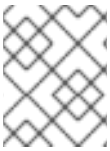
```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-examplecorp-ca-cert
spec:
  config:
    ignition:
      version: 3.1.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-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

机器的信任存储还必须支持更新节点的信任存储。

4.2.8. 续订

没有 Operator 可以自动更新 RHCOS 节点上的证书。



注意

红帽不会监控 CA 何时到期。但是，由于 CA 的长生命周期，这通常不是个问题。但是，您可能需要周期性更新信任捆绑包。

4.3. 服务 CA 证书

4.3.1. 用途

service-ca 是部署 OpenShift Container Platform 集群时创建自签名 CA 的一个 Operator。

4.3.2. 过期

不支持自定义过期条件。自签名 CA 存储在一个限定名为 **service-ca/signing-key** 的 secret 中。它在 **tls.crt** (certificate(s))、**tls.key** (private key) 和 **ca-bundle.crt** (CA bundle) 项中。

其他服务可通过使用 **service.beta.openshift.io/serving-cert-secret-name: <secret name>** 注解一个服务资源来请求一个 service serving 证书。作为响应，Operator 会为指定的 secret 生成一个新证书，**tls.crt**，以及一个私钥，**tls.key**。该证书的有效期为两年。

其他服务可通过使用 **service.beta.openshift.io/inject-cabundle:true** 注解来请求将服务 CA 的 CA 捆绑包注入 API 服务或配置映射资源，以支持通过服务 CA 生成的证书。作为响应，Operator 会将其当前 CA 捆绑包写入 API 服务的 **CABundle** 字段，或将 **service-ca.crt** 写入配置映射。

自 OpenShift Container Platform 4.3.5 起，支持自动轮转，并将其向后移植到一些 4.2.z 和 4.3.z 版本中。对于任何支持自动轮转的版本，服务 CA 证书在 26 个月内有效，并在有效期少于 13 个月时进行刷新。如果需要，您可以手动刷新服务 CA。

服务 CA 26 个月的过期时间比支持的 OpenShift Container Platform 集群的预期升级间隔更长，因此使用服务 CA 证书的非 control plane 系统将会在 CA 轮转后刷新。这会发生在轮转前使用的 CA 过期前。

警告

手动轮换的服务 CA 不会保留对上一个服务 CA 的信任。在集群中的 pod 重启完成前，您的服务可能会临时中断。pod 重启可以确保 Pod 使用由新服务 CA 发布的证书服务。

4.3.3. 管理

这些证书由系统而不是用户管理。

4.3.4. 服务

使用服务 CA 证书的服务包括：

- 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
- cluster-storage-operator
- machine-config-operator
- console-operator
- insights-operator
- machine-api-operator
- operator-lifecycle-manager
- CSI driver operator

这不是一个完整的列表。

其他资源

- 手动轮转 service serving 证书
- 使用服务提供的证书 secret 保护服务流量

4.4. 节点证书

4.4.1. 用途

节点证书由集群签名，并允许 kubelet 与 Kubernetes API 服务器通信。它们来自 kubelet CA 证书，该证书由 bootstrap 过程生成。

4.4.2. 位置

kubelet CA 证书位于 **openshift-kube-apiserver-operator** 命名空间中的 **kube-apiserver-to-kubelet-signer** secret 中。

4.4.3. 管理

这些证书由系统而不是用户管理。

4.4.4. 过期

节点证书会在 292 天后自动轮转，并在 365 天后过期。

4.4.5. 续订

Kubernetes API Server Operator 会在 292 天自动生成新的 **kube-apiserver-to-kubelet-signer** CA 证书。旧的 CA 证书在 365 天后被删除。当 kubelet CA 证书被续订或删除时，节点不会重启。

集群管理员可以通过运行以下命令手动续订 kubelet CA 证书：

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

其他资源

- 操作节点

4.5. BOOTSTRAP 证书

4.5.1. 用途

OpenShift Container Platform 4 及之后的版本中的 kubelet 使用位于 **/etc/kubernetes/kubeconfig** 中的 bootstrap 证书来启动 bootstrap。然后是 bootstrap 初始化过程 和 kubelet 授权来创建一个 CSR。

在此过程中，kubelet 在通过 bootstrap 频道进行通信时会生成一个 CSR。控制器管理器为 CSR 签名，以生成 kubelet 管理的证书。

4.5.2. 管理

这些证书由系统而不是用户管理。

4.5.3. 过期

此 bootstrap 证书有效期为 10 年。

kubelet 管理的证书的有效期为一年，并在一年经过了大约 80% 时进行自动轮转。

注意

OpenShift Lifecycle Manager (OLM) 不会更新 bootstrap 证书。

4.5.4. 自定义

您无法自定义 bootstrap 证书。

4.6. ETCD 证书

4.6.1. 用途

etcd 证书由 etcd-signer 签名; 证书来自由 bootstrap 过程生成的证书颁发机构 (CA)。

4.6.2. 过期

CA 证书有效期为 10 年。对等证书、客户端证书和服务器证书的有效期为三年。

4.6.3. 轮转 etcd 证书

在 **etcd** 证书过期前轮转它。

流程

1. 运行以下命令，验证新签名人证书的剩余生命周期：

```
$ oc get secret -n openshift-etcd etcd-signer -
ojsonpath='{.metadata.annotations.auth\.openshift\.io/certificate-not-after}'
```

2. 如果剩余的生命周期接近当前日期，通过删除签名者并等待静态 pod 推出来重新创建签名者。
   - 运行以下命令来删除签名者：

```
$ oc delete secret -n openshift-etcd etcd-signer
```

   - 运行以下命令等待静态 pod 推出：

```
$ oc wait --for=condition=Progressing=False --timeout=15m clusteroperator/etcd
```

3. 在 **etcd** 重启后，运行以下命令使用新的 **openshift-config** 命名空间中切换原始 CA，在 **openshift-etcd** 中轮转：

```
$ oc get secret etcd-signer -n openshift-etcd -ojson | jq
'del(.metadata["namespace","creationTimestamp","resourceVersion","selfLink","uid"])' | oc
apply -n openshift-config -f -
```

4. 运行以下命令，等待集群 Operator 推出并稳定：

```
$ oc adm wait-for-stable-cluster --minimum-stable-period 2m
```

4.6.4. etcd 证书轮转警报和指标签名者证书

两种警报类型告知用户待处理的 **etcd** 证书过期：

etcdSignerCAExpirationWarning

在签名者过期前 730 天。

etcdSignerCAExpirationCritical

发生 365 天，直到签名者过期。

出于以下原因，您可以轮转证书：

- 您会收到过期警报。
- 私钥被泄漏。

重要

当私钥泄漏时，您必须轮转所有证书。

OpenShift Container Platform 指标系统的 **etcd** 签名者有一个 etcd signer。在 *轮转 etcd 证书*中替换以下指标参数。

- **etcd-metric-signer** 而不是 **etcd-signer**
- **etcd-metrics-ca-bundle** 而不是 **etcd-ca-bundle**

4.6.5. 管理

这些证书仅由系统管理，并会自动轮转。

4.6.6. 服务

etcd 证书用于 etcd 对等成员间通信的加密，以及加密客户端流量。以下证书由 etcd 和其他与 etcd 通信的进程生成和使用：

- 对等证书（Peer cerfificate）：用于 etcd 成员之间的通信。
- 客户端证书（Client certificate）：用于加密服务器和客户端间的通信。目前，API 服务器只使用客户端证书，除代理外，其他服务都不应该直接连接到 etcd。客户端 secret(**etcd-client**、**etcd-metric-client**、**etcd-metric-signer** 和 **etcd-signer**)添加到 **openshift-config**、**openshift-monitoring** 和 **openshift-kube-apiserver** 命名空间中。
- 服务器证书（Server certificate）：etcd 服务器用来验证客户端请求。
- 指标证书（Metric certificate）：所有使用指标的系统都使用 metric-client 证书连接到代理。

其他资源

- 恢复到一个以前的集群状态

4.7. OLM 证书

4.7.1. 管理

Operator Lifecycle Manager (OLM)组件的所有证书 (**olm-operator**、**catalog-operator**、**packageserver** 和 **marketplace-operator**) 都由系统管理。

安装在它们的 **ClusterServiceVersion**（CSV）对象中安装包含 webhook 或 API 服务的 Operator 时，OLM 会为这些资源创建和轮转证书。**openshift-operator-lifecycle-manager** 命名空间中的资源的证书由 OLM 管理。

OLM 不会更新它在代理环境中管理的 Operator 证书。这些证书必须由用户使用订阅配置进行管理。

4.8. 聚合的 API 客户端证书

4.8.1. 用途

聚合的 API 客户端证书用于在连接到 Aggregated API 服务器时验证 KubeAPIServer。

4.8.2. 管理

这些证书由系统而不是用户管理。

4.8.3. 过期

此 CA 在 30 天内有效。

受管客户端证书在 30 天内有效。

CA 和客户端证书通过使用控制器自动轮转。

4.8.4. 自定义

您无法自定义聚合的 API 服务器证书。

4.9. MACHINE CONFIG OPERATOR 证书

4.9.1. 用途

此证书颁发机构用于在初始置备过程中保护从节点到机器配置服务器 (MCS) 的连接。

有两个证书：自签名 CA MCS CA。一个派生的证书 MCS 证书

4.9.1.1. 置备详情

使用 Red Hat Enterprise Linux CoreOS (RHCOS) 的 OpenShift Container Platform 安装会使用 Ignition 安装。这个过程分为两个部分：

1. 创建 Ignition 配置来引用 MCS 提供的完整配置的 URL。
2. 对于用户置备的 infrastucture 安装方法，Ignition 配置清单作为 **openshift-install** 命令创建的 **worker.ign** 文件。对于使用 Machine API Operator 的安装程序置备的基础架构安装方法，此配置显示为 **worker-user-data** secret。

重要

目前，不支持阻止或限制机器配置服务器端点。机器配置服务器必须公开给网络，以便新置备的机器没有现有配置或状态，才能获取其配置。在这个模型中，信任的根是证书签名请求 (CSR) 端点，即 kubelet 发送其证书签名请求以批准加入集群。因此，机器配置不应用于分发敏感信息，如 secret 和证书。

为确保机器配置服务器端点，端口 22623 和 22624 在裸机场景中是安全的，客户必须配置正确的网络策略。

其他资源

- Machine Config Operator。
- 关于 OpenShift SDN 网络插件。

4.9.1.2. 置备信任链

MCS CA 在 **security.tls.certificateAuthorities** 配置字段下注入 Ignition 配置。然后，MCS 使用 web 服务器提供的 MCS 证书提供完整的配置。

客户端会验证服务器提供的 MCS 证书对它可识别的颁发机构有信任链。在这种情况下，MCS CA 是颁发机构，它会签署 MCS 证书。这样可确保客户端正在访问正确的服务器。在这种情况下，客户端是在 initramfs 的机器上运行的 Ignition。

4.9.1.3. 集群内的关键资料

MCS CA 出现在集群中作为 **kube-system** 命名空间 **root-ca** 对象（带有 **ca.crt** 键）的配置映射。私钥不会存储在集群中，并在安装完成后丢弃。

MCS 证书在集群中作为 **openshift-machine-config-operator** 命名空间中的 secret 出现， **machine-config-server-tls** 对象中带有 **tls.crt** 和 **tls.key**。

4.9.2. 管理

目前还不支持直接修改其中任何一个证书。

4.9.3. 过期

MCS CA 有效期为 10 年。

发布的服务证书有效期为 10 年。

4.9.4. 自定义

您无法自定义 Machine Config Operator 证书。

4.10. 用户提供的默认入口证书

4.10.1. 用途

应用程序通常通过 **<route_name>.apps.<cluster_name>.<base_domain>** 来公开。**<cluster_name>** 和 **<base_domain>** 来自安装配置文件。**<route_name>** 是路由的主机字段（如果指定）或路由名称。例如，**hello-openshift-default.apps.username.devcluster.openshift.com**。**hello-openshift** 是路由的名称，路由位于 default 命名空间。您可能希望客户端访问应用程序而无需向客户端发布集群管理的 CA 证书。在提供应用程序内容时，管理员必须设置自定义默认证书。

警告

Ingress Operator 为 Ingress Controller 生成默认证书，以充当占位符，直到您配置了自定义默认证书为止。不要在生产环境集群中使用 Operator 生成的默认证书。

4.10.2. 位置

用户提供的证书必须在 **openshift-ingress** 命名空间中的 **tls** 类型 **Secret** 资源中提供。更新 **openshift-ingress-operator** 命名空间中的 **IngressController** CR，以启用用户提供的证书的使用。有关此过程的更多信息，请参阅设置自定义默认证书。

4.10.3. 管理

用户提供的证书由用户管理。

4.10.4. 过期

用户提供的证书由用户管理。

4.10.5. 服务

在集群中部署的应用程序使用用户提供的证书作为默认入口。

4.10.6. 自定义

根据需要，更新包含用户管理的证书的 secret。

其他资源

- 替换默认入口证书

4.11. 入口证书（INGRESS CERTIFICATE）

4.11.1. 用途

Ingress Operator 使用以下证书：

- 保证 Prometheus 指标的访问安全。
- 保证对路由的访问安全。

4.11.2. 位置

为了保证对 Ingress Operator 和 Ingress Controller 指标的访问安全，Ingress Operator 使用 service serving 证书。Operator 为自己的指标从 **service-ca** 控制器请求证书，**service-ca** 控制器将证书放置在 **openshift-ingress-operator** 命名空间中的名为 **metrics-tls** 的 secret 中。另外，Ingress Operator 会为每个 Ingress Controller 请求一个证书，**service-ca** 控制器会将证书放在名为 **router-metrics-certs-<name>** 的 secret 中，其中 **<name>** 是 Ingress Controller 的名称（在 **openshift-ingress** 命名空间中）。

每个 Ingress Controller 都有一个默认证书，用于没有指定其自身证书的安全路由。除非指定了自定义证书，Operator 默认使用自签名证书。Operator 使用自己的自签名签名证书为其生成的任何默认证书签名。Operator 生成此签名证书，并将其置于 **openshift-ingress-operator** 命名空间中的名为 **router-ca** 的 secret 中。当 Operator 生成默认证书时，它会将默认证书放在 **openshift-ingress** 命名空间的名为 **router-certs-<name>**（其中 **<name>** 是 Ingress Controller 的名称）的 secret 中。

警告

Ingress Operator 为 Ingress Controller 生成默认证书，以充当占位符，直到您配置了自定义默认证书为止。不要在生产环境集群中使用 Operator 生成的默认证书。

4.11.3. 工作流

图 4.1. **自定义证书工作流**

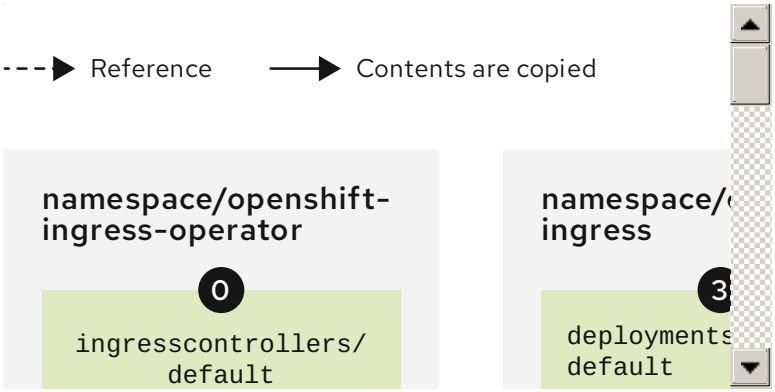
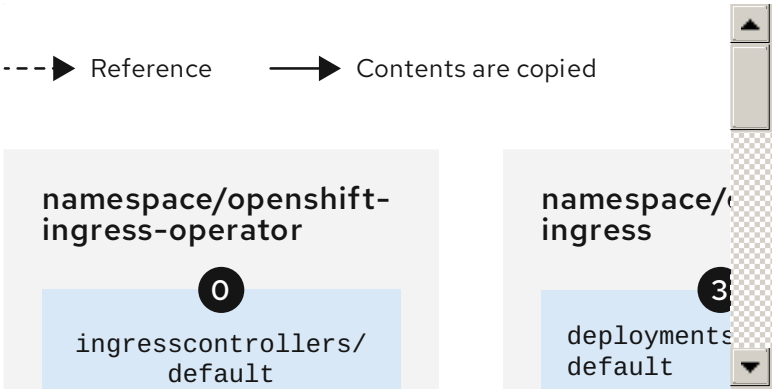


图 4.2. **默认证书工作流**



0 空的 **defaultCertificate** 项会使 Ingress Operator 使用它自己签名的 CA 来为指定的域生成 serving 证书。

1 Ingress Operator 生成的默认 CA 证书和密钥。用来为 Operator 生成的默认 serving 证书签名。

2 在默认工作流中，通配符默认服务证书，由 Ingress Operator 创建，并使用生成的默认 CA 证书签名。在自定义工作流中，这是用户提供的证书。

3 路由器部署。使用 **secrets/router-certs-default** 中的证书作为其默认前端服务器证书。

4 在默认工作流中，通配符默认服务证书（公共和私有组件）的内容在此复制，以启用 OAuth 集成。在自定义工作流中，这是用户提供的证书。

5 默认服务证书的公共（certificate）部分。替换 **configmaps/router-ca** 资源。

6 用户使用对 **ingresscontroller** serving 证书签名的 CA 证书更新集群代理配置。这可让 **auth**、**console** 和 registry 等组件信任 serving 证书。

7 包含合并的 Red Hat Enterprise Linux CoreOS (RHCOS) 和用户提供的 CA 捆绑包（如果未提供用户捆绑包）的集群范围可信 CA 捆绑包。

8 自定义 CA 证书捆绑包,用于指示其他组件（如 **auth** 和 **console**）信任配置了自定义证书的 **ingresscontroller**。

9 **trustedCA** 字段用来引用用户提供的 CA 捆绑包。

10 Cluster Network Operator 将可信 CA 捆绑包注入 **proxy-ca** 配置映射。

11 OpenShift Container Platform 4.16 及更新版本使用 **default-ingress-cert**。

4.11.4. 过期

Ingress Operator 证书的过期条件如下：

- **service-ca** 控制器创建的指标证书的过期日期为创建日期后的两年。
- Operator 的签名证书的过期日期是创建日期后的两年。
- Operator 生成的默认证书的过期日期是在创建日期后两年的时间。

您不能自定义 Ingress Operator 或 **service-ca** 控制器创建的证书的过期条款。

安装 OpenShift Container Platform 时，不能为 Ingress Operator 或 **service-ca** 控制器创建的证书指定过期条款。

4.11.5. 服务

Prometheus 使用的用来确保指标安全的证书。

Ingress Operator 使用它的签名证书来为 Ingress Controller 签名默认证书，您不要为其设置自定义默认证书。

使用安全路由的集群组件可使用默认 Ingress Controller 默认证书。

通过安全路由进入集群的入口使用 Ingress Controller 的默认证书，该证书将访问该路由，除非该路由指定了自己的证书。

4.11.6. 管理

入口证书由用户管理。如需更多信息，请参阅替换默认入口证书。

4.11.7. 续订

service-ca 控制器自动轮转其发放的证书。但是，可以使用 **oc delete secret <secret>** 来手动轮转 service serving 证书。

Ingress Operator 不轮转其自身的签名证书或它生成的默认证书。Operator 生成的默认证书的目的是作为您配置的自定义默认证书的占位者。

4.12. 监控和 OPENSHIFT LOGGING OPERATOR 组件证书

4.12.1. 过期

监控组件使用服务 CA 证书保护其流量。这些证书有效 2 年，并可在服务 CA 轮转时自动替换。轮转每每 13 个月进行一次。

如果证书在 **openshift-monitoring** 或 **openshift-logging** 命名空间中，它会被自动管理并轮转。

4.12.2. 管理

这些证书由系统而不是用户管理。

4.13. CONTROL PLANE 证书

4.13.1. 位置

Control plane 证书包括在这些命名空间中：

- 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. 管理

Control plane 证书由系统管理并自动轮转。

当 control plane 证书出现罕见的过期情形时，请参阅恢复过期的 control plane 证书

第 5 章 COMPLIANCE OPERATOR

5.1. COMPLIANCE OPERATOR 概述

OpenShift Container Platform Compliance Operator 通过自动检查许多技术实施来协助用户，并将其与行业标准、基准和基准的某些方面进行比较；Compliance Operator 并不是一个审计器。为了遵循这些各种标准，您需要与授权的审核员（如限定安全评估器(QSA)、联合授权授权局(JAB)或其他行业认可的监管机构）合作来评估您的环境。

Compliance Operator 根据有关此类标准的一般信息和实践提出建议，并可能会协助补救，但实际的合规性是您的责任。您需要与授权的审核员合作，以达到符合标准的要求。有关最新更新，请参阅 Compliance Operator 发行注记

Compliance Operator 概念
了解 Compliance Operator

了解自定义资源定义

Compliance Operator 管理
安装 Compliance Operator

更新 Compliance Operator

管理 Compliance Operator

卸载 Compliance Operator

Compliance Operator 扫描管理
支持的合规性配置集

Compliance Operator 扫描

定制 Compliance Operator

检索 Compliance Operator 原始结果

管理 Compliance Operator 补救

执行高级 Compliance Operator 任务

对 Compliance Operator 进行故障排除

使用 oc-compliance 插件

5.2. COMPLIANCE OPERATOR 发行注记

通过 Compliance Operator，OpenShift Container Platform 管理员可以描述集群所需的合规状态，并概述缺陷及修复方法。

本发行注记介绍了 OpenShift Container Platform 中 Compliance Operator 的开发。

有关 Compliance Operator 的概述，请参阅了解 Compliance Operator。

要访问最新版本，请参阅更新 Compliance Operator。

5.2.1. OpenShift Compliance Operator 1.5.0

以下公告可用于 OpenShift Compliance Operator 1.5.0：

- RHBA-2024:3533 - OpenShift Compliance Operator 1.5.0 程序错误修复和安全更新

5.2.1.1. 新功能及功能增强

- 在这个版本中，Compliance Operator 提供了一个唯一的配置集 ID，以便更轻松地使用。(CMP-2450)
- 在这个版本中，Compliance Operator 在 ROSA HCP 环境中经过测试和支持。在 ROSA HCP 上运行时，Compliance Operator 只加载 Node 配置集。这是因为红帽托管平台限制对 control plane 的访问，这使得 Platform 配置集与 Operator 的功能无关。(CMP-2581)

5.2.1.2. 程序错误修复

- CVE-2024-2961 在 Compliance Operator 1.5.0 版本中解决。(CVE-2024-2961)
- 在以前的版本中，对于 ROSA HCP 系统，配置集列表不正确。在这个版本中，Compliance Operator 提供了正确的配置集输出。(OCPBUGS-34535)
- 在这个版本中，通过在定制的配置集中设置 **ocp4-var-network-policies-namespaces-exempt-regex** 变量，可以从 **ocp4-configure-network-policies-namespaces** 中排除命名空间。(CMP-2543)

5.2.2. OpenShift Compliance Operator 1.4.1

以下公告可用于 OpenShift Compliance Operator 1.4.1：

- RHBA-2024:1830 - OpenShift Compliance Operator 程序漏洞修复和功能增强更新

5.2.2.1. 新功能及功能增强

- 在这个版本中，Compliance Operator 提供了 CIS OpenShift 1.5.0 配置集规则。(CMP-2447)
- 在这个版本中，Compliance Operator 为 **OCP4 STIG ID** 和 **SRG** 提供配置集规则。(CMP-2401)
- 在这个版本中，将过时的规则应用到 **s390x**。(CMP-2471)

5.2.2.2. 程序错误修复

- 在以前的版本中，对于使用 Red Hat Enterprise Linux (RHEL) 9 的 Red Hat Enterprise Linux CoreOS (RHCOS) 系统，应用 **ocp4-kubelet-enable-protect-kernel-sysctl-file-exist** 规则会失败。在这个版本中，规则替换为 **ocp4-kubelet-enable-protect-kernel-sysctl**。现在，应用自动补救后，基于 RHEL 9 的 RHCOS 系统会在此规则的应用程序中显示 **PASS**。(OCPBUGS-13589)
- 在以前的版本中，当使用配置集 **rhcos4-e8** 应用合规补救后，节点无法通过 SSH 访问 core 用户帐户。在这个版本中，节点可以使用 'sshkey1 选项通过 SSH 访问。(OCPBUGS-18331)
- 在以前的版本中，**STIG** 配置文件缺少来自 CaC 的规则，它们满足对 OpenShift Container Platform 公布的 **STIG** 的要求。在这个版本中，在补救时，集群满足可以使用 Compliance Operator 修复的 **STIG** 要求。(OCPBUGS-26193)

- 在以前的版本中，为多个产品创建一个带有不同类型配置集的 **ScanSettingBinding** 对象，针对绑定中的多个产品类型绕过限制。在这个版本中，产品验证允许多个产品，无论 **ScanSettingBinding** 对象中的配置集类型是什么。(OCPBUGS-26229)

- 在以前的版本中，运行 **rhcos4-service-debug-shell-disabled** 规则显示 **FAIL**，即使使用了自动补救（auto-remediation）功能。在这个版本中，运行 **rhcos4-service-debug-shell-disabled** 规则会在应用 auto-remediation 后显示 **PASS**。(OCPBUGS-28242)

- 在这个版本中，增强了使用 **rhcos4-banner-etc-issue** 规则的说明，以提供更多详细信息。(OCPBUGS-28797)

- 在以前的版本中，**api_server_api_priority_flowschema_catch_all** 规则在 OpenShift Container Platform 4.16 集群中提供了 **FAIL** 状态。在这个版本中，**api_server_api_priority_flowschema_catch_all** 规则在 OpenShift Container Platform 4.16 集群上提供 **PASS** 状态。(OCPBUGS-28918)

- 在以前的版本中，当从 **ScanSettingBinding** (SSB) 对象中显示的完成扫描中删除配置集时，Compliance Operator 不会删除旧的扫描。之后，当使用删除的配置集启动新的 SSB 时，Compliance Operator 无法更新结果。在这个版本中，新的 SSB 会显示新的合规性检查结果。(OCPBUGS-29272)

- 在以前的版本中，在 **ppc64le** 架构中，不会创建 metrics 服务。在这个版本中，当在 **ppc64le** 架构上部署 Compliance Operator v1.4.1 时，指标服务会被正确创建。(OCPBUGS-32797)

- 在以前的版本中，在 HyperShift 托管集群中，带有 **ocp4-pci-dss profile**的扫描会遇到一个不可恢复的错误，因为**过滤器无法迭代**问题。在这个版本中，对 **ocp4-pci-dss** 配置集的扫描将到达 **done** 状态，并返回 **Compliance** 或 **Non-Compliance** 测试结果。(OCPBUGS-33067)

5.2.3. OpenShift Compliance Operator 1.4.0

以下公告可用于 OpenShift Compliance Operator 1.4.0：

- RHBA-2023:7658 - OpenShift Compliance Operator 程序漏洞修复和功能增强更新

5.2.3.1. 新功能及功能增强

- 在这个版本中，使用默认 **worker** 和 **master** 节点池以外的自定义节点池的集群不再需要提供额外变量，以确保 Compliance Operator 聚合该节点池的配置文件。

- 用户现在可以通过将 **ScanSetting.suspend** 属性设置为 **True** 来暂停扫描调度。这允许用户挂起扫描调度，并在不需要删除并重新创建 **ScanSettingBinding** 的情况下重新激活它。这简化了维护期间的暂停扫描计划。(CMP-2123)

- Compliance Operator 现在支持 **Profile** 自定义资源上的可选 **version** 属性。(CMP-2125)

- Compliance Operator 现在支持 **ComplianceRules** 中的配置集名称。(CMP-2126)

- 这个版本提供了与改进的 **cronjob** API 的改进 Operator 兼容性。(CMP-2310)

5.2.3.2. 程序错误修复

- 在以前的版本中，在带有 Windows 节点的集群中，一些规则会在应用自动补救后 FAIL，因为合规性扫描不会跳过 Windows 节点。在这个版本中，扫描时会正确跳过 Windows 节点。(OCPBUGS-7355)

- 在这个版本中，对依赖多路径的 pod 的 root 卷挂载可以正确地处理 **rprivate** 默认挂载传播。(OCPBUGS-17494)
- 在以前的版本中，Compliance Operator 会在不协调规则的情况下为 **coreos_vsyscall_kernel_argument** 生成补救，即使应用补救也是如此。在版本 1.4.0 中，**coreos_vsyscall_kernel_argument** 规则可以正确地评估内核参数并生成适当的补救。(OCPBUGS-8041)
- 在此次更新之前，即使应用了 auto-remediation 后，规则 **rhcos4-audit-rules-login-events-faillock** 将失败。在这个版本中，**rhcos4-audit-rules-login-events-faillock** 失败锁定会在 auto-remediation 后被正确应用。(OCPBUGS-24594)
- 在以前的版本中，从 Compliance Operator 1.3.1 升级到 Compliance Operator 1.4.0 会导致 OVS 规则扫描结果从 **PASS** 变为 **NOT-APPLICABLE**。在这个版本中，OVS 规则扫描结果显示 **PASS** (OCPBUGS-25323)

5.2.4. OpenShift Compliance Operator 1.3.1

以下公告可用于 OpenShift Compliance Operator 1.3.1：

- RHBA-2023:5669 - OpenShift Compliance Operator 程序漏洞修复和功能增强更新

这个版本解决了底层依赖项中的 CVE。

5.2.4.1. 新功能及功能增强

- 您可以在以 FIPS 模式运行的 OpenShift Container Platform 集群中安装和使用 Compliance Operator。

重要

要为集群启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL) 计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。

当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 x86_64、ppc64le 和 s390x 架构上提交到 NIST FIPS 140-2/140-3 Validation。

5.2.4.2. 已知问题

- 在带有 Windows 节点的集群上，一些规则会在应用自动补救后 FAIL，因为合规性扫描不会跳过 Windows 节点。这与预期结果不同，因为扫描时必须跳过 Windows 节点。(OCPBUGS-7355)

5.2.5. OpenShift Compliance Operator 1.3.0

以下公告可用于 OpenShift Compliance Operator 1.3.0 ：

- RHBA-2023:5102 - OpenShift Compliance Operator 增强更新

5.2.5.1. 新功能及功能增强

- 从 Compliance Operator 1.3.0 开始，Defense Information Systems Agency Security Technical Implementation Guide (DISA-STIG) for OpenShift Container Platform 可用。如需更多信息，请参阅支持的合规性配置集。
- Compliance Operator 1.3.0 现在支持用于 NIST 800-53 Moderate-Impact Baseline 的 IBM Power® 和 IBM Z® 用于 OpenShift Container Platform 平台和节点配置集。

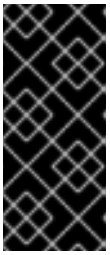
5.2.6. OpenShift Compliance Operator 1.2.0

以下公告可用于 OpenShift Compliance Operator 1.2.0 ：

- RHBA-2023:4245 - OpenShift Compliance Operator 增强更新

5.2.6.1. 新功能及功能增强

- CIS OpenShift Container Platform 4 Benchmark v1.4.0 配置集现在可用于平台和节点应用程序。要找到 CIS OpenShift Container Platform v4 Benchmark，进入 CIS Benchmarks 并点 **Download Latest CIS Benchmark**，然后注册以下载基准。



重要

升级到 Compliance Operator 1.2.0 将覆盖 CIS OpenShift Container Platform 4 Benchmark 1.1.0 配置集。

如果您的 OpenShift Container Platform 环境包含现有的 **cis** 和 **cis-node** 补救，在升级到 Compliance Operator 1.2.0 后，扫描结果可能会有一些区别。

- 现在，**scc-limit-container-allowed-capabilities** 规则提供了额外的审计安全性上下文约束 (SCC)。

5.2.7. OpenShift Compliance Operator 1.1.0

以下公告可用于 OpenShift Compliance Operator 1.1.0：

- RHBA-2023:3630 - OpenShift Compliance Operator 程序漏洞修复和功能增强更新

5.2.7.1. 新功能及功能增强

- 现在，在 **ComplianceScan** 自定义资源定义 (CRD) 状态中提供了 start 和 end 时间戳。
- 现在，通过创建一个 **Subscription** 文件，可以使用 OperatorHub 部署 Compliance Operator。如需更多信息，请参阅在托管的 control plane 上安装 Compliance Operator 。

5.2.7.2. 程序错误修复

- 在此次更新之前，一些 Compliance Operator 规则指令不存在。在这个版本中，以下规则改进了说明：
  - **classification_banner**
  - **oauth_login_template_set**
  - **oauth_logout_url_set**
  - **oauth_provider_selection_set**

    - **ocp_allowed_registries**
    - **ocp_allowed_registries_for_import**
      (OCPBUGS-10473)

- 在此次更新之前，检查准确性和规则指令并不明确。在这个版本中，以下 **sysctl** 规则改进了检查准确性和说明：
    - **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)

- 在此次更新之前，**ocp4-alert-receiver-configured** 规则不包括指令。在这个版本中，**ocp4-alert-receiver-configured** 规则包括改进的指令。(OCPBUGS-7307)

- 在此次更新之前，对于 **rhcos4-e8** 配置集，**rhcos4-sshd-set-loglevel-info** 规则会失败。在这个版本中，**sshd-set-loglevel-info** 规则的补救已更新，以应用正确的配置更改，允许在应用补救后进行后续扫描通过。(OCPBUGS-7816)

- 在此次更新之前，带有最新 Compliance Operator 安装的新 OpenShift Container Platform 在 **scheduler-no-bind-address** 规则中会失败。在这个版本中，自删除参数后，在 OpenShift Container Platform 较新版本的 OpenShift Container Platform 上禁用了 **scheduler-no-bind-address** 规则。(OCPBUGS-8347)

5.2.8. OpenShift Compliance Operator 1.0.0

以下公告可用于 OpenShift Compliance Operator 1.0.0：

- RHBA-2023:1682 - OpenShift Compliance Operator 程序错误修复更新

5.2.8.1. 新功能及功能增强

- Compliance Operator 现在是稳定的，发行频道已升级到 **stable**。将来的版本将遵循 Semantic Versioning。要访问最新版本，请参阅更新 Compliance Operator。

5.2.8.2. 程序错误修复

- 在此次更新之前，compliance_operator_compliance_scan_error_total 指标有一个 ERROR 标签，它对于每个错误信息都有一个不同的值。在这个版本中，compliance_operator_compliance_scan_error_total 指标不会增加值。(OCPBUGS-1803)

- 在此次更新之前，**ocp4-api-server-audit-log-maxsize** 规则会导致 **FAIL** 状态。在这个版本中，错误消息已从指标中删除，这会降低指标数据的基数，从而与最佳实践方式一致。(OCPBUGS-7520)

- 在此次更新之前，**rhcos4-enable-fips-mode** 规则的描述可能会造成 FIPS 可以在安装后被启用的误解。在这个版本中，**rhcos4-enable-fips-mode** 规则描述明确指定必须在安装时启用 FIPS。(OCPBUGS-8358)

5.2.9. OpenShift Compliance Operator 0.1.61

以下公告可用于 OpenShift Compliance Operator 0.1.61：

- RHBA-2023:0557 - OpenShift Compliance Operator 程序错误修复更新

5.2.9.1. 新功能及功能增强

- Compliance Operator 现在支持 Scanner Pod 的超时配置。超时在 **ScanSetting** 对象中指定。如果在超时时间内没有完成扫描，扫描会重试，直到达到最大重试次数为止。如需更多信息，请参阅配置 ScanSetting 超时。

5.2.9.2. 程序错误修复

- 在此次更新之前，Compliance Operator 补救需要变量作为输入。没有变量集的补救会在集群范围内应用，并导致节点卡住，即使它正确应用了补救。在这个版本中，Compliance Operator 会验证是否需要使用 **TailoredProfile** 进行补救提供变量。(OCPBUGS-3864)

- 在此次更新之前，**ocp4-kubelet-configure-tls-cipher-suites** 的说明不完整，需要用户手动优化查询。在这个版本中，**ocp4-kubelet-configure-tls-cipher-suites** 中提供的查询会返回实际结果来执行审计步骤。(OCPBUGS-3017)

- 在此次更新之前，kubelet 配置文件中不会生成系统保留参数，从而导致 Compliance Operator 无法取消暂停机器配置池。在这个版本中，Compliance Operator 在机器配置池评估过程中省略系统保留的参数。(OCPBUGS-4445)

- 在此次更新之前，**ComplianceCheckResult** 对象没有正确描述。在这个版本中，Compliance Operator 从规则描述中找到 **ComplianceCheckResult** 信息。(OCPBUGS-4615)

- 在此次更新之前，Compliance Operator 在解析机器配置时不会检查空的 kubelet 配置文件。因此，Compliance Operator 会 panic 和 crash。在这个版本中，Compliance Operator 实现 kubelet 配置数据结构的改进检查，只有在完全渲染时才继续。(OCPBUGS-4621)

- 在此次更新之前，Compliance Operator 根据机器配置池名称和宽限期为 kubelet 驱除生成补救，从而导致单个驱除规则出现多个补救。在这个版本中，Compliance Operator 为一条规则应用所有补救。(OCPBUGS-4338)

- 在此次更新之前，当试图创建一个 **ScanSettingBinding** 时，使用带有非默认 **MachineConfigPool** 的 **TailoredProfile** 将 **ScanSettingBinding** 标记为 **Failed** 时会出现一个回归。在这个版本中，功能会被恢复，使用 **TailoredProfile** 正确执行自定义 **ScanSettingBinding**。(OCPBUGS-6827)

- 在此次更新之前，有些 kubelet 配置参数没有默认值。在这个版本中，以下参数包含默认值 (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**

- 在此次更新之前，**selinux_confinement_of_daemons** 规则因为 kubelet 运行所需的权限而无法在 kubelet 上运行。在这个版本中，**selinux_confinement_of_daemons** 规则被禁用。(OCPBUGS-6968)

5.2.10. OpenShift Compliance Operator 0.1.59

以下公告可用于 OpenShift Compliance Operator 0.1.59：

- RHBA-2022:8538 - OpenShift Compliance Operator 程序漏洞修复更新

5.2.10.1. 新功能及功能增强

- Compliance Operator 现在支持 **ppc64le** 架构上的支付卡行业数据安全标准 (PCI-DSS) **ocp4-pci-dss** 和 **ocp4-pci-dss-node** 配置集。

5.2.10.2. 程序错误修复

- 在以前的版本中，Compliance Operator 不支持在不同架构上（如 **ppc64le**）的 Payment Card industry Data Security Standard (PCI DSS) **ocp4-pci-dss** 和 **ocp4-pci-dss-node** 配置集。现在，Compliance Operator 在 **ppc64le** 架构上支持 **ocp4-pci-dss** 和 **ocp4-pci-dss-node** 配置集。(OCPBUGS-3252)

- 在以前的版本中，在最近更新到 0.1.57 后，**rerunner** 服务帐户 (SA) 不再被集群服务版本 (CSV) 所有，这会导致在 Operator 升级过程中删除 SA。现在，CSV 拥有 0.1.59 中的 **rerunner** SA，从任何以前的版本升级都不会造成缺少的 SA。(OCPBUGS-3452)

5.2.11. OpenShift Compliance Operator 0.1.57

以下公告可用于 OpenShift Compliance Operator 0.1.57：

- RHBA-2022:6657 - OpenShift Compliance Operator 程序错误修复更新

5.2.11.1. 新功能及功能增强

- **KubeletConfig** 检查从 **Node** 改为 **Platform** 类型。**KubeletConfig** 检查 **KubeletConfig** 的默认配置。配置文件从所有节点聚合到每个节点池的一个位置。请参阅针对默认配置值评估 **KubeletConfig** 规则。

- **ScanSetting** 自定义资源现在允许用户通过 **scanLimits** 属性覆盖扫描程序 Pod 的默认 CPU 和内存限值。如需更多信息，请参阅增加 Compliance Operator 资源限制。

- 现在，可以通过 **ScanSetting** 设置 **PriorityClass** 对象。这样可确保 Compliance Operator 被优先处理，并尽可能减少集群不合规的可能性。如需更多信息，请参阅为 **ScanSetting** 扫描设置 **PriorityClass** 。

5.2.11.2. 程序错误修复

- 在以前的版本中，Compliance Operator 硬编码通知到默认的 **openshift-compliance** 命名空间。如果 Operator 安装在非默认命名空间中，通知将无法正常工作。现在，通知可以在非默认 **openshift-compliance** 命名空间中工作。(BZ#2060726)

- 在以前的版本中，Compliance Operator 无法评估 kubelet 对象使用的默认配置，从而导致结果准确和假的正状态。这个新功能会评估 kubelet 配置，现在可以准确报告。(BZ#2075041)

- 在以前的版本中，在应用自动补救后，Compliance Operator 会报告 **ocp4-kubelet-configure-event-creation** 规则为 **FAIL** 状态，因为 **eventRecordQPS** 被设置为默认值。现在，**ocp4-kubelet-configure-event-creation** 规则补救会设置默认值，规则会正确应用。(BZ#2082416)

- **ocp4-configure-network-policies** 规则需要人工干预才能有效地执行。新的描述性指令和规则更新增加使用 Calico CNI 的集群的 **ocp4-configure-network-policies** 规则的适用性。(BZ#2091794)

- 在以前的版本中，在扫描设置中使用 **debug=true** 选项时，Compliance Operator 不会清理用于扫描基础架构的 pod。这会导致 pod 在删除 **ScanSettingBinding** 后保留在集群中。现在，当一个 **ScanSettingBinding** 被删除时，pod 总是被删除。(BZ#2092913)

- 在以前的版本中，Compliance Operator 使用了一个 **operator-sdk** 命令的旧版本，该命令可导致有关已弃用功能的警报。现在，包括了 **operator-sdk** 命令的更新版本，且没有更多用于已弃用的功能的警报。(BZ#2098581)

- 在以前的版本中，如果 Compliance Operator 无法决定 kubelet 和机器配置之间的关系，则无法应用补救。现在，Compliance Operator 改进了机器配置的处理，并可确定 kubelet 配置是否为机器配置的子集。(BZ#2102511)

- 在以前的版本中，**ocp4-cis-node-master-kubelet-enable-cert-rotation** 的规则没有正确描述成功标准。因此，RotateKubelet **ClientCertificate** 的要求不明确。现在，无论 kubelet 配置文件中存在的配置如何，**ocp4-cis-node-master-kubelet-enable-cert-rotation** 的规则会准确进行。(BZ#2105153)

- 在以前的版本中，检查闲置流超时的规则不会考虑默认值，从而导致规则报告不准确。现在，更强大的检查确保了根据默认配置值的准确性增加。(BZ#2105878)

- 在以前的版本中，当在没有 Ignition 规格的情况下解析机器配置时，Compliance Operator 无法获取 API 资源，这会导致 **api-check-pods** 进程崩溃循环。现在，Compliance Operator 处理没有 Ignition 规格的机器配置池。(BZ#2117268)

- 在以前的版本中，因为 **modprobe** 配置的值不匹配，评估 **modprobe** 配置的规则也会失败。现在，相同的值用于 **modprobe** 配置检查和补救，以确保结果保持一致。(BZ#2117747)

5.2.11.3. 启用

- 指定 **Install into all namespaces in the cluster**或将 **WATCH_NAMESPACES** 环境变量设置为 **""** 时不会在影响所有命名空间。在 Compliance Operator 安装时没有指定的命名空间中安装的任何 API 资源都不再可以正常工作。API 资源可能需要在所选命名空间中创建，或默认在 **openshift-compliance** 命名空间中创建。此更改改进了 Compliance Operator 的内存用量。

5.2.12. OpenShift Compliance Operator 0.1.53

以下公告可用于 OpenShift Compliance Operator 0.1.53：

- RHBA-2022:5537 - OpenShift Compliance Operator 程序漏洞修复更新

5.2.12.1. 程序错误修复

- 在以前的版本中，**ocp4-kubelet-enable-streaming-connections** 规则包含不正确的变量比较，从而导致假的扫描结果。现在，在设置 **streamingConnectionIdleTimeout** 时，Compliance Operator 提供了准确的扫描结果。(BZ#2069891)

- 在以前的版本中，在 IBM Z® 构架中，**/etc/openvswitch/conf.db** 的组所有权不正确，从而导致 **ocp4-cis-node-worker-file-groupowner-ovs-conf-db** 检查失败。现在，这个检查在 IBM Z® 架构系统中被标记为 **NOT-APPLICABLE**。(BZ#2072597)

- 在以前的版本中，**ocp4-cis-scc-limit-container-allowed-capabilities** 规则因为部署中安全性上下文约束(SCC)规则不完整，以 **FAIL** 状态报告。现在，结果为 **MANUAL**，它与需要人工干预的其他检查一致。(BZ#2077916)

- 在以前的版本中，以下规则无法考虑 API 服务器和 TLS 证书和密钥的额外配置路径，即使正确设置了证书和密钥也会导致报告失败：

  - **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**

  现在，规则报告会准确观察 kubelet 配置文件中指定的旧文件路径。(BZ#2079813)

- 在以前的版本中，在评估超时时，**content_rule_oauth_or_oauthclient_inactivity_timeout** 规则不会考虑部署设置的可配置超时。这会导致规则失败，即使超时有效。现在，Compliance Operator 使用 **var_oauth_inactivity_timeout** 变量来设置有效的超时长度。(BZ#2081952)

- 在以前的版本中，Compliance Operator 使用在命名空间上没有适当标记命名空间的管理权限进行特权使用，从而导致有关 Pod 安全级别违反情况的警告信息。现在，Compliance Operator 有适当的命名空间标签，并对访问结果进行适当的调整，而不违反权限。(BZ#2088202)

- 在以前的版本中，为 **rhcos4-high-master-sysctl-kernel-yama-ptrace-scope** 和 **rhcos4-sysctl-kernel-core-pattern** 应用自动补救会导致后续这些规则失败，即使它们被修复。现在，在应用了补救后，规则会正确报告 **PASS**。（BZ#2094382）

- 在以前的版本中，因为内存不足例外，Compliance Operator 会以 **CrashLoopBackoff** 状态失败。现在，Compliance Operator 被改进，在内存和可以正常工作的情况下可以正确处理大型机器配置数据集。(BZ#2094854)

5.2.12.2. 已知问题

- 当在 **ScanSettingBinding** 对象中设置 **"debug":true** 时，当删除该绑定时，由 **ScanSettingBinding** 对象生成的 pod 不会被删除。作为临时解决方案，运行以下命令来删除剩余的 pod：

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

  (BZ#2092913)

5.2.13. OpenShift Compliance Operator 0.1.52

以下公告可用于 OpenShift Compliance Operator 0.1.52：

- RHBA-2022:4657 - OpenShift Compliance Operator 程序漏洞修复更新

5.2.13.1. 新功能及功能增强

- FedRAMP high SCAP 配置集现在可用于 OpenShift Container Platform 环境。如需更多信息，请参阅支持的合规性配置集。

5.2.13.2. 程序错误修复

- 在以前的版本中，因为存在 **DAC_OVERRIDE** 功能的安全环境中存在挂载权限问题，**OpenScap** 容器会崩溃。现在，可执行的挂载权限适用于所有用户。(BZ#2082151)
- 在以前的版本中，合规性规则 **ocp4-configure-network-policies** 可以被配置为 **MANUAL**。现在，合规性规则 **ocp4-configure-network-policies** 设置为 **AUTOMATIC**。(BZ#2072431)
- 在以前的版本中，Cluster Autoscaler 将无法缩减，因为 Compliance Operator 扫描 pod 在扫描后永远不会被删除。现在，pod 默认从每个节点中删除，除非明确为调试目的保存。(BZ#2075029)
- 在以前的版本中，将 Compliance Operator 应用到 **KubeletConfig** 会导致节点进入 **NotReady** 状态，因为无法立即暂停 Machine Config Pools。现在，机器配置池会被正确取消暂停，节点可以正常工作。(BZ#2071854)
- 在以前的版本中，Machine Config Operator 在最新版本中使用 **base64** 而不是 **url-encoded** 代码，从而导致 Compliance Operator 修复失败。现在，Compliance Operator 会检查编码来处理 **base64** 和 **url-encoded** Machine Config 代码，补救会正确应用。(BZ#2082431)

5.2.13.3. 已知问题

- 当在 **ScanSettingBinding** 对象中设置 **"debug":true** 时，当删除该绑定时，由 **ScanSettingBinding** 对象生成的 pod 不会被删除。作为临时解决方案，运行以下命令来删除剩余的 pod：

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

(BZ#2092913)

5.2.14. OpenShift Compliance Operator 0.1.49

以下公告可用于 OpenShift Compliance Operator 0.1.49：

- RHBA-2022:1148 - OpenShift Compliance Operator 程序错误修复和功能增强更新

5.2.14.1. 新功能及功能增强

- Compliance Operator 现在支持以下架构：
  - IBM Power®
  - IBM Z®
  - IBM® LinuxONE

5.2.14.2. 程序错误修复

- 在以前的版本中，**openshift-compliance** 内容不包括对网络类型的特定平台检查。因此，特定于 OVN 和 SDN 的检查会显示 **failed**，而不是基于网络配置的 **not-applicable**。现在，新规则包含检查联网规则的平台检查，从而更加精确地评估特定于网络的检查。(BZ#1994609)

- 在以前的版本中，**ocp4-moderate-routes-protected-by-tls** 规则会错误地检查导致规则失败的 TLS 设置，即使连接安全 SSL TLS 协议也是如此。现在，检查会正确地评估与网络指导和配置集建议一致的 TLS 设置。(BZ#2002695)

- 在以前的版本中，在请求命名空间时 **ocp-cis-configure-network-policies-namespace** 使用分页。这会导致规则失败，因为部署会截断超过 500 个命名空间的列表。现在，会请求整个命名空间列表，检查配置的网络策略的规则将可用于部署超过 500 个命名空间的部署。(BZ#2038909)

- 在以前的版本中，使用 **sshd jinja** 宏进行补救被硬编码为特定的 sshd 配置。因此，配置与规则检查的内容不一致，检查会失败。现在，sshd 配置已被参数化，规则会成功应用。(BZ#2049141)

- 在以前的版本中，**ocp4-cluster-version-operator-verify-integrity** 始终检查 Cluter Version Operator(CVO)历史记录中的第一个条目。因此，当验证后续版本的 OpenShift Container Platform 时，升级会失败。现在，**ocp4-cluster-version-operator-verify-integrity** 的合规性检查结果可以检测到验证的版本，且与 CVO 历史记录准确。(BZ#2053602)

- 在以前的版本中，**ocp4-api-server-no-adm-ctrl-plugins-disabled** 规则没有检查空准入插件列表。因此，即使启用了所有准入插件，该规则也会始终失败。现在，对 **ocp4-api-server-no-adm-ctrl-plugins-disabled** 规则的更强大的检查会准确传递所有准入控制器插件。(BZ#2058631)

- 在以前的版本中，扫描不包含针对 Linux worker 节点运行的平台检查。因此，对不是基于 Linux 的 worker 节点运行扫描会导致永不结束扫描循环。现在，扫描将根据平台类型和标签进行适当调度，并会完全成功。(BZ#2056911)

5.2.15. OpenShift Compliance Operator 0.1.48

以下公告可用于 OpenShift Compliance Operator 0.1.48：

- RHBA-2022:0416 - OpenShift Compliance Operator 程序错误修复和功能增强更新

5.2.15.1. 程序错误修复

- 在以前的版本中，与扩展开放漏洞和评估语言(OVAL)定义关联的规则带有为 **None** 的 **checkType**。这是因为 Compliance Operator 在解析规则时没有处理扩展的 OVAL 定义。在这个版本中，通过扩展 OVAL 定义的内容会被解析，这些规则现在具有 **Node** 或 **Platform** 的 **checkType**。(BZ#2040282)

- 在以前的版本中，为 **KubeletConfig** 手动创建的 **MachineConfig** 对象会阻止为补救生成 **KubeletConfig** 对象，从而使补救处于 **Pending** 状态。在这个版本中，补救会创建一个 **KubeletConfig** 对象，无论是否有为 **KubeletConfig** 手动创建的 **MachineConfig** 对象。因此，**KubeletConfig** 补救现在可以正常工作。(BZ#2040401)

5.2.16. OpenShift Compliance Operator 0.1.47

以下公告可用于 OpenShift Compliance Operator 0.1.47：

- RHBA-2022:0014 - OpenShift Compliance Operator 程序错误修复和功能增强更新

5.2.16.1. 新功能及功能增强

- Compliance Operator 现在支持支付卡行业数据安全标准(PCI DSS)的以下合规性基准：
  - ocp4-pci-dss

- ocp4-pci-dss-node

- FedRAMP 模式影响级别的额外规则和补救被添加到 OCP4-moderate、OCP4-moderate-node 和 rhcos4-moderate 配置集中。

- KubeletConfig 的补救现在包括在节点级别的配置集中。

5.2.16.2. 程序错误修复

- 在以前的版本中，如果集群运行 OpenShift Container Platform 4.6 或更早版本，则与 USBGuard 相关的规则的补救将失败。这是因为 Compliance Operator 创建的补救基于旧版本的 USBGuard，但不支持丢弃目录。现在，为运行 OpenShift Container Platform 4.6 的集群不会创建与 USBGuard 相关的规则无效的补救。如果您的集群使用 OpenShift Container Platform 4.6，则必须为 USBGuard 相关的规则手动创建补救。
另外，只会针对满足最低版本要求的规则创建补救。(BZ#1965511)

- 在以前的版本中，当渲染补救时，合规 Operator 会使用一个太严格的正则表达式来检查补救是否良好。因此，一些补救（如呈现 **sshd_config** ）不会传递正则表达式检查，因此不会被创建。一些正则表达式已被认定为不必要并被删除。现在，补救可以正确地进行。(BZ#2033009)

5.2.17. OpenShift Compliance Operator 0.1.44

以下公告可用于 OpenShift Compliance Operator 0.1.44：

- RHBA-2021:4530 - OpenShift Compliance Operator 程序错误修复和功能增强更新

5.2.17.1. 新功能及功能增强

- 在这个发行版本中，**strictNodeScan** 选项被添加到 **ComplianceScan**、**ComplianceSuite** 和 **ScanSetting** CR 中。这个选项默认为与之前行为匹配的 **true**，当扫描无法调度到某个节点上时出现错误。将选项设置为 **false** 可让 Compliance Operator 针对调度扫描更加宽松。具有临时节点的环境可将 **strictNodeScan** 值设为 false，这可以允许进行合规性扫描，即使集群中的某些节点无法调度。

- 现在，您可以通过配置 **ScanSetting** 对象的 **nodeSelector** 和 **tolerations** 属性来自定义用于调度结果服务器工作负载的节点。这些属性用于放置 **ResultServer** pod，用于挂载 PV 存储卷并存储原始资产报告格式(ARF)结果。在以前的版本中，**nodeSelector** 和 **tolerations** 参数默认选择其中一个 control plane 节点，并容限 **node-role.kubernetes.io/master 污点**。在不允许 control plane 节点挂载 PV 的环境中工作。此功能为您提供了选择节点并容许这些环境中的不同污点的方法。

- Compliance Operator 现在可以修复 **KubeletConfig** 对象。

- 现在，添加了一个包含错误消息的注释，以帮助内容开发人员区分集群中不存在的对象与无法获取的对象。

- 规则对象现在包含两个新属性，**checkType** 和 **description**。通过这些属性，您可以确定与节点检查或平台检查相关的规则，并允许您检查规则的作用。

- 此功能增强删除了需要扩展现有配置集的要求，以创建定制的配置集。这意味着 **TailoredProfile** CRD 中的 **extends** 字段不再是必需的。现在，您可以选择用于创建定制配置集的规则对象列表。请注意，您必须通过设置 **compliance.openshift.io/product-type:** 注解或为 **TailoredProfile** CR 设置 **-node** 后缀来选择您的配置集是否应用到节点或平台。

- 在本发行版本中，Compliance Operator 现在可以在其污点的所有节点上调度扫描。在以前的版本中，扫描 pod 只容许 **node-role.kubernetes.io/master 污点**，这意味着它们可以在没有污点

的节点上运行，或者仅在具有 **node-role.kubernetes.io/master** 污点的节点上运行。在将自定义污点用于其节点的部署中，这会导致扫描不会被调度到这些节点上。现在，扫描 pod 可以容限所有节点污点。

- 在本发行版本中，Compliance Operator 支持以下 North American Electric Reliability Corporation (NERC 安全配置集：
  - ocp4-nerc-cip
  - ocp4-nerc-cip-node
  - rhcos4-nerc-cip
- 在本发行版本中，Compliance Operator 支持 Red Hat OpenShift - Node level、ocp4-moderate-node、security 配置集中的 NIST 800-53 Moderate-Impact Baseline。

5.2.17.2. 模板和变量使用

- 在本发行版本中，补救模板现在允许多值变量。
- 在这个版本中，Compliance Operator 可以根据合规性配置集中设置的变量更改补救。这可用于包括特定于部署的补救值，如超时、NTP 服务器主机名或类似值。另外，**ComplianceCheckResult** 对象现在使用标签 **compliance.openshift.io/check-has-value** 列出检查使用的变量。

5.2.17.3. 程序错误修复

- 在以前的版本中，在执行扫描时，pod 的扫描程序容器中会发生意外终止。在本发行版本中，Compliance Operator 使用最新的 OpenSCAP 版本 1.3.5 来避免崩溃。
- 在以前的版本中，使用 **autoReplyRemediations** 应用补救会触发对集群节点的更新。如果某些补救不包括所有所需的输入变量，则会出现破坏性。现在，如果缺少一个或多个所需的输入变量，则会为其分配一个 **NeedsReview** 状态。如果一个或多个补救处于 **NeedsReview** 状态，机器配置池会保持暂停，且在设置所有需要的变量前不会应用补救。这有助于最小化对节点的中断。
- 用于 Prometheus 指标的 RBAC 角色和角色绑定改为 'ClusterRole' 和 'ClusterRoleBinding'，以确保监控在没有自定义的情况下正常工作。
- 在以前的版本中，如果在解析配置集时出现错误，规则或变量对象会从配置集中删除并删除。现在，如果解析过程中出现错误，**profileparser** 会使用临时注解为对象添加注解，该注解会阻止对象在解析完成后被删除。(BZ#1988259)
- 在以前的版本中，当自定义配置集中没有标题或描述时会出现一个错误。由于 XCCDF 标准需要定制配置集的标题和描述，现在需要在 **TailoredProfile** CR 中设置标题和描述。
- 在以前的版本中，当使用定制配置集时，允许使用特定的选择设置 **TailoredProfile** 变量值。这个限制现已被删除，**TailoredProfile** 变量可以设置为任何值。

5.2.18. Compliance Operator 0.1.39 发行注记

以下公告可用于 OpenShift Compliance Operator 0.1.39：

- RHBA-2021:3214 - OpenShift Compliance Operator 程序漏洞修复和功能增强更新

5.2.18.1. 新功能及功能增强

- 在以前的版本中，Compliance Operator 无法解析 Payment Card Industry Data Security Standard（PCI DSS）参考。现在，Operator 可以解析 PCI DSS 配置集提供的合规性内容。
- 在以前的版本中，Compliance Operator 无法在 moderate 配置集中为 AU-5 控制执行规则。现在，权限被添加到 Operator 中，以便它可以读取 **Prometheusrules.monitoring.coreos.com** 对象，并运行在 moderate 配置集中涵盖 AU-5 控制的规则。

5.2.19. 其他资源

- 了解 Compliance Operator

5.3. COMPLIANCE OPERATOR 概念

5.3.1. 了解 Compliance Operator

通过 Compliance Operator，OpenShift Container Platform 管理员可以描述集群所需的合规状态，并概述缺陷及修复方法。Compliance Operator 评估 OpenShift Container Platform 的 Kubernetes API 资源以及运行集群的节点的合规性。Compliance Operator 使用 NIST 认证工具 OpenSCAP 扫描并执行内容所提供的安全性策略。

重要

Compliance Operator 仅适用于 Red Hat Enterprise Linux CoreOS（RHCOS）部署。

5.3.1.1. Compliance Operator 配置集

有多个配置集可用于安装 Compliance Operator。您可以使用 **oc get** 命令来查看可用的配置集、配置集详情和特定规则。

- 查看可用的配置集：

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

输**出示例**

```
NAME                 AGE
ocp4-cis             94m
ocp4-cis-node        94m
ocp4-e8              94m
ocp4-high            94m
ocp4-high-node       94m
ocp4-moderate        94m
ocp4-moderate-node   94m
ocp4-nerc-cip        94m
ocp4-nerc-cip-node   94m
ocp4-pci-dss         94m
ocp4-pci-dss-node    94m
rhcos4-e8            94m
rhcos4-high          94m
rhcos4-moderate      94m
rhcos4-nerc-cip      94m
```

这些配置集代表不同的合规性基准。每个配置集将其应用到的产品名称添加为配置集名称的前缀。**ocp4-e8** 将 Essential 8 基准应用到 OpenShift Container Platform 产品，而 **rhcos4-e8** 将 Essential 8 基准应用到 Red Hat Enterprise Linux CoreOS (RHCOS) 产品。

- 运行以下命令，以查看 **rhcos4-e8** 配置集的详情：

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

例 5.1. 输出示例

```
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
```

- 运行以下命令，以查看 **rhcos4-audit-rules-login-events** 规则的详情：

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

例 5.2. 输出示例

```
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.
```


5.3.1.1.1. Compliance Operator 配置集类型

可用的合规性配置集有两种：Platform（平台）和 Node（节点）。

平台

平台扫描目标 OpenShift Container Platform 集群。

节点

节点扫描集群节点目标。

重要

对于具有 Node 和 Platform 应用程序的合规性配置集，如 **pci-dss** 合规性配置集，您必须在 OpenShift Container Platform 环境中运行。

5.3.1.2. 其他资源

- 支持的合规性配置集

5.3.2. 了解自定义资源定义

OpenShift Container Platform 中的 Compliance Operator 为您提供了几个自定义资源定义(CRD)来完成合规性扫描。要运行合规性扫描，它会利用预定义的安全策略，该策略从 ComplianceAsCode 社区项目衍生而来。Compliance Operator 把这些安全策略转换为 CRD，您可以使用它来运行合规性扫描，并为发现的问题获取补救。

5.3.2.1. CRD 工作流

CRD 为您提供了以下工作流来完成合规性扫描：

1. 定义合规性扫描要求
2. 配置合规性扫描设置
3. 使用合规性扫描设置处理合规性要求
4. 监控合规性扫描
5. 检查合规性扫描结果

5.3.2.2. 定义合规性扫描要求

默认情况下，Compliance Operator CRD 包含 **ProfileBundle** 和 **Profile** 对象，您可以在其中定义和设置合规性扫描要求的规则。您还可以使用 **TailoredProfile** 对象自定义默认配置集。

5.3.2.2.1. ProfileBundle 对象

安装 Compliance Operator 时，它包含 ready-to-run **ProfileBundle** 对象。Compliance Operator 解析 **ProfileBundle** 对象，并为捆绑包中的每个配置集创建一个 **Profile** 对象。它还会解析 **Rule** 和 **Variable** 对象，这些对象会被 **Profile** 对象使用。

ProfileBundle 对象示例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
  name: <profile bundle name>
  namespace: openshift-compliance
status:
  dataStreamStatus: VALID (1)
```

(1) 指明 Compliance Operator 是否能够解析内容文件。

注意

当 **contentFile** 失败时，会出现一个 **errorMessage** 属性，它提供所发生错误的详细信息。

故障排除

当您从无效的镜像回滚到已知内容镜像时，**ProfileBundle** 对象停止响应并显示 **PENDING** 状态。作为临时解决方案，您可以移到与上一个镜像不同的镜像。另外，您可以删除并重新创建 **ProfileBundle** 对象以返回到工作状态。

5.3.2.2.2. 配置集对象

Profile 对象定义可以为某个合规性标准评估的规则和变量。它包含 OpenSCAP 配置集的解析详情，如其 XCCDF 标识符和配置集会检查 **Node** 或 **Platform** 类型。您可以直接使用 **Profile** 对象，也可以使用 **TailorProfile** 对象进一步自定义它。

注意

您无法手动创建或修改 **Profile** 对象，因为它是从单个 **ProfileBundle** 对象衍生而来。通常，单个 **ProfileBundle** 对象可以包含多个 **Profile** 对象。

Profile 对象示例

```
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. 指定配置集的 XCCDF 名称。当您将 **ComplianceScan** 对象定义为扫描的配置集属性值时，请使用这个标识符。
2. 指定 **Node** 或 **Platform**。节点配置集扫描集群节点和平台配置集扫描 Kubernetes 平台。
3. 指定配置集的规则列表。每个规则都对应一个检查。

5.3.2.2.3. 规则对象

Rule 对象形成了配置文件，也会作为对象公开。使用 **Rule** 对象定义合规检查要求，并指定它是如何修复的。

Rule 对象示例

```
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. 指定检查此规则执行的类型。**Node** 配置集扫描集群节点和 **Platform** 配置集扫描 Kubernetes 平台。空值表示没有自动检查。
2. 指定规则的 XCCDF 名称，该规则直接从 datastream 解析。
3. 指定规则在失败时的严重性。

注意

Rule 对象获取适当的标签，以便轻松识别关联的 **ProfileBundle** 对象。**ProfileBundle** 也在此对象的 **OwnerReferences** 中指定。

5.3.2.2.4. TailoredProfile 对象

使用 **TailoredProfile** 对象根据您的机构要求修改默认的 **Profile** 对象。您可以启用或禁用规则，设置变量值，并为自定义提供合理化。验证后，**TailoredProfile** 对象会创建一个 **ConfigMap**，它可以被 **ComplianceScan** 对象引用。

提示

您可以通过在 **ScanSettingBinding** 对象中引用 **TailoredProfile** 对象来使用 TailoredProfile 对象。有关 **ScanSettingBinding** 的更多信息，请参阅 ScanSettingBinding 对象。

TailoredProfile 对象示例

```
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) 这是可选的。构建 **TailoredProfile** 的 **Profile** 对象的名称。如果没有设置值，则会从 **enableRules** 列表创建一个新配置集。

(2) 指定定制配置集的 XCCDF 名称。

(3) 指定 **ConfigMap** 名称，可用作 **ComplianceScan** 的 **tailoringConfigMap.name** 属性的值。

(4) 显示对象的状态，如 **READY**、**PENDING** 和 **FAILURE**。如果对象的状态为 **ERROR**，则属性 **status.errorMessage** 会为失败提供原因。

使用 **TailoredProfile** 对象时，可以使用 **TailoredProfile** 构造来创建新的 **Profile** 对象。要创建新 **配置集**，请设置以下配置参数：

- 合适的标题
- **extends** 值必须为空
- **TailoredProfile** 对象上的扫描类型注解：

  ```
  compliance.openshift.io/product-type: Platform/Node
  ```

注意

如果您没有设置 **product-type** 注解，Compliance Operator 会默认使用 **Platform** 扫描类型。在 **TailoredProfile** 对象的名称中添加 **-node** 后缀会导致 **node** 扫描类型。

5.3.2.3. 配置合规性扫描设置

在定义了合规性扫描的要求后，您可以通过指定扫描类型、扫描和扫描位置来配置它。要做到这一点，Compliance Operator 为您提供了 **ScanSetting** 对象。

5.3.2.3.1. ScanSetting 对象

使用 **ScanSetting** 对象定义并重用操作策略来运行扫描。默认情况下，Compliance Operator 会创建以下 **ScanSetting** 对象：

- **default** - 它在 master 和 worker 节点（使用 1Gi PV）上每天的 1 AM 运行一次扫描，并保留最后三条结果。补救都不会自动更新。
- **default-auto-apply** - 它在 control plane 和 worker 节点（使用 1Gi PV）上每天 1AM 运行一次扫描，并保留最后三个结果。**autoApplyRemediations** 和 **autoUpdateRemediations** 设置为 true。

ScanSetting 对象示例

```
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 设置为 **true** 以启用自动补救。设置为 **false** 可禁用自动补救。

2 设置为 **true**，为内容更新启用自动补救。设置为 **false** 以禁用内容更新自动补救。

3 以原始结果格式指定存储的扫描数量。默认值为 **3**。随着旧的结果轮转，管理员必须在轮转发生之前存储其他结果。

4 指定应当为扫描创建存储大小以存储原始结果。默认值为 **1Gi**

6 指定扫描应以 cron 格式运行的频率。

注意

要禁用轮转策略，请将值设为 **0**。

5 指定 **node-role.kubernetes.io** 标签值，以调度 **Node** 类型的扫描。这个值必须与 **MachineConfigPool** 的名称匹配。

5.3.2.4. 使用合规性扫描设置处理合规性扫描要求

当您定义了合规性扫描要求并将设置配置为运行扫描时，Compliance Operator 会使用 **ScanSettingBinding** 对象处理它。

5.3.2.4.1. ScanSettingBinding 对象

使用 **ScanSettingBinding** 对象指定您的合规要求，并引用 **Profile** 或 **TailoredProfile** 对象。然后，它会链接到一个 **ScanSetting** 对象，它为扫描提供操作限制。然后，Compliance Operator 根据 **ScanSetting** 和 **ScanSettingBinding** 对象生成 **ComplianceSuite** 对象。

ScanSettingBinding 对象示例

```
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 指定 **Profile** 或 **TailoredProfile** 对象的详情，以扫描您的环境。

2 指定操作限制，如调度和存储大小。

创建 **ScanSetting** 和 **ScanSettingBinding** 对象会导致合规性套件。要获取合规套件列表，请运行以下命令：

```
$ oc get compliancesuites
```

重要

如果删除了 **ScanSettingBinding**，则也会删除合规性套件。

5.3.2.5. 跟踪合规性扫描

创建合规套件后，您可以使用 **ComplianceSuite** 对象监控部署的扫描的状态。

5.3.2.5.1. ComplianceSuite 对象

ComplianceSuite 对象可帮助您跟踪扫描的状态。它包含创建扫描的原始设置以及总体结果。

对于 **Node** 类型扫描，您应该将扫描映射到 **MachineConfigPool**，因为它包含任何问题的补救。如果指定了标签，请确保它直接应用到池。

ComplianceSuite 对象示例

```
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 设置为 **true** 以启用自动补救。设置为 **false** 可禁用自动补救。

2 指定扫描应以 cron 格式运行的频率。

3 指定要在集群中运行的扫描规格列表。

4 表示扫描的进度。

5 表示套件的整体结果。

后台中的套件会根据 **scan** 参数创建 **ComplianceScan** 对象。您可以以编程方式获取 **ComplianceSuites** 事件。要获取套件的事件，请运行以下命令：

```
$ oc get events --field-selector involvedObject.kind=ComplianceSuite,involvedObject.name=<name of the suite>
```

重要

在手动定义 **ComplianceSuite** 时可能会创建错误，因为它包含 XCCDF 属性。

5.3.2.5.2. 高级 ComplianceScan 对象

Compliance Operator 包括用于调试或与现有工具集成的高级用户的选项。虽然建议您不要直接创建一个 **ComplianceScan** 对象，但您可以使用 **ComplianceSuite** 对象来管理它。

高级 ComplianceScan 对象示例

```
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 指定 **Node** 或 **Platform**。节点配置集扫描集群节点和平台配置集扫描 Kubernetes 平台。

2 指定您要运行的配置集的 XCCDF 标识符。

3 指定封装配置集文件的容器镜像。

4 它是可选的。指定要运行单个规则的扫描。该规则必须使用 XCCDF ID 标识，并且必须属于指定的配置集。

注意

如果您跳过 **rule** 参数，则针对指定配置集的所有可用规则运行扫描。

5 如果您在 OpenShift Container Platform 上，并希望生成补救，则 nodeSelector 标签必须与 **MachineConfigPool** 标签匹配。

注意

如果没有指定 **nodeSelector** 参数或与 **MachineConfig** 标签匹配，则扫描仍将运行，但不会创建补救。

6 指示扫描的当前阶段。

7 表示扫描的结果。

重要

如果您删除了 **ComplianceSuite** 对象，则所有关联的扫描都会被删除。

扫描完成后，它将生成结果作为 **ComplianceCheckResult** 对象的自定义资源。但是，原始结果以 ARF 格式提供。这些结果存储在持久性卷(PV)中，它具有与扫描名称关联的持久性卷声明(PVC)。您可以以编程方式获取 **ComplianceScans** 事件。要为套件生成事件，请运行以下命令：

```
oc get events --field-selector involvedObject.kind=ComplianceScan,involvedObject.name=<name of the suite>
```

5.3.2.6. 查看合规性结果

当合规性套件达到 **DONE** 阶段时，您可以查看扫描结果和可能的补救方法。

5.3.2.6.1. ComplianceCheckResult 对象

当使用特定配置集运行扫描时，会验证配置集中的多个规则。对于每个规则，都会创建一个 **ComplianceCheckResult** 对象，它为特定规则提供集群状态。

ComplianceCheckResult 对象示例

```
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. 描述扫描检查的严重性。
2. 描述检查的结果。可能的值有：
   - PASS：检查成功。
   - FAIL：检查不成功。
   - INFO：检查成功，发现一些不严重的、不被视为错误的问题。
   - MANUAL：检查无法自动评估状态，需要手动检查。
   - INCONSISTENT：不同的节点报告不同的结果.
   - ERROR：检查运行成功，但无法完成。
   - NOTAPPLICABLE: 检查没有运行，因为它不适用。

要获得套件中的所有检查结果，请运行以下命令：

```
oc get compliancecheckresults \
-l compliance.openshift.io/suite=workers-compliancesuite
```

5.3.2.6.2. ComplianceRemediation 对象

对于特定的检查，您可以有一个 datastream 指定的修复。但是，如果 Kubernetes 修复可用，则 Compliance Operator 会创建一个 **ComplianceRemediation** 对象。

ComplianceRemediation 对象示例

```
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** 表示应用了补救。**false** 表示没有应用补救。

2 包括补救的定义。

3 表示之前从以前的内容版本解析的补救。Compliance Operator 仍然保留过时的对象，以便管理员在应用新补救前有机会查看新的补救。

要从套件中获得所有补救，请运行以下命令：

```
oc get complianceremediations \
-l compliance.openshift.io/suite=workers-compliancesuite
```

要列出可自动修复的所有失败检查，请运行以下命令：

```
oc get compliancecheckresults \
-l 'compliance.openshift.io/check-status in (FAIL),compliance.openshift.io/automated-remediation'
```

要列出可手动修复的所有失败检查，请运行以下命令：

```
oc get compliancecheckresults \
-l 'compliance.openshift.io/check-status in (FAIL),!compliance.openshift.io/automated-remediation'
```

5.4. COMPLIANCE OPERATOR 管理

5.4.1. 安装 Compliance Operator

在使用 Compliance Operator 之前，您必须保证在集群中部署它。

重要

Compliance Operator 可能会报告有关受管平台（如 OpenShift Dedicated、Red Hat OpenShift Service on AWS Classic）和 Microsoft Azure Red Hat OpenShift 的不正确的结果。如需更多信息，请参阅知识库文章 Compliance Operator 报告受管服务的错误结果。

5.4.1.1. 通过 Web 控制台安装 Compliance Operator

先决条件

- 您必须具有 **admin** 权限。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 **Operators → OperatorHub**。
2. 搜索 Compliance Operator，然后点 **Install**。
3. 保留 **安装模式** 和 **命名空间** 的默认选择，以确保将 Operator 安装到 **openshift-compliance** 命名空间中。
4. 点 **Install**。

验证

确认安装成功：

1. 导航到 **Operators → Installed Operators** 页面。
2. 检查 Compliance Operator 是否已安装在 **openshift-compliance** 命名空间中，其状态为 **Succeeded**。

如果 Operator 没有成功安装：

1. 导航到 **Operators → Installed Operators** 页面，并检查 **Status** 列中是否有任何错误或故障。
2. 导航到 **Workloads → Pods** 页面，检查 **openshift-compliance** 项目中报告问题的 pod 的日志。

重要

如果 **restricted** 安全性上下文约束(SCC)已被修改为包含 **system:authenticated** 组或添加了 **requiredDropCapabilities**，则 Compliance Operator 可能会因为权限问题而无法正常工作。

您可以为 Compliance Operator scanner Pod 服务帐户创建自定义 SCC。如需更多信息，请参阅为 Compliance Operator 创建自定义 SCC 。

5.4.1.2. 使用 CLI 安装 Compliance Operator

先决条件

- 您必须具有 **admin** 权限。

流程

1. 定义一个 **Namespace** 对象：

   namespace-object.yaml示例

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     labels:
       openshift.io/cluster-monitoring: "true"
       pod-security.kubernetes.io/enforce: privileged (1)
     name: openshift-compliance
   ```

   (1) 在 OpenShift Container Platform 4.16 中，pod 安全标签必须在命名空间级别设置为 **privileged**。

2. 创建 **Namespace** 对象：

   ```
   $ oc create -f namespace-object.yaml
   ```

3. 定义一个 **OperatorGroup** 对象：

   operator-group-object.yaml示例

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

4. 创建 **OperatorGroup** 对象：

   ```
   $ oc create -f operator-group-object.yaml
   ```

5. 定义一个 **Subscription** 对象：

   subscription-object.yaml示例

   ```
   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. 创建 **Subscription** 对象：

```
$ oc create -f subscription-object.yaml
```

注意

如果要设置全局调度程序功能并启用 **defaultNodeSelector**，您必须手动创建命名空间并更新 **openshift-compliance** 命名空间的注解，或安装 Compliance Operator 的命名空间，使用 **openshift.io/node-selector: ""**。这会删除默认节点选择器并防止部署失败。

验证

1. 通过检查 CSV 文件来验证安装是否成功：

```
$ oc get csv -n openshift-compliance
```

2. 验证 Compliance Operator 是否正在运行：

```
$ oc get deploy -n openshift-compliance
```

5.4.1.3. 在 ROSA 托管 control plane (HCP) 上安装 Compliance Operator

从 Compliance Operator 1.5.0 发行版本开始，Operator 会使用 Hosted control plane 针对 Red Hat OpenShift Service on AWS 测试。

在 Red Hat OpenShift Service on AWS 上托管的 control plane 集群限制对由红帽管理的 control plane 的访问。默认情况下，Compliance Operator 将调度到 **master** 节点池中的节点，该节点在 Red Hat OpenShift Service on AWS Hosted control plane 安装中不可用。这要求您配置 **Subscription** 对象，以便 Operator 在可用节点池中调度。此步骤是在 Red Hat OpenShift Service on AWS Hosted control plane 集群上成功安装所必需的。

先决条件

- 您必须具有 **admin** 权限。

流程

1. 定义一个 **Namespace** 对象：

namespace-object.yaml 文件示例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-compliance
```

(1) 在 OpenShift Container Platform 4.16 中，pod 安全标签必须在命名空间级别设置为 **privileged**。

2. 运行以下命令来创建 **Namespace** 对象：

```
$ oc create -f namespace-object.yaml
```

3. 定义一个 **OperatorGroup** 对象：

operator-group-object.yaml 文件示例

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

4. 运行以下命令来创建 **OperatorGroup** 对象：

```
$ oc create -f operator-group-object.yaml
```

5. 定义一个 **Subscription** 对象：

subscription-object.yaml 文件示例

```
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) 更新 Operator 部署，以便在 **worker** 节点上部署。

6. 运行以下命令来创建 **Subscription** 对象：

```
$ oc create -f subscription-object.yaml
```

验证

1. 运行以下命令检查集群服务版本 (CSV) 文件来验证安装是否成功：

```
$ oc get csv -n openshift-compliance
```

2. 运行以下命令验证 Compliance Operator 是否正在运行：

```
$ oc get deploy -n openshift-compliance
```

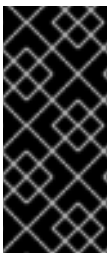
重要

如果 **restricted** 安全性上下文约束(SCC)已被修改为包含 **system:authenticated** 组或添加了 **requiredDropCapabilities**，则 Compliance Operator 可能会因为权限问题而无法正常工作。

您可以为 Compliance Operator scanner Pod 服务帐户创建自定义 SCC。如需更多信息，请参阅为 Compliance Operator 创建自定义 SCC 。

5.4.1.4. 在 Hypershift 托管 control plane 上安装 Compliance Operator

通过创建一个 **Subscription** 文件，可以使用 OperatorHub 在托管 control plane 中安装 Compliance Operator。



重要

托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

先决条件

- 您必须具有 **admin** 权限。

流程

1. 定义类似如下的 **Namespace** 对象：

   namespace-object.yaml示例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-compliance
```

   (1) 在 OpenShift Container Platform 4.16 中，pod 安全标签必须在命名空间级别设置为 **privileged**。

2. 运行以下命令来创建 **Namespace** 对象：

```
$ oc create -f namespace-object.yaml
```

3. 定义一个 **OperatorGroup** 对象：

operator-group-object.yaml示例

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

4. 运行以下命令来创建 **OperatorGroup** 对象：

```
$ oc create -f operator-group-object.yaml
```

5. 定义一个 **Subscription** 对象：

subscription-object.yaml示例

```
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. 运行以下命令来创建 **Subscription** 对象：

```
$ oc create -f subscription-object.yaml
```

验证

1. 运行以下命令，检查 CSV 文件来验证安装是否成功：

```
$ oc get csv -n openshift-compliance
```

2. 运行以下命令验证 Compliance Operator 是否正在运行：

```
$ oc get deploy -n openshift-compliance
```

其他资源

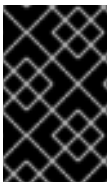
- 托管 control plane 概述

5.4.1.5. 其他资源

- Compliance Operator 在受限网络环境中被支持。如需更多信息，请参阅在受限网络中使用 Operator Lifecycle Manager。

5.4.2. 更新 Compliance Operator

作为集群管理员，您可以在 OpenShift Container Platform 集群上更新 Compliance Operator。



重要

将 OpenShift Container Platform 集群更新至 4.14 版本可能会导致 Compliance Operator 无法按预期工作。这是因为一个持续的已知问题。如需更多信息，请参阅 OCPBUGS-18025。

5.4.2.1. 准备 Operator 更新

已安装的 Operator 的订阅指定一个更新频道，用于跟踪和接收 Operator 的更新。您可以更改更新频道，以开始跟踪并从更新频道接收更新。

订阅中更新频道的名称可能会因 Operator 而异，但应遵守给定 Operator 中的常规约定。例如，频道名称可能会遵循 Operator 提供的应用程序的次发行版本更新流（**1.2**、**1.3**）或发行的频率（**stable**、**fast**）。

注意

您不能将已安装的 Operator 更改为比当前频道旧的频道。

红帽客户门户网站 Labs 包括以下应用程序，可帮助管理员准备更新其 Operator：

- Red Hat OpenShift Container Platform Operator Update Information Checker

您可以使用应用程序搜索基于 Operator Lifecycle Manager 的 Operator，并在不同版本的 OpenShift Container Platform 中验证每个更新频道的可用 Operator 版本。不包含基于 Cluster Version Operator 的 Operator。

5.4.2.2. 更改 Operator 的更新频道

您可以使用 OpenShift Container Platform Web 控制台更改 Operator 的更新频道。

提示

如果订阅中的批准策略被设置为 **Automatic**，则更新过程会在所选频道中提供新的 Operator 版本时立即启动。如果批准策略设为 **Manual**，则必须手动批准待处理的更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Operators → Installed Operators**。

2. 点击您要更改更新频道的 Operator 名称。
3. 点 **Subscription** 标签页。
4. 点 **Update channel** 下的更新频道名称。
5. 点要更改的更新频道，然后点 **Save**。
6. 对于带有 **自动批准策略** 的订阅，更新会自动开始。返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。
对于采用**手动**批准策略的订阅，您可以从 **Subscription** 选项卡中手动批准更新。

5.4.2.3. 手动**批准待**处**理的 Operator 更新**

如果已安装的 Operator 的订阅被设置为 **Manual**，则当其当前更新频道中发布新更新时，在开始安装前必须手动批准更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，进入 **Operators → Installed Operators**。
2. 处于待定更新的 Operator 会显示 **Upgrade available** 状态。点您要更新的 Operator 的名称。
3. 点 **Subscription** 标签页。任何需要批准的更新都会在 **Upgrade status** 旁边显示。例如：它可能会显示 **1 requires approval**。
4. 点 **1 requires approval**，然后点 **Preview Install Plan**。
5. 检查列出可用于更新的资源。在满意后，点 **Approve**。
6. 返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。

5.4.3. 管理 Compliance Operator

本节介绍安全性内容的生命周期，包括如何使用合规性内容的更新版本以及如何创建自定义 **ProfileBundle** 对象。

5.4.3.1. ProfileBundle CR 示例

ProfileBundle 对象需要两个信息：包含 **contentImage** 的容器镜像的 URL 以及包含合规性内容的文件。**contentFile** 参数相对于文件系统的根目录。您可以定义内置的 **rhcos4 ProfileBundle** 对象，如下例所示：

```
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 包含合规性内容的文件位置。

2 内容镜像位置。

重要

用于内容镜像的基础镜像必须包含 **coreutils**。

5.4.3.2. 更新安全性内容

安全内容作为 **ProfileBundle** 对象引用的容器镜像包括。要准确跟踪从捆绑包（如 Rules 或 Profiles）解析的 **ProfileBundles** 和 CustomResources 的更新，请使用摘要而不是标签来识别包含合规性内容的容器镜像：

```
$ oc -n openshift-compliance get profilebundles rhcos4 -oyaml
```

输**出示例**

```
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 安全性容器镜像。

每个 **ProfileBundle** 都由一个部署来支持。当 Compliance Operator 检测到容器镜像摘要已更改时，会更新部署来反映内容的变化并再次解析内容。使用摘要而不是标签可确保您使用稳定且可预测的配置集集合。

5.4.3.3. 其他资源

- Compliance Operator 在受限网络环境中被支持。如需更多信息，请参阅在受限网络中使用 Operator Lifecycle Manager。

5.4.4. 卸载 Compliance Operator

您可以使用 OpenShift Container Platform Web 控制台或 CLI 从集群中删除 OpenShift Compliance Operator。

5.4.4.1. 使用 Web 控制台从 OpenShift Container Platform 卸载 OpenShift Compliance Operator

要删除 Compliance Operator，您必须首先删除命名空间中的对象。删除对象后，您可以通过删除 **openshift-compliance** 项目来删除 Operator 及其命名空间。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 必须安装 OpenShift Compliance Operator。

流程

使用 OpenShift Container Platform Web 控制台删除 Compliance Operator：

1. 进入 **Operators → Installed Operators → Compliance Operator** 页面。
   a. 点 **All instances**。
   b. 在 **All namespaces** 中，点 Options 菜单  删除所有 ScanSettingBinding、ComplainceSuite、ComplianceScan 和 ProfileBundle 对象。
2. 切换到 **Administration → Operators → Installed Operators** 页面。
3. 点 **Compliance Operator** 条目  中的 Options 菜单并选择 **Uninstall Operator**。
4. 切换到 **Home → Projects** 页面。

5. 搜索 'compliance'。

6. 点 **openshift-compliance** 项目  旁边的 Options 菜单，然后选择 **Delete Project**。

   a. 通过在对话框中输入 **openshift-compliance** 并点 **Delete** 来确认删除。

5.4.4.2. 使用 CLI 从 OpenShift Container Platform 卸载 OpenShift Compliance Operator

要删除 Compliance Operator，您必须首先删除命名空间中的对象。删除对象后，您可以通过删除 **openshift-compliance** 项目来删除 Operator 及其命名空间。

先决条件

- 使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 必须安装 OpenShift Compliance Operator。

流程

1. 删除命名空间中的所有对象。

   a. 删除 **ScanSettingBinding** 对象：

   ```
   $ oc delete ssb --all -n openshift-compliance
   ```

   b. 删除 **ScanSetting** 对象：

   ```
   $ oc delete ss --all -n openshift-compliance
   ```

   c. 删除 **ComplianceSuite** 对象：

   ```
   $ oc delete suite --all -n openshift-compliance
   ```

   d. 删除 **ComplianceScan** 对象：

   ```
   $ oc delete scan --all -n openshift-compliance
   ```

   e. 删除 **ProfileBundle** 对象：

   ```
   $ oc delete profilebundle.compliance --all -n openshift-compliance
   ```

2. 删除 Subscription 对象：

   ```
   $ oc delete sub --all -n openshift-compliance
   ```

3. 删除 CSV 对象：

   ```
   $ oc delete csv --all -n openshift-compliance
   ```

4. 删除项目：

```
$ oc delete project openshift-compliance
```

输**出示例**

```
project.project.openshift.io "openshift-compliance" deleted
```

验证

1. 确认已删除命名空间：

```
$ oc get project/openshift-compliance
```

输**出示例**

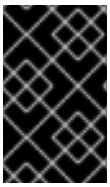
```
Error from server (NotFound): namespaces "openshift-compliance" not found
```

5.5. COMPLIANCE OPERATOR 扫描管理

5.5.1. 支持的合规性配置集

有多个配置集可用于安装 Compliance Operator(CO)。虽然您可以使用以下配置集来评估集群中的差距，但单独使用不会推断或保证与特定配置集的合规性，而不是一个审核器。

为了遵循这些各种标准，您需要与授权的审核员（如限定安全评估器(QSA)、联合授权授权局(JAB)或其他行业认可的监管机构）合作来评估您的环境。您需要与授权的审核员合作，以达到符合标准的要求。



重要

Compliance Operator 可能会报告一些受管平台（如 OpenShift Dedicated 和 Azure Red Hat OpenShift）的不正确的结果。如需更多信息，请参阅红帽知识库解决方案 #6983418。

5.5.1.1. 合规性配置集

Compliance Operator 提供以下合规配置集：

表 5.1. 支持的合规性配置集

profile	配置集标题	Application	Compliance Operator 版本	行业标准基准	支持的构架	支持的平台
rhcos4-stig	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift	节点	1.3.0+	DISA-STIG [1]	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-stig-node	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift	节点	1.3.0+	DISA-STIG [1]	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-stig	Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Openshift	平台	1.3.0+	DISA-STIG [1]	**x86_64**	
ocp4-cis-1-4	CIS Red Hat OpenShift Container Platform 4 Benchmark v1.4.0	平台	1.2.0+	CIS Benchmarks ™ [1]	**x86_64 ppc64le s390x**	

profile	配置集标题	Application	Compliance Operator 版本	行业标准基准	支持的构架	支持的平台
ocp4-cis-node-1-4	CIS Red Hat OpenShift Container Platform 4 Benchmark v1.4.0	节点 [2]	1.2.0+	CIS Benchmarks ™ [1]	**x86_64 ppc64le s390x**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-cis	CIS Red Hat OpenShift Container Platform 4 Benchmark v1.5.0	平台	1.4.1+	CIS Benchmarks ™ [1]	**x86_64 ppc64le s390x**	
ocp4-cis-node	CIS Red Hat OpenShift Container Platform 4 Benchmark v1.5.0	节点 [2]	1.4.1+	CIS Benchmarks ™ [1]	**x86_64 ppc64le s390x**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	平台	0.1.39+	ACSC 强化 Linux 工作站和服务器	**x86_64**	

profile	配置集标题	Application	Compliance Operator 版本	行业标准基准	支持的构架	支持的平台
ocp4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Platform 级别	平台	0.1.39+	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x**	
rhcos4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	节点	0.1.39+	ACSC 强化 Linux 工作站和服务器	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
rhcos4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	节点	0.1.39+	NIST SP-800-53 Release Search	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-moderate-node	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - 节点级别	节点 [2]	0.1.44+	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+

profile	配置集标题	Application	Compliance Operator 版本	行业标准基准	支持的构架	支持的平台
ocp4-nerc-cip	Red Hat OpenShift Container Platform 的 North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity 标准 - 平台级别	平台	0.1.44+	NERC CIP 标准	**x86_64**	
ocp4-nerc-cip-node	Red Hat OpenShift Container Platform 的 North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity 标准配置集 - 节点级别	节点[2]	0.1.44+	NERC CIP 标准	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
rhcos4-nerc-cip	Red Hat Enterprise Linux CoreOS 的 North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity 标准配置集	节点	0.1.44+	NERC CIP 标准	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-pci-dss	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	平台	0.1.47+	PCI 安全标准 ® Council 文档库	**x86_64 ppc64le s390x**	

profile	配置集标题	Application	Compliance Operator 版本	行业标准基准	支持的构架	支持的平台
ocp4-pci-dss-node	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	节点 [2]	0.1.47+	PCI 安全标准 ® Council 文档库	**x86_64 ppc64le s390x**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
ocp4-high	NIST 800-53 HighImpact Baseline for Red Hat OpenShift - Platform 级别	平台	0.1.52+	NIST SP-800-53 Release Search	**x86_64**	
ocp4-high-node	NIST 800-53 HighImpact Baseline for Red Hat OpenShift - 节点级别	节点 [2]	0.1.52+	NIST SP-800-53 Release Search	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+
rhcos4-high	NIST 800-53 high-Impact Baseline for Red Hat Enterprise Linux CoreOS	节点	0.1.52+	NIST SP-800-53 Release Search	**x86_64**	带有托管 control plane (ROSA HCP) 的 Red Hat OpenShift Service on AWS - 需要 1.5.0+

1. 要找到 CIS OpenShift Container Platform v4 Benchmark，进入 CIS Benchmarks 并点 **Download Latest CIS Benchmark**，然后注册以下载基准。
2. 节点配置集必须与相关的 Platform 配置集一起使用。如需更多信息，请参阅 *Compliance Operator 配置集类型*。

5.5.1.1.1. 关于扩展合规配置集

有些合规配置集会包括需要遵循行业最佳实践的控制，从而导致一些配置集会扩展其他配置集。将 Center for Internet Security (CIS) 的最佳实践与 National Institute of Standards and Technology (NIST) 安全架构结合，可以建立一个安全且合规的环境。

例如，NIST High-Impact 和 Moderate-Impact 配置集将 CIS 配置集扩展来实现合规性。因此，通过扩展合规配置集，就无需在单一集群中运行这两个配置集。

表 5.2. 配置集扩展

profile	扩展
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

5.5.1.2. 其他资源

- Compliance Operator 配置集类型

5.5.2. Compliance Operator 扫描

建议使用 **ScanSetting** 和 **ScanSettingBinding** API 来通过 Compliance Operator 运行合规性扫描。如需有关这些 API 对象的更多信息，请运行：

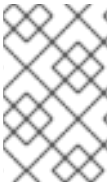
```
$ oc explain scansettings
```

或者

```
$ oc explain scansettingbindings
```

5.5.2.1. 运行合规性扫描

您可以使用互联网安全中心（CIS）配置集运行扫描。为方便起见，Compliance Operator 创建一个在启动时具有合理的默认值的 **ScanSetting** 对象。这个 **ScanSetting** 对象名为 **default**。



注意

对于 all-in-one control plane 和 worker 节点，合规性扫描在 worker 和 control plane 节点上运行两次。合规性扫描可能会导致扫描结果不一致。您可以通过在 **ScanSetting** 对象中只定义单个角色来避免结果不一致。

流程

1. 运行以下命令检查 **ScanSetting** 对象：

```
$ oc describe scansettings default -n openshift-compliance
```

输**出示例**

```
Name:         default
Namespace:    openshift-compliance
Labels:       <none>
Annotations:  <none>
API Version:  compliance.openshift.io/v1alpha1
Kind:         ScanSetting
Metadata:
  Creation Timestamp:  2022-10-10T14:07:29Z
  Generation:          1
  Managed Fields:
    API Version:  compliance.openshift.io/v1alpha1
    Fields Type:  FieldsV1
    fieldsV1:
      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:            2022-10-10T14:07:29Z
  Resource Version:  56111
  UID:               c21d1d14-3472-47d7-a450-b924287aec90
Raw Result Storage:
  Node Selector:
    node-role.kubernetes.io/master:
  Pv Access Modes:
    ReadWriteOnce 1
  Rotation:  3 2
  Size:      1Gi 3
  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 4
  worker 5
Scan Tolerations: 6
  Operator:         Exists
Schedule:           0 1 * * * 7
Show Not Applicable: false
Strict Node Scan:    true
Events:             <none>
```

1. Compliance Operator 创建一个包含扫描结果的持久性卷（PV）。默认情况下，PV 将使用访问模式 **ReadWriteOnce**，因为 Compliance Operator 无法假定集群中配置的存储类。另外，多数集群中可以使用 **ReadWriteOnce** 访问模式。如果需要获取扫描结果，可以使用一个 helper pod 来完成此操作，该 pod 也绑定卷。使用 **ReadWriteOnce** 访问模式的卷只能由一个 pod 挂载，因此请记住删除帮助程序 pod。否则，Compliance Operator 将无法重复使用卷进行后续扫描。
2. Compliance Operator 在卷中保留三个后续扫描的结果，旧的扫描会被轮转。
3. Compliance Operator 将为扫描结果分配一个 GB 存储。
4. 5. 如果扫描设置使用扫描集群节点的任何配置集，请扫描这些节点角色。
6. 默认扫描设置对象扫描所有节点。
7. 默认扫描设置对象每天 01:00 运行扫描。

作为默认扫描设置的替代，您可以使用 **default-auto-apply**，它有以下设置：

```
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 通过将 **autoUpdateRemediations** 和 **autoApplyRemediations** 标记设置为 **true**，您可以轻松地创建无需额外步骤自动修复的 **ScanSetting** 对象。

2. 创建一个 **ScanSettingBinding** 对，,它绑定到默认的 **ScanSetting** 对象，并使用 **cis** 和 **cis-node** 配置集扫描集群。例如：

```
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. 运行以下命令来创建 **ScanSettingBinding** 对象：

```
$ oc create -f <file-name>.yaml -n openshift-compliance
```

此时，**ScanSettingBinding** 对象已协调，并以 **Binding** 和 **Bound** 设置为基础。Compliance Operator 会创建一个 **ComplianceSuite** 对象和关联的 **ComplianceScan** 对象。

4. 运行以下命令跟踪合规性扫描进度：

```
$ oc get compliancescan -w -n openshift-compliance
```

扫描过程通过扫描阶段进行，最终完成后到达 **DONE** 阶段。在大多数情况下，扫描的结果是 **NON-COMPLIANT**。您可以检查扫描结果，并开始应用补救使集群兼容。如需更多信息，请参阅 *管理 Compliance Operator 补救*。

5.5.2.2. 将结果服务器 pod 调度到 worker 节点上

结果服务器 pod 挂载存储原始资产报告格式(ARF)扫描结果的持久性卷(PV)。**nodeSelector** 和 **tolerations** 属性允许您配置结果服务器 pod 的位置。

这适用于那些不允许 control plane 节点挂载持久性卷的环境。

流程

- 为 Compliance Operator 创建 **ScanSetting** 自定义资源(CR)：
  a. 定义 **ScanSetting** CR，并保存 YAML 文件，如 **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) Compliance Operator 使用此节点以 ARF 格式存储扫描结果。

(2) 结果服务器 pod 容限所有污点。

b. 要创建 **ScanSetting** CR，请运行以下命令：

```
$ oc create -f rs-workers.yaml
```

验证

- 要验证 **ScanSetting** 对象是否已创建，请运行以下命令：

```
$ oc get scansettings rs-on-workers -n openshift-compliance -o yaml
```

输出示例

```
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
```

5.5.2.3. ScanSetting 自定义资源

ScanSetting 自定义资源现在允许您通过扫描限值属性覆盖扫描程序 Pod 的默认 CPU 和内存限值。Compliance Operator 的默认值为 500Mi 内存，100m CPU 用于扫描程序容器；对于 **api-resource-collector** 容器为 200Mi 内存，100m CPU。要设置 Operator 的内存限值，如果通过 OLM 或 Operator 部署本身安装，请修改 **Subscription** 对象。

要增加 Compliance Operator 的默认 CPU 和内存限值，请参阅 *增加 Compliance Operator 资源限值*。

重要

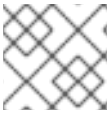
如果默认限值不足且 Operator 或扫描程序 Pod 被 Out Of Memory (OOM) 进程停止，则需要提高 Compliance Operator 或 scanner Pod 的内存限值。

5.5.2.4. 配置托管的 control plane 管理集群

如果您托管您自己的托管控制平面或 Hypershift 环境，并希望从管理集群扫描托管集群，则需要为目标托管集群设置名称和前缀命名空间。您可以通过创建一个 **TailoredProfile** 来达到此目的。

重要

此流程只适用于管理自己的托管控制平面环境的用户。



注意

托管的控制平面管理集群只支持 **ocp4-cis** 和 **ocp4-pci-dss** 配置集。

先决条件

- Compliance Operator 安装在管理集群中。

流程

1. 运行以下命令，获取要扫描的托管集群的**名称**和**命名空间** ：

```
$ oc get hostedcluster -A
```

输**出示例**

```
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. 在管理集群中，创建一个 **TailoredProfile** 扩展扫描配置集，并定义要扫描的 Hosted Cluster 的名称和命名空间：

management-tailoredprofile.yaml 示例

```
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. 变量。托管的控制平面管理集群只支持 **ocp4-cis** 和 **ocp4-pci-dss** 配置集。
2. **value** 是在前面步骤中输出的 **NAME**。
3. **value** 是在前面步骤中输出的 **NAMESPACE**。

3. 创建 **TailoredProfile**：

```
$ oc create -n openshift-compliance -f mgmt-tp.yaml
```

5.5.2.5. 应用资源请求和限值

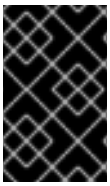
当 kubelet 作为 Pod 的一部分启动容器时，kubelet 会将该容器的请求和限值传递给容器运行时。在 Linux 中，容器运行时配置应用并强制实施您定义的限制的内核 cgroup。

CPU 限制定义容器可以使用的 CPU 时间。在每个调度间隔中，Linux 内核会检查是否超过这个限制。如果是这样，内核会在允许 cgroup 恢复执行前等待。

如果几个不同的容器(cgroups)希望在扩展系统上运行，则具有较大的 CPU 请求的工作负载会比具有小请求的工作负载分配更多的 CPU 时间。内存请求在 Pod 调度期间使用。在使用 cgroup v2 的节点上，容器运行时可能会使用内存请求作为提示来设置 **memory.min** 和 **memory.low** 值。

如果容器尝试分配超过这个限制的内存，Linux 内核的 out-of-memory 子系统会被激活，并通过停止容器中的一个进程来进行干预。Pod 或容器的内存限值也可以应用到由内存支持的卷的页面，如 emptyDir。

kubelet 将 **tmpfs emptyDir** 视为容器使用的内存进行跟踪，而不是视为本地临时存储。如果容器超过其内存请求，且在其中运行的节点变得缺少内存，则 Pod 的容器可能会被驱除。



重要

容器无法在超过 CPU 限制的情况下长时间运行。容器运行时不会停止 Pod 或容器超量使 CPU。要确定容器是否因为资源限值而无法调度或正在被终止，请参阅 *Compliance Operator 故障排除*。

5.5.2.6. 使用容器资源请求调度 Pod

创建 Pod 时，调度程序会选择要在其上运行的 Pod 的节点。每个节点都有一个最大容量，用于每种资源类型的 CPU 和它可为 Pod 提供的内存。调度程序确保调度容器的资源请求总和小于每种资源类型的节点容量。

虽然节点上的内存或 CPU 资源使用量非常低，但如果容量检查无法防止节点出现资源短缺的情况，则调度程序仍可能会拒绝在节点上放置 Pod。

对于每个容器，您可以指定以下资源限值和请求：

```
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>
```

虽然您只能为单个容器指定请求和限值，但考虑 Pod 的整体资源请求和限值也很有用。对于特定资源，容量资源请求和限值是 Pod 中每个容器这个类型的资源请求/限制总和。

容器资源请求和限值示例

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests: 1
        memory: "64Mi"
        cpu: "250m"
      limits: 2
        memory: "128Mi"
        cpu: "500m"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

```
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1. 容器请求 64 Mi 内存和 250 m CPU。
2. 容器的限值是 128 Mi 内存和 500 m CPU。

5.5.3. 定制 Compliance Operator

虽然 Compliance Operator 附带随时可用的配置集，但必须对其进行修改才能满足机构的需求和要求。修改配置集的过程称为*定制*。

Compliance Operator 提供了 **TailoredProfile** 对象来帮助定制配置集。

5.5.3.1. 创建新的定制配置集

您可以使用 **TailoredProfile** 对象从头开始编写一个定制的配置集。设置适当的 **title** 和 **description**，并将 **extends** 字段留空。向 Compliance Operator 指明此自定义配置集生成的扫描类型：

- 节点扫描：扫描操作系统。
- 平台扫描：扫描 OpenShift Container Platform 配置。

流程

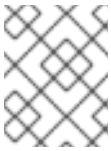
- 在 **TailoredProfile** 对象中设置以下注解：

new-profile.yaml 示例

```
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. 相应地设置 **Node** 或 **Platform**。
2. **extends** 字段是可选的。
3. 使用 **description** 字段描述新的 **TailoredProfile** 对象的功能。

4 使用 **title** 字段为您的 **TailoredProfile** 对象指定一个标题。



注意

在 **TailoredProfile** 对象的 **name** 字段中添加 **-node** 后缀与添加 **Node** 产品类型注解类似，会生成 Operating System 扫描。

5.5.3.2. 使用定制配置集扩展现有 ProfileBundle

尽管 **TailoredProfile** CR 支持最常见的定制操作，但 XCCDF 标准在定制 OpenSCAP 配置集方面具有更大的灵活性。此外，如果您的机构之前一直使用 OpenScap，则您可能有一个现有的 XCCDF 定制文件可重复使用。

ComplianceSuite 对象包含可指向自定义定制文件的可选 **TailoringConfigMap** 属性。**TailoringConfigMap** 属性的值是一个配置映射的名称，它必须包含名为 **tailoring.xml** 的键，这个键的值是定制内容。

流程

1. 浏览 Red Hat Enterprise Linux CoreOS (RHCOS) **ProfileBundle** 的可用规则：

```
$ oc get rules.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

2. 浏览同一 **ProfileBundle** 中的可用变量：

```
$ oc get variables.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

3. 创建名为 **nist-moderate-modified** 的定制配置集：

   a. 选择您要添加到 **nist-moderate-modified** 定制配置集中的规则。这个示例通过禁用两个规则并更改一个值来扩展 **rhcos4-moderate** 配置集。使用 **rationale** 值描述进行这些更改的原因：

   new-profile-node.yaml示例

```
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
```

表 5.3. spec 变量的属性

属性	描述
extends	构建此 **TailoredProfile** 的 **Profile** 对象的名称。
title	**TailoredProfile** 的人类可读标题.
disableRules	名称和理由对列表。每个名称引用要禁用的规则对象的名称。合理值是人类可读的文本，描述禁用规则的原因。
manualRules	名称和理由对列表。添加手动规则时，检查结果状态始终是 **manual**，且不会生成补救。此属性是自动的，在默认情况下，设置手动规则时没有值。
enableRules	名称和理由对列表。每个名称引用要启用的规则对象的名称。合理值是人类可读的文本，描述启用规则的原因。
description	描述 **TailoredProfile** 的人类可读文本.
setValues	名称、理由和值分组列表。每个名称都引用值集的名称。理由是人类可读的文本描述该集合。值是实际设置。

b. 添加 **tailoredProfile.spec.manualRules** 属性：

示例 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. 创建 **TailoredProfile** 对象：

```
$ oc create -n openshift-compliance -f new-profile-node.yaml 1
```

1 **TailoredProfile** 对象在默认的 **openshift-compliance** 命名空间中创建。

输出示例

```
tailoredprofile.compliance.openshift.io/nist-moderate-modified created
```

4. 定义 **ScanSettingBinding** 对象，将新的 **nist-moderate-modified** 定制配置集绑定到默认的 **ScanSetting** 对象。

new-scansettingbinding.yaml示例

```
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. 创建 **ScanSettingBinding** 对象：

```
$ oc create -n openshift-compliance -f new-scansettingbinding.yaml
```

输出示例

```
scansettingbinding.compliance.openshift.io/nist-moderate-modified created
```

5.5.4. 检索 Compliance Operator 原始结果

为 OpenShift Container Platform 集群提供合规性时，您可能需要提供扫描结果以供审核。

5.5.4.1. 从持久性卷中获取 Compliance Operator 原始结果

流程

Compliance Operator 生成原始结果并将其存储在持久性卷中。这些结果采用资产报告格式 (ARF)。

1. 探索 **ComplianceSuite** 对象：

```
$ oc get compliancesuites nist-moderate-modified \
-o json -n openshift-compliance | jq '.status.scanStatuses[].resultsStorage'
```

输出示例

```
{
    "name": "ocp4-moderate",
    "namespace": "openshift-compliance"
```

```
}
{
    "name": "nist-moderate-modified-master",
    "namespace": "openshift-compliance"
}
{
    "name": "nist-moderate-modified-worker",
    "namespace": "openshift-compliance"
}
```

这显示了可以访问原始结果的持久性卷声明。

2. 使用其中一个结果的名称和命名空间来验证原始数据位置：

```
$ oc get pvc -n openshift-compliance rhcos4-moderate-worker
```

输出示例

```
NAME                    STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
rhcos4-moderate-worker   Bound pvc-548f6cfe-164b-42fe-ba13-a07cfbc77f3a   1Gi     RWO
gp2         92m
```

3. 通过生成挂载卷并复制结果的 Pod 来获取原始结果：

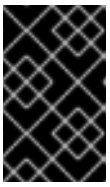
```
$ oc create -n openshift-compliance -f pod.yaml
```

pod.yaml 示例

```
apiVersion: "v1"
kind: Pod
metadata:
  name: pv-extract
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: pv-extract-pod
      image: registry.access.redhat.com/ubi9/ubi
      command: ["sleep", "3000"]
      volumeMounts:
      - mountPath: "/workers-scan-results"
        name: workers-scan-vol
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: workers-scan-vol
      persistentVolumeClaim:
        claimName: rhcos4-moderate-worker
```

4. Pod 运行后，下载结果：

```
$ oc cp pv-extract:/workers-scan-results -n openshift-compliance .
```



重要

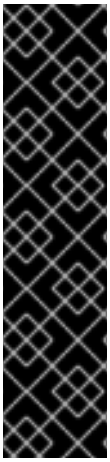
生成挂载持久性卷的 Pod 会将声明保留为 **Bound**。如果使用中的卷存储类的权限设置为 **ReadWriteOnce**，则该卷每次只能被一个 Pod 挂载。您必须在完成后删除 Pod，否则 Operator 将无法调度 Pod 并继续在此位置存储结果。

5. 提取完成后，可以删除 Pod：

```
$ oc delete pod pv-extract -n openshift-compliance
```

5.5.5. 管理 Compliance Operator 结果和补救

每个 **ComplianceCheckResult** 都代表一次合规性规则检查的结果。如果该规则可自动修复，则创建一个具有相同名称的 **ComplianceRemediation** 对象，由 **ComplianceCheckResult** 拥有。除非请求，否则不会自动应用补救，这使 OpenShift Container Platform 管理员有机会审阅补救的用途，且仅在验证后应用补救。



重要

联邦信息处理标准(FIPS)合规性的完整补救需要为集群启用 FIPS 模式。要启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL)计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。

在以下构架中支持 FIPS 模式：

- **x86_64**
- **ppc64le**
- **s390x**

5.5.5.1. 过滤合规性检查结果

默认情况下，**ComplianceCheckResult** 对象使用几个有用的标签标记，允许您查询检查，并决定生成结果后的后续步骤。

列出属于特定套件的检查：

```
$ oc get -n openshift-compliance compliancecheckresults \
  -l compliance.openshift.io/suite=workers-compliancesuite
```

列出属于特定扫描的检查：

```
$ oc get -n openshift-compliance compliancecheckresults \
-l compliance.openshift.io/scan=workers-scan
```

不是所有的 **ComplianceCheckResult** 对象都会创建 **ComplianceRemediation** 对象。只有可自动修复的 **ComplianceCheckResult** 对象。如果 **ComplianceCheckResult** 对象带有 **compliance.openshift.io/automated-remediation** 标签，则该对象具有相关的补救。补救的名称与检查

的名称相同。

列出可自动修复的所有失败检查：

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/automated-remediation'
```

列出所有失败的检查（按严重性排序）：

```
$ oc get compliancecheckresults -n openshift-compliance \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/check-severity=high'
```

输**出示例**

```
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-enable-fips-mode                 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-enable-fips-mode                 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
ocp4-moderate-fips-mode-enabled-on-all-nodes                   FAIL     high
```

列出必须手动修复的所有失败检查：

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,!compliance.openshift.io/automated-remediation'
```

手动补救步骤通常存储在 **ComplianceCheckResult** 对象的 **description** 属性中。

表 5.4. ComplianceCheckResult 状态

ComplianceCheckResult 状态	描述
PASS	合规检查运行完成并通过。
FAIL	合规检查运行完并失败。
INFO	合规检查运行完毕，并发现一些不严重的、不被视为错误的问题。
手动	合规检查没有方法自动评估成功或失败，必须手动检查。

ComplianceCheckResult 状态	描述
INCONSISTENT	合规检查报告来自不同来源的结果，通常是集群节点。
ERROR	合规性检查运行，但无法正确完成。
NOT-APPLICABLE	合规检查未运行，因为它不适用或未选中。

5.5.5.2. 审阅补救

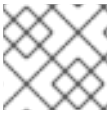
审阅拥有补救的 **ComplianceRemediation** 对象和 **ComplianceCheckResult** 对象。**ComplianceCheckResult** 对象包含有关检查作用和试图避免的强化的人类可读描述，以及其他 **metadata**，如重要性和相关的安全控制。**ComplianceRemediation** 对象代表可以解决 **ComplianceCheckResult** 中描述的问题的一种方式。第一次扫描后，检查状态为 **MissingDependencies** 的补救。

下面是名为 **sysctl-net-ipv4-conf-all-accept-redirects** 的检查和补救示例。此示例经过修订，仅显示 **spec** 和 **status**，省略了 **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
```

补救有效负载存储在 **spec.current** 属性中。有效负载可以是任何 Kubernetes 对象，但因为此补救是通过节点扫描生成的，上例中的补救有效负载是 **MachineConfig** 对象。对于平台扫描，补救有效负载通常是其他类型的对象（如 **ConfigMap** 或 **Secret**），但是否应用这种补救通常取决于管理员，否则 Compliance Operator 需要一组非常广泛的权限才能操作任何通用 Kubernetes 对象。本文稍后会提供补救平台检查的示例。

要查看应用补救时的具体操作，**MachineConfig** 对象内容将使用 Ignition 对象进行配置。有关格式的更多信息，请参阅 Ignition 规格。在示例中，**spec.config.storage.files[0].path** 属性指定由该补救创建的文件 (**/etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf**)，**spec.config.storage.files[0].contents.source** 属性指定该文件的内容。



注意

文件的内容是 URL 编码的。

使用以下 Python 脚本查看内容：

```
$ echo "net.ipv4.conf.all.accept_redirects%3D0" | python3 -c "import sys, urllib.parse;
print(urllib.parse.unquote(''.join(sys.stdin.readlines())))"
```

输出示例

```
net.ipv4.conf.all.accept_redirects=0
```

重要

Compliance Operator 不会自动解决补救之间可能出现的依赖关系问题。应用补救后，用户应执行重新扫描以确保准确的结果。

5.5.5.3. 使用自定义机器配置池时应用补救

当您创建自定义 **MachineConfigPool** 时，在 **MachineConfigPool** 中添加一个标签，以便 **KubeletConfig** 中的 **machineConfigPoolSelector** 可以与 **MachineConfigPool** 的标签匹配。

重要

不要在 **KubeletConfig** 文件中设置 **protectKernelDefaults: false**，因为 Compliance Operator 完成应用补救后，**MachineConfigPool** 对象可能无法意外暂停。

流程

1. 列出节点。

```
$ oc get nodes -n openshift-compliance
```

输出示例

```
NAME                                 STATUS  ROLES  AGE    VERSION
ip-10-0-128-92.us-east-2.compute.internal  Ready   master 5h21m  v1.29.4
ip-10-0-158-32.us-east-2.compute.internal  Ready   worker 5h17m  v1.29.4
ip-10-0-166-81.us-east-2.compute.internal  Ready   worker 5h17m  v1.29.4
ip-10-0-171-170.us-east-2.compute.internal Ready   master 5h21m  v1.29.4
ip-10-0-197-35.us-east-2.compute.internal  Ready   master 5h22m  v1.29.4
```

2. 为节点添加标签。

```
$ oc -n openshift-compliance \
label node ip-10-0-166-81.us-east-2.compute.internal \
node-role.kubernetes.io/<machine_config_pool_name>=
```

输出示例

```
node/ip-10-0-166-81.us-east-2.compute.internal labeled
```

3. 创建自定义 **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) **labels** 字段定义要为机器配置池(MCP)添加的标签名称。

4. 验证 MCP 是否已成功创建。

```
$ oc get mcp -w
```

5.5.5.4. 根据默认配置值评估 KubeletConfig 规则

OpenShift Container Platform 基础架构可能会在运行时包含不完整的配置文件，节点会假定缺少配置选项的默认配置值。某些配置选项可以作为命令行参数传递。因此，Compliance Operator 无法验证节点上的配置文件是否已完成，因为它可能会在规则检查中缺失选项。

要防止出现假的负结果（默认配置值通过检查，但实际应该失败），Compliance Operator 使用 Node/Proxy API 获取节点池中每个节点的配置，然后节点池中的所有配置选项都存储在代表该节点池中所有节点配置的文件中。这提高了扫描结果的准确性。

使用带有默认 **master** 和 **worker** 节点池配置的此功能不需要额外的配置更改。

5.5.5.5. 扫描自定义节点池

Compliance Operator 不会维护每个节点池配置的副本。Compliance Operator 将单一节点池中的所有节点的一致性配置选项聚合到配置文件的一个副本中。然后，Compliance Operator 使用特定节点池的配置文件来评估针对该池中的节点的规则。

流程

1. 将 **example** 角色添加到要存储在 **ScanSettingBinding** CR 中的 **ScanSetting** 对象中：

```
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. 创建使用 **ScanSettingBinding** CR 的扫描：

```
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
```

验证

- 平台 KubeletConfig 规则通过 **Node/Proxy** 对象检查。您可以运行以下命令来查找这些规则：

```
$ oc get rules -o json | jq '.items[] | select(.checkType == "Platform") | select(.metadata.name
| contains("ocp4-kubelet-")) | .metadata.name'
```

5.5.5.6. 修复 KubeletConfig 子池

KubeletConfig 补救标签可以应用到 **MachineConfigPool** 子池。

流程

- 在子池 **MachineConfigPool** CR 中添加标签：

```
$ oc label mcp <sub-pool-name> pools.operator.machineconfiguration.openshift.io/<sub-
pool-name>=
```

5.5.5.7. 应用补救

布尔值属性 **spec.apply** 控制 Compliance Operator 是否应该应用补救。您可以通过将属性设置为 **true** 来应用补救：

```
$ oc -n openshift-compliance \
patch complianceremediations/<scan-name>-sysctl-net-ipv4-conf-all-accept-redirects \
--patch '{"spec":{"apply":true}}' --type=merge
```

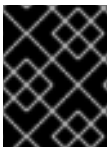
在 Compliance Operator 处理应用的补救后，**status.ApplicationState** 属性会更改为 **Applied** 或在出错时更改为 **Error**。应用机器配置补救时，该补救与其他应用的补救一起渲染为名为 **75-$scan-name-$suite-name** 的 **MachineConfig** 对象。**MachineConfig** 对象随后由 Machine Config Operator 渲染，最终由在每个节点上运行的机器控制守护进程实例应用到机器配置池中的所有节点。

请注意，当 MachineConfigOperator 将新的 **MachineConfig** 对象应用到池中的节点时，属于池的所有节点都会重启。应用多个补救时，这可能会不方便，每个补救都会重新渲染组合 **75-$scan-name-$suite-name MachineConfig** 对象。要防止立即应用补救，您可以通过将 **MachineConfigPool** 对象的 **.spec.paused** 属性设置为 **true** 来暂停机器配置池。

Compliance Operator 可以自动应用补救。在 **ScanSetting** 顶层对象中设置 **autoApplyRemediations: true**。

警告

只有经过仔细考虑才能自动应用补救。



重要

Compliance Operator 不会自动解决补救之间可能出现的依赖关系问题。应用补救后，用户应执行重新扫描以确保准确的结果。

5.5.5.8. 手动补救平台检查

检查平台扫描通常必须由管理员手动修复，原因有两个：

- 并不总是能够自动决定必须设置的值。其中一项检查要求提供允许的 registry 列表，但扫描程序并不知道组织要允许哪些 registry。
- 不同的检查会修改不同的 API 对象，需要自动补救以拥有 **root** 或超级用户访问权限来修改集群中的对象，但不建议这样做。

流程

1. 以下示例使用 **ocp4-ocp-allowed-registries-for-import** 规则，这在默认 OpenShift Container Platform 安装中会失败。检查规则 **oc get rule. compliance/ocp4-ocp-allowed-registries-for-import -oyaml**，该规则通过设置 **allowedRegistriesForImport** 属性来限制允许用户从中导入镜像的 registry，该规则的 *warning* 属性还会显示已检查的 API 对象，因此可以修改它并修复问题：

   ```
   $ oc edit image.config.openshift.io/cluster
   ```

   输出示例

   ```
   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. 重新运行扫描：

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

5.5.5.9. 更新补救

使用新版本的合规性内容时，可能会提供与之前版本不同的新补救版本。Compliance Operator 将保留应用的旧版本补救。OpenShift Container Platform 管理员还收到要审核和应用的新版本通知。之前应用的 ComplianceRemediation 对象已更新，它的状态已更改为 **Outdated**。过时的对象已标识，因此可轻松搜索。

以前应用的补救内容将存储在 **ComplianceRemediation** 对象的 **spec.outdated** 属性中，新的更新内容将存储在 **spec.current** 属性中。将内容更新至新版本后，管理员需要审查补救。只要 **spec.outdated** 属性存在，它将用来渲染生成的 **MachineConfig** 对象。删除 **spec.outdated** 属性后，Compliance Operator 会重新渲染生成的 **MachineConfig** 对象，导致 Operator 将配置推送到节点。

流程

1. 搜索任何过时的补救：

```
$ oc -n openshift-compliance get complianceremediations \
-l complianceoperator.openshift.io/outdated-remediation=
```

输出示例

```
NAME                            STATE
workers-scan-no-empty-passwords   Outdated
```

当前应用的补救存储在 **Outdated** 属性中，未应用的新补救存储在 **Current** 属性中。如果您对新版本满意，可删除 **Outdated** 字段。如果要保留更新的内容，可删除 **Current** 和 **Outdated** 属性。

2. 应用更新的补救版本：

```
$ oc -n openshift-compliance patch complianceremediations workers-scan-no-empty-passwords \
--type json -p '[{"op":"remove", "path":/spec/outdated}]'
```

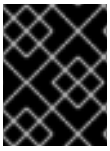
3. 补救状态将从 **Outdated** 切换为 **Applied**：

```
$ oc get -n openshift-compliance complianceremediations workers-scan-no-empty-passwords
```

输**出示例**

```
NAME                           STATE
workers-scan-no-empty-passwords   Applied
```

4. 节点将应用更新的补救版本并重新引导。



重要

Compliance Operator 不会自动解决补救之间可能出现的依赖关系问题。应用补救后，用户应执行重新扫描以确保准确的结果。

5.5.5.10. 取消应用补救

可能需要取消应用之前应用的补救。

流程

1. 将 **apply** 标志设置为 **false** ：

```
$ oc -n openshift-compliance \
patch complianceremediations/rhcos4-moderate-worker-sysctl-net-ipv4-conf-all-accept-redirects \
--patch '{"spec":{"apply":false}}' --type=merge
```

2. 补救状态将更改为 **NotApplied**，组合 **MachineConfig** 对象会被重新渲染，不包含补救。

重要

所有包含补救的受影响节点都将被重新引导。

重要

Compliance Operator 不会自动解决补救之间可能出现的依赖关系问题。应用补救后，用户应执行重新扫描以确保准确的结果。

5.5.5.11. 删除 KubeletConfig 补救

KubeletConfig 补救包括在节点级别的配置集中。要删除 KubeletConfig 补救，您必须手动将其从 **KubeletConfig** 对象中删除。本例演示了如何删除 **one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** 补救的合规性检查。

流程

1. 找到 **one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** 补救的 **scan-name** 和合规检查：

```
$ oc -n openshift-compliance get remediation \ one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available -o yaml
```

输**出示例**

```
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) 补救的扫描名称。

(2) 添加到 **KubeletConfig** 对象的补救。

注意

如果补救调用 **evictionHard** kubelet 配置，您必须指定所有 **evictionHard** 参数：**memory.available**、**nodefs.available**、**nodefs.inodesFree**、**imagefs.available** 和 **imagefs.inodesFree**。如果没有指定所有参数，则只应用指定的参数，补救无法正常工作。

2. 删除补救：

a. 为补救对象将 **apply** 设置为 false：

```
$ 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. 使用 **scan-name**，找到补救应用到的 **KubeletConfig** 对象：

```
$ oc -n openshift-compliance get kubeletconfig \
--selector compliance.openshift.io/scan-name=one-rule-tp-node-master
```

输**出示例**

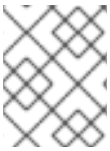
```
NAME                                AGE
compliance-operator-kubelet-master   2m34s
```

c. 从 **KubeletConfig** 对象手动删除补救 **imagefs.available: 10%** ：

```
$ oc edit -n openshift-compliance KubeletConfig compliance-operator-kubelet-master
```

重要

所有包含补救的受影响节点都将被重新引导。



注意

您还必须在定制的配置集中排除任何调度的扫描规则，这些配置集自动应用补救，否则会在下一次调度的扫描过程中重新应用补救。

5.5.5.12. Inconsistent ComplianceScan

ScanSetting 对象列出了合规性扫描从 **ScanSetting** 或 **ScanSettingBinding** 对象扫描的节点角色。每个节点角色通常映射到机器配置池。

重要

机器配置池中的所有机器都应该相同，池中节点的所有扫描结果都应该相同。

如果某些结果与其他结果不同，Compliance Operator 将标记一些节点将报告为 **INCONSISTENT** 的 **ComplianceCheckResult** 对象。所有 **ComplianceCheckResult** 对象也都标有 **compliance.openshift.io/inconsistent-check**。

因为池中的机器数量可能非常大，所以 Compliance Operator 会尝试找到最常用的状态，并列出与常见状态不同的节点。最常见的状态存储在 **compliance.openshift.io/most-common-status** 注解中，注解 **compliance.openshift.io/inconsistent-source** 包含与最常见状态不同的 **hostname:status** 检查状态对。如果没有找到常见状态，则所有 **hostname:status** 对都列在 **compliance.openshift.io/inconsistent-source annotation** 中。

如果可能，仍会创建补救，以便集群可以整合到兼容状态。但是，并非总是能够创建补救，必须手动纠正节点之间的差异。必须使用 **compliance.openshift.io/rescan=** 选项为扫描添加注解来重新运行合规性扫描，以得到一致的结果：

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

5.5.5.13. 其他资源

- 修改节点。

5.5.6. 执行高级 Compliance Operator 任务

Compliance Operator 包含适用于高级用户的选项，用于调试或与现有工具集成。

5.5.6.1. 直接使用 ComplianceSuite 和 ComplianceScan 对象

虽然建议用户利用 **ScanSetting** 和 **ScanSettingBinding** 对象来定义套件和扫描，但也有直接定义 **ComplianceSuite** 对象的有效用例：

- 仅指定单个规则进行扫描。这可与 **debug: true** 属性一起用于调试，提高 OpenSCAP 扫描程序的详细程度，否则调试模式会变得非常冗长。将测试限制为一条规则有助于减少调试信息的数量。
- 提供自定义 nodeSelector。要使补救适用，nodeSelector 必须与一个池匹配。
- 使用定制文件将 Scan 指向定制配置映射。
- 不需要从捆绑包解析配置集的消耗成本时用于测试或开发。

以下示例显示仅使用一条规则扫描 worker 机器的 **ComplianceSuite**：

```
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: ""
```

上面提到的 **ComplianceSuite** 对象和 **ComplianceScan** 对象以 OpenSCAP 期望的格式指定多个属性。

要找到配置集、内容或规则值，您可以先从 **ScanSetting** 和 **ScanSettingBinding** 创建类似的 Suite 或检查从 **ProfileBundle** 对象中解析的对象，如规则或配置集。这些对象包含可以从 **ComplianceSuite** 中引用它们的 **xccdf_org** 标识符。

5.5.6.2. 为 ScanSetting 扫描设置 PriorityClass

在大规模环境中，默认的 **PriorityClass** 对象可能太低，以保证 Pod 在其上执行扫描。对于必须保持合规或保证自动扫描的集群，建议设置 **PriorityClass** 变量，以确保 Compliance Operator 始终在资源约束的情况下赋予优先级。

流程

- 设置 **PriorityClass** 变量：

```
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 如果无法找到 **ScanSetting** 中引用的 **PriorityClass**，Operator 会将 **PriorityClass** 留空，发出警告，并在没有 **PriorityClass** 的情况下继续调度扫描。

5.5.6.3. 使用原始定制配置集

尽管 **TailoredProfile** CR 支持最常见的定制操作，但 XCCDF 标准在定制 OpenSCAP 配置集方面具有更大的灵活性。此外，如果您的机构之前一直使用 OpenScap，则您可能有一个现有的 XCCDF 定制文件可重复使用。

ComplianceSuite 对象包含可指向自定义定制文件的可选 **TailoringConfigMap** 属性。**TailoringConfigMap** 属性的值是一个配置映射的名称，它必须包含名为 **tailoring.xml** 的键，这个键的值是定制内容。

流程

1. 从一个文件创建 **ConfigMap** 对象：

```
$ oc -n openshift-compliance \
create configmap nist-moderate-modified \
--from-file=tailoring.xml=/path/to/the/tailoringFile.xml
```

2. 在属于 Suite 的 Scan 中引用定制文件：

```
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: ""
```

5.5.6.4. 执行重新扫描

通常，您希望按指定时间表重新运行扫描，如每周一或每天。在修复节点上的问题后，重新运行一次扫描也很有用。要执行单次扫描，可使用 **compliance.openshift.io/rescan=** 选项注解扫描：

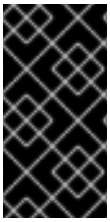
```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

一个重新扫描会为 **rhcos-moderate** 配置集生成四个额外的 **mc** ：

```
$ oc get mc
```

输出示例

```
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
```



重要

应用扫描设置 **default-auto-apply** 标签时，补救会自动应用并过时的补救更新。如果存在由于依赖项或已经过时的补救没有被应用的补救，重新扫描会应用补救，并可能会触发重启。只有使用 **MachineConfig** 对象触发器重启的补救。如果没有要应用的更新或依赖项，则不会重启。

5.5.6.5. 为结果设置自定义存储大小

虽然 **ComplianceCheckResult** 等自定义资源表示一次检查跨所有扫描节点的聚合结果，但审阅扫描程序生成的原始结果会很有用。原始结果以 ARF 格式生成，可能较大（每个节点几十兆字节），将其存储在由 **etcd** 键-值存储支持的 Kubernetes 资源中是不切实际的。相反，每次扫描都会创建一个默认为 1GB 大小的 PV。根据您的环境，您可能想要相应地增大 PV 大小。这可以使用在 **ScanSetting** 和 **ComplianceScan** 资源中公开的 **rawResultStorage.size** 属性完成。

相关的参数是 **rawResultStorage.rotation**，它控制在旧的扫描被轮转前 PV 中保留的扫描次数。默认值为 3，将轮转策略设置为 0 可禁用轮转。根据默认轮转策略和每个原始 ARF 扫描报告 100MB 的估计大小，您可以计算出环境的正确 PV 大小。

5.5.6.5.1. 使用自定义结果存储值

由于 OpenShift Container Platform 可以在各种公有云或裸机中部署，因此 Compliance Operator 无法决定可用的存储配置。默认情况下，Compliance Operator 会尝试使用集群的默认存储类创建 PV 来存储结果，但可以使用 **rawResultStorage.StorageClassName** 属性配置自定义存储类。

重要

如果您的集群没有指定默认存储类，则必须设置此属性。

将 **ScanSetting** 自定义资源配置为使用标准存储类，并创建大小为 10GB 的持久性卷，并保留最后 10 个结果：

ScanSetting CR 示例

```
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 * * *'
```

5.5.6.6. 应用套件扫描生成的补救

虽然您可以使用 **ComplianceSuite** 对象中的 **autoApplyRemediations** 布尔值参数但您可以使用 **compliance.openshift.io/apply-remediations** 为对象添加注解。这允许 Operator 应用所有创建的补救。

流程

- 运行以下命令应用 **compliance.openshift.io/apply-remediations** 注解：

```
$ oc -n openshift-compliance \
annotate compliancesuites/workers-compliancesuite compliance.openshift.io/apply-remediations=
```

5.5.6.7. 自动更新补救

在某些情况下带有较新内容的扫描可能会将补救标记为 **OUTDATED**。作为管理员您可以应用 **compliance.openshift.io/remove-outdated** 注解来应用新的补救并删除过时的补救。

流程

- 应用 **compliance.openshift.io/remove-outdated** 注解：

```
$ oc -n openshift-compliance \
annotate compliancesuites/workers-compliancesuite compliance.openshift.io/remove-outdated=
```

或者在 **ScanSetting** 或 **ComplianceSuite** 对象中设置 autoUpdateRemediations 标志以自动更新补救。

5.5.6.8. 为 Compliance Operator 创建自定义 SCC

在一些环境中，您必须创建一个自定义安全性上下文约束(SCC)文件，以确保 Compliance Operator **api-resource-collector** 使用正确的权限。

先决条件

- 您必须具有 **admin** 权限。

流程

1. 在名为 **restricted-adjusted-compliance.yaml** 的 YAML 文件中定义 SCC：

   SecurityContextConstraints 对象定义

```
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 SCC 的优先级必须高于适用于 **system:authenticated** 组的任何其他 SCC。

2 Compliance Operator Scanner pod 使用的服务帐户。

2. 创建 SCC：

```
$ oc create -n openshift-compliance  -f restricted-adjusted-compliance.yaml
```

输**出示例**

```
securitycontextconstraints.security.openshift.io/restricted-adjusted-compliance created
```

验证

1. 验证是否已创建 SCC：

```
$ oc get -n openshift-compliance scc restricted-adjusted-compliance
```

输**出示例**

```
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"]
```

5.5.6.9. 其他资源

- 管理安全性上下文约束

5.5.7. 对 Compliance Operator 进行故障排除

本节介绍如何对 Compliance Operator 进行故障排除。该信息可用于诊断问题或在错误报告中提供信息。一些常规提示：

- 出现重大事件时，Compliance Operator 会发出 Kubernetes 事件。您可以使用以下命令查看集群中的所有事件：

```
$ oc get events -n openshift-compliance
```

  或者使用以下命令查看 Scan 等对象的事件：

```
$ oc describe -n openshift-compliance compliancescan/cis-compliance
```

- Compliance Operator 由多个控制器组成，大约每个 API 对象一个。仅过滤对应于有问题的 API 对象的控制器很有用。如果无法应用 **ComplianceRemediation**，请查看 **remediationctrl** 控制器中的信息。可以通过使用 **jq** 进行解析来过滤单个控制器中的信息：

```
$ oc -n openshift-compliance logs compliance-operator-775d7bddbd-gj58f \
    | jq -c 'select(.logger == "profilebundlectrl")'
```

- 在 UTC 中，自 UNIX 时间戳以来的时间戳以秒为单位记录。要将其转换为人类可读的日期，请使用 **date -d @timestamp --utc**，例如：

```
$ date -d @1596184628.955853 --utc
```

- 很多自定义资源允许设置 **debug** 选项，最重要的是 **ComplianceSuite** 和 **ScanSetting**。启用此选项会增加 OpenSCAP 扫描程序 Pod 以及其他一些帮助程序 Pod 的详细程度。

- 如果单个规则意外传递或失败，则仅使用该规则运行单次扫描或 suite 从相应的 **ComplianceCheckResult** 对象中查找规则 ID 并将其用作 **Scan** CR 中的 **rule** 属性值会很有帮助。然后再启用 **debug** 选项，扫描程序 Pod 中的 **scanner** 容器日志会显示原始 OpenSCAP 日志。

5.5.7.1. 扫描剖析

以下各节概述了 Compliance Operator 扫描的组件和阶段。

5.5.7.1.1. 合规性源

合规性内容存储在从 **ProfileBundle** 对象生成的 **Profile** 对象中。Compliance Operator 为集群创建一个 **ProfileBundle** 对象，并为集群节点创建另一个对象。

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

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

ProfileBundle 对象由带有 **Bundle** 名称标签的部署处理。要使用 **Bundle** 排除问题，可以查找部署并查看部署中的 Pod 日志：

```
$ 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
```

5.5.7.1.2. ScanSetting 和 ScanSettingBinding 对象生命周期和调试

通过有效的合规性内容源，可以使用高级 **ScanSetting** 和 **ScanSettingBinding** 对象来生成 **ComplianceSuite** 和 **ComplianceScan** 对象：

```
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
```

ScanSetting 和 **ScanSettingBinding** 对象均由标记为 **logger=scansettingbindingctrl** 的同一控制器处理。这些对象没有状态。任何问题都以事件的形式传递：

```
Events:
  Type    Reason        Age    From                  Message
  ----    ------        ----   ----                  -------
  Normal  SuiteCreated  9m52s  scansettingbindingctrl  ComplianceSuite openshift-compliance/my-companys-compliance-requirements created
```

现在创建一个 **ComplianceSuite** 对象。流继续协调新创建的 **ComplianceSuite**。

5.5.7.1.3. ComplianceSuite 自定义资源生命周期和调试

ComplianceSuite CR 是一个围绕 **ComplianceScan** CR 的 wrapper。**ComplianceSuite** CR 由标记为 **logger=suitectrl** 的控制器处理。该控制器处理从 Suite 创建 Scan 的过程，将单个扫描状态协调并整合为一个 Suite 状态。如果将 Suite 设置为定期执行，**suitectrl** 也会处理创建 **CronJob** CR 的事件，该 CR 在初始运行完成后会在 Suite 中重新运行 Scan：

```
$ oc get cronjobs
```

输出示例

```
NAME                         SCHEDULE    SUSPEND   ACTIVE   LAST SCHEDULE   AGE
<cron_name>                    0 1 * * *   False     0        <none>          151m
```

对于最重要的问题，会发出事件。使用 **oc describe compliancesuites/<name>** 查看它们。**Suite** 对象还有一个 **Status** 子资源，当属于这个 **Suite** 的任何 **Scan** 对象更新其 Status 子资源时，这个子资源会更新。创建所有预期扫描后，控制会被传递给扫描控制器。

5.5.7.1.4. ComplianceScan 自定义资源生命周期和调试

ComplianceScan CR 由 **scanctrl** 控制器处理。这也是进行实际扫描以及创建扫描结果的地方。每次扫描都经过几个阶段：

5.5.7.1.4.1. 待处理阶段

在此阶段验证扫描是否正确。如果存储大小等参数无效，则扫描会转换为 DONE，并包含 ERROR 结果，否则进入启动阶段。

5.5.7.1.4.2. 启动阶段

在此阶段，一些配置映射包含扫描程序 Pod 的环境或直接包含扫描程序 Pod 要评估的脚本。列出配置映射：

```
$ oc -n openshift-compliance get cm \
-l compliance.openshift.io/scan-name=rhcos4-e8-worker,complianceoperator.openshift.io/scan-
script=
```

扫描程序 pod 将使用这些配置映射。如果您需要修改扫描程序行为，更改扫描程序调试级别或打印原始结果，修改配置映射是一种好方法。随后，每次扫描都会创建一个持久性卷声明，以存储原始 ARF 结果：

```
$ oc get pvc -n openshift-compliance -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

PVC 由每次扫描进行的 **ResultServer** 部署挂载。**ResultServer** 是一个简单的 HTTP 服务器，单个扫描程序 Pod 会将完整 ARF 结果上传到其中。每个服务器都可以在不同节点中运行。完整 ARF 结果可能非常大，您不能假定可以创建同时从多个节点挂载的卷。扫描完成后，**ResultServer** 部署会缩减。包含原始结果的 PVC 可从另一个自定义 Pod 挂载，并可获取或检查结果。扫描程序 Pod 和 **ResultServer** 之间的流量受 mutual TLS 协议保护。

最后，扫描程序 Pod 在此阶段启动；一个扫描程序 Pod 用于一个 **Platform** 扫描实例，每个匹配节点的一个扫描程序 Pod 用于一个 **node** 扫描实例。每节点 Pod 使用节点名称标识。每个 Pod 始终使用 **ComplianceScan** 名称标识：

```
$ oc get pods -lcompliance.openshift.io/scan-name=rhcos4-e8-worker,workload=scanner --show-
labels
```

输**出示例**

```
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
```

+ 扫描然后进入 Running 阶段。

5.5.7.1.4.3. 运行阶段

运行阶段会等待扫描程序 Pod 完成。运行阶段使用以下术语和进程：

- **init 容器**：有一个名为 **content-container** 的 init 容器。它运行 **contentImage** 容器，并执行单个命令，该命令将 **contentFile** 复制到与这个 Pod 中其他容器共享的 **/content** 目录中。
- **scanner**：此容器运行扫描。对于节点扫描，该容器将节点文件系统作为 **/host** 挂载，并挂载 init 容器交付的内容。该容器还挂载启动阶段创建的 **entrypoint ConfigMap** 并执行它。入口点 **ConfigMap** 中的默认脚本执行 OpenSCAP，并将结果文件存储在 Pod 容器之间共享的 **/results** 目录中。可查看此 Pod 中的日志以确定 OpenSCAP 扫描程序检查了哪些内容。可使用 **debug** 标记查看更详细的输出。
- **logcollector**：logcollector 容器会等待扫描程序容器完成。然后，它会将完整的 ARF 结果上传到 **ResultServer**，并以 **ConfigMap** 的形式单独上传 XCCDF 结果以及扫描结果和 OpenSCAP 结果代码。这些结果配置映射使用扫描名称 (**compliance.openshift.io/scan-name=rhcos4-e8-worker**) 进行标记：

  ```
  $ oc describe cm/rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
  ```

  输**出示例**

  ```
  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"?>
  ...
  ```

用于 **Platform** 扫描的扫描程序 Pod 类似，以下情况除外：

- 名为 **api-resource-collector** 的一个额外 init 容器读取由 content-container init 容器提供的 OpenSCAP 内容，确定内容需要检查的 API 资源，并将这些 API 资源存储到 **scanner** 容器将从中读取它们的共享目录中。
- **scanner** 容器不需要挂载主机文件系统。

扫描程序 Pod 完成后，扫描将进入聚合阶段。

5.5.7.1.4.4. 聚合阶段

在聚合阶段，扫描控制器会生成另一个名为聚合器 Pod 的 Pod。它的目的是获取生成的 **ConfigMap** 对象，读取结果，并为每个检查结果创建对应的 Kubernete 对象。如果可自动补救检查失败，将创建一个 **ComplianceRemediation** 对象。为了提供人类可读的检查和补救元数据，聚合器 Pod 也会使用 init 容器挂载 OpenSCAP 内容。

当一个配置映射由聚合器 Pod 处理时，会使用 **compliance-remediations/processed** 标签进行标识。这个阶段会生成 **ComplianceCheckResult** 对象：

```
$ oc get compliancecheckresults -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

输**出示例**

```
NAME                                                STATUS   SEVERITY
rhcos4-e8-worker-accounts-no-uid-except-zero        PASS     high
rhcos4-e8-worker-audit-rules-dac-modification-chmod FAIL     medium
```

和 **ComplianceRemediation** 对象：

```
$ oc get complianceremediations -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

输**出示例**

```
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
```

创建这些 CR 后，聚合器 Pod 将退出，扫描进入完成阶段。

5.5.7.1.4.5. 完成阶段

在最后的扫描阶段，会根据需要清理扫描资源，如果扫描是一次性的，**ResultServer** 部署会被缩减，如果扫描是连续性的，会被删除；下一个扫描实例将重新创建部署。

也可以在完成阶段通过注解来触发扫描重新运行：

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

扫描到达完成阶段后，除非将补救设置为使用 **autoApplyRemediations: true** 自动应用，否则不会自动

执行任何其他操作。OpenShift Container Platform 管理员现在将审阅补救并根据需要应用它们。如果将补救设置为自动应用，**ComplianceSuite** 控制器会在完成阶段接管，暂停扫描映射到的机器配置池，并一次性应用所有补救。如果应用了补救，**ComplianceRemediation** 控制器将接管。

5.5.7.1.5. ComplianceRemediation 控制器生命周期和调试

示例扫描报告了一些结果。通过将 **apply** 属性切换为 **true** 可启用其中一个补救：

```
$ oc patch complianceremediations/rhcos4-e8-worker-audit-rules-dac-modification-chmod --patch
'{"spec":{"apply":true}}' --type=merge
```

ComplianceRemediation 控制器 (**logger=remediationctrl**) 协调修改后的对象。协调的结果是会更改已协调补救对象的状态，也会更改包含所有已应用补救的已渲染的每套件 **MachineConfig** 对象。

MachineConfig 对象始终以 **75-** 开头，并以扫描和套件命名：

```
$ oc get mc | grep 75-
```

输**出示例**

```
75-rhcos4-e8-worker-my-companys-compliance-requirements                           3.2.0
2m46s
```

当前包含 **mc** 的补救列在机器配置的注解中：

```
$ oc describe mc/75-rhcos4-e8-worker-my-companys-compliance-requirements
```

输**出示例**

```
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:
```

ComplianceRemediation 控制器的算法如下：

- 所有当前应用的补救都读取到初始补救集合中。
- 如果要应用协调后的补救，会将其添加到该集合中。
- **MachineConfig** 对象从集合中渲染，并使用集合中的补救名称进行注解。如果集合为空（未应用最后一个补救）,将删除渲染的 **MachineConfig** 对象。
- 只有渲染的机器配置与集群中已应用的 MC 不同时，才会更新（或创建，或删除）应用的 MC。
- 创建或修改 **MachineConfig** 对象会触发与 **machineconfiguration.openshift.io/role** 标签匹配的节点重新引导 - 请参阅 MachineConfig Operator 文档以了解更多详细信息。

根据需要更新渲染的机器配置并更新协调的补救对象状态后，补救循环即告终止。在本例中，应用补救会触发重新引导。重新引导后，注解扫描以重新运行它：

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

扫描将运行并完成。检查补救是否通过：

```
$ oc -n openshift-compliance \
get compliancecheckresults/rhcos4-e8-worker-audit-rules-dac-modification-chmod
```

输出示例

```
NAME                                             STATUS   SEVERITY
rhcos4-e8-worker-audit-rules-dac-modification-chmod   PASS     medium
```

5.5.7.1.6. 有用的标签

Compliance Operator 生成的每个 pod 都专门使用其所属的扫描及其工作来标识。扫描标识符使用 **compliance.openshift.io/scan-name** 标签进行标识。工作负载标识符使用 **workload** 标签进行标识。

Compliance Operator 调度以下工作负载：

- **scanner**：执行合规性扫描。
- **resultserver**：存储合规性扫描的原始结果。
- **aggregator**：汇总结果，检测不一致和输出结果对象（检查结果和补救）。
- **suitererunner**：将标记一个要重新运行的套件（设定时间表时）。
- **profileparser**：解析数据流并创建适当的配置集、规则和变量。

需要对某个工作负载进行调试和记录日志时，请运行：

```
$ oc logs -l workload=<workload_name> -c <container_name>
```

5.5.7.2. 增加 Compliance Operator 资源限值

在某些情况下，Compliance Operator 可能需要的内存超过默认限制允许的数量。缓解此问题的最佳方法是设置自定义资源限制。

要增加扫描程序 Pod 的默认内存和 CPU 限值，请参阅 *'ScanSetting' 自定义资源*。

流程

1. 要将 Operator 的内存限值增加到 500 Mi，请创建名为 **co-memlimit-patch.yaml** 的以下补丁文件：

```
spec:
  config:
    resources:
      limits:
        memory: 500Mi
```

2. 应用补丁文件：

```
$ oc patch sub compliance-operator -nopenshift-compliance --patch-file co-memlimit-patch.yaml --type=merge
```

5.5.7.3. 配置 Operator 资源限制

resources 字段为 Operator Lifecycle Manager (OLM) 创建的 Pod 中所有容器定义资源约束。

注意

此流程中应用的资源约束会覆盖现有的资源限制。

流程

- 通过编辑 **Subscription** 对象，注入 0.25 cpu 和 64 Mi 内存的请求，以及每个容器 0.5 cpu 和 128 Mi 内存：

```
kind: Subscription
metadata:
  name: custom-operator
spec:
  package: etcd
  channel: alpha
  config:
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

5.5.7.4. 配置 ScanSetting 超时

ScanSetting 对象有一个 timeout 选项，可在 **ComplianceScanSetting** 对象中指定一个持续时间字符串，如 **1h30m**。如果扫描没有在指定的超时时间内完成，则扫描会重新尝试，直到达到 **maxRetryOnTimeout** 限制为止。

流程

- 要在 ScanSetting 中设置 **timeout** 和 **maxRetryOnTimeout**，请修改现有 **ScanSetting** 对象：

```
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. **timeout** 变量定义为持续时间字符串，如 **1h30m**。默认值为 **30m**。要禁用超时，请将值设为 **0s**。
2. **maxRetryOnTimeout** 变量定义尝试重试的次数。默认值为 **3**。

5.5.7.5. 获取支持

如果您在执行本文档所述的某个流程或 OpenShift Container Platform 时遇到问题，请访问 红帽客户门户网站。

通过红帽客户门户网站：

- 搜索或者浏览红帽知识库，了解与红帽产品相关的文章和解决方案。
- 提交问题单给红帽支持。
- 访问其他产品文档。

要识别集群中的问题，您可以在 OpenShift Cluster Manager 中使用 Insights。Insights 提供了问题的详细信息，并在有可用的情况下，提供了如何解决问题的信息。

如果您对本文档有任何改进建议，或发现了任何错误，请为相关文档组件提交 JIRA 问题。请提供具体详情，如章节名称和 OpenShift Container Platform 版本。

5.5.8. 使用 oc-compliance 插件

虽然 Compliance Operator 为集群自动执行许多检查和补救，但将集群设置为合规的完整过程通常需要管理员与 Compliance Operator API 和其他组件交互。**oc-compliance** 插件使进程变得更加容易。

5.5.8.1. 安装 oc-compliance 插件

流程

1. 提取 **oc-compliance** 镜像以获取 **oc-compliance** 二进制文件：

```
$ 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/
```

输出示例

```
W0611 20:35:46.486903   11354 manifest.go:440] Chose linux/amd64 manifest from the manifest list.
```

现在，您可以运行 **oc-compliance**。

5.5.8.2. 获取原始结果

当合规性扫描完成后，生成的 **ComplianceCheckResult** 自定义资源（CR）中会列出单个检查的结果。但是，管理员或审核员可能需要扫描的完整详情。OpenSCAP 工具创建带有详细结果的高级记录格式

（ARF）格式的文件。这个 ARF 文件太大，无法存储在配置映射或其他标准 Kubernetes 资源中，因此会创建一个持久性卷（PV）来包含它。

流程

- 使用 Compliance Operator 从 PV 获取结果是一个包括四步的过程。但是，在 **oc-compliance** 插件中，您可以使用单个命令：

```
$ oc compliance fetch-raw <object-type> <object-name> -o <output-path>
```

- **<object-type>** 可以是 **scansettingbinding**、**compliancescan** 或 **compliancesuite**，具体取决于扫描通过哪个对象启动。
- **<object-name>** 是用于收集 ARF 文件的绑定、套件或扫描对象的名称，**<output-path>** 是放置结果的本地目录。
  例如：

```
$ oc compliance fetch-raw scansettingbindings my-binding -o /tmp/
```

输**出示例**

```
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
```

查看目录中的文件列表：

```
$ ls /tmp/ocp4-cis-node-master/
```

输**出示例**

```
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
```

提取结果：

```
$ 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
```

查看结果：

```
$ ls resultsdir/worker-scan/
```

输**出示例**

```
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
```

5.5.8.3. 重新运行扫描

虽然可以作为调度的作业运行扫描，但通常必须根据需要重新运行扫描，特别是在应用修复或进行其他更改集群时。

流程

- 使用 Compliance Operator 重新运行扫描需要使用扫描对象上的注解。但是，通过 **oc-compliance** 插件，您可以使用单个命令重新运行扫描。输入以下命令为名为 **my-binding** 的 **ScanSettingBinding** 对象重新运行扫描：

```
$ oc compliance rerun-now scansettingbindings my-binding
```

输出示例

```
Rerunning scans from 'my-binding': ocp4-cis
Re-running scan 'openshift-compliance/ocp4-cis'
```

5.5.8.4. 使用 ScanSettingBinding 自定义资源

当使用由 Compliance Operator 提供的 **ScanSetting** 和 **ScanSettingBinding** 自定义资源（CR）时，可以在使用一组常用的扫描选项时，对多个配置集运行扫描，如 **schedule**、**machine roles**、**tolerations** 等。虽然这比使用多个 **ComplianceSuite** 或 **ComplianceScan** 对象更容易，但可能会让新用户混淆。

oc compliance bind 子命令可帮助您创建 **ScanSettingBinding** CR。

流程

1. 运行：

```
$ oc compliance bind [--dry-run] -N <binding name> [-S <scansetting name>]
<objtype/objname> [..<objtype/objname>]
```

   - 如果省略 **-S** 标志，则使用 Compliance Operator 提供的 **default** 扫描设置。
   - 对象类型是 Kubernetes 对象类型，可以是 **profile** 或 **tailorprofile**。可以提供多个对象。
   - 对象名称是 Kubernetes 资源的名称，如 **.metadata.name**。
   - 添加 **--dry-run** 选项，以显示所创建对象的 YAML 文件。
     例如，给定以下配置集和扫描设置：

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

输出示例

```
NAME          AGE
ocp4-cis      9m54s
```

```
ocp4-cis-node     9m54s
ocp4-e8           9m54s
ocp4-moderate     9m54s
ocp4-ncp          9m54s
rhcos4-e8         9m54s
rhcos4-moderate   9m54s
rhcos4-ncp        9m54s
rhcos4-ospp       9m54s
rhcos4-stig       9m54s
```

```
$ oc get scansettings -n openshift-compliance
```

输**出示例**

```
NAME                 AGE
default              10m
default-auto-apply   10m
```

2. 要将 **default** 设置应用到 **ocp4-cis** 和 **ocp4-cis-node** 配置集，请运行：

```
$ oc compliance bind -N my-binding profile/ocp4-cis profile/ocp4-cis-node
```

输**出示例**

```
Creating ScanSettingBinding my-binding
```

创建 **ScanSettingBinding** CR 后，绑定配置集开始使用相关设置扫描两个配置集。总体而言，这是开始使用 Compliance Operator 扫描的最快方法。

5.5.8.5. 打印控制

合规标准通常分为层次结构，如下所示：

- 基准是特定标准的一组控制的顶级定义。例如，FedRAMP Moderate 或 Center for Internet Security（CIS）v.1.6.0。
- 控制描述了遵守基准必须满足的一系列要求。例如，FedRAMP AC-01（访问控制政策和程序）。
- 规则是单一检查，特定于要纳入合规性的系统，以及一个或多个这些规则映射到控制。
- Compliance Operator 处理将规则分组到配置集中的单个基准测试。可能很难确定哪些控制配置集中的规则集是否满足要求。

流程

- **oc compliance control** 子命令提供有关给定配置集满足的标准和控制标准的报告：

```
$ oc compliance controls profile ocp4-cis-node
```

输**出示例**

```
+-----------+----------+
| FRAMEWORK | CONTROLS |
```

```
+-----------+----------+
| CIS-OCP   | 1.1.1    |
+           +----------+
|           | 1.1.10   |
+           +----------+
|           | 1.1.11   |
+           +----------+
...
```

5.5.8.6. 获取合规补救详情

Compliance Operator 提供了补救对象，用于自动执行与集群兼容所需的更改。**fetch-fixes** 子命令可帮助您准确了解使用了哪些配置修复。使用 **fetch-fixes** 子命令，将修复对象从配置集、规则或 **ComplianceRemediation** 对象提取到要检查的目录中。

流程

1. 查看配置集的补救：

```
$ oc compliance fetch-fixes profile ocp4-cis -o /tmp
```

输出示例

```
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 当配置集中没有对应的修复规则时会出现 **No fixes to persist**警告，因为规则无法自动修复，或者没有提供相应的补救。

2. 您可以查看 YAML 文件示例。**head** 命令将显示前 10 行：

```
$ head /tmp/ocp4-api-server-audit-log-maxsize.yaml
```

输出示例

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  maximumFileSizeMegabytes: 100
```

3. 查看扫描后创建的 **ComplianceRemediation** 对象的补救：

```
$ oc get complianceremediations -n openshift-compliance
```

输**出示例**

```
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
```

输**出示例**

```
Persisted compliance remediation fix to /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

4. 您可以查看 YAML 文件示例。**head** 命令将显示前 10 行：

```
$ head /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

输**出示例**

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  encryption:
    type: aescbc
```

警告

在直接应用补救前请小心。有些补救可能不适用于批量，如中等配置集中的 usbguard 规则。在这些情况下，允许 Compliance Operator 应用规则，因为它解决了依赖项并确保集群处于良好状态。

5.5.8.7. 查看 ComplianceCheckResult 对象详情

扫描运行结束时，会针对各个扫描规则创建 **ComplianceCheckResult** 对象。**view-result** 子命令提供了 **ComplianceCheckResult** 对象详细信息人类可读的输出。

流程

- 运行：

```
$ oc compliance view-result ocp4-cis-scheduler-no-bind-address
```

第 6 章 FILE INTEGRITY OPERATOR

6.1. FILE INTEGRITY OPERATOR 发行注记

OpenShift Container Platform 的 File Integrity Operator 在 RHCOS 节点上持续运行文件完整性检查。

本发行注记介绍了 OpenShift Container Platform 中 File Integrity Operator 的开发。

有关 File Integrity Operator 的概述，请参阅了解 File Integrity Operator。

要访问最新版本，请参阅更新 File Integrity Operator。

6.1.1. OpenShift File Integrity Operator 1.3.4

以下公告可用于 OpenShift File Integrity Operator 1.3.4：

- RHBA-2024:2946 OpenShift File Integrity Operator 程序漏洞修复和功能增强更新

6.1.1.1. 程序错误修复

在以前的版本中，File Integrity Operator 会因为大量证书轮转而发出 **NodeHasIntegrityFailure** 警报。在这个版本中，警报和失败状态会被正确触发。(**OCPBUGS-31257**)

6.1.2. OpenShift File Integrity Operator 1.3.3

以下公告可用于 OpenShift File Integrity Operator 1.3.3：

- RHBA-2023:5652 OpenShift File Integrity Operator 程序漏洞修复和功能增强更新

这个版本解决了底层依赖项中的 CVE。

6.1.2.1. 新功能及功能增强

- 您可以在以 FIPS 模式运行的 OpenShift Container Platform 集群中安装和使用 File Integrity Operator。

重要

要为集群启用 FIPS 模式，您必须从配置为以 FIPS 模式操作的 Red Hat Enterprise Linux (RHEL) 计算机运行安装程序。有关在 RHEL 中配置 FIPS 模式的更多信息，请参阅在 FIPS 模式中安装该系统。

当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 x86_64、ppc64le 和 s390x 架构上提交到 NIST FIPS 140-2/140-3 Validation。

6.1.2.2. 程序错误修复

- 在以前的版本中，一些带有私有默认挂载传播的 FIO pod 与 **hostPath: path: /** 卷挂载会中断依赖多路径的 CSI 驱动程序。这个问题已被解决，CSI 驱动程序可以正常工作。(当使用多路径时，一些 OpenShift Operator pod 阻止卸载 CSI 卷。)

- 这个版本解决了 CVE-2023-39325。(**CVE-2023-39325**)

6.1.3. OpenShift File Integrity Operator 1.3.2

以下公告可用于 OpenShift File Integrity Operator 1.3.2：

- RHBA-2023:5107 OpenShift File Integrity Operator 程序错误修复更新

这个版本解决了底层依赖项中的 CVE。

6.1.4. OpenShift File Integrity Operator 1.3.1

以下公告可用于 OpenShift File Integrity Operator 1.3.1：

- RHBA-2023:3600 OpenShift File Integrity Operator 程序错误修复更新

6.1.4.1. 新功能及功能增强

- FIO 现在将 kubelet 证书作为默认文件包括在 OpenShift Container Platform 管理时发出警告。(OCPBUGS-14348)
- FIO 现在可以正确地将电子邮件定向到红帽技术支持的地址。(OCPBUGS-5023)

6.1.4.2. 程序错误修复

- 在以前的版本中，当节点从集群中移除时，FIO 不会清理 **FileIntegrityNodeStatus** CRD。FIO 已更新，以便在删除节点时正确清理节点状态 CRD。(OCPBUGS-4321)
- 在以前的版本中，FIO 也错误地表示新节点无法进行完整性检查。FIO 已更新，在向集群添加新节点时正确显示节点状态 CRD。这提供了正确的节点状态通知。(OCPBUGS-8502)
- 在以前的版本中，当 FIO 协调 **FileIntegrity** CRD 时，它将暂停扫描，直到协调完成为止。这会导致在节点上不受协调影响的积极重新初始化过程。这个问题还为机器配置池造成不必要的 daemonset，它们与 **FileIntegrity** 无关。FIO 可以正确地处理这些情况，并只暂停受文件完整性更改影响的节点的 AIDE 扫描。(CMP-1097)

6.1.4.3. 已知问题

在 FIO 1.3.1 中，增加 IBM Z® 集群中的节点可能会导致文件完整性节点状态 **Failed**。如需更多信息，请参阅在 IBM Power® 中添加节点可能会导致文件完整性节点状态失败 。

6.1.5. OpenShift File Integrity Operator 1.2.1

以下公告可用于 OpenShift File Integrity Operator 1.2.1：

- RHBA-2023:1684 OpenShift File Integrity Operator 程序错误修复更新
- 此发行版本包括更新的容器依赖项。

6.1.6. OpenShift File Integrity Operator 1.2.0

以下公告可用于 OpenShift File Integrity Operator 1.2.0 ：

- RHBA-2023:1273 OpenShift File Integrity Operator 增强更新

6.1.6.1. 新功能及功能增强

- File Integrity Operator 自定义资源 (CR) 现在包含一个 **initialDelay** 功能，该功能指定了在启动第一个 AIDE 完整性检查前等待的秒数。如需更多信息，请参阅创建 FileIntegrity 自定义资源。
- File Integrity Operator 现在是稳定的，发行频道已升级到 **stable**。将来的版本将遵循 Semantic Versioning。要访问最新版本，请参阅更新 File Integrity Operator。

6.1.7. OpenShift File Integrity Operator 1.0.0

以下公告可用于 OpenShift File Integrity Operator 1.0.0：

- RHBA-2023:0037 OpenShift File Integrity Operator 程序错误修复更新

6.1.8. OpenShift File Integrity Operator 0.1.32

以下公告可用于 OpenShift File Integrity Operator 0.1.32：

- RHBA-2022:7095 OpenShift File Integrity Operator 程序错误修复更新

6.1.8.1. 程序错误修复

- 在以前的版本中，File Integrity Operator 发布的警报没有设置命名空间，因此很难了解警报来自于哪个命名空间。现在，Operator 会设置适当的命名空间，提供有关该警报的更多信息。(BZ#2112394)
- 在以前的版本中，File Integrity Operator 不会更新 Operator 启动时的指标服务，从而导致指标目标无法访问。在这个版本中，File Integrity Operator 可确保在 Operator 启动时更新 metrics 服务。(BZ#2115821)

6.1.9. OpenShift File Integrity Operator 0.1.30

以下公告可用于 OpenShift File Integrity Operator 0.1.30：

- RHBA-2022:5538 Integrity Operator 程序漏洞修复和功能增强更新

6.1.9.1. 新功能及功能增强

- File Integrity Operator 现在在以下构架中被支持：
  - IBM Power®
  - IBM Z® 和 IBM® LinuxONE

6.1.9.2. 程序错误修复

- 在以前的版本中，File Integrity Operator 发布的警报没有设置命名空间，因此很难了解警报的来源位置。现在，Operator 会设置适当的命名空间，从而增加对警报的了解。(BZ#2101393)

6.1.10. OpenShift File Integrity Operator 0.1.24

以下公告可用于 OpenShift File Integrity Operator 0.1.24：

- RHBA-2022:1331 OpenShift File Integrity Operator 程序错误修复更新

6.1.10.1. 新功能及功能增强

- 现在，您可以使用 **config.maxBackups** 属性配置 **FileIntegrity** 自定义资源(CR)中的最大备份数量。此属性指定从 **re-init** 进程保留的 AIDE 数据库和日志备份数量，以保留在节点上。超出配置数目之外的旧备份会自动修剪。默认值为五个备份。

6.1.10.2. 程序错误修复

- 在以前的版本中，将 Operator 从 0.1.21 之前的版本升级到 0.1.22 可能会导致 **re-init** 功能失败。这是因为 Operator 无法更新 **configMap** 资源标签。现在，升级到最新版本会修复资源标签。(BZ#2049206)

- 在以前的版本中，当强制默认 **configMap** 脚本内容强制时，会比较错误的数据密钥。这会导致在 Operator 升级后，**aide-reinit** 脚本无法正确更新，并导致 **re-init** 进程失败。现在，**daemonSet** 运行完毕，AIDE 数据库 **re-init** 过程可以成功执行。(BZ#2072058)

6.1.11. OpenShift File Integrity Operator 0.1.22

以下公告可用于 OpenShift File Integrity Operator 0.1.22:

- RHBA-2022:0142 OpenShift File Integrity Operator 程序错误修复更新

6.1.11.1. 程序错误修复

- 在以前的版本中，安装有 File Integrity Operator 的系统可能会因为 **/etc/kubernetes/aide.reinit** 文件而中断 OpenShift Container Platform 更新。如果 **/etc/kubernetes/aide.reinit** 文件存在，则会出现这种情况，但稍后在 **ostree** 验证前被删除。在这个版本中，**/etc/kubernetes/aide.reinit** 被移到 **/run** 目录中，以便它不会与 OpenShift Container Platform 更新冲突。(BZ#2033311)

6.1.12. OpenShift File Integrity Operator 0.1.21

以下公告可用于 OpenShift File Integrity Operator 0.1.21：

- RHBA-2021:4631 OpenShift File Integrity Operator 程序漏洞修复和功能增强更新

6.1.12.1. 新功能及功能增强

- web 控制台的 Monitoring 仪表板中会显示与 **FileIntegrity** 扫描结果和处理指标相关的指标。结果使用 **file_integrity_operator_** 前缀标记。

- 如果节点在超过 1 秒的情况下存在完整性失败，则 operator 命名空间警报中提供的默认 **PrometheusRule** 带有一个警告。

- 以下动态 Machine Config Operator 和 Cluster Version Operator 相关文件路径不包括在默认的 AIDE 策略中，以帮助在节点更新过程中阻止假的正状态：

  - /etc/machine-config-daemon/currentconfig
  - /etc/pki/ca-trust/extracted/java/cacerts
  - /etc/cvo/updatepayloads
  - /root/.kube

- AIDE 守护进程进程具有 v0.1.16 的稳定性改进，并且对 AIDE 数据库初始化时可能发生的错误更具弹性。

6.1.12.2. 程序错误修复

- 在以前的版本中，当 Operator 自动升级时，过时的守护进程集不会被删除。在这个版本中，过期的守护进程集会在自动升级过程中被删除。

6.1.13. 其他资源

- 了解 File Integrity Operator

6.2. 安装 FILE INTEGRITY OPERATOR

6.2.1. 使用 Web 控制台安装 File Integrity Operator

先决条件

- 您必须具有 **admin** 权限。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 **Operators → OperatorHub**。
2. 搜索 File Integrity Operator，然后点 **Install**。
3. 保留 **安装模式** 和 **命名空间** 的默认选择，以确保将 Operator 安装到 **openshift-file-integrity** 命名空间中。
4. 点 **Install**。

验证

确认安装成功：

1. 导航到 **Operators → Installed Operators** 页面。
2. 检查是否在 **openshift-file-integrity** 命名空间中安装 Operator，其状态为 **Succeeded**。

如果 Operator 没有成功安装：

1. 导航到 **Operators → Installed Operators** 页面，并检查 **Status** 列中是否有任何错误或故障。
2. 导航到 **Workloads → Pods** 页面，检查 **openshift-file-integrity** 项目中报告问题的 pod 的日志。

6.2.2. 使用 CLI 安装 File Integrity Operator

先决条件

- 您必须具有 **admin** 权限。

流程

1. 运行以下命令，创建一个 **Namespace** 对象 YAML 文件：

```
$ oc create -f <file-name>.yaml
```

输出示例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged (1)
  name: openshift-file-integrity
```

(1) 在 OpenShift Container Platform 4.16 中，pod 安全标签必须在命名空间级别设置为 **privileged**。

2. 创建 **OperatorGroup** 对象 YAML 文件：

```
$ oc create -f <file-name>.yaml
```

输出示例

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

3. 创建 **Subscription** 对象 YAML 文件：

```
$ oc create -f <file-name>.yaml
```

输出示例

```
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
```

验证

1. 通过检查 CSV 文件来验证安装是否成功：

```
$ oc get csv -n openshift-file-integrity
```

2. 验证 File Integrity Operator 是否正在运行：

```
$ oc get deploy -n openshift-file-integrity
```

6.2.3. 其他资源

- 在受限网络环境中支持 File Integrity Operator。如需更多信息，请参阅在受限网络中使用 Operator Lifecycle Manager。

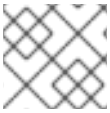
6.3. 更新 FILE INTEGRITY OPERATOR

作为集群管理员，您可以更新 OpenShift Container Platform 集群上的 File Integrity Operator。

6.3.1. 准备 Operator 更新

已安装的 Operator 的订阅指定一个更新频道，用于跟踪和接收 Operator 的更新。您可以更改更新频道，以开始跟踪并从更新频道接收更新。

订阅中更新频道的名称可能会因 Operator 而异，但应遵守给定 Operator 中的常规约定。例如，频道名称可能会遵循 Operator 提供的应用程序的次发行版本更新流（**1.2**、**1.3**）或发行的频率（**stable**、**fast**）。



注意

您不能将已安装的 Operator 更改为比当前频道旧的频道。

红帽客户门户网站 Labs 包括以下应用程序，可帮助管理员准备更新其 Operator：

- Red Hat OpenShift Container Platform Operator Update Information Checker

您可以使用应用程序搜索基于 Operator Lifecycle Manager 的 Operator，并在不同版本的 OpenShift Container Platform 中验证每个更新频道的可用 Operator 版本。不包含基于 Cluster Version Operator 的 Operator。

6.3.2. 更改 Operator 的更新频道

您可以使用 OpenShift Container Platform Web 控制台更改 Operator 的更新频道。

提示

如果订阅中的批准策略被设置为 **Automatic**，则更新过程会在所选频道中提供新的 Operator 版本时立即启动。如果批准策略设为 **Manual**，则必须手动批准待处理的更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 web 控制台的 **Administrator** 视角中，导航到 **Operators → Installed Operators**。
2. 点击您要更改更新频道的 Operator 名称。
3. 点 **Subscription** 标签页。

4. 点 **Update channel** 下的更新频道名称。
5. 点要更改的更新频道，然后点 **Save**。
6. 对于带有 **自动批准策略** 的订阅，更新会自动开始。返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。
   对于采用**手动**批准策略的订阅，您可以从 **Subscription** 选项卡中手动批准更新。

6.3.3. 手动批准待处理的 Operator 更新

如果已安装的 Operator 的订阅被设置为 **Manual**，则当其当前更新频道中发布新更新时，在开始安装前必须手动批准更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，进入 **Operators → Installed Operators**。
2. 处于待定更新的 Operator 会显示 **Upgrade available** 状态。点您要更新的 Operator 的名称。
3. 点 **Subscription** 标签页。任何需要批准的更新都会在 **Upgrade status** 旁边显示。例如：它可能会显示 **1 requires approval**。
4. 点 **1 requires approval**，然后点 **Preview Install Plan**。
5. 检查列出可用于更新的资源。在满意后，点 **Approve**。
6. 返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。

6.4. 了解 FILE INTEGRITY OPERATOR

File Integrity Operator 是一个 OpenShift Container Platform Operator，其在集群节点上持续运行文件完整性检查。它部署一个守护进程集，在每个节点上初始化并运行特权高级入侵检测环境 (AIDE) 容器，从而提供一个状态对象和在守护进程集初始运行时修改的文件日志。

重要

目前，仅支持 Red Hat Enterprise Linux CoreOS (RHCOS) 节点。

6.4.1. 创建 FileIntegrity 自定义资源

FileIntegrity 自定义资源 (CR) 的实例代表一组对一个或多个节点进行的持续文件完整性扫描。

每个 **FileIntegrity** CR 都由一个在符合 **FileIntegrity** CR 规格的节点上运行 AIDE 的守护进程集支持。

流程

1. 创建名为 **worker-fileintegrity.yaml** 的 **FileIntegrity** CR 示例，以便在 worker 节点上启用扫描：

示例 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. 定义调度节点扫描的选择器。
2. 在带有自定义污点的节点上指定调度**容限**。如果没有指定，则应用允许在主节点上运行的默认容限。
3. 定义包含要使用的 AIDE 配置的 **ConfigMap**。
4. AIDE 完整性检查之间暂停的秒数。在节点中频繁进行 AIDE 检查需要大量资源，因此可以指定较长的时间间隔。默认值为 900 秒 (15 分钟)。
5. 从 re-init 进程保留的最大 AIDE 数据库和日志备份数，以保留在节点上。守护进程会自动修剪超出这个数字的旧备份。默认设为 5。
6. 启动第一个 AIDE 完整性检查前等待的秒数。默认设为 0。
7. **FileIntegrity** 实例的运行状态。状态为 **Initializing**、**Pending** 或 **Active**。

Initializing	**FileIntegrity** 对象目前正在初始化或重新初始化 AIDE 数据库。
待处理	**FileIntegrity** 部署仍然被创建。
Active	扫描处于活动状态且持续。

2. 将 YAML 文件应用到 **openshift-file-integrity** 命名空间：

```
$ oc apply -f worker-fileintegrity.yaml -n openshift-file-integrity
```

验证

- 运行以下命令确认 **FileIntegrity** 对象已创建成功：

```
$ oc get fileintegrities -n openshift-file-integrity
```

输**出示例**

```
NAME                  AGE
worker-fileintegrity   14s
```

6.4.2. 检查 FileIntegrity 自定义资源状态

FileIntegrity 自定义资源 (CR) 通过 .**status.phase** 子资源报告其状态。

流程

- 要查询 **FileIntegrity** CR 状态，请运行：

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status.phase }"
```

输**出示例**

```
Active
```

6.4.3. FileIntegrity 自定义资源阶段

- **待处理** - 创建自定义资源 (CR) 后的阶段。
- **Active** - 后备守护进程集启动并运行的阶段。
- **初始化** - AIDE 数据库重新初始化的阶段。

6.4.4. 了解 FileIntegrityNodeStatuses 对象

FileIntegrity CR 的扫描结果会在名为 **FileIntegrityNodeStatuses** 的另一个对象中报告。

```
$ oc get fileintegritynodestatuses
```

输**出示例**

```
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
```

注意

可能需要经过一段时间 **FileIntegrityNodeStatus** 对象才会可用。

每个节点都有一个结果对象。每个 **FileIntegrityNodeStatus** 对象的 **nodeName** 属性都对应于被扫描的节点。文件完整性扫描的状态在 **results** 数组中表示，其中包含扫描条件。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

fileintegritynodestatus 对象报告 AIDE 运行的最新状态，并在 **status** 字段中公开状态为 **Failed**、**Succeeded** 或 **Errored**。

```
$ oc get fileintegritynodestatuses -w
```

输出示例

```
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
```

6.4.5. FileIntegrityNodeStatus CR 状态类型

这些条件会在对应的 **FileIntegrityNodeStatus** CR 状态的结果数组中报告：

- **Succeeded** - 通过完整性检查；AIDE 检查中所涵盖的文件和目录自数据库上次初始化以来没有被修改。
- **Failed** - 完整性检查失败；AIDE 检查中所涵盖的一些文件和目录自数据库上次初始化以来已被修改。
- **Errored** - AIDE 扫描程序遇到内部错误。

6.4.5.1. FileIntegrityNodeStatus CR 成功示例

成功状态条件的输出示例

```
[
  {
```

```
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:57Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:46:03Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:48Z"
  }
]
```

在这种情况下，所有三个扫描都成功，目前没有其他条件。

6.4.5.2. FileIntegrityNodeStatus CR 失败状态示例

要模拟失败条件，请修改 AIDE 跟踪的其中一个文件。例如，在其中一个 worker 节点上修改 **/etc/resolv.conf**：

```
$ oc debug node/ip-10-0-130-192.ec2.internal
```

输出示例

```
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 ...
```

一段时间后，相应的 **FileIntegrityNodeStatus** 对象的结果数组中会报告 **Failed** 条件。之前的 **Succeeded** 条件被保留，便于您查明检查失败的时间。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io/worker-fileintegrity-ip-10-0-130-
192.ec2.internal -ojsonpath='{.results}' | jq -r
```

或者，如果您没有提到对象名称，则运行：

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

输出示例

```
[
  {
```

```
    "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"
  }
]
```

Failed 条件指向一个配置映射，该映射详细介绍了具体的失败及失败原因：

```
$ oc describe cm aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
```

输**出示例**

```
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+a8yTgHPnIHMaRIS+so61EN8VOpg

Events:  <none>
```

由于配置映射数据大小限制，超过 1MB 的 AIDE 日志会作为 base64 编码的 gzip 存档添加到失败配置映射中。在这种情况下，您要将以上命令的输出输出管道到 **base64 --decode | gunzip**。压缩的日志由配置映射中存在 **file-integrity.openshift.io/compressed** 注解键来表示。

6.4.6. 了解事件

FileIntegrity 和 **FileIntegrityNodeStatus** 对象的状态由 *事件* 记录。事件的创建时间反映了最新变化，如从 **Initializing** 变为 **Active**，它不一定是最新的扫描结果。但是，最新的事件始终反映最新的状态。

```
$ oc get events --field-selector reason=FileIntegrityStatus
```

输**出示例**

```
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
```

当节点扫描失败时，会使用 **add/changed/removed** 和配置映射信息创建事件。

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

输**出示例**

```
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
```

更改添加、更改或删除的文件数量会导致新的事件，即使节点的状态尚未转换。

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

输**出示例**

```
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
```

6.5. 配置自定义 FILE INTEGRITY OPERATOR

6.5.1. 查看 FileIntegrity 对象属性

和任何 Kubernetes 自定义资源 (CR) 一样，您可以运行 **oc explain fileintegrity**，然后使用以下方法查看各个属性：

```
$ oc explain fileintegrity.spec
```

```
$ oc explain fileintegrity.spec.config
```

6.5.2. 重要属性

表 6.1. 重要的 spec 和 spec.config 属性

属性	描述
spec.nodeSelector	键值对映射必须与节点标签匹配，才能在该节点上调度 AIDE Pod。典型的用途是仅设置一个键值对，其中 **node-role.kubernetes.io/worker: ""** 在所有 worker 节点上调度 AIDE，**node.openshift.io/os_id: "rhcos"** 在所有 Red Hat Enterprise Linux CoreOS (RHCOS) 节点上调度 AIDE。
spec.debug	布尔值属性。如果设为 **true**，在 AIDE 守护进程集中运行的守护进程会输出额外信息。
spec.tolerations	在带有自定义污点的节点上指定调度容限。如果没有指定，将应用默认容限，允许在 control plane 节点上运行容限。

属性	描述
spec.config.gracePeriod	AIDE 完整性检查之间暂停的秒数。在节点中频繁进行 AIDE 检查需要大量资源，因此可以指定较长的时间间隔。默认为 **900** 秒或 15 分钟。
maxBackups	从 **re-init** 进程保留的最大 AIDE 数据库和日志备份数，以保留在节点上。守护进程会自动修剪超出这个数字的旧备份。
spec.config.name	包含自定义 AIDE 配置的 configMap 的名称。如果省略，则会创建一个默认配置。
spec.config.namespace	包含自定义 AIDE 配置的 configMap 的命名空间。如果未设置，FIO 会生成适合 RHCOS 系统的默认配置。
spec.config.key	在由 **name** 和 **namespace** 指定的配置映射中包含实际 AIDE 配置的关键。默认值为 **aide.conf**。
spec.config.initialDelay	启动第一个 AIDE 完整性检查前等待的秒数。默认设为 0。此属性是可选的。

6.5.3. 检查默认配置

默认 File Integrity Operator 配置存储在与 **FileIntegrity** CR 名称相同的配置映射中。

流程

- 要检查默认配置，请运行：

```
$ oc describe cm/worker-fileintegrity
```

6.5.4. 了解默认的 File Integrity Operator 配置

下面是配置映射的 **aide.conf** 键的摘录：

```
@@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
```

FileIntegrity 实例的默认配置涵盖以下目录下的文件：

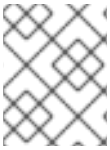
- **/root**
- **/boot**
- **/usr**
- **/etc**

不涵盖以下目录：

- **/var**
- **/opt**
- **/etc/** 下一些特定于 OpenShift Container Platform 的排除项

6.5.5. 提供自定义 AIDE 配置

任何配置 AIDE 内部行为的条目，如 **DBDIR**、**LOGDIR**、**database** 和 **database_out** 都会被 Operator 覆盖。对于要监视是否发生了完整性更改的所有路径，Operator 会在其前面为 **/hostroot/** 添加前缀。这样，要重复使用可能通常不适用于容器化环境并从根目录启动的现有 AIDE 配置，就会更方便。



注意

/hostroot 是运行 AIDE 的 Pod 挂载主机文件系统的目录。更改配置会触发重新初始化数据库。

6.5.6. 定义自定义 File Integrity Operator 配置

本例重点基于为 **worker-fileintegrity** CR 提供的默认配置，为在 control plane 节点上运行的扫描程序定义自定义配置。如果您计划部署作为守护进程集运行的自定义软件，并将其数据存储在 control plane 节点上的 **/opt/mydaemon** 下，则此工作流可能很有用。

流程

1. 复制默认配置。
2. 使用必须监视或排除的文件编辑默认配置。
3. 将已编辑的内容存储至新配置映射中。
4. 通过 **spec.config** 中的属性将 **FileIntegrity** 对象指向新的配置映射。
5. 提取默认配置：

```
$ oc extract cm/worker-fileintegrity --keys=aide.conf
```

这将创建一个名为 **aide.conf** 的文件，您可对其进行编辑。为了说明 Operator 如何对路径进行后期处理，本例添加一个不含前缀的排除目录：

```
$ vim aide.conf
```

输**出示例**

```
/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
```

排除特定于 control plane 节点的路径：

```
!/opt/mydaemon/
```

将其他内容存储在 **/etc** 中：

```
/hostroot/etc/ CONTENT_EX
```

6. 根据该文件创建配置映射：

```
$ oc create cm master-aide-conf --from-file=aide.conf
```

7. 定义引用该配置映射的 **FileIntegrity** CR 清单：

```
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
```

Operator 处理所提供的配置映射文件，并使用与 **FileIntegrity** 对象相同的名称将结果存储在配置映射中：

```
$ oc describe cm/master-fileintegrity | grep /opt/mydaemon
```

输**出示例**

```
!/hostroot/opt/mydaemon
```

6.5.7. 更改自定义文件完整性配置

要更改文件完整性配置，切勿更改生成的配置映射。相反，可通过 **spec.name**、**namespace** 和 **key** 属性更改链接到 **FileIntegrity** 对象的配置映射。

6.6. 执行高级自定义 FILE INTEGRITY OPERATOR 任务

6.6.1. 重新初始化数据库

如果 File Integrity Operator 检测到计划进行的更改，则可能需要重新初始化数据库。

流程

- 使用 **file-integrity.openshift.io/re-init** 注解 **FileIntegrity** 自定义资源（CR）：

```
$ oc annotate fileintegrities/worker-fileintegrity file-integrity.openshift.io/re-init=
```

旧的数据库和日志文件已备份，新的数据库被初始化。旧的数据库和日志保留在 **/etc/kubernetes** 下的节点上，参见使用 **oc debug** 生成的 Pod 中的以下输出：

输**出示例**

```
 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
```

为了提供一些永久记录，生成的配置映射不归 **FileIntegrity** 所有，因此需要手动清理。因此，任何之前的完整性失败仍在 **FileIntegrityNodeStatus** 对象中可见。

6.6.2. 机器配置集成

在 OpenShift Container Platform 4 中，集群节点配置通过 **MachineConfig** 对象提供。您可以假定因 **MachineConfig** 对象导致的对文件进行的更改是预期行为，不应该导致文件完整性扫描失败。为禁止由于 **MachineConfig** 对象更新导致对文件进行更改，File Integrity Operator 会监视节点对象；在更新节点期间，会暂停 AIDE 扫描。更新完成后，数据库会重新初始化并恢复扫描。

此暂停和恢复逻辑只适用于通过 **MachineConfig** API 进行的更新，因为它们反映在节点对象注解中。

6.6.3. 探索守护进程集

每个 **FileIntegrity** 对象代表多个节点上的扫描。扫描本身由守护进程集管理的 Pod 执行。

要查找代表 **FileIntegrity** 对象的守护进程集，请运行：

```
$ oc -n openshift-file-integrity get ds/aide-worker-fileintegrity
```

要列出该守护进程集中的 Pod，请运行：

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

要查看单个 AIDE Pod 的日志，调用其中一个 Pod 上的 **oc logs**。

```
$ oc -n openshift-file-integrity logs pod/aide-worker-fileintegrity-mr8x6
```

输**出示例**

```
Starting the AIDE runner daemon
initializing AIDE db
initialization finished
running aide check
...
```

由 AIDE 守护进程创建的配置映射不会保留，在 File Integrity Operator 处理后会被删除。但是，如果失败和出错，这些配置映射的内容会被复制到 **FileIntegrityNodeStatus** 对象指向的配置映射中。

6.7. 对 FILE INTEGRITY OPERATOR 进行故障排除

6.7.1. 常规故障排除

问题

您通常希望在 File Integrity Operator 中排除问题。

解决方案

在 **FileIntegrity** 对象中启用 debug 标记。**debug** 标记会增加在 **DaemonSet** pod 中运行并运行 AIDE 检查的守护进程详细程度。

6.7.2. 检查 AIDE 配置

问题

您需要检查 AIDE 配置。

解决方案

AIDE 配置存储在与 **FileIntegrity** 对象同名的配置映射中。所有 AIDE 配置的配置映射都标有 **file-integrity.openshift.io/aide-conf**。

6.7.3. 确定 FileIntegrity 对象的阶段

问题

您需要确定 **FileIntegrity** 对象是否存在并查看其当前状态。

解决方案

要查看 **FileIntegrity** 对象的当前状态，请运行：

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status }"
```

一旦创建 **FileIntegrity** 对象和后备守护进程集，状态就应切换为 **Active**。如果没有，请检查 Operator Pod 日志。

6.7.4. 确定守护进程集的 Pod 是否在预期节点上运行

问题

您需要确认守护进程集是否存在，其 Pod 是否在您期望的节点上运行。

解决方案

运行：

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

注意

添加 **-owide** 包括运行 pod 的节点的 IP 地址。

要检查守护进程 pod 的日志，请运行 **oc logs**。

检查 AIDE 命令的返回值，以查看检查通过还是失败。

第 7 章 SECURITY PROFILES OPERATOR

7.1. SECURITY PROFILES OPERATOR 概述

OpenShift Container Platform Security Profiles Operator (SPO) 提供了一种将安全计算 (seccomp) 配置集和 SELinux 配置集定义为自定义资源，将配置集同步到给定命名空间中的每个节点。有关最新更新，请参阅发行注记。

SPO 可以在协调循环时将自定义资源分发到每个节点，确保配置集保持最新状态。请参阅 了解安全配置集 Operator。

SPO 管理 SELinux 策略和 seccomp 配置集，用于命名空间工作负载。如需更多信息，请参阅启用安全配置集 Operator。

您可以创建 seccomp 和 SELinux 配置集，将策略绑定到 pod，记录工作负载，并同步命名空间中的所有 worker 节点。

使用 Advanced Security Profile Operator 任务 来启用日志增强、配置 Webhook 和指标，或者将配置集限制为单个命名空间。

根据需要 xref:[Troubleshoot the Security Profiles Operator]，或联系红帽支持。

您可以在删除 Operator 前删除配置集来卸载 Security Profiles Operator。

7.2. SECURITY PROFILES OPERATOR 发行注记

Security Profiles Operator 提供了将安全计算 (seccomp) 和 SELinux 配置集定义为自定义资源，将配置集同步到给定命名空间中的每个节点。

本发行注记介绍了 OpenShift Container Platform 中 Security Profiles Operator 的开发。

有关 Security Profiles Operator 的概述，请参阅 xref:[Security Profiles Operator Overview]。

7.2.1. Security Profiles Operator 0.8.2

以下公告可用于 Security Profiles Operator 0.8.2：

- RHBA-2023:5958 - OpenShift Security Profiles Operator 程序错误修复更新

7.2.1.1. 程序错误修复

- 在以前的版本中，**SELinuxProfile** 对象不会从同一命名空间中继承自定义属性。在这个版本中，这个问题已被解决，**SELinuxProfile** 对象属性会从与预期相同的命名空间继承。(OCPBUGS-17164)
- 在以前的版本中，RawSELinuxProfiles 会在创建过程中挂起，且不会达到 **Installed** 状态。在这个版本中，这个问题已被解决，RawSELinuxProfiles 已被成功创建。(OCPBUGS-19744)
- 在以前的版本中，将 **enableLogEnricher** 应用到 **true** 会导致 **seccompProfile log-enricher-trace** pod 处于 **Pending** 状态。在这个版本中，**log-enricher-trace** pod 会如预期到达 **Installed** 状态。(OCPBUGS-22182)
- 在以前的版本中，Security Profiles Operator 生成高卡性指标，从而导致 Prometheus pod 使用大量内存。在这个版本中，以下指标将不再在 Security Profiles Operator 命名空间中应用：

- **rest_client_request_duration_seconds**
- **rest_client_request_size_bytes**
- **rest_client_response_size_bytes**
(OCPBUGS-22406)

7.2.2. Security Profiles Operator 0.8.0

以下公告可用于 Security Profiles Operator 0.8.0：

- RHBA-2023:4689 - OpenShift Security Profiles Operator 程序错误修复更新

7.2.2.1. 程序错误修复

- 在以前的版本中，当试图在断开连接的集群中安装 Security Profiles Operator 时，因为 SHA 重新标记问题，提供的安全哈希值不正确。在这个版本中，SHA 与断开连接的环境保持一致。(OCPBUGS-14404)

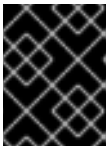
7.2.3. Security Profiles Operator 0.7.1

以下公告可用于 Security Profiles Operator 0.7.1：

- RHSA-2023:2029 - OpenShift Security Profiles Operator 程序错误修复更新

7.2.3.1. 新功能及功能增强

- Security Profiles Operator (SPO) 现在会自动为 RHEL 8 和 9 的 RHCOS 系统选择适当的 **selinuxd** 镜像。



重要

为断开连接的环境镜像镜像必须镜像 Security Profiles Operator 提供的 **selinuxd** 镜像。

- 现在，您可以在 **spod** 守护进程中启用内存优化。如需更多信息，请参阅 spod 守护进程中启用内存优化。

注意

默认情况下不启用 SPO 内存优化。

- 守护进程资源要求现在可以配置。如需更多信息，请参阅自定义守护进程资源要求。
- 优先级类名称现在可以在 **spod** 配置中配置。如需更多信息，请参阅为 spod 守护进程 pod 设置自定义优先级类名称。

7.2.3.2. 弃用和删除的功能

- 现在，默认的 **nginx-1.19.1** seccomp 配置集已从 Security Profiles Operator 部署中删除。

7.2.3.3. 程序错误修复

- 在以前的版本中，Security Profiles Operator (SPO) SELinux 策略不会继承容器模板的低级别策略定义。如果您选择了另一个模板，如 net_container，策略将无法正常工作，因为它只需要仅存在于容器模板中的低级别策略定义。当 SPO SELinux 策略试图将 SELinux 策略从 SPO 自定义格式转换为通用中间语言 (CIL) 格式时，会发生此问题。在这个版本中，容器模板会附加到需要从 SPO 转换到 CIL 的任何 SELinux 策略中。另外，SPO SELinux 策略可以从任何支持的策略模板中继承低级策略定义。(OCPBUGS-12879)

已知问题

- 卸载 Security Profiles Operator 时，**MutatingWebhookConfiguration** 对象不会被删除，必须手动删除。作为临时解决方案，在卸载 Security Profiles Operator 后删除 **MutatingWebhookConfiguration** 对象。这些步骤在卸载安全配置集 Operator 中定义。(OCPBUGS-4687)

7.2.4. Security Profiles Operator 0.5.2

以下公告可用于 Security Profiles Operator 0.5.2：

- RHBA-2023:0788 - OpenShift Security Profiles Operator 程序错误修复更新

这个版本解决了底层依赖项中的 CVE。

已知问题

- 卸载 Security Profiles Operator 时，**MutatingWebhookConfiguration** 对象不会被删除，必须手动删除。作为临时解决方案，在卸载 Security Profiles Operator 后删除 **MutatingWebhookConfiguration** 对象。这些步骤在卸载安全配置集 Operator 中定义。(OCPBUGS-4687)

7.2.5. 安全配置集 Operator 0.5.0

以下公告可用于 Security Profiles Operator 0.5.0 ：

- RHBA-2022:8762 - OpenShift Security Profiles Operator 程序错误修复更新

已知问题

- 卸载 Security Profiles Operator 时，**MutatingWebhookConfiguration** 对象不会被删除，必须手动删除。作为临时解决方案，在卸载 Security Profiles Operator 后删除 **MutatingWebhookConfiguration** 对象。这些步骤在卸载安全配置集 Operator 中定义。(OCPBUGS-4687)

7.3. 了解安全配置集 OPERATOR

OpenShift Container Platform 管理员可以使用 Security Profiles Operator 来定义集群中增加的安全措施。

重要

Security Profiles Operator 仅支持 Red Hat Enterprise Linux CoreOS (RHCOS) worker 节点。不支持 Red Hat Enterprise Linux (RHEL) 节点。

7.3.1. 关于安全配置集

安全配置集可以在集群中的容器级别提高安全性。

seccomp 安全配置集列出了进程可以进行的系统调用。权限比 SELinux 更广泛，可限制操作，如全系统范围内的 **write**。

SELinux 安全配置集提供了一个基于标签的系统，用于限制系统中进程、应用程序或文件的访问和使用。环境中的所有文件都具有定义权限的标签。SELinux 配置集可以在给定的结构中定义访问，如目录。

7.4. 启用安全配置集 OPERATOR

在使用 Security Profiles Operator 之前，您必须确保在集群中部署 Operator。

重要

Security Profiles Operator 仅支持 Red Hat Enterprise Linux CoreOS (RHCOS) worker 节点。不支持 Red Hat Enterprise Linux (RHEL) 节点。

重要

Security Profiles Operator 只支持 **x86_64** 架构。

7.4.1. 安装 Security Profiles Operator

先决条件

- 您必须具有 **admin** 权限。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 **Operators → OperatorHub**。
2. 搜索 Security Profiles Operator，然后点 **Install**。
3. 保留**安装模式**和**命名空间**的默认选择，以确保将 Operator 安装到 **openshift-security-profiles** 命名空间中。
4. 点 **Install**。

验证

确认安装成功：

1. 导航到 **Operators → Installed Operators** 页面。
2. 检查 Security Profiles Operator 是否在 **openshift-security-profiles** 命名空间中安装，其状态是否为 **Succeeded**。

如果 Operator 没有成功安装：

1. 导航到 **Operators → Installed Operators** 页面，并检查 **Status** 列中是否有任何错误或故障。
2. 进入到 **Workloads → Pods** 页面，检查 **openshift-security-profiles** 项目中报告问题的 pod 的日志。

7.4.2. 使用 CLI 安装 Security Profiles Operator

先决条件

- 您必须具有 **admin** 权限。

流程

1. 定义一个 **Namespace** 对象：

   namespace-object.yaml示例

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

2. 创建 **Namespace** 对象：

   ```
   $ oc create -f namespace-object.yaml
   ```

3. 定义一个 **OperatorGroup** 对象：

   operator-group-object.yaml示例

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: security-profiles-operator
     namespace: openshift-security-profiles
   ```

4. 创建 **OperatorGroup** 对象：

   ```
   $ oc create -f operator-group-object.yaml
   ```

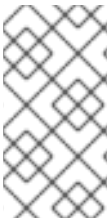
5. 定义一个 **Subscription** 对象：

   subscription-object.yaml示例

   ```
   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. 创建 **Subscription** 对象：

```
$ oc create -f subscription-object.yaml
```



注意

如果要设置全局调度程序功能并启用 **defaultNodeSelector**，您必须手动创建命名空间并更新 **openshift-security-profiles** 命名空间的注解，或安装 Security Profiles Operator 的命名空间，使用 **openshift.io/node-selector: ""**。这会删除默认节点选择器并防止部署失败。

验证

1. 通过检查以下 CSV 文件来验证安装是否成功：

```
$ oc get csv -n openshift-security-profiles
```

2. 运行以下命令，验证 Security Profiles Operator 是否正常工作：

```
$ oc get deploy -n openshift-security-profiles
```

7.4.3. 配置日志记录详细程度

Security Profiles Operator 支持默认日志记录详细程度 **0**，以及更加详细的 **1**。

流程

- 要启用增强的日志详细程度，请运行以下命令来修补 **spod** 配置并调整值：

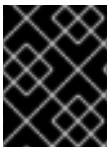
```
$ oc -n openshift-security-profiles patch spod \
    spod --type=merge -p '{"spec":{"verbosity":1}}'
```

输**出示例**

```
securityprofilesoperatordaemon.security-profiles-operator.x-k8s.io/spod patched
```

7.5. 管理 SECCOMP 配置集

创建和管理 seccomp 配置集，并将它们绑定到工作负载。



重要

Security Profiles Operator 仅支持 Red Hat Enterprise Linux CoreOS (RHCOS) worker 节点。不支持 Red Hat Enterprise Linux (RHEL) 节点。

7.5.1. 创建 seccomp 配置集

使用 **SeccompProfile** 对象来创建配置集。

seccompProfile 对象可以限制容器内的系统调用，限制应用程序的访问。

流程

1. 运行以下命令来创建项目：

```
$ oc new-project my-namespace
```

2. 创建 **SeccompProfile** 对象：

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  namespace: my-namespace
  name: profile1
spec:
  defaultAction: SCMP_ACT_LOG
```

seccomp 配置集将保存在 **/var/lib/kubelet/seccomp/operator/<namespace>/<name>.json** 中。

init 容器创建 Security Profiles Operator 的根目录，以便在没有 **root** 组或用户 ID 权限的情况下运行 Operator。会创建一个符合链接，从无根配置存储 **/var/lib/openshift-security-profiles** 到 kubelet root **/var/lib/kubelet/seccomp/operator** 中的默认 **seccomp** root 路径。

7.5.2. 将 seccomp 配置集应用到 pod

创建 pod 以应用其中一个创建的配置集。

流程

1. 创建定义 **securityContext** 的 pod 对象：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: Localhost
      localhostProfile: operator/my-namespace/profile1.json
  containers:
    - name: test-container
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

2. 运行以下命令，查看 **seccompProfile.localhostProfile** 属性的配置集路径：

```
$ oc -n my-namespace get seccompprofile profile1 --output wide
```

输**出示例**

```
NAME      STATUS     AGE   SECCOMPPROFILE.LOCALHOSTPROFILE
profile1   Installed  14s   operator/my-namespace/profile1.json
```

3. 运行以下命令，查看 localhost 配置集的路径：

```
$ oc get sp profile1 --output=jsonpath='{.status.localhostProfile}'
```

输**出示例**

```
operator/my-namespace/profile1.json
```

4. 将 **localhostProfile** 输出应用到补丁文件：

```
spec:
  template:
    spec:
      securityContext:
        seccompProfile:
          type: Localhost
          localhostProfile: operator/my-namespace/profile1.json
```

5. 运行以下命令，将配置集应用到任何其他工作负载，如 **Deployment** 对象：

```
$ oc -n my-namespace patch deployment myapp --patch-file patch.yaml --type=merge
```

输**出示例**

```
deployment.apps/myapp patched
```

验证

- 运行以下命令确认配置集是否已正确应用：

```
$ oc -n my-namespace get deployment myapp --
output=jsonpath='{.spec.template.spec.securityContext}' | jq .
```

输**出示例**

```
{
  "seccompProfile": {
    "localhostProfile": "operator/my-namespace/profile1.json",
    "type": "localhost"
  }
}
```

7.5.2.1. 使用 ProfileBindings 将工作负载绑定到配置集

您可以使用 **ProfileBinding** 资源将安全配置集绑定到容器的 **SecurityContext**。

流程

1. 要将使用 **quay.io/security-profiles-operator/test-nginx-unprivileged:1.21** 镜像的 pod 绑定到示例 **SeccompProfile** 配置集，请在与 pod 和 **SeccompProfile** 对象相同的命名空间中创建一个 **ProfileBinding** 对象：

```
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
```

(1) **kind:** 变量引用配置集的名称。

(2) **name:** 变量引用配置集的名称。

2. 运行以下命令，使用 **enable-binding=true** 标记命名空间：

```
$ oc label ns my-namespace spo.x-k8s.io/enable-binding=true
```

3. 定义名为 **test-pod.yaml** 的 pod：

```
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. 创建 pod：

```
$ oc create -f test-pod.yaml
```

注意

如果 pod 已存在，您必须重新创建 pod 才能使绑定正常工作。

验证

- 运行以下命令确认 pod 会继承 **ProfileBinding** ：

```
$ oc get pod test-pod -o jsonpath='{.spec.containers[*].securityContext.seccompProfile}'
```

输**出示例**

```
{"localhostProfile":"operator/my-namespace/profile.json","type":"Localhost"}
```

7.5.3. 从工作负载记录配置集

Security Profiles Operator 可以使用 **ProfileRecording** 对象记录系统调用，从而更轻松地为应用程序创建基准配置集。

当使用日志增强器来记录 seccomp 配置集时，请验证日志增强功能是否已启用。如需更多信息，请参阅*附加资源*。

注意

具有 **privileged: true** 安全上下文保留的容器可防止基于日志的记录。特权容器不受到 seccomp 策略的影响，基于日志的记录利用特殊的 seccomp 配置集来记录事件。

流程

1. 运行以下命令来创建项目：

```
$ oc new-project my-namespace
```

2. 运行以下命令，使用 **enable-recording=true** 标记命名空间：

```
$ oc label ns my-namespace spo.x-k8s.io/enable-recording=true
```

3. 创建包含 **recorder: logs** 变量的 **ProfileRecording** 对象：

```
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. 创建一个工作负载来记录：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: my-namespace
  name: my-pod
  labels:
    app: my-app
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: nginx
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

```
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: redis
      image: quay.io/security-profiles-operator/redis:6.2.1
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

5. 输入以下命令确认 pod 处于 **Running** 状态：

```
$ oc -n my-namespace get pods
```

输**出示例**

```
NAME    READY   STATUS    RESTARTS   AGE
my-pod   2/2     Running   0          18s
```

6. 确认增强器表示它接收这些容器的审计日志：

```
$ oc -n openshift-security-profiles logs --since=1m --selector name=spod -c log-enricher
```

输**出示例**

```
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"
```

验证

1. 删除 pod：

```
$ oc -n my-namepace delete pod my-pod
```

2. 确认 Security Profiles Operator 协调两个 seccomp 配置集：

```
$ oc get seccompprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

seccompprofile 的输出示例

```
NAME                  STATUS      AGE
test-recording-nginx   Installed   2m48s
test-recording-redis   Installed   2m48s
```

7.5.3.1. 每个容器配置集实例合并

默认情况下，每个容器实例记录都记录到单独的配置文件中。Security Profiles Operator 可将每个容器配置集合并到一个配置集中。当使用 **ReplicaSet** 或 **Deployment** 对象部署应用程序时，合并配置集很有用。

流程

1. 编辑 **ProfileRecording** 对象使其包含 **mergeStrategy: containers** 变量：

```
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. 运行以下命令标记命名空间：

```
$ 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. 使用以下 YAML 创建工作负载：

```
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. 要记录单个配置集，请运行以下命令删除部署：

```
$ oc delete deployment nginx-deploy -n my-namespace
```

5. 要合并配置集，请运行以下命令删除配置集记录：

```
$ oc delete profilerecording test-recording -n my-namespace
```

6. 要启动合并操作并生成结果配置集，请运行以下命令：

```
$ oc get seccompprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

seccompprofiles 的输出示例

```
NAME                          STATUS      AGE
test-recording-nginx-record   Installed   55s
```

7. 要查看任何容器使用的权限，请运行以下命令：

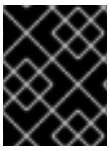
```
$ oc get seccompprofiles test-recording-nginx-record -o yaml
```

其他资源

- 管理安全性上下文约束
- 在 OpenShift 中管理 SCC
- 使用日志增强
- 关于安全配置集

7.6. 管理 SELINUX 配置集

创建和管理 SELinux 配置集并将其绑定到工作负载。



重要

Security Profiles Operator 仅支持 Red Hat Enterprise Linux CoreOS (RHCOS) worker 节点。不支持 Red Hat Enterprise Linux (RHEL) 节点。

7.6.1. 创建 SELinux 配置集

使用 **SelinuxProfile** 对象来创建配置集。

SelinuxProfile 对象有几个功能，允许改进安全强化和可读性：

- 限制配置集从继承到当前命名空间或系统范围的配置集。因为系统中通常会安装很多配置集，但集群工作负载只能使用子集，所以可继承的系统配置集列在 **spec.selinuxOptions.allowedSystemProfiles** 的 **spod** 实例中。
- 执行权限、类和标签的基本验证。
- 添加新的关键字 **@self**，用于描述使用该策略的进程。这允许在工作负载和命名空间间轻松使用策略，因为策略的使用取决于名称和命名空间。

- 与直接在 SELinux CIL 语言中写入配置文件相比，添加了更高的安全强化和易读功能。

流程

1. 运行以下命令来创建项目：

```
$ oc new-project nginx-deploy
```

2. 通过创建以下 **SelinuxProfile** 对象来创建可与非特权工作负载一起使用的策略：

```
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. 运行以下命令，等待 **selinuxd** 安装策略：

```
$ oc wait --for=condition=ready -n nginx-deploy selinuxprofile nginx-secure
```

输**出示例**

```
selinuxprofile.security-profiles-operator.x-k8s.io/nginx-secure condition met
```

策略被放入由 Security Profiles Operator 拥有的容器中的 **emptyDir** 中。策略以通用中间语言 (CIL) 格式保存，格式为 **/etc/selinux.d/<name>_<namespace>.cil**。

4. 运行以下命令来访问 pod：

```
$ oc -n openshift-security-profiles rsh -c selinuxd ds/spod
```

验证

1. 运行以下命令，使用 **cat** 查看文件内容：

```
$ cat /etc/selinux.d/nginx-secure_nginx-deploy.cil
```

输**出示例**

```
(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. 运行以下命令验证策略是否已安装：

```
$ semodule -l | grep nginx-secure
```

输**出示例**

```
nginx-secure_nginx-deploy
```

7.6.2. 将 SELinux 配置集应用到 pod

创建 pod 以应用其中一个创建的配置集。

对于 SELinux 配置集，必须标记命名空间以允许 特权 工作负载。

流程

1. 运行以下命令，将 **scc.podSecurityLabelSync=false** 标签应用到 **nginx-deploy** 命名空间：

```
$ oc label ns nginx-deploy security.openshift.io/scc.podSecurityLabelSync=false
```

2. 运行以下命令，将 **privileged** 标签应用到 **nginx-deploy** 命名空间：

```
$ oc label ns nginx-deploy --overwrite=true pod-security.kubernetes.io/enforce=privileged
```

3. 运行以下命令来获取 SELinux 配置集使用字符串：

```
$ oc get selinuxprofile.security-profiles-operator.x-k8s.io/nginx-secure -n nginx-deploy -
ojsonpath='{.status.usage}'
```

输**出示例**

```
nginx-secure_nginx-deploy.process
```

4. 在 **.spec.containers[].securityContext.seLinuxOptions** 属性中应用工作负载清单中的输出字符串：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-secure
  namespace: nginx-deploy
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
```

```
    type: RuntimeDefault
  containers:
    - image: nginxinc/nginx-unprivileged:1.21
      name: nginx
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
        seLinuxOptions:
          # NOTE: This uses an appropriate SELinux type
          type: nginx-secure_nginx-deploy.process
```

重要

在创建工作负载前，SELinux **type** 必须存在。

7.6.2.1. 应用 SELinux 日志策略

要记录策略违反或 AVC 拒绝，请将 **SElinuxProfile** 配置集设置为 **permissive**。

重要

此流程定义日志策略。它没有设置强制策略。

流程

- 在 **SElinuxProfile** 中添加 **permissive: true** ：

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha2
kind: SelinuxProfile
metadata:
  name: nginx-secure
  namespace: nginx-deploy
spec:
  permissive: true
```

7.6.2.2. 使用 ProfileBindings 将工作负载绑定到配置集

您可以使用 **ProfileBinding** 资源将安全配置集绑定到容器的 **SecurityContext**。

流程

1. 要将使用 **quay.io/security-profiles-operator/test-nginx-unprivileged:1.21** 镜像的 pod 绑定到示例 **SelinuxProfile** 配置集，请在与 pod 和 **SelinuxProfile** 对象相同的命名空间中创建一个 **ProfileBinding** 对象：

```
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
```

1 **kind:** 变量引用配置集的名称。

2 **name:** 变量引用配置集的名称。

2. 运行以下命令，使用 **enable-binding=true** 标记命名空间：

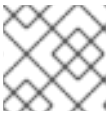
```
$ oc label ns my-namespace spo.x-k8s.io/enable-binding=true
```

3. 定义名为 **test-pod.yaml** 的 pod：

```
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. 创建 pod：

```
$ oc create -f test-pod.yaml
```



注意

如果 pod 已存在，您必须重新创建 pod 才能使绑定正常工作。

验证

- 运行以下命令确认 pod 会继承 **ProfileBinding** ：

```
$ oc get pod test-pod -o jsonpath='{.spec.containers[*].securityContext.seLinuxOptions.type}'
```

输出示例

```
profile_nginx-binding.process
```

7.6.2.3. 复制控制器和 SecurityContextConstraints

当您为复制控制器（如部署或守护进程集）部署 SELinux 策略时，请注意控制器生成的 **Pod** 对象不会与创建工作负载的用户的身份运行。除非选择了 **ServiceAccount**，否则 Pod 可能会恢复到使用不允许使用自定义安全策略的受限 **SecurityContextConstraints** (SCC)。

流程

1. 运行以下命令来创建项目：

```
$ oc new-project nginx-secure
```

2. 创建以下 **RoleBinding** 对象，以允许在 **nginx-secure** 命名空间中使用 SELinux 策略：

```
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. 创建 **Role** 对象：

```
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. 创建 **ServiceAccount** 对象：

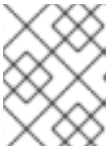
```
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: null
  name: spo-deploy-test
  namespace: nginx-secure
```

5. 创建 **Deployment** 对象：

```
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 在创建 Deployment 前，必须存在 **.seLinuxOptions.type**。



注意

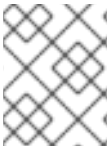
SELinux 类型没有在工作负载中指定，并由 SCC 处理。当 Pod 由部署和 **ReplicaSet** 创建时，pod 将以适当的配置集运行。

确保您的 SCC 仅可供正确的服务帐户使用。如需更多信息，请参阅*附加资源*。

7.6.3. 从工作负载记录配置集

Security Profiles Operator 可以使用 **ProfileRecording** 对象记录系统调用，从而更轻松地为应用程序创建基准配置集。

当使用日志增强器来记录 SELinux 配置集时，请验证日志增强功能是否已启用。如需更多信息，请参阅*附加资源*。



注意

具有 **privileged: true** 安全上下文保留的容器可防止基于日志的记录。特权容器不受到 SELinux 策略的影响，基于日志的记录利用特殊的 SELinux 配置集记录事件。

流程

1. 运行以下命令来创建项目：

```
$ oc new-project my-namespace
```

2. 运行以下命令，使用 **enable-recording=true** 标记命名空间：

```
$ oc label ns my-namespace spo.x-k8s.io/enable-recording=true
```

3. 创建包含 **recorder: logs** 变量的 **ProfileRecording** 对象：

```
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. 创建一个工作负载来记录：

```
apiVersion: v1
kind: Pod
metadata:
  namespace: my-namespace
  name: my-pod
  labels:
    app: my-app
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: nginx
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: redis
      image: quay.io/security-profiles-operator/redis:6.2.1
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

5. 输入以下命令确认 pod 处于 **Running** 状态：

```
$ oc -n my-namespace get pods
```

输**出示例**

```
NAME    READY   STATUS    RESTARTS   AGE
my-pod   2/2     Running   0          18s
```

6. 确认增强器表示它接收这些容器的审计日志：

```
$ oc -n openshift-security-profiles logs --since=1m --selector name=spod -c log-enricher
```

输出示例

```
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"
```

验证

1. 删除 pod：

```
$ oc -n my-namepace delete pod my-pod
```

2. 确认 Security Profiles Operator 协调两个 SELinux 配置集：

```
$ oc get selinuxprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

selinuxprofile 的输出示例

```
NAME                  USAGE                                  STATE
test-recording-nginx  test-recording-nginx_my-namespace.process  Installed
test-recording-redis  test-recording-redis_my-namespace.process  Installed
```

7.6.3.1. 每个容器配置集实例合并

默认情况下，每个容器实例记录都记录到单独的配置文件中。Security Profiles Operator 可将每个容器配置集合并到一个配置集中。当使用 **ReplicaSet** 或 **Deployment** 对象部署应用程序时，合并配置集很有用。

流程

1. 编辑 **ProfileRecording** 对象使其包含 **mergeStrategy: containers** 变量：

```
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. 运行以下命令标记命名空间：

```
$ 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. 使用以下 YAML 创建工作负载：

```
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. 要记录单个配置集，请运行以下命令删除部署：

```
$ oc delete deployment nginx-deploy -n my-namespace
```

5. 要合并配置集，请运行以下命令删除配置集记录：

```
$ oc delete profilerecording test-recording -n my-namespace
```

6. 要启动合并操作并生成结果配置集，请运行以下命令：

```
$ oc get selinuxprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

selinuxprofiles 的输出示例

```
NAME                        USAGE                                      STATE
test-recording-nginx-record   test-recording-nginx-record_my-namespace.process   Installed
```

7. 要查看任何容器使用的权限，请运行以下命令：

```
$ oc get selinuxprofiles test-recording-nginx-record -o yaml
```

7.6.3.2. 关于 SELinuxContext: RunAsAny

SELinux 策略的记录是通过一个 webhook 来实现的，它将一个特殊的 SELinux 类型注入被记录的 pod。SELinux 类型使 pod 以 **permissive** 模式运行，将所有 AVC 拒绝记录到 **audit.log** 中。默认情况下，不允许使用自定义 SELinux 策略运行工作负载，而是使用自动生成的类型。

要记录工作负载，工作负载必须使用具有使用 SCC 权限的服务帐户，允许 Webhook 注入 permissive SELinux 类型。**privileged** SCC 包含 **seLinuxContext: RunAsAny**。

另外，如果集群启用了 Pod Security Admission，则命名空间必须使用 **pod-security.kubernetes.io/enforce: privileged** 标记，因为只有 **privileged** Pod Security Standard 允许使用自定义 SELinux 策略。

其他资源

- 管理安全性上下文约束
- 在 OpenShift 中管理 SCC
- 使用日志增强
- 关于安全配置集

7.7. 高级安全配置集 OPERATOR 任务

使用高级任务来启用指标、配置 Webhook 或限制系统调用。

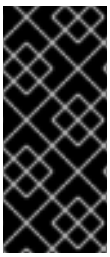
7.7.1. 限制 seccomp 配置集中允许的 syscalls

默认情况下，Security Profiles Operator 不限制 **seccomp** 配置集中的 **syscalls**。您可以在 **spod** 配置中定义允许的 **syscalls** 列表。

流程

- 要定义 **allowedSyscalls** 列表，请运行以下命令来调整 **spec** 参数：

```
$ oc -n openshift-security-profiles patch spod spod --type merge \
    -p '{"spec":{"allowedSyscalls": ["exit", "exit_group", "futex", "nanosleep"]}}'
```



重要

Operator 将仅安装 **seccomp** 配置集，该配置集在允许列表中定义了 **syscalls** 的子集。所有不符合这个规则集的配置集都会被拒绝。

当在 **spod** 配置中修改了允许的 **syscalls** 列表时，Operator 将识别已安装的不合规的配置集，并自动删除它们。

7.7.2. 容器运行时的基本系统调用

您可以使用 **baseProfileName** 属性为给定运行时指定最少需要 **syscalls** 来启动一个容器。

流程

- 编辑 **SeccompProfile** kind 对象，并将 **baseProfileName: runc-v1.0.0** 添加到 **spec** 字段中：

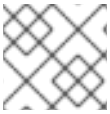
```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
```

```
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
```

7.7.3. 在 spod 守护进程中启用内存优化

在 **spod** 守护进程内运行的控制器会在启用配置集记录时监视集群中的所有 pod。这会导致大型集群中的内存用量很高，从而导致 **spod** 守护进程内存不足或崩溃。

为防止崩溃，**spod** 守护进程只能配置为仅加载标记为配置集记录的 pod。



注意

默认情况下不启用 SPO 内存优化。

流程

1. 运行以下命令来启用内存优化：

```
$ oc -n openshift-security-profiles patch spod spod --type=merge -p '{"spec":
{"enableMemoryOptimization":true}}'
```

2. 要记录 pod 的安全配置集，pod 必须使用 **spo.x-k8s.io/enable-recording: "true"** 标记：

```
apiVersion: v1
kind: Pod
metadata:
  name: my-recording-pod
  labels:
    spo.x-k8s.io/enable-recording: "true"
# ...
```

7.7.4. 自定义守护进程资源要求

可以使用 **spod** 配置中的字段 **daemonResourceRequirements** 来调整守护进程容器的默认资源要求。

流程

- 要指定守护进程容器的内存和 cpu 请求和限值，请运行以下命令：

```
$ oc -n openshift-security-profiles patch spod spod --type merge -p \
    '{"spec":{"daemonResourceRequirements": { \
    "requests": {"memory": "256Mi", "cpu": "250m"}, \
    "limits": {"memory": "512Mi", "cpu": "500m"}}}}'
```

7.7.5. 为 spod 守护进程 pod 设置自定义优先级类名称

spod 守护进程 pod 的默认优先级类名称设置为 **system-node-critical**。通过在 **priorityClassName** 字段中设置值，可以在 **spod** 配置中配置自定义优先级类名称。

流程

- 运行以下命令来配置优先级类名称：

```
$ oc -n openshift-security-profiles patch spod spod --type=merge -p '{"spec":
{"priorityClassName":"my-priority-class"}}'
```

输**出示例**

```
securityprofilesoperatordaemon.openshift-security-profiles.x-k8s.io/spod patched
```

7.7.6. 使用指标

openshift-security-profiles 命名空间提供指标端点，这些端点由 kube-rbac-proxy 容器进行保护。所有指标都由 **openshift-security-profiles** 命名空间中的 **metrics** 服务公开。

Security Profiles Operator 包含一个集群角色和对应的绑定 **spo-metrics-client**，用于从集群中检索指标。有两个指标路径可用：

- **metrics.openshift-security-profiles/metrics**: 用于控制器运行时指标
- **metrics.openshift-security-profiles/metrics-spod**: 用于 Operator 守护进程指标

流程

1. 要查看 metrics 服务的状态，请运行以下命令：

```
$ oc get svc/metrics -n openshift-security-profiles
```

输**出示例**

```
NAME      TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
metrics   ClusterIP   10.0.0.228   <none>        443/TCP   43s
```

2. 要检索指标，请运行以下命令使用 **openshift-security-profiles** 命名空间中的默认 **ServiceAccount** 令牌查询服务端点：

```
$ 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'
```

输**出示例**

```
# 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. 要从不同的命名空间中检索指标，请运行以下命令将 **ServiceAccount** 链接到 **spo-metrics-client ClusterRoleBinding** ：

```
$ oc get clusterrolebinding spo-metrics-client -o wide
```

输**出示例**

```
NAME                ROLE                              AGE   USERS   GROUPS   SERVICEACCOUNTS
spo-metrics-client  ClusterRole/spo-metrics-client    35m                    openshift-security-profiles/default
```

7.7.6.1. controller-runtime 指标

controller-runtime **metrics** 和 DaemonSet 端点 **metrics-spod** 提供了一组默认的指标。守护进程提供了其他指标，该指标始终以 **security_profiles_operator_** 前缀。

表 7.1. 可用的 controller-runtime 指标

指标键	可能的标签	类型	用途
seccomp_profile_total	**operation={delete,update}**	计数	seccomp 配置集操作的数量。
seccomp_profile_audit_total	**node**, **namespace**, **pod**, **container**, **executable**, **syscall**	计数	seccomp 配置集审计操作的数量。需要启用日志增强。
seccomp_profile_bpf_total	**node**, **mount_namespace**, **profile**	计数	seccomp 配置集 bpf 操作的数量。需要启用 bpf 记录器。
seccomp_profile_error_total	**reason={ SeccompNotSupportedOnNode, InvalidSeccompProfile, CannotSaveSeccompProfile, CannotRemoveSeccompProfile, CannotUpdateSeccompProfile, CannotUpdateNodeStatus }**	计数	seccomp 配置集数量。
selinux_profile_total	**operation={delete,update}**	计数	SELinux 配置集操作的数量。

指标键	可能的标签	类型	用途
selinux_profile_audit_total	**node**, **namespace**, **pod**, **container**, **executable**, **scontext**,**tcontext**	计数	SELinux 配置集审核操作的数量。需要启用日志增强。
selinux_profile_error_total	**reason={ CannotSaveSelinuxPolicy, CannotUpdatePolicyStatus, CannotRemoveSelinuxPolicy, CannotContactSelinuxd, CannotWritePolicyFile, CannotGetPolicyStatus }**	计数	SELinux 配置集的数量错误。

7.7.7. 使用日志增强

Security Profiles Operator 包含一个日志功能，它默认是禁用的。日志增强容器以 **privileged** 权限运行，以便从本地节点读取审计日志。日志增强在主机 PID 命名空间 **hostPID** 中运行。

重要

日志增强必须具有读取主机进程的权限。

流程

1. 运行以下命令修补 **spod** 配置以启用日志增强：

```
$ oc -n openshift-security-profiles patch spod spod \
    --type=merge -p '{"spec":{"enableLogEnricher":true}}'
```

输**出示例**

```
securityprofilesoperatordaemon.security-profiles-operator.x-k8s.io/spod patched
```

注意

Security Profiles Operator 将自动重新部署 **spod** 守护进程集。

2. 运行以下命令来查看审计日志：

```
$ oc -n openshift-security-profiles logs -f ds/spod log-enricher
```

输出示例

```
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}
```

7.7.7.1. 使用日志功能跟踪应用程序

您可以使用 Security Profiles Operator 日志增强来跟踪应用程序。

流程

1. 要跟踪应用程序，请创建一个 **SeccompProfile** 日志记录配置集：

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  name: log
  namespace: default
spec:
  defaultAction: SCMP_ACT_LOG
```

2. 创建 pod 对象以使用配置集：

```
apiVersion: v1
kind: Pod
metadata:
  name: log-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: Localhost
      localhostProfile: operator/default/log.json
  containers:
  - name: log-container
    image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

3. 运行以下命令检查日志增强的输出：

```
$ oc -n openshift-security-profiles logs -f ds/spod log-enricher
```

例 7.1. 输出示例

```
…
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"
…
```

7.7.8. 配置 Webhook

配置集绑定和配置集记录对象可以使用 Webhook。配置集绑定和记录对象配置是 **MutatingWebhookConfiguration** CR，由 Security Profiles Operator 管理。

要更改 webhook 配置，**spod** CR 会公开一个 **webhookOptions** 字段，允许修改 **failurePolicy**、**namespaceSelector** 和 **objectSelector** 变量。这可让您将 webhook 设置为 "soft-fail"，或将它们限制为命名空间的子集，以便即使 Webhook 失败，其他命名空间或资源不受影响。

流程

1. 通过创建以下补丁文件，将 **record.spo.io** Webhook 配置设置为仅记录带有 **spo-record=true** 标签的 pod：

```
spec:
  webhookOptions:
    - name: recording.spo.io
      objectSelector:
        matchExpressions:
          - key: spo-record
            operator: In
            values:
              - "true"
```

2. 运行以下命令来修补 **spod/spod** 实例：

```
$ oc -n openshift-security-profiles patch spod \
    spod -p $(cat /tmp/spod-wh.patch) --type=merge
```

3. 要查看生成的 **MutatingWebhookConfiguration** 对象，请运行以下命令：

```
$ oc get MutatingWebhookConfiguration \
    spo-mutating-webhook-configuration -oyaml
```

7.8. 对 SECURITY PROFILES OPERATOR 进行故障排除

对 Security Profiles Operator 进行故障排除以诊断问题或在错误报告中提供信息。

7.8.1. 检查 seccomp 配置集

损坏的 **seccomp** 配置集可能会破坏您的工作负载。不允许其他工作负载映射 **/var/lib/kubelet/seccomp/operator** 的任何部分，以确保用户无法滥用系统。

流程

1. 运行以下命令确认配置集已被协调：

```
$ oc -n openshift-security-profiles logs openshift-security-profiles-<id>
```

例 7.2. 输出示例

```
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. 运行以下命令确认 **seccomp** 配置集已保存到正确的路径中：

```
$ oc exec -t -n openshift-security-profiles openshift-security-profiles-<id> \
    -- ls /var/lib/kubelet/seccomp/operator/my-namespace/my-workload
```

输**出示例**

```
profile-block.json
profile-complain.json
```

7.9. 卸载安全配置集 OPERATOR

您可以使用 OpenShift Container Platform Web 控制台从集群中删除 Security Profiles Operator。

7.9.1. 使用 Web 控制台卸载 Security Profiles Operator

要删除 Security Profiles Operator，您必须首先删除 **seccomp** 和 SELinux 配置集。删除配置集后，您可以通过删除 **openshift-security-profiles** 项目来删除 Operator 及其命名空间。

先决条件

- 访问使用具有 **cluster-admin** 权限的账户的 OpenShift Container Platform 集群。
- 已安装 Security Profiles Operator。

流程

使用 OpenShift Container Platform Web 控制台删除 Security Profiles Operator：

1. 导航到 **Operators → Installed Operators** 页面。
2. 删除所有 **seccomp** 配置集、SELinux 配置集和 Webhook 配置。
3. 切换到 **Administration → Operators → Installed Operators** 页面。
4. 点 **Security Profiles Operator** 条目  中的 Options 菜单并选择 **Uninstall Operator**。

5. 切换到 **Home → Projects** 页面。
6. 搜索**安全配置文件**。
7. 点 **openshift-security-profiles** 项目  旁边的 Options 菜单，然后选择 **Delete Project**。
   a. 通过在对话框中输入 **openshift-security-profiles** 并点 **Delete** 来确认删除。
8. 运行以下命令来删除 **MutatingWebhookConfiguration** 对象：

```
$ oc delete MutatingWebhookConfiguration spo-mutating-webhook-configuration
```

第 8 章 NBDE TANG SERVER OPERATOR

8.1. NBDE TANG SERVER OPERATOR 概述

网络绑定磁盘加密 (NBDE) 使用一个或多个专用网络绑定服务器提供 LUKS 加密卷的自动解锁。NBDE 的客户端称为 Clevis 解密策略框架，服务器端由 Tang 表示。

NBDE Tang Server Operator 允许自动化在 OpenShift Container Platform (OCP) 环境中部署一个或多个 Tang 服务器。

8.2. NBDE TANG SERVER OPERATOR 发行注记

以下发行注记介绍了 OpenShift Container Platform 中 Security Profiles Operator 的开发。

- RHEA-2023:7491 - NBDE Tang Server Operator 1.0 的发布
- RHEA-2024:0854 - NBDE Tang Server Operator 1.0.1 已从 "alpha" 频道移到 "stable" 频道

8.3. 了解 NBDE TANG SERVER OPERATOR

您可以使用 NBDE Tang Server Operator 自动在需要网络绑定磁盘加密(NBDE)的 OpenShift Container Platform 集群中自动部署 Tang 服务器，利用 OpenShift Container Platform 提供的工具来实现此自动化。

NBDE Tang Server Operator 简化了安装过程，并使用 OpenShift Container Platform 环境提供的原生功能，如多副本部署、扩展、流量负载均衡等。Operator 还提供某些操作的自动化，这些操作在手动执行时容易出错，例如：

- 服务器部署和配置
- 密钥轮转
- 删除隐藏的密钥

NBDE Tang Server Operator 使用 Operator SDK 实施，并允许通过自定义资源定义 (CRD) 在 OpenShift 中部署一个或多个 Tang 服务器。

8.3.1. 其他资源

- Tang-Operator: Providing NBDE in OpenShift Red Hat Hybrid Cloud 博客文章
- tang-operator Github 项目
- RHEL 9 安全强化文档中的 使用基于策略的解密配置加密卷的自动解锁 章节

8.4. 安装 NBDE TANG SERVER OPERATOR

您可以使用 Web 控制台或 CLI 通过 **oc** 命令安装 NBDE Tang Operator。

8.4.1. 使用 Web 控制台安装 NBDE Tang Server Operator

您可以使用 Web 控制台从 OperatorHub 安装 NBDE Tang Server Operator。

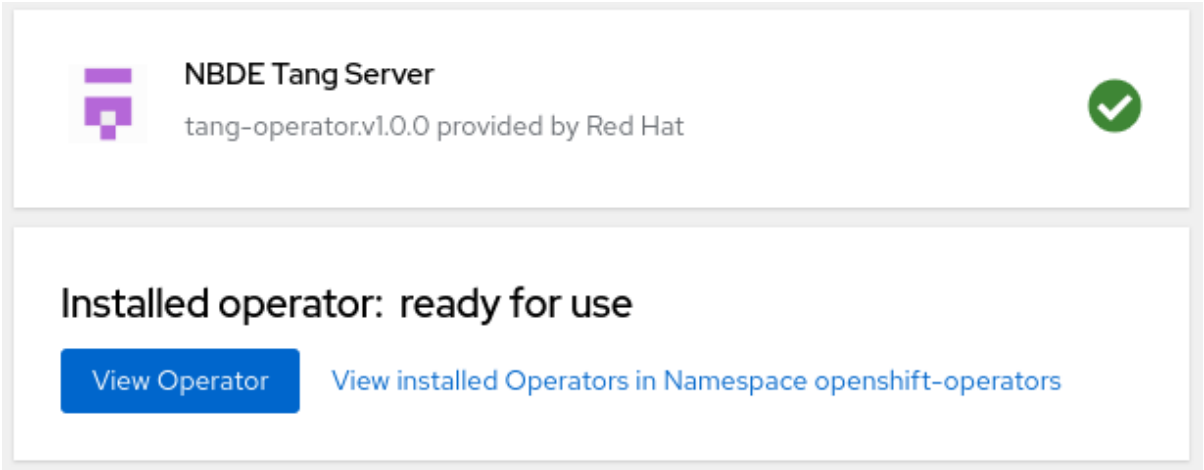
先决条件

- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。

流程

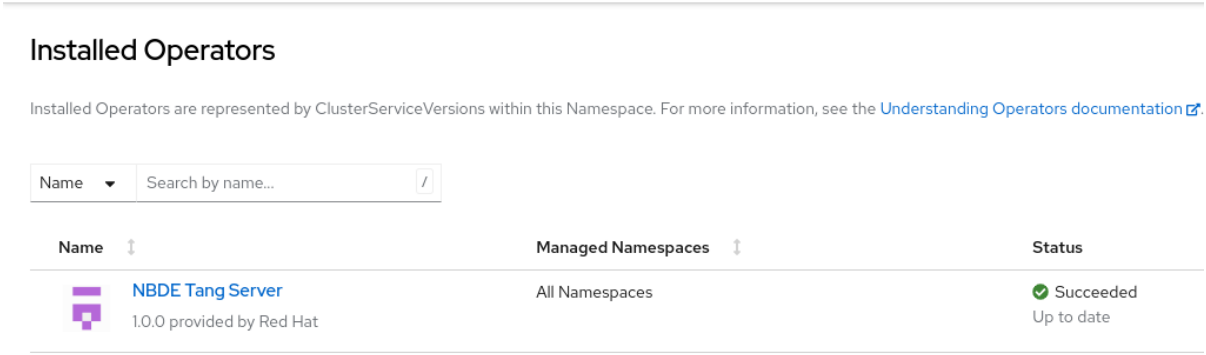
1. 在 OpenShift Container Platform Web 控制台中导航至 **Operators → OperatorHub**。
2. 搜索 NBDE Tang Server Operator：

3. 点 **Install**。
4. 在 **Operator 安装** 屏幕上，将 **Update channel**,**Version**,**Installation mode**,**Installed Namespace**, 和 **Update approval** 字段保持默认值。
5. 点 **Install** 确认安装选项后，控制台会显示安装确认。



验证

1. 导航到 **Operators → Installed Operators** 页面。
2. 检查 NBDE Tang Server Operator 是否已安装，其状态是否为 **Succeeded**。



8.4.2. 使用 CLI 安装 NBDE Tang Server Operator

您可以使用 CLI 从 OperatorHub 安装 NBDE Tang Server Operator。

先决条件

- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。
- 已安装 OpenShift CLI（**oc**）。

流程

1. 使用以下命令列出 OperatorHub 上的可用 Operator，并将输出限制为与 Tang 相关的结果：

```
$ oc get packagemanifests -n openshift-marketplace | grep tang
```

输出示例

```
tang-operator        Red Hat
```

在本例中，对应的 packagemanifest 名称是 **tang-operator**。

2. 创建一个 **Subscription** 对象 YAML 文件，以便为 NBDE Tang Server Operator 订阅一个命名空间，如 **tang-operator.yaml** ：

tang-operator 的订阅 YAML 示例

```
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) 指定您要订阅 Operator 的频道名称。

(2) 指定要订阅的 Operator 的名称。

3 指定提供 Operator 的 CatalogSource 的名称。

4 CatalogSource 的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub CatalogSource。

3. 将订阅应用到集群：

```
$ oc apply -f tang-operator.yaml
```

验证

- 检查 NBDE Tang Server Operator 控制器是否在 **openshift-operators** 命名空间中运行：

```
$ oc -n openshift-operators get pods
```

输**出示例**

```
NAME                                        READY   STATUS    RESTARTS   AGE
tang-operator-controller-manager-694b754bd6-4zk7x   2/2     Running   0          12s
```

8.5. 使用 NBDE TANG SERVER OPERATOR 配置和管理 TANG 服务器

使用 NBDE Tang Server Operator，您可以部署和快速配置 Tang 服务器。在部署的 Tang 服务器上，您可以列出现有的密钥并轮转它们。

8.5.1. 使用 NBDE Tang Server Operator 部署 Tang 服务器

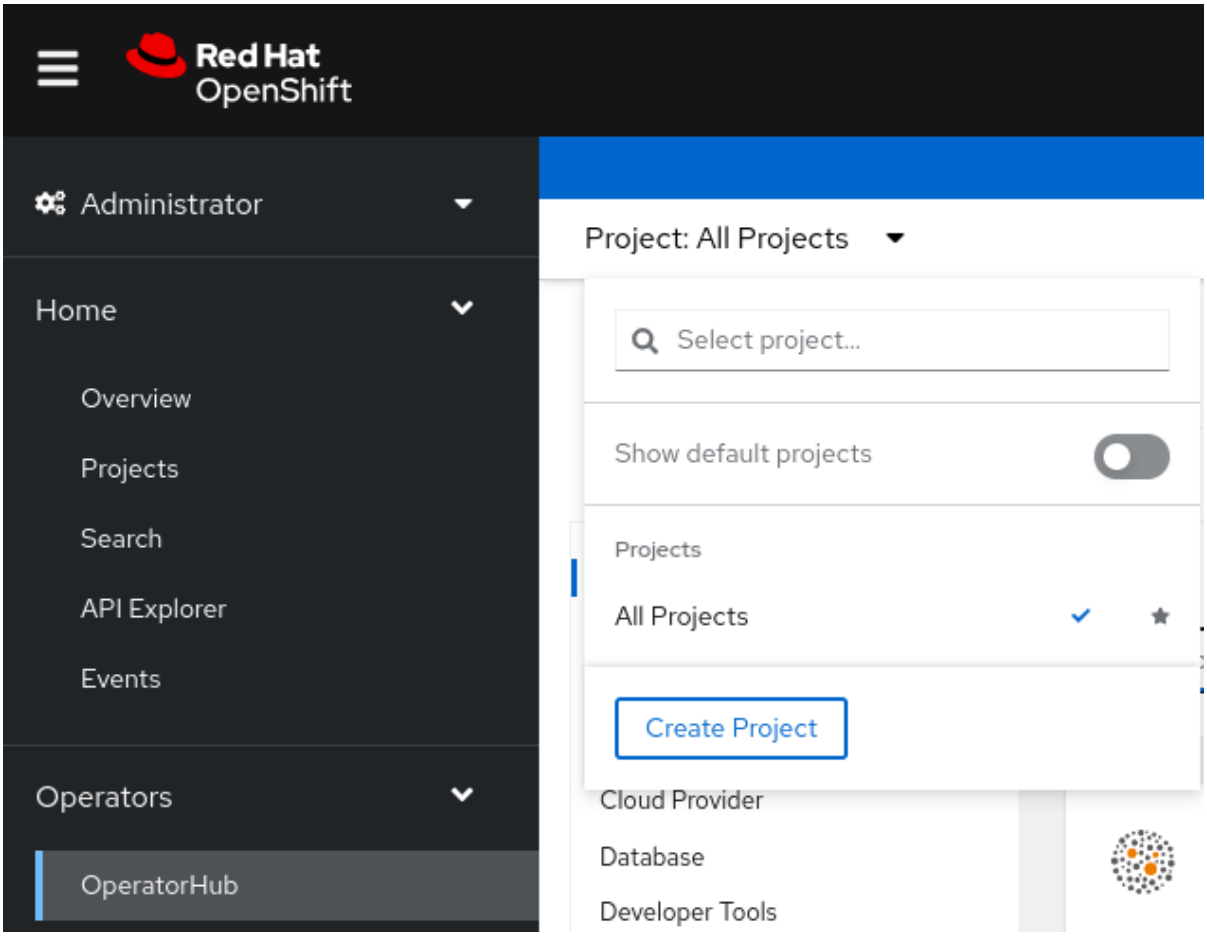
您可以使用 web 控制台中的 NBDE Tang Server Operator 部署和快速配置一个或多个 Tang 服务器。

先决条件

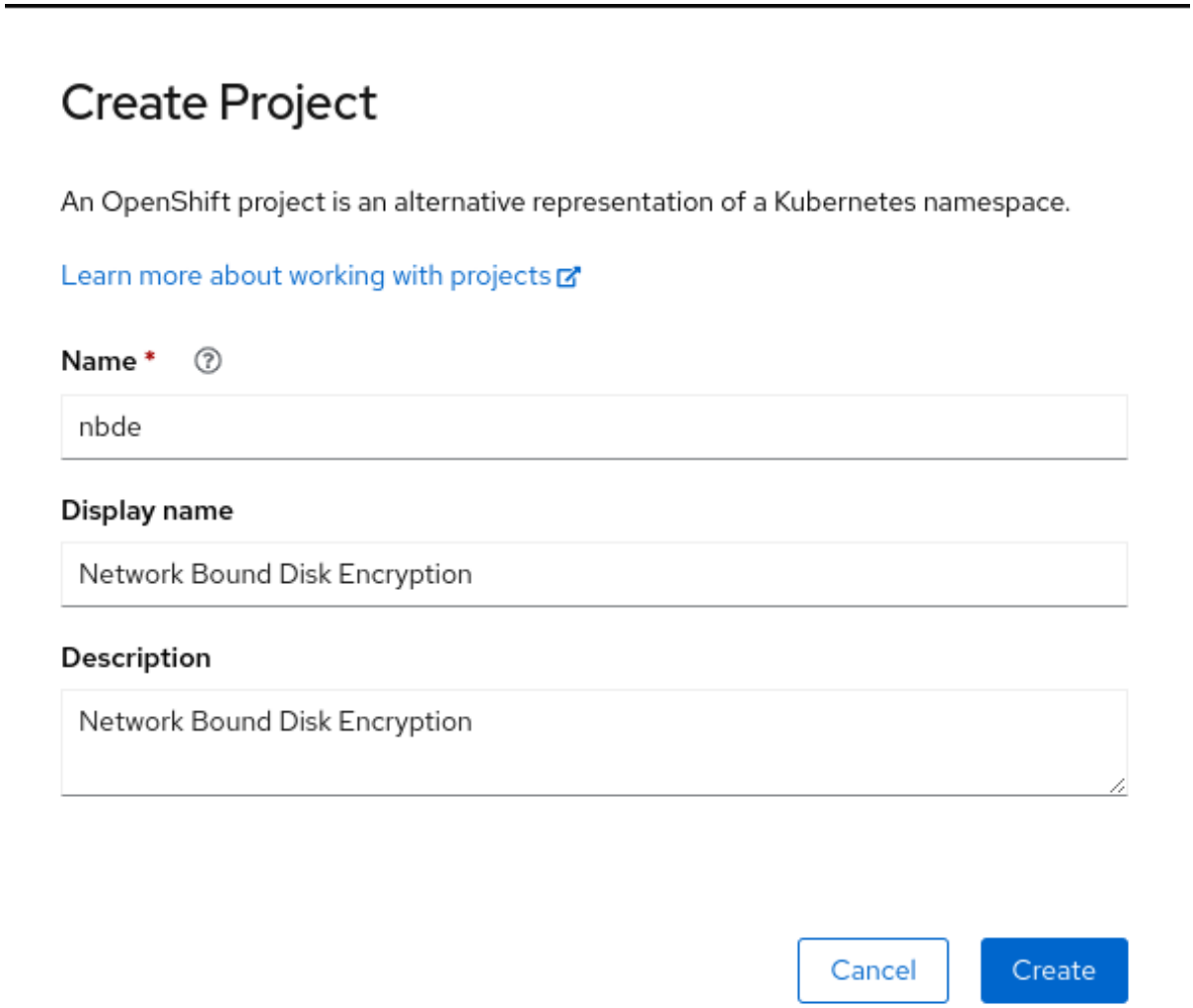
- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。
- 您已在 OCP 集群上安装了 NBDE Tang Server Operator。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 **Operators → OperatorHub**。
2. 选择 **Project**，然后点 **Create Project** ：

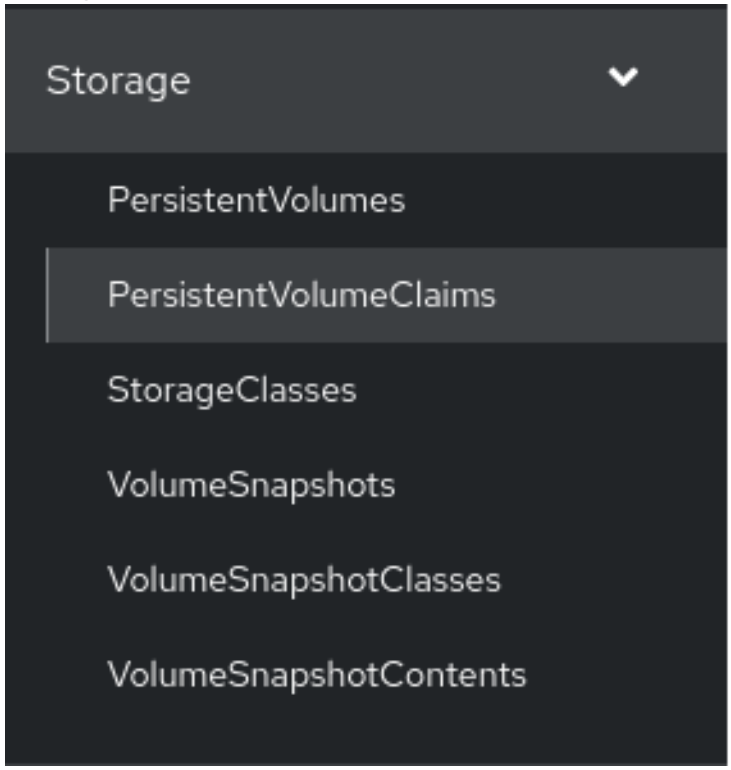


3. 在 **Create Project** 页面中，填写所需信息，例如：



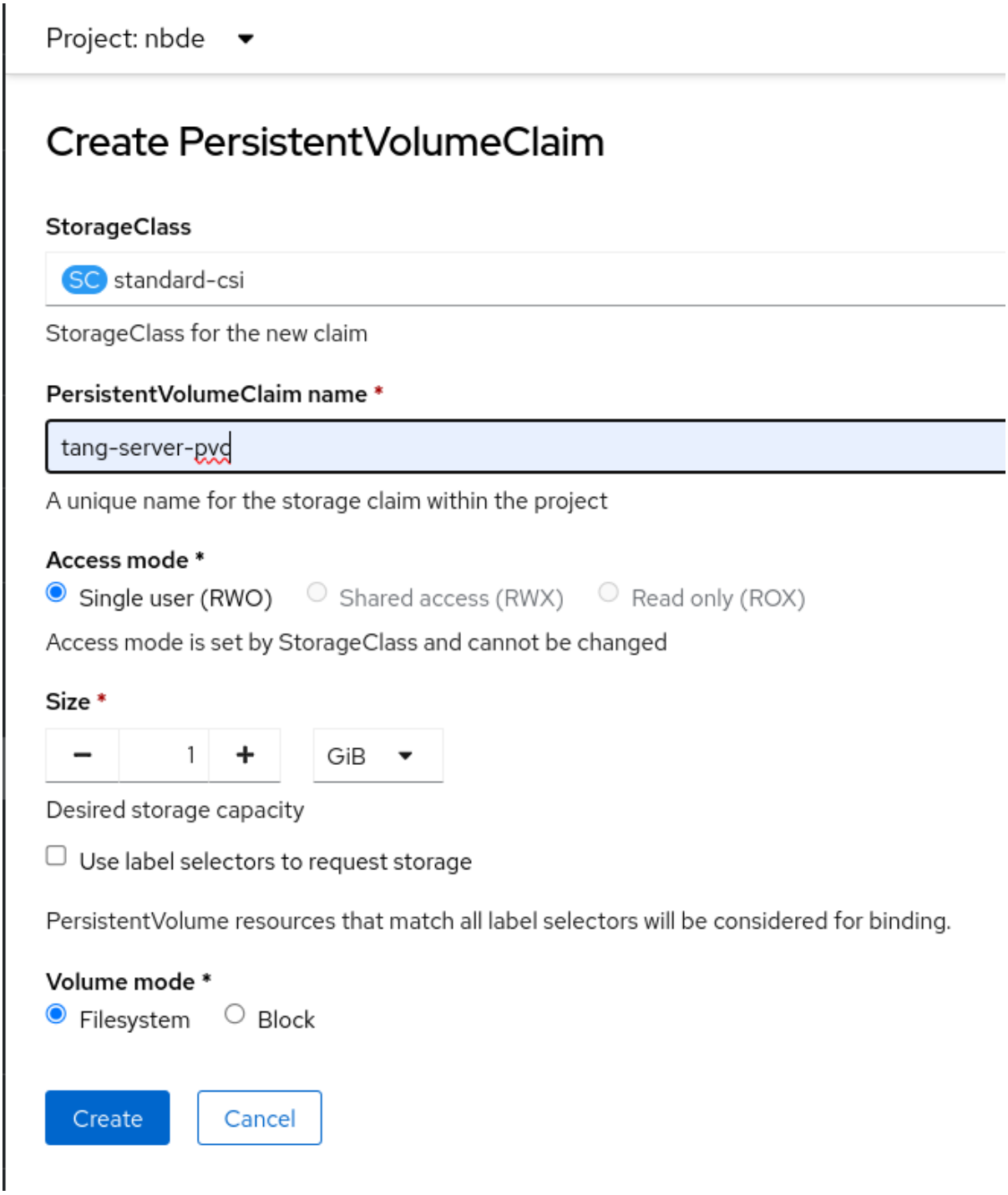
4. 点 **Create**。

5. NBDE Tang 服务器副本需要持久性卷声明 (PVC) 来存储加密密钥。在 Web 控制台中，进入到 **Storage → PersistentVolumeClaims** ：



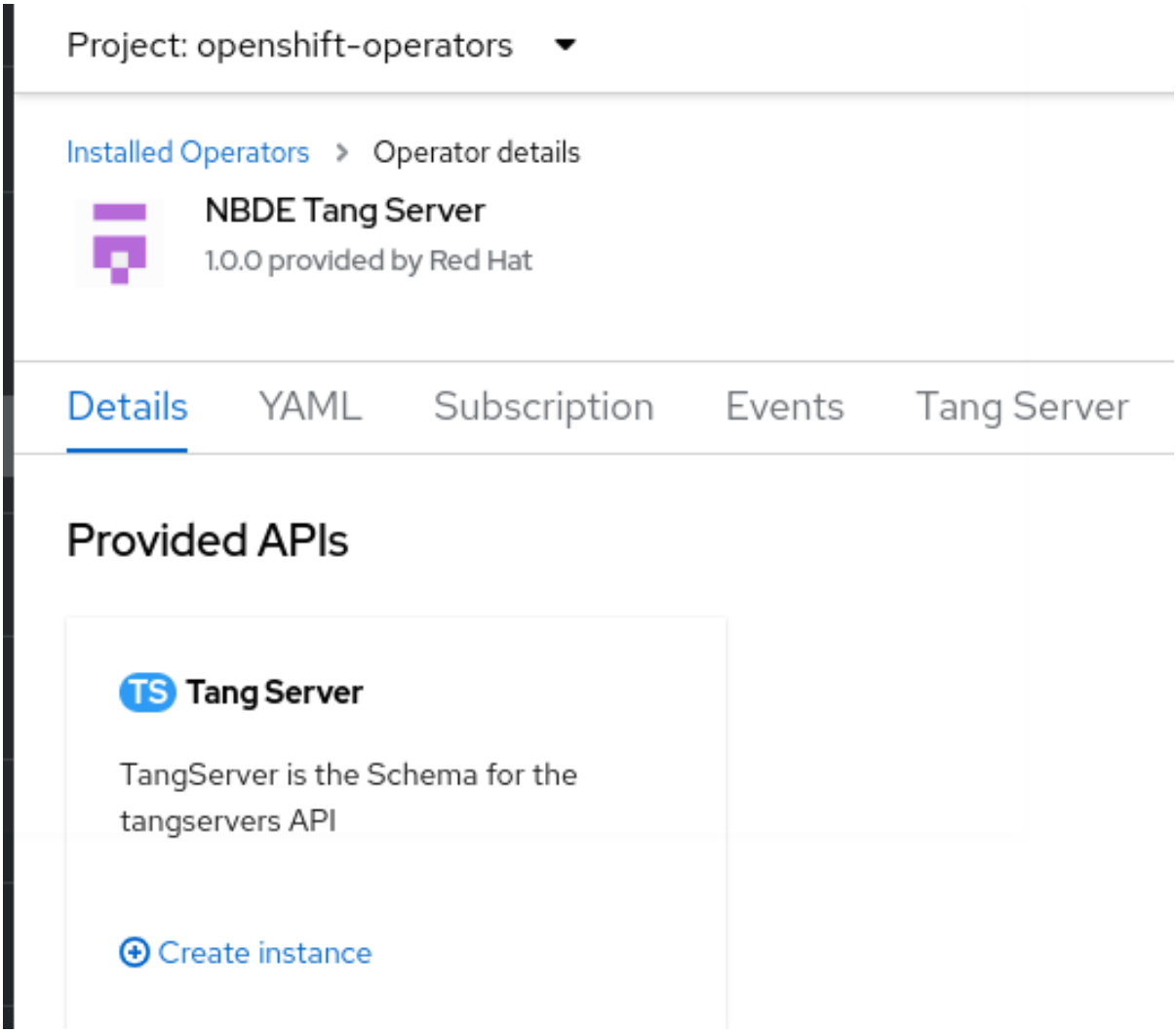
6. 在以下 **PersistentVolumeClaims** 屏幕上，点 **Create PersistentVolumeClaim**。

7. 在 **Create PersistentVolumeClaim** 页面中，选择一个适合您的部署场景的存储。考虑要轮转加密密钥的频率。为您的 PVC 命名并选择声明的存储容量，例如：

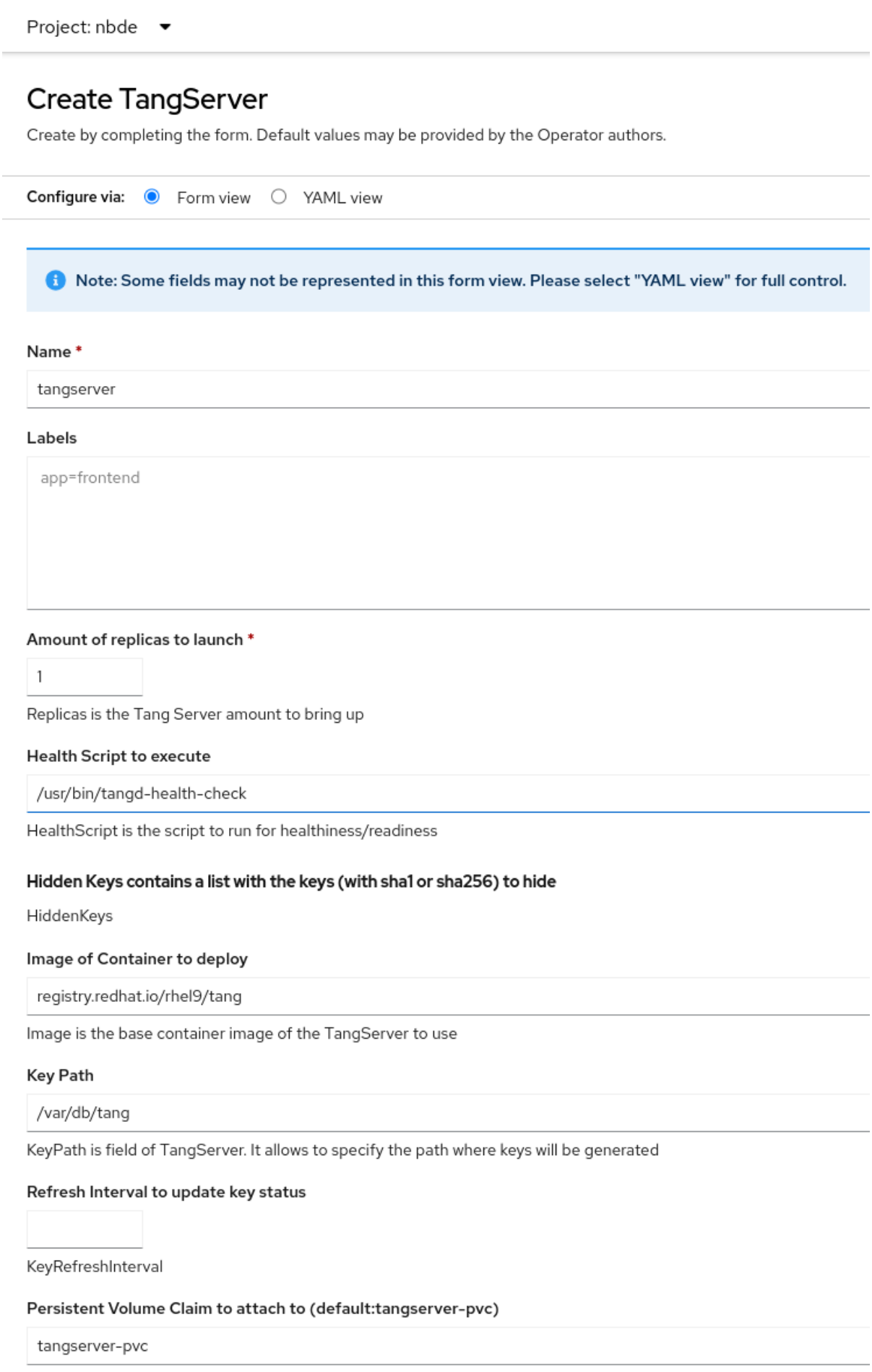


8. 进入到 **Operators → Installed Operators**，然后点 **NBDE Tang Server**。

9. 单击 **Create instance**。



10. 在 **Create TangServer** 页面中，选择 Tang Server 实例的名称，副本数，并指定之前创建的持久性卷声明的名称，例如：



11. 在输入所需的值后，将更改设置与场景中的默认值不同，点 **Create**。

8.5.2. 使用 NBDE Tang Server Operator 轮转密钥

使用 NBDE Tang Server Operator 时，您还可以轮转 Tang 服务器密钥。轮转它们的确切间隔取决于您的应用程序、密钥大小以及机构策略。

先决条件

- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。
- 您已在 OpenShift 集群中使用 NBDE Tang Server Operator 部署了 Tang 服务器。
- 已安装 OpenShift CLI（**oc**）。

流程

1. 列出 Tang 服务器上的现有密钥，例如：

```
$ oc -n nbde describe tangserver
```

输**出示例**

```
…
Status:
  Active Keys:
 File Name:   QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
 Generated:   2022-02-08 15:44:17.030090484 +0000
 sha1:        PvYQKtrTuYsMV2AomUeHrUWkCGg
 sha256:      QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

2. 创建一个 YAML 文件，将您的活跃密钥移到隐藏的密钥中，例如 **minimal-keyretrieve-rotate-tangserver.yaml** ：

tang-operator 的 key-rotation YAML 示例

```
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) 指定活跃密钥的 SHA-1 thumbprint 来轮转它。

3. 应用 YAML 文件：

```
$ oc apply -f minimal-keyretrieve-rotate-tangserver.yaml
```

验证

1. 根据您的配置，在一定时间内，检查之前的 **activeKey** 值是新的 **hiddenKey** 值，并且新生成 **activeKey** 密钥文件，例如：

```
$ oc -n nbde describe tangserver
```

输**出示例**

```
…
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. 使用 NBDE Tang Server Operator 删除隐藏的密钥

轮转 Tang 服务器密钥后，之前活跃的密钥将变为隐藏，并且不再由 Tang 实例公告。您可以使用 NBDE Tang Server Operator 删除加密密钥。

WARNING

除非您确定所有绑定的 Clevis 客户端都使用新密钥，否则不要删除任何隐藏的密钥。

先决条件

- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。
- 您已在 OpenShift 集群中使用 NBDE Tang Server Operator 部署了 Tang 服务器。
- 已安装 OpenShift CLI（**oc**）。

流程

1. 列出 Tang 服务器上的现有密钥，例如：

```
$ oc -n nbde describe tangserver
```

输**出示例**

```
…
```

```
Status:
  Active Keys:
 File Name:   PvYQKtrTuYsMV2AomUeHrUWkCGg.jwk
 Generated:   2022-02-08 15:44:17.030090484 +0000
 sha1:      PvYQKtrTuYsMV2AomUeHrUWkCGg
 sha256:      QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

2. 创建一个 YAML 文件以删除所有隐藏键，如 **hidden-keys-deletion-tangserver.yaml** ：

tang-operator 的 hidden-keys-deletion YAML 示例

```
apiVersion: daemons.redhat.com/v1alpha1
kind: TangServer
metadata:
  name: tangserver
  namespace: nbde
  finalizers:
    - finalizer.daemons.tangserver.redhat.com
spec:
  replicas: 1
  hiddenKeys: [] 1
```

1 空数组作为 **hiddenKeys** 条目的值表示您想要在 Tang 服务器上保留没有隐藏的键。

3. 应用 YAML 文件：

```
$ oc apply -f hidden-keys-deletion-tangserver.yaml
```

验证

1. 根据您的配置，在一定时间之后，检查以前的活跃密钥仍然存在，但没有隐藏的密钥可用，例如：

```
$ oc -n nbde describe tangserver
```

输出示例

```
…
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. 识别使用 NBDE TANG SERVER OPERATOR 部署的 TANG 服务器的 URL

在将 Clevis 客户端配置为使用 Tang 服务器公告的加密密钥之前，您必须识别服务器的 URL。

8.6.1. 使用 Web 控制台识别 NBDE Tang Server Operator 的 URL

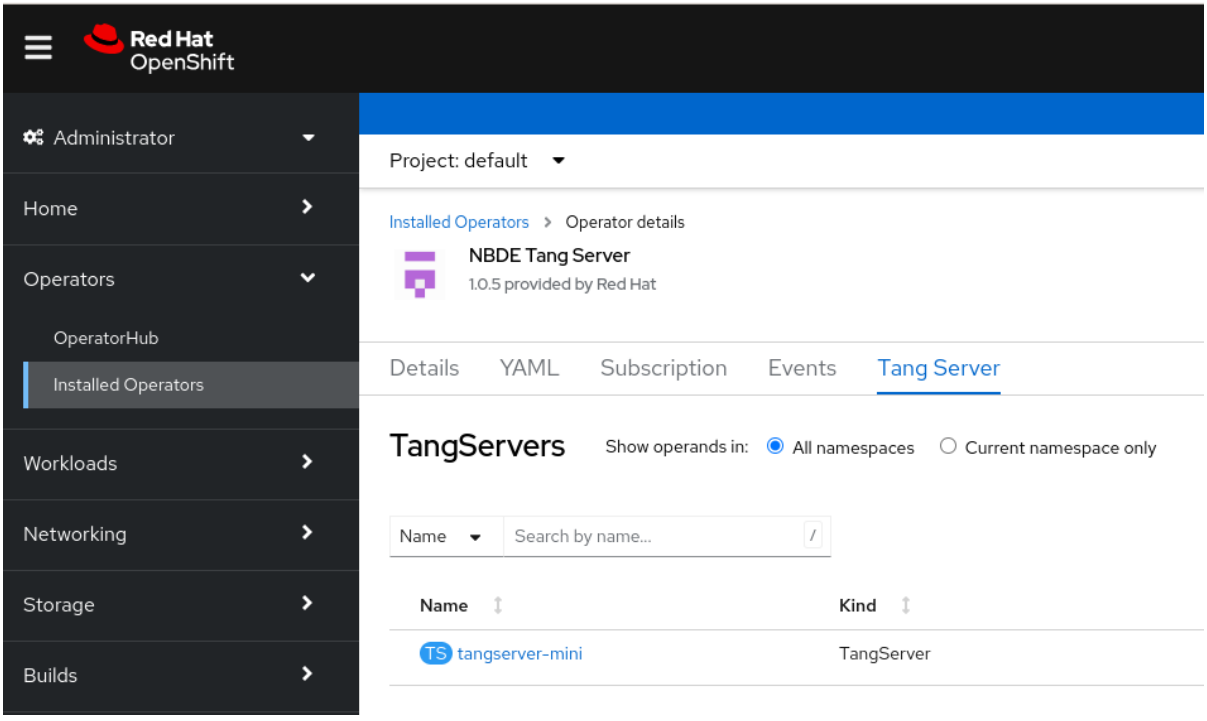
您可以使用 OpenShift Container Platform Web 控制台识别 OperatorHub 中通过 NBDE Tang Server Operator 部署的 Tang 服务器的 URL。识别 URL 后，您可以在包含 LUKS 加密卷的客户端中使用 **clevis luks bind** 命令，您要使用 Tang 服务器公告的密钥自动解锁。有关使用 Clevis 配置客户端的详细信息，请参阅 RHEL 9 安全强化文档中的配置 LUKS 加密卷的手动注册 部分。

先决条件

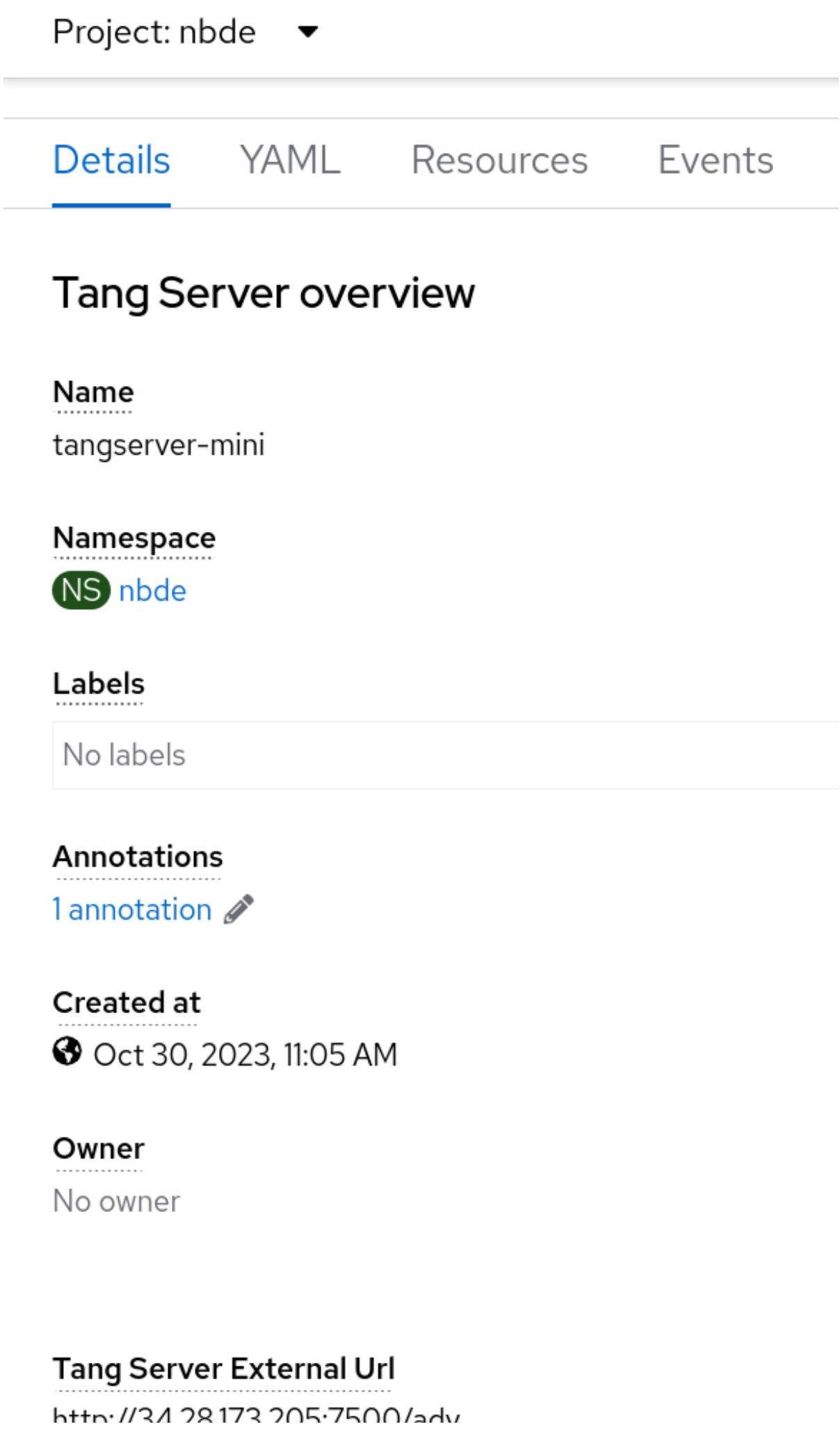
- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。
- 您在 OpenShift 集群中使用 NBDE Tang Server Operator 部署了 Tang 服务器。

流程

1. 在 OpenShift Container Platform web 控制台中进入至 **Operators → Installed Operators → Tang Server**。
2. 在 NBDE Tang Server Operator 详情页面中，选择 **Tang Server**。



3. 此时会出现为集群部署的 Tang 服务器列表。点您要与 Clevis 客户端绑定的 Tang 服务器的名称。
4. Web 控制台显示所选 Tang 服务器的概述。您可以在屏幕的 **Tang Server External Url** 部分找到 Tang 服务器的 URL：


Project: nbde
Details
YAML
Resources
Events
Tang Server overview
Name
tangserver-mini
Namespace
NS nbde
Labels
No labels
Annotations
1 annotation
Created at
Oct 30, 2023, 11:05 AM
Owner
No owner
Tang Server External Url
http://34.28.173.205:7500/adv

http://34.28.173.205:7500/adv

在本例中，Tang 服务器的 URL 是 **http://34.28.173.205:7500**。

验证

- 您可以使用 **curl**、**wget** 或类似工具来检查 Tang 服务器是否广告，例如：

```
$ curl 2> /dev/null http://34.28.173.205:7500/adv  | jq
```

输**出示例**

```
{
  "payload": "eyJrZXlzIj…eSJdfV19",
  "protected": "eyJhbGciOiJFUzUxMiIsImN0eSI6Imp3ay1zZXQranNvbiJ9",
  "signature": "AUB0qSFx0FJLeTU…aV_GYWlDx50vCXKNyMMCRx"
}
```

8.6.2. 使用 CLI 识别 NBDE Tang Server Operator 的 URL

您可以使用 CLI 识别 OperatorHub 中通过 NBDE Tang Server Operator 部署的 Tang 服务器的 URL。识别 URL 后，您可以在包含 LUKS 加密卷的客户端中使用 **clevis luks bind** 命令，您要使用 Tang 服务器公告的密钥自动解锁。有关使用 Clevis 配置客户端的详细信息，请参阅 RHEL 9 安全强化文档中的配置 LUKS 加密卷的手动注册 部分。

先决条件

- 您必须在 OpenShift Container Platform 集群中具有 **cluster-admin** 权限。
- 已安装 OpenShift CLI（**oc**）。
- 您在 OpenShift 集群中使用 NBDE Tang Server Operator 部署了 Tang 服务器。

流程

1. 列出 Tang 服务器的详情，例如：

```
$ oc -n nbde describe tangserver
```

输**出示例**

```
…
Spec:
…
Status:
  Ready:               1
  Running:             1
  Service External URL:  http://34.28.173.205:7500/adv
  Tang Server Error:     No
Events:
…
```

2. 使用 **Service External URL:** 项的值，但没有 **/adv** 部分。在本例中，Tang 服务器的 URL 是 **http://34.28.173.205:7500**。

验证

- 您可以使用 **curl**、**wget** 或类似工具来检查 Tang 服务器是否广告，例如：

```
$ curl 2> /dev/null http://34.28.173.205:7500/adv  | jq
```

输**出示例**

```
{
  "payload": "eyJrZXlzIj…eSJdfV19",
  "protected": "eyJhbGciOiJFUzUxMiIsImN0eSI6Imp3ay1zZXQranNvbiJ9",
  "signature": "AUB0qSFx0FJLeTU…aV_GYWlDx50vCXKNyMMCRx"
}
```

8.6.3. 其他资源

- RHEL 9 安全强化文档中的 配置 LUKS 加密卷的手动注册 部分。

第 9 章 CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

9.1. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT OVERVIEW

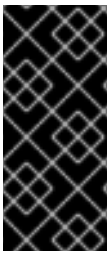
Red Hat OpenShift 的 cert-manager Operator 是一个集群范围的服务，可提供应用程序证书生命周期管理。Red Hat OpenShift 的 cert-manager Operator 允许您与外部证书颁发机构集成并提供证书置备、续订和停用。

9.1.1. 关于 Red Hat OpenShift 的 cert-manager Operator

cert-manager 项目在 Kubernetes API 中引入了证书颁发机构和证书作为资源类型，从而能够根据集群中工作的开发人员提供证书。Red Hat OpenShift 的 cert-manager Operator 提供了一种支持的方法，可将 cert-manager 集成到 OpenShift Container Platform 集群中。

Red Hat OpenShift 的 cert-manager Operator 提供了以下功能：

- 支持与外部证书颁发机构集成
- 管理证书的工具
- 开发人员能够自助使用证书
- 自动证书续订



重要

对于 OpenShift Container Platform，不要同时使用 cert-manager Operator for Red Hat OpenShift 和社区版本的 cert-manager Operator。

另外，您不应该在一个 OpenShift 集群的多个命名空间中为 OpenShift Container Platform 安装 cert-manager Operator。

9.1.2. 支持的签发者类型

Red Hat OpenShift 的 cert-manager Operator 支持以下签发者类型：

- 自动证书管理环境 (ACME)
- 证书颁发机构 (CA)
- 自签名
- Vault
- Venafi

9.1.3. 证书请求方法

对于 Red Hat OpenShift，可以使用 cert-manager Operator 请求证书：

使用 cert-manager.io/CertificateRequest 对象

使用此方法，服务开发人员会创建一个 **CertificateRequest** 对象，其有效的 **issuerRef** 指向配置的签发者（由服务基础架构管理员配置）。服务基础架构管理员随后接受或拒绝证书请求。只有接受的证书请求才会创建对应的证书。

使用 cert-manager.io/Certificate 对象

使用此方法时，服务开发人员使用有效的 **issuerRef** 创建一个 **Certificate** 对象，并从指向证书对象的 secret 获取 **Certificate**。

9.1.4. 为 Red Hat OpenShift 版本支持 cert-manager Operator

OpenShift Container Platform 4.16 支持以下 Red Hat OpenShift 的 cert-manager Operator 版本：

- cert-manager Operator for Red Hat OpenShift 1.13

9.1.5. 其他资源

- cert-manager 项目文档

9.2. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT RELEASE NOTES

Red Hat OpenShift 的 cert-manager Operator 是一个集群范围的服务，可提供应用程序证书生命周期管理。

本发行注记介绍了 Red Hat OpenShift 的 cert-manager Operator 的开发。

如需更多信息，请参阅关于 Red Hat OpenShift 的 cert-manager Operator 。

9.2.1. cert-manager Operator for Red Hat OpenShift 1.13.1

发布日期： 2024 年 5 月 15 日

以下公告可用于 Red Hat OpenShift 1.13.1 的 cert-manager Operator：

- RHEA-2024:2849

Red Hat OpenShift 的 cert-manager Operator 版本 **1.13.1** 基于上游 cert-manager 版本 **v1.13.6**。如需更多信息，请参阅 v1.13.6 的 cert-manager 项目发行注记。

9.2.1.1. CVE

- CVE-2023-45288
- CVE-2023-48795
- CVE-2024-24783

9.2.2. cert-manager Operator for Red Hat OpenShift 1.13.0

发布日期： 2024 年 1 月 16 日

以下公告可用于 Red Hat OpenShift 1.13.0 的 cert-manager Operator：

- RHEA-2024:0259

Red Hat OpenShift 的 cert-manager Operator 版本 **1.13.0** 基于上游 cert-manager 版本 **v1.13.3**。如需更多信息，请参阅 v1.13.0 的 cert-manager 项目发行注记。

9.2.2.1. 新功能及功能增强

- 现在，您可以使用 Red Hat OpenShift 的 cert-manager Operator 管理 API 服务器和 Ingress Controller 的证书。如需更多信息，请参阅使用签发者配置证书。
- 在这个版本中，Red Hat OpenShift 的 cert-manager Operator 范围（以前仅限于 AMD64 架构上的 OpenShift Container Platform），现在已被扩展为支持对 IBM Z® (**s390x**)、IBM Power® (**ppc64le**) 和 ARM64 架构上的证书的管理。
- 在这个版本中，您可以使用 DNS over HTTPS (DoH) 在 ACME DNS-01 质询验证过程中执行自我检查。可以使用命令行标志 **--dns01-recursive-nameservers-only** 和 **--dns01-recursive-nameservers** 来控制 DNS 的自我检查方法。如需更多信息，请参阅 通过覆盖 cert-manager Operator API 中的参数来自定义 cert-manager。

9.2.2.2. CVE

- CVE-2023-39615
- CVE-2023-3978
- CVE-2023-37788
- CVE-2023-29406

9.3. 为 RED HAT OPENSHIFT 安装 CERT-MANAGER OPERATOR

默认情况下，Red Hat OpenShift 的 cert-manager Operator 不会在 OpenShift Container Platform 中安装。您可以使用 Web 控制台为 Red Hat OpenShift 安装 cert-manager Operator。

9.3.1. 为 Red Hat OpenShift 安装 cert-manager Operator

9.3.1.1. 使用 Web 控制台为 Red Hat OpenShift 安装 cert-manager Operator

您可以使用 Web 控制台为 Red Hat OpenShift 安装 cert-manager Operator。

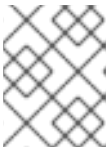
先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 访问 OpenShift Container Platform web 控制台。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 导航至 **Operators → OperatorHub**。
3. 在过滤器框中输入 **cert-manager Operator for Red Hat OpenShift**。
4. 选择 **cert-manager Operator for Red Hat OpenShift**

5. 从 **Version** 下拉列表中选择 cert-manager Operator for Red Hat OpenShift，然后点 **Install**。



注意

请参阅以下 "Additional resources" 部分的 Red Hat OpenShift 版本支持的 cert-manager Operator。

6. 在 **Install Operator** 页面中：

   a. 如果需要，更新**更新频道**。频道默认为 **stable-v1**，它为 Red Hat OpenShift 安装 cert-manager Operator 的最新稳定版本。

   b. 为 Operator 选择 **Installed Namespace**。默认 Operator 命名空间为 **cert-manager-operator**。
   如果 **cert-manager-operator** 命名空间不存在，则会为您创建它。

   c. 选择一个 **更新批准策略**。

      - **Automatic** 策略允许 Operator Lifecycle Manager（OLM）在有新版本可用时自动更新 Operator。
      - **Manual** 策略需要拥有适当凭证的用户批准 Operator 更新。

   d. 点 **Install**。

验证

1. 导航到 **Operators → Installed Operators**。
2. 验证 **cert-manager Operator for Red Hat OpenShift**是否在 **cert-manager-operator** 命名空间中 **Status** 为 **Succeeded**。
3. 输入以下命令验证 cert-manager pod 是否正在运行：

```
$ oc get pods -n cert-manager
```

输出示例

```
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
```

只有在 cert-manager pod 启动并运行后，才可将 cert-manager Operator 用于 Red Hat OpenShift。

9.3.1.2. 使用 CLI 为 Red Hat OpenShift 安装 cert-manager Operator

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。

流程

1. 运行以下命令，创建一个名为 **cert-manager-operator** 的新项目：

```
$ oc new-project cert-manager-operator
```

2. 创建一个 **OperatorGroup** 对象：

   a. 创建包含以下内容的 YAML 文件，如 **operatorGroup.yaml** ：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
spec:
  targetNamespaces:
  - "cert-manager-operator"
```

   b. 运行以下命令来创建 **OperatorGroup** 对象：

```
$ oc create -f operatorGroup.yaml
```

3. 创建 **Subscription** 对象：

   a. 创建一个 YAML 文件，如 **subscription.yaml**，用于定义 **Subscription** 对象：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
spec:
  channel: stable-v1
  name: openshift-cert-manager-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic
  startingCSV: cert-manager-operator.v1.13.0
```

   b. 运行以下命令来创建 **Subscription** 对象：

```
$ oc create -f subscription.yaml
```

验证

1. 运行以下命令验证 OLM 订阅是否已创建：

```
$ oc get subscription -n cert-manager-operator
```

输出示例

```
NAME                              PACKAGE                           SOURCE             CHANNEL
openshift-cert-manager-operator   openshift-cert-manager-operator   redhat-operators   stable-v1
```

2. 运行以下命令验证 Operator 是否已成功安装：

```
$ oc get csv -n cert-manager-operator
```

输**出示例**

```
NAME                         DISPLAY                                       VERSION   REPLACES
PHASE
cert-manager-operator.v1.13.0   cert-manager Operator for Red Hat OpenShift   1.13.0
cert-manager-operator.v1.12.1   Succeeded
```

3. 运行以下命令，验证 Red Hat OpenShift 的 cert-manager Operator 的状态是否为 **Running** ：

```
$ oc get pods -n cert-manager-operator
```

输**出示例**

```
NAME                                                READY   STATUS    RESTARTS   AGE
cert-manager-operator-controller-manager-695b4d46cb-r4hld   2/2     Running   0          7m4s
```

4. 运行以下命令，验证 cert-manager pod 的状态是否为 **Running** ：

```
$ oc get pods -n cert-manager
```

输**出示例**

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-58b7f649c4-dp6l4              1/1     Running   0          7m1s
cert-manager-cainjector-5565b8f897-gx25h   1/1     Running   0          7m37s
cert-manager-webhook-9bc98cbdd-f972x       1/1     Running   0          7m40s
```

其他资源

- 为 Red Hat OpenShift 版本支持 cert-manager Operator

9.3.2. 了解 Red Hat OpenShift 的 cert-manager Operator 的更新频道

更新频道是您可以为集群中的 Red Hat OpenShift 声明 cert-manager Operator 版本的机制。Red Hat OpenShift 的 cert-manager Operator 提供了以下更新频道：

- **stable-v1**
- **stable-v1.y**

9.3.2.1. stable-v1 频道

在为 Red Hat OpenShift 安装 cert-manager Operator 时，**stable-v1** 频道是默认频道和推荐的频道。**stable-v1** 频道为 Red Hat OpenShift 安装并更新 cert-manager Operator 的最新版本。如果要使用 Red Hat OpenShift 的 cert-manager Operator 的最新稳定版本，请选择 **stable-v1** 频道。

stable-v1 频道提供以下更新批准策略：

自动

如果为 Red Hat OpenShift 选择已安装 cert-manager Operator 的自动更新，则 **stable-v1** 频道中提供了 Red Hat OpenShift 的 cert-manager Operator 的新版本。Operator Lifecycle Manager (OLM) 会自动升级 Operator 的运行实例，而无需人为干预。

Manual

如果选择手动更新，则当 Red Hat OpenShift 的 cert-manager Operator 的更新版本可用时，OLM 会创建一个更新请求。作为集群管理员，您必须手动批准该更新请求，才能将 Red Hat OpenShift 的 cert-manager Operator 更新至新版本。

9.3.2.2. stable-v1.y 频道

Red Hat OpenShift 的 cert-manager Operator 的 y-stream 版本从 **stable-v1.y** 频道（如 **stable-v1.10**, **stable-v1.11**, 和 **stable-v1.12**）安装更新。如果要使用 y-stream 版本并更新至 Red Hat OpenShift 的 cert-manager Operator 的 z-stream 版本，请选择 **stable-v1.y** 频道。

stable-v1.y 频道提供以下更新批准策略：

自动

如果为 Red Hat OpenShift 选择已安装 cert-manager Operator 的自动更新，则 **stable-v1.y** 频道提供了 Red Hat OpenShift 的 cert-manager Operator 的新 z-stream 版本。OLM 会在无需人工干预的情况下自动升级 Operator 的运行实例。

Manual

如果选择手动更新，则当 Red Hat OpenShift 的 cert-manager Operator 的更新版本可用时，OLM 会创建一个更新请求。作为集群管理员，您必须手动批准该更新请求，才能将 Red Hat OpenShift 的 cert-manager Operator 更新至 z-stream 版本的新版本。

9.3.3. 其他资源

- 在集群中添加 Operator
- 更新安装的 Operator

9.4. 为 RED HAT OPENSHIFT 配置 CERT-MANAGER OPERATOR 的出口代理

如果在 OpenShift Container Platform 中配置了集群范围的出口代理，Operator Lifecycle Manager (OLM) 会自动配置使用集群范围代理管理的 Operator。OLM 使用 **HTTP_PROXY**、**HTTPS_PROXY**、**NO_PROXY** 环境变量自动更新所有 Operator 的部署。

您可以将代理 HTTPS 连接所需的 CA 证书注入 Red Hat OpenShift 的 cert-manager Operator 中。

9.4.1. 为 Red Hat OpenShift 为 cert-manager Operator 注入自定义 CA 证书

如果 OpenShift Container Platform 集群启用了集群范围代理，您可以将代理 HTTPS 连接所需的 CA 证书注入 Red Hat OpenShift 的 cert-manager Operator 所需的 CA 证书。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您已为 OpenShift Container Platform 启用集群范围代理。

流程

1. 运行以下命令，在 **cert-manager** 命名空间中创建配置映射：

```
$ oc create configmap trusted-ca -n cert-manager
```

2. 运行以下命令，将 OpenShift Container Platform 信任的 CA 捆绑包注入配置映射中：

```
$ oc label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true -n cert-manager
```

3. 运行以下命令，为 Red Hat OpenShift 更新 cert-manager Operator 的部署以使用配置映射：

```
$ 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"}]}}}'
```

验证

1. 运行以下命令验证部署是否已推出：

```
$ 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
```

输**出示例**

```
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. 运行以下命令，验证 CA 捆绑包是否已挂载为卷：

```
$ oc get deployment cert-manager -n cert-manager -o=jsonpath=
{.spec.template.spec.'containers[0].volumeMounts'}
```

输**出示例**

```
[{"mountPath":"/etc/pki/tls/certs/cert-manager-tls-ca-bundle.crt","name":"trusted-
ca","subPath":"ca-bundle.crt"}]
```

3. 运行以下命令，验证 CA 捆绑包的来源是否为 **trusted-ca** 配置映射：

```
$ oc get deployment cert-manager -n cert-manager -o=jsonpath=
{.spec.template.spec.volumes}
```

输**出示例**

```
[{"configMap":{"defaultMode":420,"name":"trusted-ca"},"name":"trusted-ca"}]
```

9.4.2. 其他资源

- 在 Operator Lifecycle Manager 中配置代理支持

9.5. 自定义 CERT-MANAGER OPERATOR API 字段

您可以通过覆盖环境变量和参数来为 Red Hat OpenShift API 字段自定义 cert-manager Operator。

警告

要覆盖不支持的参数，您可以在 **CertManager** 资源中添加 **spec.unsupportedConfigOverrides** 部分，但不支持使用 **spec.unsupportedConfigOverrides**。

9.5.1. 通过覆盖 cert-manager Operator API 中的环境变量来自定义 cert-manager

您可以通过在 **CertManager** 资源中添加 **spec.controllerConfig** 部分来覆盖 Red Hat OpenShift 的 cert-manager Operator 支持的环境变量。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问 OpenShift Container Platform 集群。

流程

1. 运行以下命令来编辑 **CertManager** 资源：

```
$ oc edit certmanager cluster
```

2. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

```
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 将 **<proxy_url>** 替换为代理服务器 URL。

3 将 **<ignore_proxy_domains>** 替换为以逗号分隔的域列表。代理服务器会忽略这些域。

3. 保存更改并退出文本编辑器以应用您的更改。

验证

1. 运行以下命令，验证 cert-manager 控制器 pod 是否已重新部署：

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

输出示例

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          39s
```

2. 运行以下命令，验证是否为 cert-manager pod 更新环境变量：

```
$ oc get pod <redeployed_cert-manager_controller_pod> -n cert-manager -o yaml
```

输出示例

```
    env:
    ...
    - name: HTTP_PROXY
      value: http://<PROXY_URL>
    - name: HTTPS_PROXY
      value: https://<PROXY_URL>
    - name: NO_PROXY
      value: <IGNORE_PROXY_DOMAINS>
```

9.5.2. 通过覆盖 cert-manager Operator API 中的参数来自定义 cert-manager

您可以通过在 **CertManager** 资源中添加 **spec.controllerConfig** 部分来覆盖 Red Hat OpenShift 的 cert-manager Operator 支持的参数。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问 OpenShift Container Platform 集群。

流程

1. 运行以下命令来编辑 **CertManager** 资源：

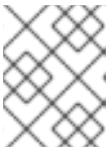
```
$ oc edit certmanager cluster
```

2. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
```

```
...
spec:
 ...
 controllerConfig:
  overrideArgs:
    - '--dns01-recursive-nameservers=<server_address>' 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. 提供以逗号分隔的名称服务器列表，以查询 DNS-01 自我检查。命名空间的指定形式可以是 **<host>:<port>**（例如 **1.1.1.1:53**），或使用 DNS over HTTPS (DoH)（例如 **https://1.1.1.1/dns-query**）。
2. 指定只使用递归名称服务器，而不是检查与该域关联的权威名称服务器。
3. 提供以逗号分隔的 **<host>:<port>** 名称服务器列表，以查询自动证书管理环境(ACME) HTTP01 自我检查。例如，**--acme-http01-solver-nameservers=1.1.1.1:53**。
4. 7. 8. 指定设置日志级别详细程度来确定日志消息的详细程度。
5. 指定指标端点的主机和端口。默认值为 **--metrics-listen-address=0.0.0.0:9402**。
6. 在配置 ACME Issuer 以使用 **--issuer-ambient-credentials** 参数时，您必须使用 ambient 凭证来解决 DNS-01 质询。



注意

仅从 cert-manager Operator for Red Hat OpenShift version 1.13.0 或更高版本开始才支持 DNS over HTTPS (DoH)。

3. 保存更改并退出文本编辑器以应用您的更改。

验证

- 运行以下命令，验证是否为 cert-manager pod 更新参数：

```
$ oc get pods -n cert-manager -o yaml
```

输出示例

```
...
 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
```

9.5.3. 在删除证书时自动删除 TLS secret

您可以通过在 **CertManager** 资源中添加 **spec.controllerConfig** 部分，为 Red Hat OpenShift 的 cert-manager Operator 启用 **--enable-certificate-owner-ref** 标志。**--enable-certificate-owner-ref** 标志将证书资源设置为存储 TLS 证书的 secret 的所有者。

警告

如果您为 Red Hat OpenShift 卸载 cert-manager Operator，或从集群中删除证书资源，secret 会被自动删除。这可能导致网络连接问题，具体取决于使用证书 TLS secret 的位置。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问 OpenShift Container Platform 集群。
- 您已为 Red Hat OpenShift 1.12.0 或更高版本安装了 cert-manager Operator。

流程

1. 运行以下命令，检查 **Certificate** 对象及其 secret 是否可用：

```
$ oc get certificate
```

输**出示例**

```
NAME                                                READY   SECRET                                              AGE
certificate-from-clusterissuer-route53-ambient   True    certificate-from-clusterissuer-route53-ambient   8h
```

2. 运行以下命令来编辑 **CertManager** 资源：

```
$ oc edit certmanager cluster
```

3. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
# ...
spec:
# ...
  controllerConfig:
    overrideArgs:
      - '--enable-certificate-owner-ref'
```

4. 保存更改并退出文本编辑器以应用您的更改。

验证

- 运行以下命令，验证 cert-manager controller pod 是否更新了 **--enable-certificate-owner-ref** 标志：

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager -o yaml
```

输**出示例**

```
# ...
  metadata:
    name: cert-manager-6e4b4d7d97-zmdnb
    namespace: cert-manager
# ...
  spec:
    containers:
    - args:
      - --enable-certificate-owner-ref
```

9.5.4. 覆盖 cert-manager 组件的 CPU 和内存限值

为 Red Hat OpenShift 安装 cert-manager Operator 后，您可以为 cert-manager 组件（如 cert-manager controller、CA injector 和 Webhook）从 cert-manager Operator 为 Red Hat OpenShift API 配置 CPU 和内存限值。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问 OpenShift Container Platform 集群。
- 您已为 Red Hat OpenShift 1.12.0 或更高版本安装了 cert-manager Operator。

流程

1. 输入以下命令检查 cert-manager 控制器、CA injector 和 Webhook 的部署是否可用：

```
$ oc get deployment -n cert-manager
```

输**出示例**

```
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. 在设置 CPU 和内存限值前，请输入以下命令检查 cert-manager 控制器、CA injector 和 Webhook 的现有配置：

```
$ oc get deployment -n cert-manager -o yaml
```

输**出示例**

```
# ...
  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 默认情况下，**spec.resources** 字段为空。cert-manager 组件没有 CPU 和内存限值。

3. 要为 cert-manager 控制器、CA injector 和 Webhook 配置 CPU 和内存限值，请输入以下命令：

```
$ 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 定义 cert-manager 控制器 pod 中单个容器可以请求的最大 CPU 和内存量。

2 5 您可以指定 cert-manager 控制器 pod 可以请求的 CPU 限制。默认值为 **10m**。

3 6 您可以指定 cert-manager 控制器 pod 可以请求的内存限值。默认值为 **32Mi**。

4 为 cert-manager 控制器 pod 定义调度程序设置的 CPU 和内存量。

7 定义 CA 注入程序 pod 中单个容器可以请求的最大 CPU 和内存量。

8 11 您可以指定 CA injector pod 可以请求的 CPU 限制。默认值为 **10m**。

9 12 您可以指定 CA 注入程序 pod 可以请求的内存限值。默认值为 **32Mi**。

10 为 CA 注入器 pod 定义调度程序设置的 CPU 和内存量。

13 定义最大 CPU 和内存量，定义 Webhook pod 中单个容器可以请求的最大 CPU 和内存量。

14 17 您可以指定 Webhook pod 可以请求的 CPU 限值。默认值为 **10m**。

15 18 您可以指定 Webhook pod 可以请求的内存限值。默认值为 **32Mi**。

16 定义 Webhook pod 的调度程序设置的 CPU 和内存量。

输**出示例**

```
certmanager.operator.openshift.io/cluster patched
```

验证

1. 验证 cert-manager 组件的 CPU 和内存限值是否已更新：

```
$ oc get deployment -n cert-manager -o yaml
```

输**出示例**

```
# ...
  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
# ...
```

9.6. 为 RED HAT OPENSHIFT 验证 CERT-MANAGER OPERATOR

您可以通过配置云凭证，为集群中的 Red Hat OpenShift 验证 cert-manager Operator。

9.6.1. 在 AWS 上进行身份验证

先决条件

- 您已为 Red Hat OpenShift 1.11.1 或更高版本安装了 cert-manager Operator。
- 您已将 Cloud Credential Operator 配置为以 *mint* 或 *passthrough* 模式运行。

流程

1. 创建一个 **CredentialsRequest** 资源 YAML 文件，如 **sample-credential-request.yaml**，如下所示：

```
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. 运行以下命令来创建 **CredentialsRequest** 资源：

```
$ oc create -f sample-credential-request.yaml
```

3. 运行以下命令，为 Red Hat OpenShift 更新 cert-manager Operator 的订阅对象：

```
$ 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"}]}}}'
```

验证

1. 运行以下命令，获取重新部署的 cert-manager 控制器 pod 的名称：

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

输出示例

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          15m39s
```

2. 运行以下命令，验证 cert-manager 控制器 pod 是否使用挂载在 **mountPath** 中指定的路径下的 AWS 凭证卷更新：

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

输出示例

```
...
spec:
 containers:
 - args:
   ...
   - mountPath: /.aws
     name: cloud-credentials
 ...
```

```
volumes:
...
- name: cloud-credentials
  secret:
    ...
    secretName: aws-creds
```

9.6.2. 使用 AWS 安全令牌服务进行身份验证

先决条件

- 您已提取并准备好 **ccoctl** 二进制文件。
- 已使用手动模式的 Cloud Credential Operator 配置了一个带有 AWS STS 的 OpenShift Container Platform 集群。

流程

1. 运行以下命令，创建一个目录来存储 **CredentialsRequest** 资源 YAML 文件：

```
$ mkdir credentials-request
```

2. 通过应用以下 yaml，在 **credentials-request** 目录下创建一个 **CredentialsRequest** 资源 YAML 文件，如 **sample-credential-request.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. 运行以下命令，使用 **ccoctl** 工具处理 **CredentialsRequest** 对象：

```
$ 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>
```

输**出示例**

```
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
```

从输出中复制 **<aws_role_arn>** 以在下一步中使用。例如，**"arn:aws:iam::XXXXXXXXXXXX:role/<user_defined_name>-cert-manager-aws-creds"**

4. 运行以下命令，将 **eks.amazonaws.com/role-arn="<aws_role_arn>"** 注解添加到服务帐户：

```
$ oc -n cert-manager annotate serviceaccount cert-manager eks.amazonaws.com/role-arn="<aws_role_arn>"
```

5. 要创建新 pod，请运行以下命令删除现有 cert-manager 控制器 pod：

```
$ oc delete pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

AWS 凭证在一分钟内应用到一个新的 cert-manager 控制器 pod。

验证

1. 运行以下命令，获取更新的 cert-manager 控制器 pod 的名称：

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

输**出示例**

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          39s
```

2. 运行以下命令验证 AWS 凭证是否已更新：

```
$ oc set env -n cert-manager po/<cert_manager_controller_pod_name> --list
```

输**出示例**

```
# 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
```

其他资源

- 配置 Cloud Credential Operator 工具

9.6.3. 在 GCP 上进行身份验证

先决条件

- 您已为 Red Hat OpenShift 1.11.1 或更高版本安装了 cert-manager Operator。
- 您已将 Cloud Credential Operator 配置为以 *mint* 或 *passthrough* 模式运行。

流程

1. 通过应用以下 yaml 创建 **CredentialsRequest** 资源 YAML 文件，如 **sample-credential-request.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
```

注意

dns.admin 角色为管理 Google Cloud DNS 资源的服务帐户提供管理员特权。要确保 cert-manager 使用具有最小权限的服务帐户运行，您可以创建一个具有以下权限的自定义角色：

- **dns.resourceRecordSets.***
- **dns.changes.***
- **dns.managedZones.list**

2. 运行以下命令来创建 **CredentialsRequest** 资源：

```
$ oc create -f sample-credential-request.yaml
```

3. 运行以下命令，为 Red Hat OpenShift 更新 cert-manager Operator 的订阅对象：

```
$ 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"}]}}}'
```

验证

1. 运行以下命令，获取重新部署的 cert-manager 控制器 pod 的名称：

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

输**出示例**

```
NAME                                READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt        1/1     Running   0          15m39s
```

2. 运行以下命令，验证 cert-manager 控制器 pod 是否使用挂载在 **mountPath** 中指定的路径下的 GCP 凭证卷更新：

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

输**出示例**

```
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
```

9.6.4. 使用 GCP Workload Identity 进行身份验证

先决条件

- 已提取并准备好 **ccoctl** 二进制文件。
- 安装了 Red Hat OpenShift 1.11.1 或更高版本的 cert-manager Operator。
- 您已在手动模式中使用 Cloud Credential Operator 配置了一个带有 GCP Workload Identity 的 OpenShift Container Platform 集群。

流程

流程

1. 运行以下命令，创建一个目录来存储 **CredentialsRequest** 资源 YAML 文件：

```
$ mkdir credentials-request
```

2. 在 **credentials-request** 目录中，创建一个包含以下 **CredentialsRequest** 清单的 YAML 文件：

```
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
```

注意

dns.admin 角色为管理 Google Cloud DNS 资源的服务帐户提供管理员特权。要确保 cert-manager 使用具有最小权限的服务帐户运行，您可以创建一个具有以下权限的自定义角色：

- **dns.resourceRecordSets.***
- **dns.changes.***
- **dns.managedZones.list**

3. 运行以下命令，使用 **ccoctl** 工具处理 **CredentialsRequest** 对象：

```
$ 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>
```

示例命令

```
$ 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. 运行以下命令应用在集群 manifests 目录中生成的 secret：

```
$ ls <path_to_output_dir>/manifests/*-credentials.yaml | xargs -I{} oc apply -f {}
```

5. 运行以下命令，为 Red Hat OpenShift 更新 cert-manager Operator 的订阅对象：

```
$ 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"}]}}}'
```

验证

1. 运行以下命令，获取重新部署的 cert-manager 控制器 pod 的名称：

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

输**出示例**

```
NAME                       READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          15m39s
```

2. 运行以下命令，验证 cert-manager 控制器 pod 是否已使用挂载到 **mountPath** 中指定的路径下的 GCP 工作负载身份凭证卷更新：

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

输**出示例**

```
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:
```

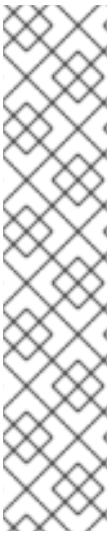
```
    - key: service_account.json
      path: application_default_credentials.json
  secretName: gcp-credentials
```

其他资源

- 配置 Cloud Credential Operator 工具
- 带组件的短期凭证的手动模式
- Cloud Credential Operator 的默认行为

9.7. 配置 ACME 签发者

Red Hat OpenShift 的 cert-manager Operator 支持使用自动证书管理环境 (ACME) CA 服务器（如 *Let's Encrypt*）来发布证书。通过在 **Issuer** API 对象中指定 secret 详情来配置显式凭证。方便的凭证从环境、元数据服务或本地文件中提取，这些文件没有在 **Issuer** API 对象中明确配置。



注意

Issuer 对象是命名空间范围。它只能从同一命名空间中发布证书。您还可以使用 **ClusterIssuer** 对象在集群中的所有命名空间中发布证书。

定义 ClusterIssuer 对象的 YAML 文件示例

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: acme-cluster-issuer
spec:
  acme:
    ...
```

注意

默认情况下，您可以将 **ClusterIssuer** 对象与 ambient 凭证一起使用。要将 **Issuer** 对象与 ambient 凭证搭配使用，您必须为 cert-manager 控制器启用 **--issuer-ambient-credentials** 设置。

9.7.1. 关于 ACME 签发者

Red Hat OpenShift 的 cert-manager Operator 的 ACME 签发者类型代表自动的证书颁发机构 (ACME) 证书颁发机构 (CA) 服务器。ACME CA 服务器依赖于一个 *质询（challenge）* 来验证客户端是否拥有请求证书的域名。如果质询成功，Red Hat OpenShift 的 cert-manager Operator 可以发布证书。如果失败，Red Hat OpenShift 的 cert-manager Operator 不会发布证书。

注意

Let's Encrypt 和 internet ACME 服务器不支持私有 DNS 区域。

9.7.1.1. 支持的 ACME 质询类型

Red Hat OpenShift 的 cert-manager Operator 支持 ACME 签发者的以下质询类型：

HTTP-01

使用 HTTP-01 质询类型，您可以在域中的 HTTP URL 端点上提供一个计算的密钥。如果 ACME CA 服务器可以从 URL 获取密钥，它可以验证您作为域的所有者。
如需更多信息，请参阅上游 cert-manager 文档中的 HTTP01。

DNS-01

使用 DNS-01 质询类型，您可以在 DNS TXT 记录中提供一个计算的密钥。如果 ACME CA 服务器可以通过 DNS 查找获取密钥，它可以验证您作为域的所有者。
如需更多信息，请参阅上游 cert-manager 文档中的 DNS01。

9.7.1.2. 支持的 DNS-01 供应商

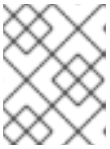
Red Hat OpenShift 的 cert-manager Operator 支持以下 ACME 签发者的 DNS-01 供应商：

- Amazon Route 53
- Azure DNS

注意

Red Hat OpenShift 的 cert-manager Operator 不支持使用 Microsoft Entra ID pod 身份为 pod 分配受管身份。

- Google Cloud DNS
- Webhook
  红帽测试并支持 DNS 供应商，使用带有 OpenShift Container Platform 上的 cert-manager 的外部 Webhook。OpenShift Container Platform 测试并支持以下 DNS 供应商：
  - cert-manager-webhook-ibmcis



注意

使用未列出的 DNS 供应商可能会与 OpenShift Container Platform 一起正常工作，但供应商没有被红帽测试，因此不被红帽支持。

9.7.2. 配置 ACME 签发者以解决 HTTP-01 质询

您可以使用 cert-manager Operator for Red Hat OpenShift 设置 ACME 签发者来解决 HTTP-01 质询。此流程使用 *Let's Encrypt* 作为 ACME CA 服务器。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您有一个要公开的服务。在此过程中，该服务名为 **sample-workload**。

流程

1. 创建 ACME 集群签发者。
   a. 创建定义 **ClusterIssuer** 对象的 YAML 文件：

acme-cluster-issuer.yaml 文件示例

```
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. 为集群签发者提供名称。
2. 将 **<secret_private_key>** 替换为要存储 ACME 帐户私钥的 secret 名称。
3. 指定用于访问 ACME 服务器的 **directory** 端点 URL。这个示例使用 *Let's Encrypt* staging 环境。
4. 指定 Ingress 类。

b. 可选：如果您在没有指定 **ingressClassName** 的情况下创建对象，请使用以下命令修补现有的入口：

```
$ oc patch ingress/<ingress-name> --type=merge --patch '{"spec":
{"ingressClassName":"openshift-default"}}' -n <namespace>
```

c. 运行以下命令来创建 **ClusterIssuer** 对象：

```
$ oc create -f acme-cluster-issuer.yaml
```

2. 创建一个 Ingress 来公开用户工作负载的服务。

a. 创建定义 **Namespace** 对象的 YAML 文件：

namespace.yaml 文件示例

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-ingress-namespace 1
```

1. 指定 Ingress 的命名空间。

b. 运行以下命令来创建 **Namespace** 对象：

```
$ oc create -f namespace.yaml
```

c. 创建定义 **Ingress** 对象的 YAML 文件：

ingress.yaml 文件示例

```
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. 指定 Ingress 的名称。
2. 指定您为 Ingress 创建的命名空间。
3. 指定您创建的集群签发者。
4. 指定 Ingress 类。
5. 指定 Ingress 类。
6. 将 **<hostname>** 替换为要与证书关联的 Subject Alternative Name。此名称用于在证书中添加 DNS 名称。
7. 指定要存储创建证书的 secret。
8. 将 **<hostname>** 替换为主机名。您可以使用 **<host_name>.<cluster_ingress_domain>** 语法来利用 ***.<cluster_ingress_domain>** 通配符 DNS 记录并为集群提供证书。例如，您可以使用 **apps.<cluster_base_domain>**。否则，您必须确保为所选主机名存在 DNS 记录。
9. 指定要公开的服务名称。本例使用名为 **sample-workload** 的服务。

d. 运行以下命令来创建 **Ingress** 对象：

```
$ oc create -f ingress.yaml
```

9.7.3. 使用 AWS Route53 的显式凭证配置 ACME 签发者

您可以使用 cert-manager Operator for Red Hat OpenShift 设置自动证书管理环境(ACME)签发者，通过使用 AWS 上的显式凭证来解决 DNS-01 质询。此流程使用 *Let's Encrypt* 作为 ACME 证书颁发机构(CA)服务器，并演示了如何解决 Amazon Route 53 的 DNS-01 质询。

先决条件

- 您必须提供显式 **accessKeyID** 和 **secretAccessKey** 凭证。如需更多信息，请参阅上游 cert-manager 文档中的 Route53。

注意

您可以在 AWS 上运行的 OpenShift Container Platform 集群中使用带有显式凭证的 Amazon Route 53。

流程

1. 可选：覆盖 DNS-01 自我检查的名称服务器设置。
   只有在目标公共托管区与集群的默认私有托管区重叠时才需要这一步。

   a. 运行以下命令来编辑 **CertManager** 资源：

   ```
   $ oc edit certmanager cluster
   ```

   b. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

   ```
   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) 添加 **spec.controllerConfig** 部分。

   (2) 指定只使用递归名称服务器，而不是检查与该域关联的权威名称服务器。

   (3) 提供以逗号分隔的 **<host>:<port>** 名称服务器列表来查询 DNS-01 自我检查。您必须使用 **1.1.1.1:53** 值来避免公共和私有区重叠。

   c. 保存文件以使改变生效。

2. 可选：为签发者创建一个命名空间：

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

3. 运行以下命令，创建一个 secret 来存储 AWS 凭证：

```
$ oc create secret generic aws-secret --from-literal=awsSecretAccessKey=
<aws_secret_access_key> \ 1
   -n my-issuer-namespace
```

1 将 **<aws_secret_access_key>** 替换为您的 AWS secret 访问密钥。

4. 创建签发者：

   a. 创建定义 **Issuer** 对象的 YAML 文件：

   issuer.yaml 文件示例

```
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 为签发者提供名称。

2 指定您为签发者创建的命名空间。

3 指定用于访问 ACME 服务器的 **directory** 端点 URL。这个示例使用 *Let's Encrypt* staging 环境。

4 将 **<email_address>** 替换为您的电子邮件地址。

5 将 **<secret_private_key>** 替换为要存储 ACME 帐户私钥的 secret 名称。

6 将 **<aws_key_id>** 替换为您的 AWS 密钥 ID。

7 将 **<hosted_zone_id>** 替换为您的托管区 ID。

8 将 **<region_name>** 替换为 AWS 区域名称。例如，**us-east-1**。

9 指定您创建的 secret 的名称。

10 指定您创建的 secret 中的密钥，用于存储 AWS secret 访问密钥。

b. 运行以下命令来创建 **Issuer** 对象：

```
$ oc create -f issuer.yaml
```

9.7.4. 在 AWS 上使用 ambient 凭证配置 ACME 签发者

您可以使用 Red Hat OpenShift 的 cert-manager Operator 设置 ACME 签发者，通过使用 AWS 上的 ambient 凭证来解决 DNS-01 质询。此流程使用 *Let's Encrypt* 作为 ACME CA 服务器，并演示了如何解决 Amazon Route 53 的 DNS-01 质询。

先决条件

- 如果您的集群被配置为使用 AWS 安全令牌服务 (STS)，则遵循为 *Red Hat OpenShift 为 AWS Security Token Service 集群配置 cert-manager Operator 的云凭证*一节中的内容。
- 如果您的集群没有使用 AWS STS，按照 *为 Red Hat OpenShift on AWS 的 cert-manager Operator 配置云凭证*中的内容进行操作。

流程

1. 可选：覆盖 DNS-01 自我检查的名称服务器设置。
   只有在目标公共托管区与集群的默认私有托管区重叠时才需要这一步。

   a. 运行以下命令来编辑 **CertManager** 资源：

   ```
   $ oc edit certmanager cluster
   ```

   b. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

   ```
   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 添加 **spec.controllerConfig** 部分。

   2 指定只使用递归名称服务器，而不是检查与该域关联的权威名称服务器。

   3 提供以逗号分隔的 **<host>:<port>** 名称服务器列表来查询 DNS-01 自我检查。您必须使用 **1.1.1.1:53** 值来避免公共和私有区重叠。

c. 保存文件以使改变生效。

2. 可选：为签发者创建一个命名空间：

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

3. 修改 **CertManager** 资源以添加 **--issuer-ambient-credentials** 参数：

```
$ oc patch certmanager/cluster \
  --type=merge \
  -p='{"spec":{"controllerConfig":{"overrideArgs":["--issuer-ambient-credentials"]}}}'
```

4. 创建签发者：

a. 创建定义 **Issuer** 对象的 YAML 文件：

issuer.yaml 文件示例

```
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. 为签发者提供名称。
2. 指定您为签发者创建的命名空间。
3. 指定用于访问 ACME 服务器的 **directory** 端点 URL。这个示例使用 *Let's Encrypt* staging 环境。
4. 将 **<email_address>** 替换为您的电子邮件地址。
5. 将 **<secret_private_key>** 替换为要存储 ACME 帐户私钥的 secret 名称。
6. 将 **<hosted_zone_id>** 替换为您的托管区 ID。

b. 运行以下命令来创建 **Issuer** 对象：

```
$ oc create -f issuer.yaml
```

9.7.5. 使用 GCP Cloud DNS 的显式凭证配置 ACME 签发者

您可以使用 Red Hat OpenShift 的 cert-manager Operator 设置 ACME 签发者，以便在 GCP 上使用显式凭证来解决 DNS-01 质询。此流程使用 *Let's Encrypt* 作为 ACME CA 服务器，并演示了如何解决 Google CloudDNS 的 DNS-01 质询。

先决条件

- 您已为 Google CloudDNS 设置具有所需角色的 Google Cloud 服务帐户。如需更多信息，请参阅上游 cert-manager 文档中的 Google CloudDNS。

注意

您可以在没有在 GCP 上运行的 OpenShift Container Platform 集群中使用带有显式凭证的 Google CloudDNS。

流程

1. 可选：覆盖 DNS-01 自我检查的名称服务器设置。
只有在目标公共托管区与集群的默认私有托管区重叠时才需要这一步。

   a. 运行以下命令来编辑 **CertManager** 资源：

   ```
   $ oc edit certmanager cluster
   ```

   b. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

   ```
   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) 添加 **spec.controllerConfig** 部分。

   (2) 指定只使用递归名称服务器，而不是检查与该域关联的权威名称服务器。

   (3) 提供以逗号分隔的 **<host>:<port>** 名称服务器列表来查询 DNS-01 自我检查。您必须使用 **1.1.1.1:53** 值来避免公共和私有区重叠。

   c. 保存文件以使改变生效。

2. 可选：为签发者创建一个命名空间：

   ```
   $ oc new-project my-issuer-namespace
   ```

3. 运行以下命令，创建一个 secret 来存储 GCP 凭证：

```
$ 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. 创建签发者：

   a. 创建定义 **Issuer** 对象的 YAML 文件：

   issuer.yaml 文件示例

```
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. 为签发者提供名称。
   2. 将 **<issuer_namespace>** 替换为您的签发者命名空间。
   3. 将 **<secret_private_key>** 替换为要存储 ACME 帐户私钥的 secret 名称。
   4. 指定用于访问 ACME 服务器的 **directory** 端点 URL。这个示例使用 *Let's Encrypt* staging 环境。
   5. 将 **<project_id>** 替换为包含 Cloud DNS 区的 GCP 项目的名称。
   6. 指定您创建的 secret 的名称。
   7. 指定您创建的 secret 中存储 GCP secret 访问密钥的密钥。

   b. 运行以下命令来创建 **Issuer** 对象：

```
$ oc create -f issuer.yaml
```

9.7.6. 使用 GCP 上的 ambient 凭证配置 ACME 签发者

您可以使用 Red Hat OpenShift 的 cert-manager Operator 设置 ACME 签发者，通过使用 GCP 上的 ambient 凭证来解决 DNS-01 质询。此流程使用 *Let's Encrypt* 作为 ACME CA 服务器，并演示了如何解决 Google CloudDNS 的 DNS-01 质询。

先决条件

- 如果您的集群被配置为使用 GCP Workload Identity，则遵循 *为带有 GCP Workload Identity 的 Red Hat OpenShift 为 cert-manager Operator 配置云凭证* 中的内容。
- 如果您的集群没有使用 GCP Workload Identity，按照 *为 Red Hat OpenShift on GCP 上的 cert-manager Operator 配置云凭证* 中的内容进行操作。

流程

1. 可选：覆盖 DNS-01 自我检查的名称服务器设置。
   只有在目标公共托管区与集群的默认私有托管区重叠时才需要这一步。

   a. 运行以下命令来编辑 **CertManager** 资源：

   ```
   $ oc edit certmanager cluster
   ```

   b. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

   ```
   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) 添加 **spec.controllerConfig** 部分。

   (2) 指定只使用递归名称服务器，而不是检查与该域关联的权威名称服务器。

   (3) 提供以逗号分隔的 **<host>:<port>** 名称服务器列表来查询 DNS-01 自我检查。您必须使用 **1.1.1.1:53** 值来避免公共和私有区重叠。

   c. 保存文件以使改变生效。

2. 可选：为签发者创建一个命名空间：

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

3. 修改 **CertManager** 资源以添加 **--issuer-ambient-credentials** 参数：

   ```
   $ oc patch certmanager/cluster \
     --type=merge \
     -p='{"spec":{"controllerConfig":{"overrideArgs":["--issuer-ambient-credentials"]}}}'
   ```

4. 创建签发者：

   a. 创建定义 **Issuer** 对象的 YAML 文件：

issuer.yaml 文件示例

```
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. 为签发者提供名称。
2. 将 **<secret_private_key>** 替换为要存储 ACME 帐户私钥的 secret 名称。
3. 指定用于访问 ACME 服务器的 **directory** 端点 URL。这个示例使用 *Let's Encrypt* staging 环境。
4. 将 **<gcp_project_id>** 替换为包含 Cloud DNS 区的 GCP 项目的名称。

b. 运行以下命令来创建 **Issuer** 对象：

```
$ oc create -f issuer.yaml
```

9.7.7. 使用 Microsoft Azure DNS 的显式凭证配置 ACME 签发者

您可以使用 Red Hat OpenShift 的 cert-manager Operator 设置 ACME 签发者，以便在 Microsoft Azure 上使用显式凭证来解决 DNS-01 质询。此流程使用 *Let's Encrypt* 作为 ACME CA 服务器，并演示了如何解决 Azure DNS 的 DNS-01 质询。

先决条件

- 您已为 Azure DNS 设置具有所需角色的服务主体。如需更多信息，请参阅上游 cert-manager 文档中的 Azure DNS。

注意

对于不在 Microsoft Azure 上运行的 OpenShift Container Platform 集群，您可以按照以下步骤操作。

流程

1. 可选：覆盖 DNS-01 自我检查的名称服务器设置。
   只有在目标公共托管区与集群的默认私有托管区重叠时才需要这一步。

   a. 运行以下命令来编辑 **CertManager** 资源：

```
$ oc edit certmanager cluster
```

b. 使用以下覆盖参数添加 **spec.controllerConfig** 部分：

```
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) 添加 **spec.controllerConfig** 部分。

(2) 指定只使用递归名称服务器，而不是检查与该域关联的权威名称服务器。

(3) 提供以逗号分隔的 **<host>:<port>** 名称服务器列表来查询 DNS-01 自我检查。您必须使用 **1.1.1.1:53** 值来避免公共和私有区重叠。

c. 保存文件以使改变生效。

2. 可选：为签发者创建一个命名空间：

```
$ oc new-project my-issuer-namespace
```

3. 运行以下命令，创建一个 secret 来存储 Azure 凭证：

```
$ 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) 将 **<secret_name>** 替换为您的 secret 名称。

(2) 将 **<azure_secret_access_key_name>** 替换为您的 Azure secret 访问密钥名称。

(3) 将 **<azure_secret_access_key_value>** 替换为您的 Azure secret 键。

4. 创建签发者：

a. 创建定义 **Issuer** 对象的 YAML 文件：

issuer.yaml 文件示例

```
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. 为签发者提供名称。
2. 将 **<issuer_namespace>** 替换为您的签发者命名空间。
3. 将 **<secret_private_key>** 替换为要存储 ACME 帐户私钥的 secret 名称。
4. 指定用于访问 ACME 服务器的 **directory** 端点 URL。这个示例使用 *Let's Encrypt* staging 环境。
5. 将 **<azure_client_id>** 替换为您的 Azure 客户端 ID。
6. 将 **<secret_name>** 替换为客户端 secret 的名称。
7. 将 **<azure_secret_access_key_name>** 替换为客户端 secret 密钥名称。
8. 将 **<azure_subscription_id>** 替换为您的 Azure 订阅 ID。
9. 将 **<azure_tenant_id>** 替换为您的 Azure 租户 ID。
10. 将 **<azure_dns_zone_resource_group>** 替换为 Azure DNS 区资源组的名称。
11. 将 **<azure_dns_zone>** 替换为 Azure DNS 区的名称。

b. 运行以下命令来创建 **Issuer** 对象：

```
$ oc create -f issuer.yaml
```

9.7.8. 其他资源

- 为 AWS 安全令牌服务集群的 Red Hat OpenShift 配置 cert-manager Operator 的云凭证
- 为 AWS 上的 Red Hat OpenShift 配置 cert-manager Operator 的云凭证
- 使用 GCP Workload Identity 为 Red Hat OpenShift 配置云凭证

- 为 GCP 上的 Red Hat OpenShift 配置 cert-manager Operator 的云凭证

9.8. 使用签发者配置证书

通过将 cert-manager Operator 用于 Red Hat OpenShift，您可以为集群中的工作负载管理证书、处理续订和颁发等任务，以及与集群外部交互的组件。

9.8.1. 为用户工作负载创建证书

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 您已为 Red Hat OpenShift 安装了 cert-manager Operator。

流程

1. 创建签发者。如需更多信息，请参阅"添加资源"部分中的"配置签发者"。
2. 创建证书：
   a. 创建一个 YAML 文件，如 **certificate.yaml**，用于定义 **Certificate** 对象：

   certificate.yaml 文件示例

```
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 为证书提供名称。
   2 指定签发者的命名空间。
   3 指定通用名称 (CN)。
   4 指定要创建包含证书的 secret 名称。
   5 指定域名。
   6 指定签发者的名称。

   b. 运行以下命令来创建 **Certificate** 对象：

```
$ oc create -f certificate.yaml
```

验证

- 运行以下命令验证证书是否已创建并就绪：

  ```
  $ oc get certificate -w -n <issuer_namespace>
  ```

  证书处于 **Ready** 状态后，集群中的工作负载就可以使用生成的证书 secret 开始。

9.8.2. 为 API 服务器创建证书

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 您已为 Red Hat OpenShift 1.13.0 或更高版本安装了 cert-manager Operator。

流程

1. 创建签发者。如需更多信息，请参阅"添加资源"部分中的"配置签发者"。
2. 创建证书：
   a. 创建一个 YAML 文件，如 **certificate.yaml**，用于定义 **Certificate** 对象：

   certificate.yaml 文件示例

   ```
   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) 为证书提供名称。

   (2) 指定通用名称 (CN)。

   (3) 指定要创建包含证书的 secret 名称。

   (4) 指定 API 服务器的 DNS 名称。

   (5) 指定签发者的名称。

b. 运行以下命令来创建 **Certificate** 对象：

```
$ oc create -f certificate.yaml
```

3. 添加名为 certificate 的 API 服务器。如需更多信息，请参阅"添加资源"部分中的"添加名为 certificate 的 API 服务器"部分。

注意

要确保证书已更新，请在创建证书后再次运行 **oc login** 命令。

验证

- 运行以下命令验证证书是否已创建并就绪：

```
$ oc get certificate -w -n openshift-config
```

证书处于 **Ready** 状态后，集群中的 API 服务器就可以使用生成的证书 secret 开始。

9.8.3. 为 Ingress Controller 创建证书

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 您已为 Red Hat OpenShift 1.13.0 或更高版本安装了 cert-manager Operator。

流程

1. 创建签发者。如需更多信息，请参阅"添加资源"部分中的"配置签发者"。
2. 创建证书：

a. 创建一个 YAML 文件，如 **certificate.yaml**，用于定义 **Certificate** 对象：

certificate.yaml 文件示例

```
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. 为证书提供名称。
2. 指定通用名称 (CN)。
3. 指定要创建包含证书的 secret 名称。
4. 5. 指定入口的 DNS 名称。
6. 指定签发者的名称。

b. 运行以下命令来创建 **Certificate** 对象：

```
$ oc create -f certificate.yaml
```

3. 替换默认入口证书。如需更多信息，请参阅"添加资源"部分中的"替换默认入口证书"部分。

验证

- 运行以下命令验证证书是否已创建并就绪：

```
$ oc get certificate -w -n openshift-ingress
```

证书处于 **Ready** 状态后，集群中的 Ingress Controller 就可以使用生成的证书 secret 开始。

9.8.4. 其他资源

- 配置签发者
  - 支持的签发者类型
  - 配置 ACME 签发者
- 添加名为 certificate 的 API 服务器
- 替换默认入口证书

9.9. 监控 RED HAT OPENSHIFT 的 CERT-MANAGER OPERATOR

您可以使用 Prometheus Operator 提供的格式为 Red Hat OpenShift 公开 cert-manager Operator 的控制器指标。

9.9.1. 使用 Red Hat OpenShift 的 cert-manager Operator 的服务监控器启用监控

您可以使用服务监控器执行自定义指标提取，为 Red Hat OpenShift 为 cert-manager Operator 启用监控和指标集合。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 安装了 Red Hat OpenShift 的 cert-manager Operator。

流程

1. 运行以下命令添加标签以启用集群监控：

```
$ oc label namespace cert-manager openshift.io/cluster-monitoring=true
```

2. 创建服务监控器：

   a. 创建定义 **Role**、**RoleBinding** 和 **ServiceMonitor** 对象的 YAML 文件：

   monitoring.yaml 文件示例

```
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. 运行以下命令来创建 **Role**、**RoleBinding** 和 **ServiceMonitor** 对象：

```
$ oc create -f monitoring.yaml
```

其他资源

- 为用户定义的项目设置指标集合

9.9.2. 为 Red Hat OpenShift 查询 cert-manager Operator 的指标

为 Red Hat OpenShift 启用 cert-manager Operator 监控后，您可以使用 OpenShift Container Platform Web 控制台查询其指标。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您已为 Red Hat OpenShift 安装了 cert-manager Operator。
- 您已为 Red Hat OpenShift 为 cert-manager Operator 启用监控和指标集合。

流程

1. 在 OpenShift Container Platform web 控制台中进入到 **Observe → Metrics**。
2. 使用以下格式之一添加查询：
   - 指定端点：

```
{instance="<endpoint>"} 1
```

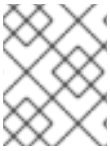
1 将 **<endpoint>** 替换为 **cert-manager** 服务的端点值。您可以运行以下命令来找到端点值：**oc describe service cert-manager -n cert-manager**。

   - 指定 **tcp-prometheus-servicemonitor** 端口：

```
{endpoint="tcp-prometheus-servicemonitor"}
```

9.10. 为 CERT-MANAGER 和 CERT-MANAGER OPERATOR 配置日志级别

要排除 cert-manager 组件和 Red Hat OpenShift 的 cert-manager Operator 的问题，您可以配置日志级别详细程度。



注意

要为不同的 cert-manager 组件使用不同的日志级别，请参阅*自定义 cert-manager Operator API 字段*。

9.10.1. 为 cert-manager 设置日志级别

您可以为 cert-manager 设置日志级别，以确定日志消息的详细程度。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 您已为 Red Hat OpenShift 1.11.1 或更高版本安装了 cert-manager Operator。

流程

1. 运行以下命令来编辑 **CertManager** 资源：

```
$ oc edit certmanager.operator cluster
```

2. 通过编辑 **spec.logLevel** 部分来设置日志级别值：

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
...
spec:
  logLevel: <log_level> 1
```

1 **CertManager** 资源的有效日志级别值为 **Normal**、**Debug**、**Trace** 和 **TraceAll**。要审计日志并在没有问题时执行常见操作，请将 **logLevel** 设置为 **Normal**。要通过查看详细日志来排除小问题，请将 **logLevel** 设置为 **Debug**。要通过查看更详细的日志来排除主要问题，您可以将 **logLevel** 设置为 **Trace**。要排除严重问题，请将 **logLevel** 设置为 **TraceAll**。默认 **logLevel** 为 **Normal**。

注意

TraceAll 生成大量日志。将 **logLevel** 设置为 **TraceAll** 后，您可能会遇到性能问题。

3. 保存更改并退出文本编辑器以应用您的更改。
   应用更改后，cert-manager 组件控制器、CA 注入器和 Webhook 的详细程度被更新。

9.10.2. 为 Red Hat OpenShift 为 cert-manager Operator 设置日志级别

您可以为 Red Hat OpenShift 为 cert-manager Operator 设置日志级别，以确定 Operator 日志消息的详细程度。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。

- 您已为 Red Hat OpenShift 1.11.1 或更高版本安装了 cert-manager Operator。

流程

- 运行以下命令，为 Red Hat OpenShift 更新 cert-manager Operator 的订阅对象，以提供 Operator 日志的详细程度：

```
$ 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 使用所需的日志级别号替换 **v**。**v** 的有效值的范围为 **1`到 `10** 范围。默认值为 **2**。

验证

1. 重新部署 cert-manager Operator pod。运行以下命令，验证 Red Hat OpenShift 的 cert-manager Operator 的日志级别是否已更新：

```
$ oc set env deploy/cert-manager-operator-controller-manager -n cert-manager-operator --list | grep -e OPERATOR_LOG_LEVEL -e container
```

输**出示例**

```
# 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. 运行 **oc logs** 命令验证 Red Hat OpenShift 的 cert-manager Operator 的日志级别是否已更新：

```
$ oc logs deploy/cert-manager-operator-controller-manager -n cert-manager-operator
```

9.10.3. 其他资源

- 自定义 cert-manager Operator API 字段

9.11. 为 RED HAT OPENSHIFT 卸载 CERT-MANAGER OPERATOR

您可以通过卸载 Operator 并删除其相关资源，从 OpenShift Container Platform 中删除 Red Hat OpenShift 的 cert-manager Operator。

9.11.1. 为 Red Hat OpenShift 卸载 cert-manager Operator

您可以使用 Web 控制台为 Red Hat OpenShift 卸载 cert-manager Operator。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 访问 OpenShift Container Platform web 控制台。

- 安装了 Red Hat OpenShift 的 cert-manager Operator。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 为 Red Hat OpenShift Operator 卸载 cert-manager Operator。
   a. 导航到 **Operators → Installed Operators**。
   b. 点 **cert-manager Operator for Red Hat OpenShift**条目旁边的 Options 菜单  并选择 **Uninstall Operator**。
   c. 在确认对话框中，点 **Uninstall**。

9.11.2. 为 Red Hat OpenShift 资源删除 cert-manager Operator

为 Red Hat OpenShift 卸载 cert-manager Operator 后，您可以选择从集群中移除其相关资源。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 访问 OpenShift Container Platform web 控制台。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 删除 cert-manager 组件的部署，如 **cert-manager**、**cainjector** 和 **Webhook**，存在于 **cert-manager** 命名空间中。
   a. 点 **Project** 下拉菜单查看所有可用项目的列表，然后选择 **cert-manager** 项目。
   b. 进入到 **Workloads → Deployments**。
   c. 选择您要删除的部署。
   d. 点 **Actions** 下拉菜单，然后选择 **Delete Deployment** 以查看确认对话框。
   e. 点 **Delete** 以删除部署。
   f. 或者，使用命令行界面 (CLI) 删除 **cert-manager** 组件（如 **cert-manager**、**cainjector** 和 **webhook** ）的部署。

```
$ oc delete deployment -n cert-manager -l app.kubernetes.io/instance=cert-manager
```

3. 可选：删除由 Red Hat OpenShift 的 cert-manager Operator 安装的自定义资源定义 (CRD)：
   a. 进入到 **Administration → CustomResourceDefinitions**。
   b. 在 **Name** 字段中输入 **certmanager** 来过滤 CRD。

c. 点击以下 CRD  旁边的 Options 菜单，然后选择 **Delete Custom Resource Definition**

- 证书
- **CertificateRequest**
- **CertManager** (**operator.openshift.io**)
- **Challenge**
- **ClusterIssuer**
- **Issuer**
- **顺序**

4. 可选：删除 **cert-manager-operator** 命名空间。

a. 导航至 **Administration** → **Namespaces**。

b. 点 **cert-manager-operator** 旁边的 Options 菜单  并选择 **Delete Namespace**。

c. 在确认对话框中，在字段中输入 **cert-manager-operator** 并点 **Delete**。

第 10 章 查看审计日志

OpenShift Container Platform auditing（审计）提供一组安全相关的按时间排序的记录，记录各个用户、管理员或其他系统组件影响系统的一系列活动。

10.1. 关于 API 审计日志

审计在 API 服务器级别运作，记录所有传入到服务器的请求。每个审计日志包含以下信息：

表 10.1. 审计日志字段

字段	描述
level	生成事件的审计级别。
auditID	为每个请求生成的唯一审计 ID。
stage	生成此事件实例时请求处理的阶段。
requestURI	客户端向服务器发送的请求 URI。
verb	与请求相关联的 Kubernetes 操作动词。对于非资源请求，这是小写 HTTP 方法。
user	经过身份验证的用户信息。
impersonatedUser	可选。如果请求模拟了另一个用户，则为被模拟的用户信息。
sourceIPs	可选。源 IP，请求发起的源和任何中间代理。
userAgent	可选。客户端报告的用户代理字符串。请注意，用户代理由客户端提供，且必须不可信任。
objectRef	可选。这个请求的目标对象引用。这不适用于 **List** 类型请求，或者非资源请求。
responseStatus	可选。响应的状态，即使 **ResponseObject** 不是 **Status** 类型也会生成。对于成功的响应，这只会包括代码。对于非状态类型错误响应，这将自动生成出错信息。
requestObject	可选。请求中的 API 对象，采用 JSON 格式。在进行 version conversion、defaulting、admission 或 merging 之前，在请求中的 **RequestObject** 记录（可能会被转换为 JSON 格式）。这是一个外部版本化的对象类型，可能自身并不是一个有效的对象。对于非资源请求，这会被忽略，且只在 Request 级别或更高级别中被记录。
responseObject	可选。响应中返回的 API 对象，使用 JSON 格式。在转换为外部类型后，**ResponseObject** 被记录，并被序列化为 JSON 数据。在非资源请求中会省略它，且仅在 Response 级别中记录。

字段	描述
requestReceivedTimestamp	请求到达 API 服务器的时间。
stageTimestamp	请求到达当前审计阶段的时间。
annotations	可选。一个无结构的键值映射，它存储在一个审计事件中，可以通过在请求服务链中调用的插件来设置它，包括认证、授权和准入插件。请注意，这些注解用于审计事件，且与所提交对象的 **metadata.annotations** 没有关联。标识信息组件的键应该是唯一的以避免名称冲突，例如 **podsecuritypolicy.admission.k8s.io/policy**。值应该较短。注解包含在 Metadata 级别中。

Kubernetes API 服务器的输出示例：

{"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. 查看审计日志

您可以查看每个 control plane 节点的 OpenShift API 服务器、Kubernetes API 服务器、OpenShift OAuth API 服务器和 OpenShift OAuth 服务器的日志。

流程

查看审计日志：

- 查看 OpenShift API 服务器审计日志：
  a. 列出每个 control plane 节点可用的 OpenShift API 服务器审计日志：

```
$ oc adm node-logs --role=master --path=openshift-apiserver/
```

输出示例

```
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. 通过提供节点名称和日志名称来查看特定的 OpenShift API 服务器审计日志：

```
$ oc adm node-logs <node_name> --path=openshift-apiserver/<log_name>
```

例如：

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=openshift-
apiserver/audit-2021-03-09T00-12-19.834.log
```

输**出示例**

```
{"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\""}}
```

- 查看 Kubernetes API 服务器审计日志：

a. 列出每个 control plane 节点可用的 Kubernetes API 服务器审计日志：

```
$ oc adm node-logs --role=master --path=kube-apiserver/
```

输**出示例**

```
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. 通过提供节点名称和日志名称来查看特定的 Kubernetes API 服务器审计日志：

```
$ oc adm node-logs <node_name> --path=kube-apiserver/<log_name>
```

例如：

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=kube-apiserver/audit-
2021-03-09T14-07-27.129.log
```

输**出示例**

```
{"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\""}}
```

- 查看 OpenShift OAuth API 服务器审计日志：

  a. 列出每个 control plane 节点可用的 OpenShift OAuth API 服务器审计日志：

  ```
  $ oc adm node-logs --role=master --path=oauth-apiserver/
  ```

  输**出示例**

  ```
  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. 通过提供节点名称和日志名称来查看特定的 OpenShift OAuth API 服务器审计日志：

  ```
  $ oc adm node-logs <node_name> --path=oauth-apiserver/<log_name>
  ```

  例如：

  ```
  $ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-apiserver/audit-2021-03-09T13-06-26.128.log
  ```

  输**出示例**

  ```
  {"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"},"response
Status":{"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\""}}
```

- 查看 OpenShift OAuth 服务器审计日志：

  a. 列出每个 control plane 节点可用的 OpenShift OAuth 服务器审计日志：

  ```
  $ oc adm node-logs --role=master --path=oauth-server/
  ```

  输出示例

  ```
  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. 通过提供节点名称和日志名称来查看特定的 OpenShift OAuth 服务器审计日志：

  ```
  $ oc adm node-logs <node_name> --path=oauth-server/<log_name>
  ```

  例如：

  ```
  $ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-server/audit-
  2022-05-11T18-57-32.395.log
  ```

  输出示例

  ```
  {"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":"kubea
  dmin","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
  ```

  authentication.openshift.io/decision 注解的可能值 **allow**、**deny** 或 **error**。

10.3. 过滤审计日志

您可以使用 **jq** 或另一个 JSON 解析工具来过滤 API 服务器审计日志。

注意

日志记录到 API 服务器审计日志的信息量是由设置的审计日志策略控制的。

以下流程提供了使用 **jq** 在 control plane 节点 **node-1.example.com** 上过滤审计日志的示例。有关使用 jq 的详情，请参考 **jq 手册**。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您已安装了 **jq**。

流程

- 根据用户过滤 OpenShift API 服务器审计日志：

```
$ oc adm node-logs node-1.example.com  \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.user.username == "myusername")'
```

- 根据用户代理过滤 OpenShift API 服务器审计日志：

```
$ 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")'
```

- 通过特定 API 版本过滤 Kubernetes API 服务器审计日志，仅输出用户代理：

```
$ oc adm node-logs node-1.example.com  \
  --path=kube-apiserver/audit.log \
  | jq 'select(.requestURI | startswith("/apis/apiextensions.k8s.io/v1beta1")) | .userAgent'
```

- 通过排除动词来过滤 OpenShift OAuth API 服务器审计日志：

```
$ oc adm node-logs node-1.example.com  \
  --path=oauth-apiserver/audit.log \
  | jq 'select(.verb != "get")'
```

- 根据标识用户名和失败并显示错误的事件过滤 OpenShift OAuth 服务器审计日志：

```
$ 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. 收集审计日志

您可以使用 must-gather 工具来收集审计日志以调试集群，您可以检查或发送到红帽支持。

流程

1. 使用 **-- /usr/bin/gather_audit_logs** 运行 **oc adm must-gather** 命令：

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

2. 从工作目录中刚刚创建的 **must-gather** 目录创建一个压缩文件。例如，在使用 Linux 操作系统的计算机上运行以下命令：

```
$ tar cvaf must-gather.tar.gz must-gather.local.472290403699006248 1
```

1 将 **must-gather-local.472290403699006248** 替换为实际目录名称。

3. 在红帽客户门户网站的**客户支持**页面中，将压缩文件附加到您的支持问题单中。

10.5. 其他资源

- must-gather 工具
- API 审计日志事件结构
- 配置审计日志策略
- 关于日志转发

第 11 章 配置审计日志策略

您可以通过选择要使用的审计日志策略配置集来控制记录到 API 服务器审计日志的信息量。

11.1. 关于审计日志策略配置集

审计日志配置集定义如何记录发送到 OpenShift API 服务器、Kubernetes API 服务器、OpenShift OAuth API 服务器和 OpenShift OAuth 服务器的请求。

OpenShift Container Platform 提供以下预定义的审计策略配置集：

配置集	描述
default	仅记录读取和写入请求的日志元数据；除了 OAuth 访问令牌请求外，不记录请求正文。这是默认策略。
WriteRequestBodies	除了记录所有请求的元数据外，还会记录对 API 服务器的写入请求（**create**, **update**, **patch**, **delete**, **deletecollection**）。这个配置集的资源开销比 **Default** 配置集大。[1]
AllRequestBodies	除了记录所有请求的元数据外，对 API 服务器的每个读写请求（**get**、**list**、**create**、**update**、**patch**）都进行日志记录。这个配置集的资源开销最大。[1]
None	没有记录请求，包括 OAuth 访问令牌请求和 OAuth 授权令牌请求。当设置此配置集时，会忽略自定义规则。 **警告** 除非完全了解在对问题进行故障排除时无法记录数据的风险，否则不要使用 **None** 配置集禁用审计日志记录。如果禁用审计日志记录且出现支持情况，您可能需要启用审计日志记录并重现问题，才能正确排除故障。

1. 敏感资源（如 **Secret**、**Route** 和 **OAuthClient** 对象）仅记录在元数据级别上。OpenShift OAuth 服务器事件仅记录在元数据级别上。

默认情况下，OpenShift Container Platform 使用 **Default** 审计日志配置集。您可以使用另一个审计策略配置集来记录请求的具体数据，但注意这会消耗更多资源（如 CPU、内存和 I/O）。

11.2. 配置审计日志策略

您可以配置审计日志策略,使其在记录 API 服务器的请求时使用。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 编辑 **APIServer** 资源：

```
$ oc edit apiserver cluster
```

2. 更新 **spec.audit.profile** 字段：

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: WriteRequestBodies  1
```

1 设置为 **Default**、**WriteRequestBodies**、**AllRequestBodies** 或 **None**。默认配置集为 **Default**。

警告

不建议使用 **None** 配置集禁用审计日志记录，除非您完全意识到在对问题进行故障排除时无法记录数据的风险。如果禁用审计日志记录且出现支持情况，您可能需要启用审计日志记录并重现问题，才能正确排除故障。

3. 保存文件以使改变生效。

验证

- 验证是否已推出 Kubernetes API 服务器 pod 的新修订版本。所有节点更新至新修订版本可能需要几分钟时间。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

查看 Kubernetes API 服务器的 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新的修订版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision**：

```
AllNodesAtLatestRevision
3 nodes are at revision 12  1
```

1 在本例中，最新的修订版本号为 **12**。

如果输出显示的信息类似于以下消息之一，则更新仍在进行中。等待几分钟后重试。

- **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. 使用自定义规则配置审计日志策略

您可以配置定义自定义规则的审计日志策略。您可以指定多个组，并定义要用于该组的配置集。

这些自定义规则优先于顶级配置集字段。自定义规则从上到下评估，第一个匹配项会被应用。

重要

如果顶级配置集字段设为 **None**，则忽略自定义规则。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 编辑 **APIServer** 资源：

```
$ oc edit apiserver cluster
```

2. 添加 **spec.audit.customRules** 字段：

```
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. 添加一个或多个组，并指定要用于该组的配置集。这些自定义规则优先于顶级配置集字段。自定义规则从上到下评估，第一个匹配项会被应用。
2. 设置为 **Default**、**WriteRequestBodies** 或 **AllRequestBodies**。如果您没有设置此顶级配置集字段，则默认为 **Default** 配置集。

警告

如果要使用自定义规则，请不要将顶级配置集字段设置为 **None**。如果顶级配置集字段设为 **None**，则忽略自定义规则。

3. 保存文件以使改变生效。

验证

- 验证是否已推出 Kubernetes API 服务器 pod 的新修订版本。所有节点更新至新修订版本可能需要几分钟时间。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

查看 Kubernetes API 服务器的 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新的修订版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision**：

```
AllNodesAtLatestRevision
3 nodes are at revision 12 (1)
```

(1) 在本例中，最新的修订版本号为 **12**。

如果输出显示的信息类似于以下消息之一，则更新仍在进行中。等待几分钟后重试。

  - **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. 禁用审计日志

您可以为 OpenShift Container Platform 禁用审计日志记录。当您禁用审计日志记录时，即使 OAuth 访问令牌请求和 OAuth 授权令牌请求也不会记录。

警告

不建议使用 **None** 配置集禁用审计日志记录，除非您完全意识到在对问题进行故障排除时无法记录数据的风险。如果禁用审计日志记录且出现支持情况，您可能需要启用审计日志记录并重现问题，才能正确排除故障。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 编辑 **APIServer** 资源：

```
$ oc edit apiserver cluster
```

2. 将 **spec.audit.profile** 字段设置为 **None** ：

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: None
```

注意

您还可以通过在 **spec.audit.customRules** 字段中指定自定义规则，只为特定组禁用审计日志记录。

3. 保存文件以使改变生效。

验证

- 验证是否已推出 Kubernetes API 服务器 pod 的新修订版本。所有节点更新至新修订版本可能需要几分钟时间。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

查看 Kubernetes API 服务器的 **NodeInstallerProgressing** 状态条件，以验证所有节点是否处于最新的修订版本。在更新成功后，输出会显示 **AllNodesAtLatestRevision** ：

```
AllNodesAtLatestRevision
3 nodes are at revision 12 (1)
```

(1) 在本例中，最新的修订版本号为 **12**。

如果输出显示的信息类似于以下消息之一，则更新仍在进行中。等待几分钟后重试。

  - **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**

第 12 章 配置 TLS 安全配置集

TLS 安全配置文件为服务器提供了一种方式，以规范在连接到服务器时可以使用什么加密方式。这样可确保 OpenShift Container Platform 组件使用加密库，它们不允许已知的不安全协议、密码或算法。

集群管理员可选择要用于以下每个组件的 TLS 安全配置集：

- Ingress Controller
- control plane
  这包括 Kubernetes API 服务器、Kubernetes 控制器管理器、Kubernetes 调度程序、OpenShift API 服务器、OpenShift OAuth API 服务器、OpenShift OAuth 服务器和 etcd。
- kubelet，当作为 Kubernetes API 服务器的 HTTP 服务器时

12.1. 了解 TLS 安全配置集

您可以使用 TLS（Transport Layer Security）安全配置集来定义各种 OpenShift Container Platform 组件需要哪些 TLS 密码。OpenShift Container Platform TLS 安全配置集基于 Mozilla 推荐的配置。

您可以为每个组件指定以下 TLS 安全配置集之一：

表 12.1. TLS 安全配置集

profile	描述
Old	此配置集用于旧的客户端或库。该配置集基于旧的向后兼容性建议配置。 **Old** 配置集要求最低 TLS 版本 1.0。 **注意** 对于 Ingress Controller，最小 TLS 版本从 1.0 转换为 1.1。
Intermediate	这个配置集是大多数客户端的建议配置。它是 Ingress Controller、kubelet 和 control plane 的默认 TLS 安全配置集。该配置集基于 Intermediate 兼容性推荐的配置。 **Intermediate** 配置集需要最小 TLS 版本 1.2。
Modern	此配置集主要用于不需要向后兼容的现代客户端。这个配置集基于 Modern 兼容性推荐的配置。 **Modern** 配置集需要最低 TLS 版本 1.3。

profile	描述
Custom	此配置集允许您定义要使用的 TLS 版本和密码。 **警告** 使用 **Custom** 配置集时要谨慎，因为无效的配置可能会导致问题。

注意

当使用预定义的配置集类型时，有效的配置集配置可能会在发行版本之间有所改变。例如，使用在版本 X.Y.Z 中部署的 Intermediate 配置集指定了一个规格，升级到版本 X.Y.Z+1 可能会导致应用新的配置集配置，从而导致推出部署。

12.2. 查看 TLS 安全配置集详情

您可以查看以下组件的预定义 TLS 安全配置集的最小 TLS 版本和密码：Ingress Controller、control plane 和 kubelet。

重要

最低 TLS 版本和配置集密码列表的有效配置可能因组件而异。

流程

- 查看特定 TLS 安全配置集的详情：

```
$ oc explain <component>.spec.tlsSecurityProfile.<profile> 1
```

1 对于 **<component>**，指定 **ingresscontroller**、**apiserver** 或 **kubeletconfig**。对于 **<profile>**，指定 **old**、**intermediate** 或 **custom**。

例如，检查 control plane 的 **intermediate** 配置集中包含的密码：

```
$ oc explain apiserver.spec.tlsSecurityProfile.intermediate
```

输**出示例**

```
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
```

- 查看组件的 **tlsSecurityProfile** 字段的所有详情：

```
$ oc explain <component>.spec.tlsSecurityProfile (1)
```

(1) 对于 **<component>**，指定 **ingresscontroller**、**apiserver** 或 **kubeletconfig**。

例如，检查 Ingress Controller 的 **tlsSecurityProfile** 字段的所有详情：

```
$ oc explain ingresscontroller.spec.tlsSecurityProfile
```

输**出示例**

```
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 列出 **intermediate** 配置集的密码和最小版本。

2 这里列出了 **modern** 配置集的密码和最小版本。

3 这里列出了 **old** 配置集的密码和最小版本。

12.3. 为 INGRESS CONTROLLER 配置 TLS 安全配置集

要为 Ingress Controller 配置 TLS 安全配置集，请编辑 **IngressController** 自定义资源（CR）来指定预定义或自定义 TLS 安全配置集。如果没有配置 TLS 安全配置集，则默认值基于为 API 服务器设置的 TLS 安全配置集。

配置 Old TLS 安全配置集的 IngressController CR 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS 安全配置集定义 Ingress Controller 的 TLS 连接的最低 TLS 版本和 TLS 密码。

您可以在 **Status.Tls Profile** 和 **Spec.Tls Security Profile** 下看到 **IngressController** 自定义资源（CR）中配置的 TLS 安全配置集的密码和最小 TLS 版本。对于 **Custom** TLS 安全配置集，这两个参数下列出了特定的密码和最低 TLS 版本。

注意

HAProxy Ingress Controller 镜像支持 TLS **1.3** 和 **Modern** 配置集。

Ingress Operator 还会将 **Old** 或 **Custom** 配置集的 TLS **1.0** 转换为 **1.1**。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 编辑 **openshift-ingress-operator** 项目中的 **IngressController** CR，以配置 TLS 安全配置集：

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. 添加 **spec.tlsSecurityProfile** 字段：

Custom 配置集的 IngressController CR 示例

```
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) 指定 TLS 安全配置集类型（**Old**、**Intermediate** 或 **Custom**）。默认值为 **Intermediate**。

(2) 为所选类型指定适当的字段：

- **old: {}**
- **intermediate: {}**
- **custom:**

(3) 对于 **custom** 类型，请指定 TLS 密码列表和最低接受的 TLS 版本。

3. 保存文件以使改变生效。

验证

- 验证 **IngressController** CR 中是否设置了配置集：

```
$ oc describe IngressController default -n openshift-ingress-operator
```

输出示例

```
Name:         default
Namespace:    openshift-ingress-operator
```

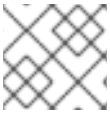
```
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

12.4. 为 CONTROL PLANE 配置 TLS 安全配置集

要为 control plane 配置 TLS 安全配置集，请编辑 **APIServer** 自定义资源（CR）来指定预定义或自定义 TLS 安全配置集。在 **APIServer** CR 中设置 TLS 安全配置集将设置传播到以下 control plane 组件：

- Kubernetes API 服务器
- Kubernetes 控制器管理器
- Kubernetes 调度程序
- OpenShift API 服务器
- OpenShift OAuth API 服务器
- OpenShift OAuth 服务器
- etcd

如果没有配置 TLS 安全配置集，则默认 TLS 安全配置集为 **Intermediate**。



注意

Ingress Controller 的默认 TLS 安全配置集基于为 API 服务器设置的 TLS 安全配置集。

配置 Old TLS 安全配置集的 APIServer CR 示例

```
apiVersion: config.openshift.io/v1
kind: APIServer
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS 安全配置集定义与 control plane 组件通信的最低 TLS 版本和所需的 TLS 密码。

您可以在 **Spec.Tls Security Profile** 下的 **APIServer** 自定义资源（CR）中看到配置的 TLS 安全配置集。对于 **Custom** TLS 安全配置集，会列出特定的密码和最小 TLS 版本。

注意

control plane 不支持 TLS **1.3** 作为最小 TLS 版本；不支持 **Modern** 配置集，因为它需要 TLS **1.3**。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 编辑默认 **APIServer** CR 以配置 TLS 安全配置集：

```
$ oc edit APIServer cluster
```

2. 添加 **spec.tlsSecurityProfile** 字段：

Custom 配置集的 APIServer CR 示例

```
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) 指定 TLS 安全配置集类型（**Old**、**Intermediate** 或 **Custom**）。默认值为 **Intermediate**。

(2) 为所选类型指定适当的字段：

- **old: {}**
- **intermediate: {}**
- **custom:**

(3) 对于 **custom** 类型，请指定 TLS 密码列表和最低接受的 TLS 版本。

3. 保存文件以使改变生效。

验证

- 验证 **APIServer** CR 中是否设置了 TLS 安全配置集：

```
$ oc describe apiserver cluster
```

输出示例

```
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
 ...
```

- 验证 TLS 安全配置集是否在 **etcd** CR 中设置：

```
$ oc describe etcd cluster
```

输出示例

```
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. 为 KUBELET 配置 TLS 安全配置集

要在作为 HTTP 服务器时为 kubelet 配置 TLS 安全配置集，请创建一个 **KubeletConfig** 自定义资源（CR）来为特定节点指定预定义或自定义 TLS 安全配置集。如果没有配置 TLS 安全配置集，则默认 TLS 安全配置集为 **Intermediate**。

kubelet 使用其 HTTP/GRPC 服务器与 Kubernetes API 服务器通信，后者向 pod 发送命令，收集日志，并通过 kubelet 对 pod 运行 exec 命令。

在 worker 节点上配置 Old TLS 安全配置集的 KubeletConfig CR 示例

```
apiVersion: config.openshift.io/v1
kind: KubeletConfig
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
#...
```

您可以在配置的节点上的 **kubelet.conf** 文件中看到配置 TLS 安全配置集的密码和最小 TLS 版本。

先决条件

- 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform。

流程

1. 创建 **KubeletConfig** CR 来配置 TLS 安全配置集：

   Custom 配置集的 KubeletConfig CR 示例

```
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. 指定 TLS 安全配置集类型（**Old**、**Intermediate** 或 **Custom**）。默认值为 **Intermediate**。
2. 为所选类型指定适当的字段：
   - **old: {}**
   - **intermediate: {}**
   - **custom:**
3. 对于 **custom** 类型，请指定 TLS 密码列表和最低接受的 TLS 版本。
4. 可选：为您要应用 TLS 安全配置集的节点指定机器配置池标签。

2. 创建 **KubeletConfig** 对象：

```
$ oc create -f <filename>
```

根据集群中的 worker 节点数量，等待配置的节点被逐个重启。

验证

要验证是否设置了配置集，请在节点处于 **Ready** 状态后执行以下步骤：

1. 为配置的节点启动 debug 会话：

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

2. 将 **/host** 设置为 debug shell 中的根目录：

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

3. 查看 **kubelet.conf** 文件：

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

输出示例

```
  "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",
#...
```

第 13 章 配置 SECCOMP 配置集

OpenShift Container Platform 容器或 pod 运行一个应用程序，它执行一个或多个定义明确的任务。应用程序通常只需要底层操作系统内核 API 的一小部分。安全计算模式 seccomp 是一个 Linux 内核功能，可用于将容器中运行的进程限制为仅使用可用系统调用的子集。

restricted-v2 SCC 适用于 4.16 中的所有新创建的 pod。默认 seccomp 配置集 **runtime/default** 应用到这些 pod。

seccomp 配置集作为 JSON 文件存储在磁盘上。

重要

seccomp 配置集不能应用到特权容器。

13.1. 验证应用到 POD 的默认 SECCOMP 配置集

OpenShift Container Platform 附带了一个默认的 seccomp 配置集，它被引用为 **runtime/default**。在 4.16 中，新创建的 pod 将安全上下文约束(SCC) 设置为 **restricted-v2**，默认的 seccomp 配置集应用到 pod。

流程

1. 您可以运行以下命令来验证 pod 上设置了安全性上下文约束(SCC) 和默认 seccomp 配置集：

   a. 验证在命名空间中运行的 pod:

   ```
   $ oc get pods -n <namespace>
   ```

   例如，要验证在 **workshop** 命名空间中运行的 pod，请运行以下命令：

   ```
   $ oc get pods -n workshop
   ```

   输**出示例**

   ```
   NAME              READY   STATUS      RESTARTS   AGE
   parksmap-1-4xkwf  1/1     Running     0          2m17s
   parksmap-1-deploy 0/1     Completed   0          2m22s
   ```

   b. 检查 pod：

   ```
   $ oc get pod parksmap-1-4xkwf -n workshop -o yaml
   ```

   输**出示例**

   ```
   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. 如果您的工作负载无法访问其他 SCC，则默认添加 **restricted-v2** SCC。
2. 4.16 中新创建的 pod 会将 seccomp 配置集配置为 **runtime/default**，如 SCC 强制。

13.1.1. 升级的集群

在升级到 4.16 的集群中，所有经过身份验证的用户都可以访问 **restricted** 和 **restricted-v2** SCC。

由 SCC **restricted** 接受的负载，例如升级的 OpenShift Container Platform v4.10 集群可以被 **restricted-v2** 接受。这是因为 **restricted-v2** 在 **restricted** 和 **restricted-v2** 间是更加严格的 restrictive SCC。

注意

工作负载必须能够使用 **retricted-v2** 运行。

相反，对于需要 **privilegeEscalation: true** 的工作负载，此工作负载将继续对任何经过身份验证的用户使用 **restricted** SCC。这是因为 **restricted-v2** 不允许 **privilegeEscalation**。

13.1.2. 新安装的集群

对于新安装的 OpenShift Container Platform 4.11 或更高版本的集群，**restricted-v2** 会将 **restricted** SCC 替换为可供任何经过身份验证的用户使用的 SCC。具有 **privilegeEscalation: true** 的工作负载不会接受到集群中，因为 **restricted-v2** 是默认可供经过身份验证的用户使用的唯一 SCC。

privilegeEscalation 被 **restricted** 允许但不被 **restricted-v2** 允许。更多功能会被 **restricted-v2** 决绝，但会被 **restricted** SCC 允许。

具有 **privilegeEscalation: true** 的工作负载可能会被接受到新安装的 OpenShift Container Platform v4.11 或更高版本的集群中。要使用 RoleBinding 运行以下命令，为运行工作负载（或任何其他可以接受此工作负载的 ServiceAccount）授予 **restricted** SCC 的访问权限：

-

```
$ oc -n <workload-namespace> adm policy add-scc-to-user <scc-name> -z <serviceaccount_name>
```

在 OpenShift Container Platform 4.16 中，添加 pod 注解 **seccomp.security.alpha.kubernetes.io/pod: runtime/default** 和 **container.seccomp.security.alpha.kubernetes.io/<container_name>: runtime/default** 已被弃用。

13.2. 配置自定义 SECCOMP 配置集

您可以配置自定义 seccomp 配置集，允许您根据应用要求更新过滤器。这使得集群管理员能够更好地控制在 OpenShift Container Platform 中运行的工作负载的安全性。

seccomp 安全配置集列出了进程可以进行的系统调用（系统调用）。权限比 SELinux 更广泛，可限制操作，如全系统范围内的 **write**。

13.2.1. 创建 seccomp 配置集

您可以使用 **MachineConfig** 对象来创建配置集。

seccomp 可以限制容器内的系统调用（系统调用），限制应用的访问权限。

先决条件

- 有集群管理员权限。
- 您已创建了自定义安全上下文约束 (SCC)。如需更多信息，请参阅*附加资源*。

流程

- 创建 **MachineConfig** 对象：

```
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. 设置自定义 seccomp 配置集

前提条件

- 有集群管理员权限。

- 您已创建了自定义安全上下文约束 (SCC)。如需更多信息，请参阅"附加资源"。
- 您已创建了自定义 seccomp 配置集。

流程

1. 使用 Machine Config 将自定义 seccomp 配置集上传到 **/var/lib/kubelet/seccomp/<custom-name>.json**。有关详细信息，请参阅"附加资源"。
2. 通过引用创建的自定义 seccomp 配置集来更新自定义 SCC：

```
seccompProfiles:
- localhost/<custom-name>.json 1
```

1 提供自定义 seccomp 配置集的名称。

13.2.3. 将自定义 seccomp 配置集应用到工作负载

前提条件

- 集群管理员已设置了自定义 seccomp 配置集。如需了解更多详细信息，请参阅 "设置自定义 seccomp 配置集"。

流程

- 通过设置 **securityContext.seccompProfile.type** 字段，将 seccomp 配置集应用到工作负载，如下所示：

  示例

```
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: <custom-name>.json 1
```

1 提供自定义 seccomp 配置集的名称。

  另外，您可以使用 pod 注解 **seccomp.security.alpha.kubernetes.io/pod: localhost/<custom-name>.json**。但是，此方法在 OpenShift Container Platform 4.16 中已弃用。

在部署过程中，准入控制器会验证以下内容：

- 用户角色允许的当前 SCC 的注解。
- Pod 允许包含 seccomp 配置集的 SCC。

如果 pod 允许 SCC，kubelet 会使用指定的 seccomp 配置集运行 pod。

重要

确保 seccomp 配置集已部署到所有 worker 节点。

注意

自定义 SCC 必须具有自动分配给 pod 的适当优先级，或满足 Pod 所需的其他条件，如允许 CAP_NET_ADMIN。

13.3. 其他资源

- 管理安全性上下文约束
- 机器配置概述

第 14 章 允许从其他主机对 API 服务器进行基于 JAVASCRIPT 的访问

14.1. 允许从其他主机对 API 服务器进行基于 JAVASCRIPT 的访问

默认的 OpenShift Container Platform 配置只允许 Web 控制台向 API 服务器发送请求。

如果需要使用不同的主机名从 JavaScript 应用程序访问 API 服务器或 OAuth 服务器，您可以配置额外的主机名来允许。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

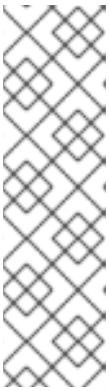
1. 编辑 **APIServer** 资源：

```
$ oc edit apiserver.config.openshift.io cluster
```

2. 在 **spec** 部分下添加 **additionalCORSAllowedOrigins** 字段，并指定一个或多个额外主机名：

```
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 主机名用 Golang 正则表达式指定，与来自对 API 服务器和 OAuth 服务器的 HTTP 请求的 CORS 标头匹配。



注意

此示例采用以下语法：

- **(?i)** 使它不区分大小写。
- // 固定到域的开头，并且匹配 **http:** 或 **https:** 后面的双斜杠。
- **\.** 对域名中的点进行转义。
- **(:|\z)** 匹配域名末尾 **(\z)** 或端口分隔符 **(:)**。

3. 保存文件以使改变生效。

第 15 章 加密 ETCD 数据

15.1. 关于 ETCD 加密

默认情况下，OpenShift Container Platform 不加密 etcd 数据。在集群中启用对 etcd 进行加密的功能可为数据的安全性提供额外的保护层。例如，如果 etcd 备份暴露给不应该获得这个数据的人员，它会帮助保护敏感数据。

启用 etcd 加密时，以下 OpenShift API 服务器和 Kubernetes API 服务器资源将被加密：

- Secrets
- 配置映射
- Routes
- OAuth 访问令牌
- OAuth 授权令牌

当您启用 etcd 加密时，会创建加密密钥。您必须具有这些密钥才能从 etcd 备份中恢复。

注意

etcd 加密只加密值，而不加密键。资源类型、命名空间和对象名称是未加密的。

如果在备份过程中启用了 etcd 加密，***static_kuberesources_<datetimestamp>.tar.gz*** 文件包含 etcd 快照的加密密钥。为安全起见，请将此文件与 etcd 快照分开存储。但是，需要这个文件才能从相应的 etcd 快照恢复以前的 etcd 状态。

15.2. 支持的加密类型

在 OpenShift Container Platform 中，支持以下加密类型来加密 etcd 数据：

AES-CBC

使用带有 PKCS#7 padding 和 32 字节密钥的 AES-CBC 来执行加密。加密密钥每周轮转。

AES-GCM

使用带有随机 nonce 和 32 字节密钥的 AES-GCM 来执行加密。加密密钥每周轮转。

15.3. 启用 ETCD 加密

您可以启用 etcd 加密来加密集群中的敏感资源。

警告

在初始加密过程完成前，不要备份 etcd 资源。如果加密过程还没有完成，则备份可能只被部分加密。

启用 etcd 加密后，可能会出现一些更改：

- etcd 加密可能会影响几个资源的内存消耗。
- 您可能会注意到对备份性能具有临时影响，因为领导必须提供备份服务。
- 磁盘 I/O 可能会影响接收备份状态的节点。

您可以在 AES-GCM 或 AES-CBC 加密中加密 etcd 数据库。

注意

要将 etcd 数据库从一个加密类型迁移到另一个加密类型，您可以修改 API 服务器的 **spec.encryption.type** 字段。将 etcd 数据迁移到新的加密类型会自动进行。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 修改 **APIServer** 对象：

```
$ oc edit apiserver
```

2. 将 **spec.encryption.type** 字段设置为 **aesgcm** 或 **aescbc** ：

```
spec:
  encryption:
    type: aesgcm 1
```

1 设置为 **aesgcm** 用于 AES-GCM 加密，或设置为 **aescbc** 用于 AES-CBC 加密。

3. 保存文件以使改变生效。
   加密过程开始。根据 etcd 数据库的大小，这个过程可能需要 20 分钟或更长时间才能完成。

4. 验证 etcd 加密是否成功。

   a. 查看 OpenShift API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功加密：

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

   在成功加密后输出显示 **EncryptionCompleted**：

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

如果输出显示 **EncryptionInProgress**，加密仍在进行中。等待几分钟后重试。

b. 查看 Kubernetes API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功加密：

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

在成功加密后输出显示 **EncryptionCompleted**：

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

如果输出显示 **EncryptionInProgress**，加密仍在进行中。等待几分钟后重试。

c. 查看 OpenShift OAuth API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功加密：

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

在成功加密后输出显示 **EncryptionCompleted**：

```
EncryptionCompleted
All resources encrypted: oauthaccesstokens.oauth.openshift.io,
oauthauthorizetokens.oauth.openshift.io
```

如果输出显示 **EncryptionInProgress**，加密仍在进行中。等待几分钟后重试。

15.4. 禁用 ETCD 加密

您可以在集群中禁用 etcd 数据的加密。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 修改 **APIServer** 对象：

```
$ oc edit apiserver
```

2. 将 **encryption** 字段类型设置为 **identity**：

```
spec:
  encryption:
    type: identity (1)
```

(1) **identity** 类型是默认值，意味着没有执行任何加密。

3. 保存文件以使改变生效。
   解密过程开始。根据集群的大小，这个过程可能需要 20 分钟或更长的时间才能完成。

4. 验证 etcd 解密是否成功。

   a. 查看 OpenShift API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功解密：

   ```
   $ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
   (@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   在成功解密后输出显示 **DecryptionCompleted**：

   ```
   DecryptionCompleted
   Encryption mode set to identity and everything is decrypted
   ```

   如果输出显示 **DecryptionInProgress**，解密仍在进行中。等待几分钟后重试。

   b. 查看 Kubernetes API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功解密：

   ```
   $ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
   (@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   在成功解密后输出显示 **DecryptionCompleted**：

   ```
   DecryptionCompleted
   Encryption mode set to identity and everything is decrypted
   ```

   如果输出显示 **DecryptionInProgress**，解密仍在进行中。等待几分钟后重试。

   c. 查看 OpenShift OAuth API 服务器的 **Encrypted** 状态条件，以验证其资源是否已成功解密：

   ```
   $ oc get authentication.operator.openshift.io -o=jsonpath='{range
   .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   在成功解密后输出显示 **DecryptionCompleted**：

   ```
   DecryptionCompleted
   Encryption mode set to identity and everything is decrypted
   ```

   如果输出显示 **DecryptionInProgress**，解密仍在进行中。等待几分钟后重试。

第 16 章 对 POD 进行安全漏洞扫描

使用 Red Hat Quay Container Security Operator，您可以访问 OpenShift Container Platform Web 控制台中用于集群中活跃 pod 的容器镜像，访问 OpenShift Container Platform Web 控制台中的漏洞扫描结果。Red Hat Quay Container Security Operator：

- 监视与所有或指定命名空间中的 pod 关联的容器
- 查询容器来自漏洞信息的容器 registry，提供镜像的 registry 正在运行镜像扫描（如 Quay.io 或带有 Clair 扫描的 Red Hat Quay registry）
- 通过 Kubernetes API 中的 **ImageManifestVuln** 对象公开漏洞

根据这里的说明，Red Hat Quay Container Security Operator 安装在 **openshift-operators** 命名空间中，因此 OpenShift Container Platform 集群中的所有命名空间都可以使用它。

16.1. 安装 RED HAT QUAY CONTAINER SECURITY OPERATOR

您可以从 OpenShift Container Platform Web 控制台 Operator Hub 或 CLI 安装 Red Hat Quay Container Security Operator。

先决条件

- 已安装 **oc** CLI。
- 具有 OpenShift Container Platform 集群的管理员特权。
- 您有来自集群中运行的 Red Hat Quay 或 Quay.io registry 的容器。

流程

1. 您可以使用 OpenShift Container Platform Web 控制台安装 Red Hat Quay Container Security Operator：
   a. 在 Web 控制台中，进入到 **Operators → OperatorHub** 并选择 **Security**。
   b. 选择 **Red Hat Quay Container Security Operator**Operator，然后选择 **Install**。
   c. 在 **Red Hat Quay Container Security Operator**页面中，选择 **Install**。**Update channel**, **Installation mode**, 和 **Update approval** 会自动选择。**Installed Namespace** 字段默认为 **openshift-operators**。您可以根据需要调整这些设置。
   d. 选择 **Install**。在 **Installed Operators** 页中几分钟后会出现 **Red Hat Quay Container Security Operator**。
   e. 可选：您可以将自定义证书添加到 Red Hat Quay Container Security Operator。例如，在当前目录中创建一个名为 **quay.crt** 的证书。然后，运行以下命令将自定义证书添加到 Red Hat Quay Container Security Operator 中：

      ```
      $ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n openshift-operators
      ```

   f. 可选：如果您添加了自定义证书，请重启 Red Hat Quay Container Security Operator pod 以使新证书生效。

2. 另外，您可以使用 CLI 安装 Red Hat Quay Container Security Operator：

   a. 输入以下命令来检索 Container Security Operator 及其频道的最新版本：

```
$ oc get packagemanifests container-security-operator \
  -o jsonpath='{range .status.channels[*]}{@.currentCSV} {@.name}{"\n"}{end}' \
  | awk '{print "STARTING_CSV=" $1 " CHANNEL=" $2 }' \
  | sort -nr \
  | head -1
```

   输**出示例**

```
STARTING_CSV=container-security-operator.v3.8.9 CHANNEL=stable-3.8
```

   b. 使用上一命令的输出，为 Red Hat Quay Container Security Operator 创建一个 **Subscription** 自定义资源，并将它保存为 **container-security-operator.yaml**。例如：

```
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. 为 **spec.channel** 参数指定您在上一步中获取的值。

   2. 指定您在上一步中为 **spec.startingCSV** 参数获取的值。

   c. 输入以下命令应用配置：

```
$ oc apply -f container-security-operator.yaml
```

   输**出示例**

```
subscription.operators.coreos.com/container-security-operator created
```

16.2. 使用 RED HAT QUAY CONTAINER SECURITY OPERATOR

以下流程演示了如何使用 Red Hat Quay Container Security Operator。

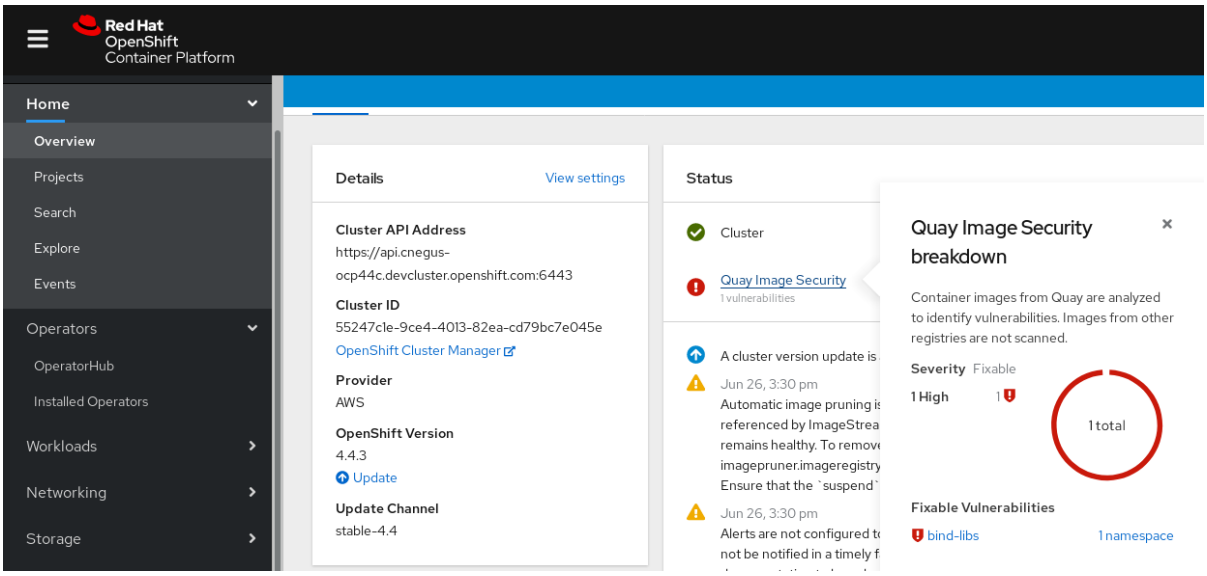
先决条件

- 已安装 Red Hat Quay Container Security Operator。

流程

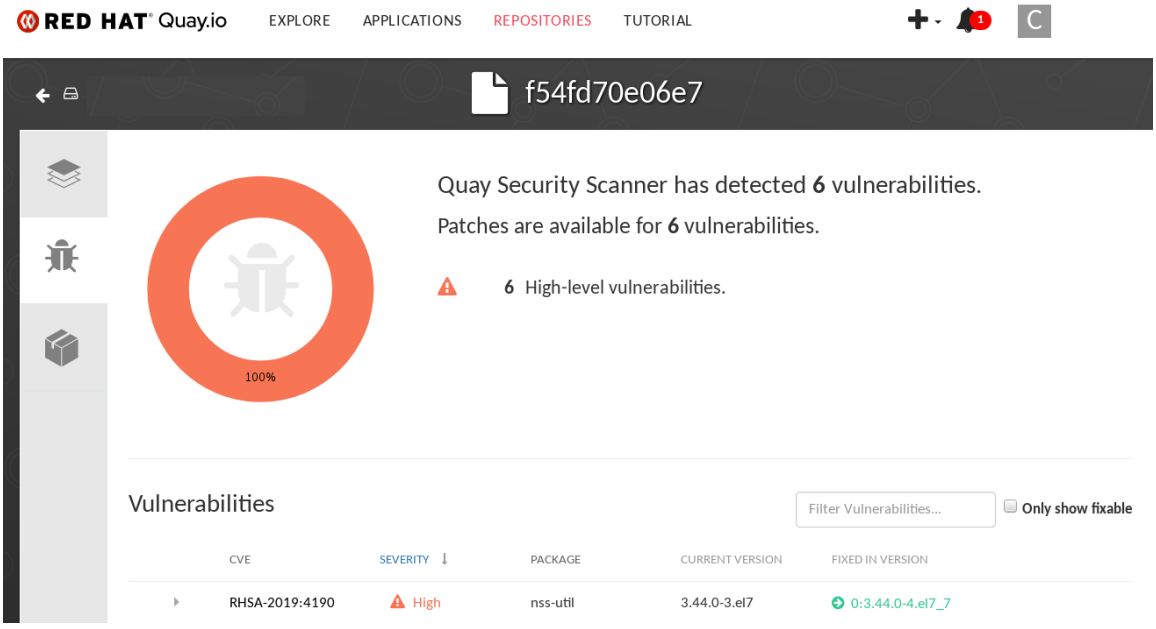
1. 在 OpenShift Container Platform web 控制台中进入至 **Home → Overview**。在 **Status** 部分下，**Quay Image Security** 提供发现的漏洞数量。

2. 点 **Quay Image Security** 以显示 **Quay Image Security breakdown**，其详细说明了漏洞的严重性、漏洞是否可以修复以及漏洞总数。例如：

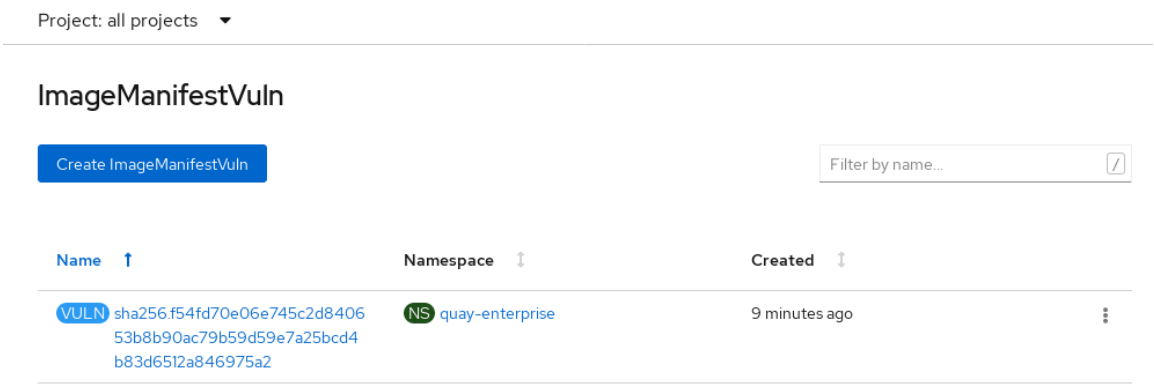


3. 您可以通过两种方式之一解决检测到的漏洞：

   a. 选择到这个漏洞的链接。这会带您进入容器来自的容器 registry，您可以在其中查看有关漏洞的信息。以下示例显示了从 Quay.io registry 中检测到的漏洞：



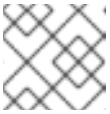
   b. 选择**命名空间**链接。这会把您带到 **ImageManifestVuln** 页面，您可以在其中查看所选镜像的名称以及该镜像运行的所有命名空间。例如，下图显示一个有特定安全漏洞镜像在 **quay-enterprise** 命名空间中运行：



4. 在了解了哪些镜像存在安全漏洞，以及如何修复这些漏洞，以及运行镜像的命名空间后，您可以执行以下操作来提高安全性：

   a. 提醒您机构中运行镜像的人员，并请求它们更正相关漏洞。

   b. 通过删除启动镜像所在 Pod 的部署或其他对象来停止镜像运行。



注意

如果删除 pod，可能需要几分钟时间才能在仪表板上重置漏洞信息。

16.3. 通过 CLI 查询镜像安全漏洞

使用 **oc** 命令，可以显示 Red Hat Quay Container Security Operator 检测到的漏洞信息。

先决条件

- 您已在 OpenShift Container Platform 实例上安装了 Red Hat Quay Container Security Operator。

流程

1. 输入以下命令查询检测到的容器镜像漏洞：

```
$ oc get vuln --all-namespaces
```

输**出示例**

```
NAMESPACE     NAME            AGE
default       sha256.ca90...  6m56s
skynet        sha256.ca90...  9m37s
```

2. 要显示特定漏洞的详情，请将漏洞名称及其命名空间附加到 **oc describe** 命令中。以下示例显示了一个活跃的容器，其镜像包含带有漏洞的 RPM 软件包：

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
```

输**出示例**

```
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...
```

第 17 章 网络绑定磁盘加密 (NBDE)

17.1. 关于磁盘加密技术

通过 Network-Bound Disk Encryption (NBDE)，您可以加密物理和虚拟机上的硬盘驱动器的根卷，而无需在重启机器时手动输入密码。

17.1.1. 磁盘加密技术比较

要了解 Network-Bound Disk Encryption (NBDE) 对于在边缘服务器中静态数据的安全，请在运行 Red Hat Enterprise Linux (RHEL) 的系统上将密钥托管和 TPM 磁盘加密与 NBDE 进行对比。

下表提供了需要考虑威胁模型和每个加密解决方案复杂性的一些权衡。

场景	密钥托管	TPM 磁盘加密（没有 Clevis）	NBDE
针对单磁盘失窃的保护	X	X	X
针对整个服务器偏移的保护	X		X
系统可以独立于网络重启		X	
没有定期重新密钥		X	
密钥永远不会通过网络传输		X	X
OpenShift 支持		X	X

17.1.1.1. 密钥托管

密钥托管是用于存储加密密钥的传统系统。网络上的密钥服务器存储具有加密引导磁盘节点的加密密钥，并在查询时返回。关键管理、传输加密和身份验证的复杂性不能成为引导磁盘加密的合理选择。

虽然在 Red Hat Enterprise Linux (RHEL) 中可用，但基于托管的磁盘加密设置和管理是一个手动过程，不适用于 OpenShift Container Platform 的自动化操作，包括自动添加节点，且当前不受 OpenShift Container Platform 支持。

17.1.1.2. TPM 加密

受信任的平台模块 (TPM) 磁盘加密最适合在远程保护位置的数据中心或安装。dm-crypt 和 BitLocker 使用 TPM 绑定密钥加密磁盘，然后将 TPM 绑定密钥存储在 TPM 中（附加到节点的主板）中。这种方法的主要优点是没有外部依赖项，节点可以在引导时解密自己的磁盘，而无需任何外部交互。

如果磁盘被从节点盗取并被外部分析，TPM 磁盘加密可以防止数据解密。但是，对于不安全的位置，这可能是不够的。例如，如果攻击者窃取整个节点，攻击者可以在打开节点电源时截获数据，因为节点会解密自己的磁盘。这适用于具有物理 TPM2 芯片的节点，以及具有虚拟可信平台模块 (VTPM) 访问的虚拟机。

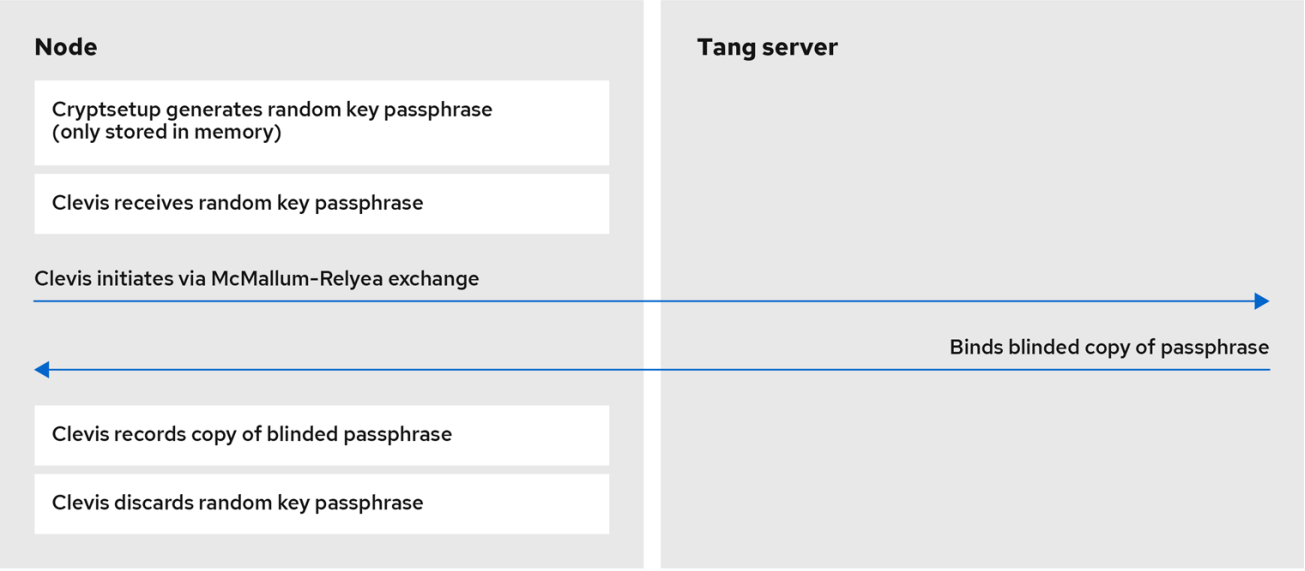
17.1.1.3. 网络绑定磁盘加密 (NBDE)

网络绑定磁盘加密 (NBDE) 通过网络以安全匿名的方式将加密密钥与外部服务器或一组服务器有效连接。这不是一个关键托管项，因为节点不存储加密密钥或将其传送到网络上，否则行为方式类似。

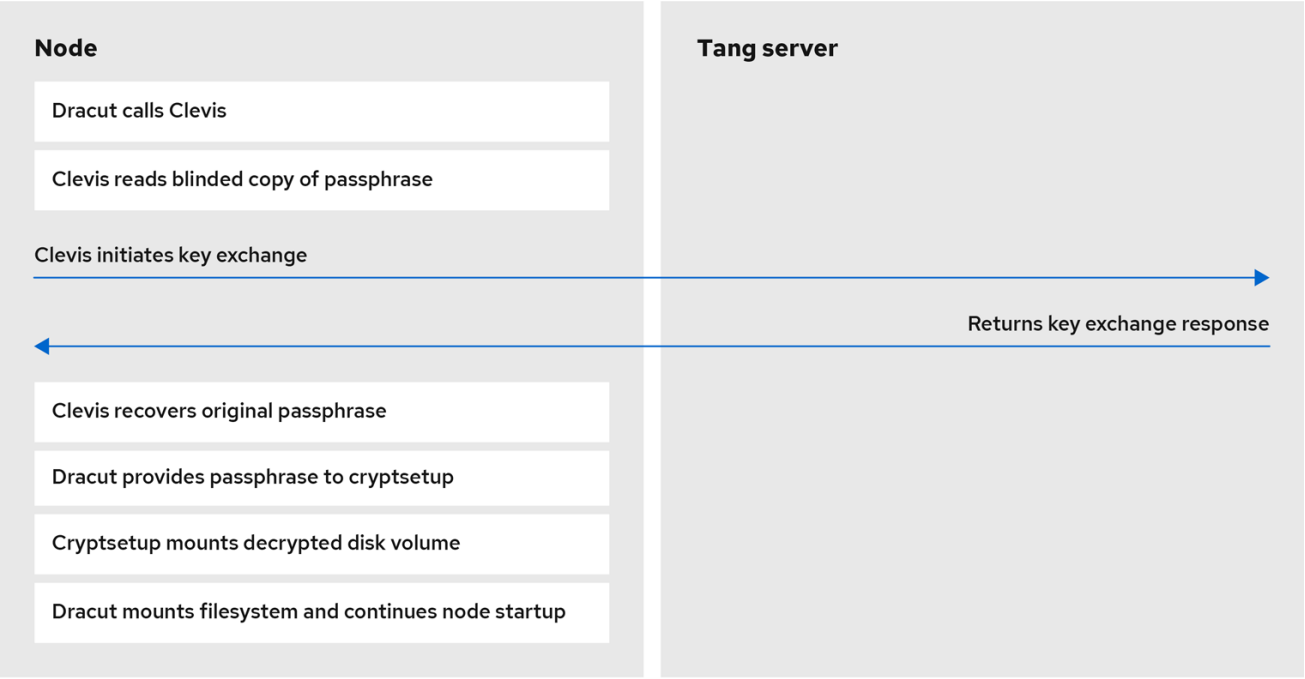
Clevis 和 Tang 是通用客户端和服务器组件，提供网络绑定加密。Red Hat Enterprise Linux CoreOS (RHCOS) 将这些组件与 Linux Unified Key Setup-on-disk-format (LUKS) 结合使用来加密和解密 root 和非 root 存储卷，以完成 Network-Bound Disk Encryption。

当节点启动时，它会尝试通过执行加密握手来联系预定义的 Tang 服务器集合。如果它可以访问所需的 Tang 服务器数量，节点可以构建其磁盘解密密钥并解锁磁盘以继续启动。如果节点因为网络中断或服务器不可用而无法访问 Tang 服务器，则该节点无法引导并继续无限期重试，直到 Tang 服务器再次可用为止。由于密钥有效与网络中节点的存在相关联，因此尝试获得静态数据的攻击者需要同时获取节点上的磁盘，以及访问 Tang 服务器的网络访问权限。

下图演示了 NBDE 的部署模型。



下图说明了重启期间 NBDE 的行为。



17.1.1.4. Secret 共享加密

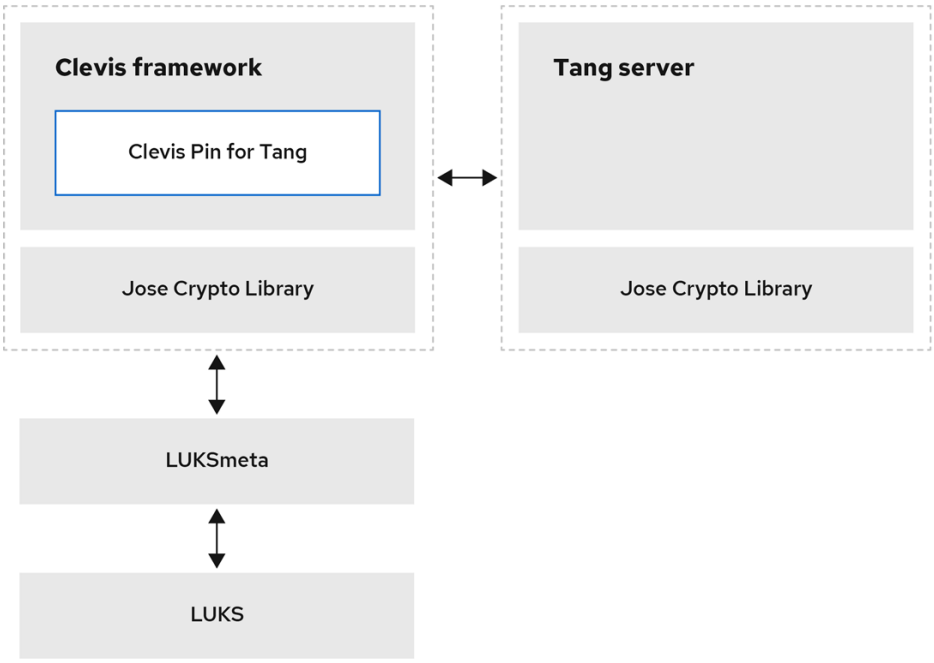
Shamir 的机密共享 (ss) 是一种加密算法，用于安全地划分、分发和重新集合密钥。使用这个算法，OpenShift Container Platform 可以支持更复杂的密钥保护组合。

当您将集群节点配置为使用多个 Tang 服务器时，OpenShift Container Platform 使用 ss 来设置解密策略，该策略将在至少一个指定服务器可用时成功。您可以创建层以提高安全性。例如，您可以定义一个策略，其中 OpenShift Container Platform 需要 TPM 和给定的 Tang 服务器列表之一来解密磁盘。

17.1.2. Tang 服务器磁盘加密

以下组件和技术实施网络绑定磁盘加密 (NBDE)。

图 17.1. 使用 LUKS1 加密的卷时的 NBDE 方案。luksmeta 软件包不用于 LUKS2 卷。



Tang 是一个将数据绑定到网络存在的服务器。它使得当节点绑定到特定安全网络时，包含可用数据的节点。Tang 是无状态的，不需要传输层安全 (TLS) 或身份验证。与基于 escrow 的解决方案不同，密钥服务器存储所有加密密钥并了解每个加密密钥，Tang 永远不会与任何节点密钥交互，因此它永远不会从节点获得任何识别信息。

Clevis 是一种可自动解密的可插入框架，提供 Linux Unified Key Setup-on-disk-format (LUKS) 卷的自动解锁。Clevis 软件包在节点上运行，并提供功能的客户端。

Clevis pin 是 Clevis 框架的一个插件。有三个固定类型：

TPM2

将磁盘加密绑定到 TPM2。

tang

将磁盘加密绑定到 Tang 服务器以启用 NBDE。

Shamir 的机密共享 (sss)

允许其他固定的更复杂的组合.它允许更多细微的策略，例如：

- 必须能够访问这三个 Tang 服务器中的一个
- 必须能够访问这五个 Tang 服务器中的三个
- 必须能够访问这三个 Tang 服务器中的至少一个 TPM2 和D

17.1.3. Tang 服务器位置规划

在规划 Tang 服务器环境时，请考虑 Tang 服务器的物理和网络位置。

物理位置

Tang 服务器的地理位置相对而言并不重要，只要它们能够适当保护不受未授权访问或失窃的影响，并提供运行关键服务所需的可用性和可访问性。

具有 Clevis 客户端的节点不需要本地 Tang 服务器，只要 Tang 服务器始终可用。灾难恢复需要冗余电源和冗余网络连接，而不管其位置如何。

网络位置

任何可通过网络访问 Tang 服务器的节点都可以解密自己的磁盘分区，或者由同一 Tang 服务器加密的任何其他磁盘。
为 Tang 服务器选择网络位置，以确保给定主机存在或不存在网络连接允许解密。例如，防火墙保护可能已实施，以禁止从任何类型的虚拟客户机或公共网络或位于该建筑不安全区域的任何网络jack的访问。

此外，还要维护生产和开发网络之间的网络隔离。这有助于定义适当的网络位置，并添加额外的安全层。

不要在同一资源上部署 Tang 服务器，如在同一个 **rolebindings.rbac.authorization.k8s.io** 集群，它们负责解锁。但是，Tang 服务器和其他安全资源集群可能是一个有用的配置，以启用对多个额外集群和集群资源的支持。

17.1.4. Tang 服务器大小要求

围绕可用性、网络和物理位置的要求推动了要使用多少 Tang 服务器的决策，而不是服务器容量的担忧。

Tang 服务器不维护使用 Tang 资源加密的数据状态。Tang 服务器可以完全独立，或者仅共享其关键材料，从而可以很好地扩展。

Tang 服务器可以通过两种方式处理关键资料：

- 多个 Tang 服务器共享关键资料：
  - 您必须对同一 URL 后面的 Tang 服务器共享密钥进行负载均衡。配置可以像循环 DNS 一样简单，也可以使用物理负载均衡器。
  - 您可以从单个 Tang 服务器扩展到多个 Tang 服务器。当 Tang 服务器共享密钥资料和相同的 URL 时，扩展 Tang 服务器不需要在节点上重新密钥或客户端重新配置。
  - 客户端节点设置和密钥轮转只需要一个 Tang 服务器。
- 多个 Tang 服务器生成自己的关键资料：
  - 您可以在安装时配置多个 Tang 服务器。
  - 您可以在负载均衡器后面扩展单个 Tang 服务器。
  - 所有 Tang 服务器都必须在客户端节点设置或密钥轮转过程中可用。
  - 当客户端节点使用默认配置引导时，Clevis 客户端会联系所有 Tang 服务器。只有 *n* 个 Tang 服务器必须在线才能进行解密。*n* 的默认值是 1。
  - 红帽不支持更改 Tang 服务器行为的安装后配置。

17.1.5. 日志记录注意事项

集中记录 Tang 流量具有优势，因为您可以发现意外的解密请求等事件。例如：

- 请求解密与其引导序列不匹配的密码短语的节点

- 在已知维护活动之外请求解密的节点，如循环密钥

17.2. TANG 服务器安装注意事项

安装群集节点时，必须启用 Network-Bound Disk Encryption (NBDE)。但是，您可以在安装时初始化后随时更改磁盘加密策略。

17.2.1. 安装场景

在规划 Tang 服务器安装时请考虑以下建议：

- 小环境可以使用单一的关键材料组，即使使用多个 Tang 服务器：
  - 密钥轮转更为简单。
  - Tang 服务器可以轻松扩展，以实现高可用性。
- 大型环境可从多组关键材料中受益：
  - 在不同的物理环境中安装不需要在地理区域之间复制和同步关键材料。
  - 在大型环境中，密钥轮转更为复杂。
  - 节点安装和重新密钥要求所有 Tang 服务器的网络连接。
  - 网络流量可能会因为在解密过程中查询所有 Tang 服务器的节点引导而发生小幅增长。请注意，尽管只有一个 Clevis 客户端查询必须成功，但 Clevis 查询所有 Tang 服务器。
- 进一步的复杂性：
  - 额外的手动重新配置可以允许 **any N of M servers online** 的 Shamir 的 secret 共享 (sss) 来解密磁盘分区。在这种情况下，解密磁盘需要多组关键资料，并在初始安装后通过 Clevis 客户端手动管理 Tang 服务器和节点。
- 高级别建议：
  - 对于单个 RAN 部署，一组有限的 Tang 服务器可以在对应的域控制器 (DC) 中运行。
  - 对于多个 RAN 部署，您必须确定是否在每个对应的 DC 中运行 Tang 服务器，或者一个全局 Tang 环境是否更能满足系统的其他需求和要求。

17.2.2. 安装 Tang 服务器

要部署一个或多个 Tang 服务器，您可以根据您的场景从以下选项中选择：

1. 使用 NBDE Tang Server Operator 部署 Tang 服务器
2. 在 RHEL 系统上部署 SELinux 处于 enforcing 模式的 Tang 服务器
3. 在 RHEL web 控制台中配置 Tang 服务器
4. 将 Tang 部署为容器
5. 使用 nbde_server 系统角色设置多个 Tang 服务器

17.2.2.1. 计算要求

Tang 服务器的计算要求非常低。任何您要用于将服务器部署到生产中的典型服务器评级配置都可以调配足够的计算容量。

高可用性注意事项仅取决于可用性，而非额外的计算能力，以满足客户的需求。

17.2.2.2. 在引导时自动启动

由于 Tang 服务器使用的密钥材料的敏感性质，您应该记住，Tang 服务器引导序列期间手动干预的开销可能很有用。

默认情况下，如果 Tang 服务器启动且没有预期的本地卷中的关键资料，它将创建新的资料并为其提供服务。您可以通过从预先存在的密钥材料开始或中止启动并等待手动干预来避免这种默认行为。

17.2.2.3. HTTP 与 HTTPS

到 Tang 服务器的流量可以加密 (HTTPS) 或纯文本 (HTTP)。加密此流量没有显著的安全优势，在进行解密后，它删除了运行 Clevis 客户端的节点中与传输层安全 (TLS) 证书检查相关的复杂性或故障条件。

虽然可以对节点的 Clevis 客户端和 Tang 服务器之间的未加密流量执行被动监控，但使用此流量确定关键资料的能力最好是未来的理论问题。任何此类流量分析都需要大量捕获的数据。密钥轮转会立即将其无效。最后，任何能够执行被动监控的威胁操作者已获得必要的网络访问权限，以执行到 Tang 服务器的手动连接，并可对捕获的 Clevis 标头进行简单的手动解密。

但是，因为安装站点中存在的其他网络策略可能需要流量加密，而无论应用程序是什么，请考虑将此决定保留给集群管理员。

其他资源

- RHEL 8 安全强化 文档中 使用基于策略的解密配置加密卷的自动解锁
- 官方 Tang 服务器容器
- 在安装过程中加密和镜像磁盘

17.3. TANG 服务器加密密钥管理

重新创建加密密钥的加密机制基于节点上存储的 *blinded key* 以及相关 Tang 服务器的私钥。为防止攻击者同时获得 Tang 服务器私钥和节点的加密磁盘的可能性，建议定期重新密钥。

您必须对每个节点执行重新密钥操作，然后才能从 Tang 服务器中删除旧密钥。以下小节提供了重新密钥和删除旧密钥的步骤。

17.3.1. 为 Tang 服务器备份密钥

Tang 服务器使用 **/usr/libexec/tangd-keygen** 生成新密钥，并将其默认存储在 **/var/db/tang** 目录中。要在失败时恢复 Tang 服务器，请备份这个目录。密钥是敏感的，因为它们能够对已使用它们的所有主机执行引导磁盘解密，因此必须相应地保护密钥。

流程

- 将 **/var/db/tang** 目录中的备份密钥复制到可以恢复密钥的 temp 目录。

17.3.2. 为 Tang 服务器恢复密钥

您可以通过从备份访问密钥来恢复 Tang 服务器的密钥。

流程

- 将备份文件夹中的密钥恢复到 **/var/db/tang/** 目录。
  当 Tang 服务器启动时，它会公告并使用这些恢复的密钥。

17.3.3. 重新密钥 (Rekeying) Tang 服务器

此流程使用一组包含三个 Tang 服务器，各自具有唯一密钥作为一个示例。

使用冗余 Tang 服务器可减少节点无法自动引导的几率。

重新加密 Tang 服务器和所有关联的 NBDE 加密节点是一个三个步骤。

先决条件

- 在一个或多个节点上安装正常工作的网络级磁盘加密 (NBDE) 安装。

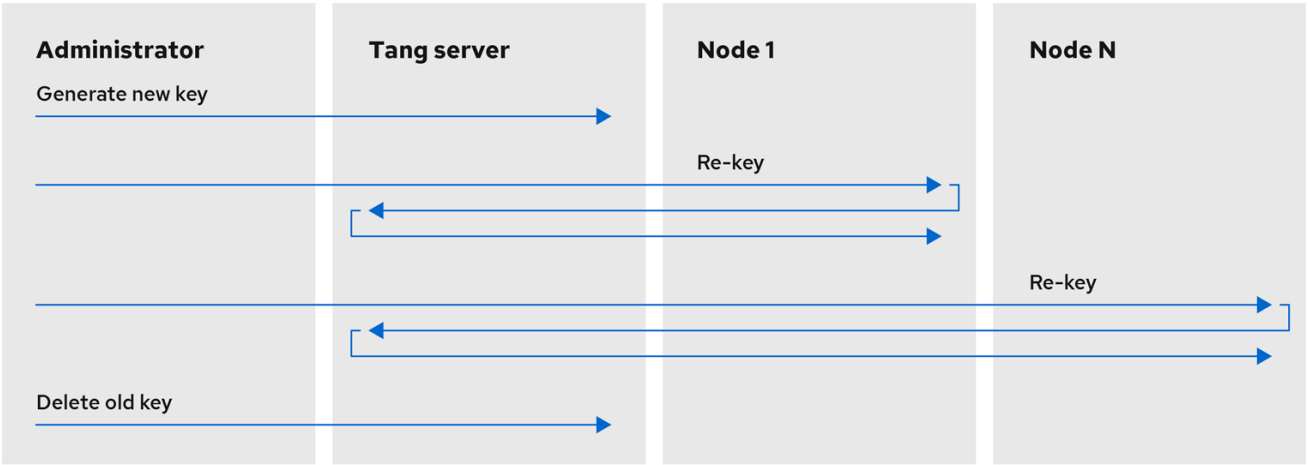
流程

1. 生成新的 Tang 服务器密钥。
2. 重新密钥所有 NBDE 加密节点，以便它们使用新密钥。
3. 删除旧的 Tang 服务器密钥。

注意

在所有 NBDE 加密的节点完成重新密钥前删除旧密钥会导致这些节点过度依赖于任何其他配置的 Tang 服务器。

图 17.2. 重新密钥 Tang 服务器的工作流示例



17.3.3.1. 生成新的 Tang 服务器密钥

先决条件

- 运行 Tang 服务器的 Linux 计算机上的 root shell。
- 为了便于验证 Tang 服务器密钥轮转，请使用旧密钥加密小测试文件：

```
# echo plaintext | clevis encrypt tang '{"url":"http://localhost:7500”}' -y >/tmp/encrypted.oldkey
```

- 验证加密成功，并且可以解密文件来生成相同的字符串的**明文** ：

```
# clevis decrypt </tmp/encrypted.oldkey
```

流程

1. 找到并访问存储 Tang 服务器密钥的目录。这通常是 **/var/db/tang** 目录。检查当前公告的密钥 thumbprint:

```
# tang-show-keys 7500
```

输**出示例**

```
36AHjNH3NZDSnlONLz1-V4ie6t8
```

2. 输入 Tang 服务器密钥目录：

```
# cd /var/db/tang/
```

3. 列出当前的 Tang 服务器密钥：

```
# ls -A1
```

输**出示例**

```
36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
```

在正常的 Tang 服务器操作过程中，此目录中有两个 **.jwk** 文件：一个用于签名和验证，另一个用于密钥生成。

4. 禁用旧密钥的公告：

```
# for key in *.jwk; do \
  mv -- "$key" ".$key"; \
done
```

设置 Network-Bound Disk 加密 (NBDE) 或请求密钥的新客户端将不再看到旧密钥。现有的客户端仍然可以访问和使用旧密钥，直到它们被删除为止。Tang 服务器读取但不会公告存储在 UNIX 隐藏文件中的密钥，这些文件以 **.** 字符开头。

5. 生成新密钥：

```
# /usr/libexec/tangd-keygen /var/db/tang
```

6. 列出当前的 Tang 服务器密钥，以验证旧密钥不再公告，因为它们现在是隐藏的文件，并存在新密钥：

```
# ls -A1
```

输出示例

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

Tang 自动公告新密钥。

注意

较新的 Tang 服务器安装包括帮助程序 **/usr/libexec/tangd-rotate-keys** 目录，负责同时禁用公告并生成新密钥。

7. 如果您在共享相同关键资料的负载均衡器后面运行多个 Tang 服务器，请确保在继续之前，在此进行的更改会正确同步整个服务器集。

验证

1. 验证 Tang 服务器是否在广播新密钥，而不是公告旧密钥：

```
# tang-show-keys 7500
```

输出示例

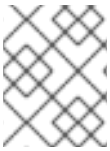
```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. 验证旧密钥（尽管未公告）仍然可用于解密请求：

```
# clevis decrypt </tmp/encrypted.oldkey
```

17.3.3.2. 重新密钥所有 NBDE 节点

您可以使用 **DaemonSet** 对象来更新远程集群中的所有节点的密钥，而不会给远程集群造成任何停机时间。



注意

如果在重新密钥过程中节点断电，则可能无法引导，且必须通过 Red Hat Advanced Cluster Management (RHACM) 或 GitOps 管道重新部署。

先决条件

- **cluster-admin** 对具有 Network-Bound Disk Encryption (NBDE) 节点的所有集群进行访问。
- 所有 Tang 服务器都必须可供每个 NBDE 节点进行 rekeying 进行访问，即使 Tang 服务器的密钥没有改变。

- 获取每个 Tang 服务器的 Tang 服务器 URL 和密钥 thumbprint。

流程

1. 根据以下模板创建 **DaemonSet** 对象。此模板设置三台冗余 Tang 服务器，但可轻松适应其他情况。更改 **NEW_TANG_PIN** 环境中的 Tang 服务器 URL 和 thumbprints 以适合您的环境：

```
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
```

在这种情况下，即使您对 **tangserver01** 进行 rekeying，您不仅要为 **tangserver01** 指定新的指纹，还必须为所有其他 Tang 服务器指定当前的指纹。如果未指定重新密钥操作的所有指纹，则可能会存在中间人攻击机的风险。

2. 要将守护进程集发布到必须重新密钥的每个集群，请运行以下命令：

```
$ oc apply -f tang-rekey.yaml
```

但是，为了大规模运行，请将守护进程集包装在 ACM 策略中。此 ACM 配置必须包含一个用于部署守护进程集的策略，第二个策略用于检查所有守护进程集 pod 是否都是 READY，以及一个放置规则，以将其应用到适当的集群集合。

注意

验证守护进程集是否已成功重新加密所有服务器后，请删除 守护进程集。如果您不删除守护进程集，则必须在下一次重新密钥操作之前删除它。

验证

分发守护进程集后，监控守护进程集，以确保成功完成重新密钥。示例守护进程集中的脚本在重新密钥失败时会终止并显示错误，如果成功，则保持 **CURRENT** 状态。还有一个就绪度探测，它会在重新密钥成功完成时将 pod 标记为 **READY**。

- 这是在重新密钥完成前守护进程集的输出列表示例：

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

输**出示例**

```
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR            AGE
tang-rekey   1         1         0       1            0           kubernetes.io/os=linux   11s
```

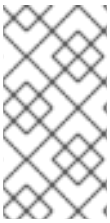
- 这是成功完成重新密钥后守护进程集的输出列表示例：

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

输出示例

```
NAME        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR           AGE
tang-rekey  1         1         1       1            1           kubernetes.io/os=linux   13h
```

完成重新密钥通常需要几分钟时间。



注意

如果您使用 ACM 策略将守护进程集分发到多个集群，则必须包含一个检查每个守护进程集的 READY 计数等于 DESIRED 计数的合规性策略。这样，遵守此类策略就表明所有守护进程集 pod 都是 READY，并且重新密钥已成功完成。您还可以使用 ACM 搜索来查询所有守护进程集的状态。

17.3.3.3. Tang 服务器的临时重新密钥错误故障排除

要确定重新密钥 Tang 服务器的错误条件是否为临时，请执行以下步骤。临时错误条件可能包括：

- 临时网络中断
- Tang 服务器维护

通常，当发生这些类型的临时错误条件时，您可以等待守护进程集成功解决错误，或者您可以删除守护进程集，且不会重试直到临时错误条件解决为止。

流程

1. 使用普通的 Kubernetes pod 重启策略，重启执行重新密钥操作的 pod。
2. 如果有任何关联的 Tang 服务器不可用，请尝试重新密钥，直到所有服务器重新上线。

17.3.3.4. Tang 服务器的永久重新密钥错误故障排除

如果在重新密钥 Tang 服务器后，**READY** 计数在延长时间后不等于 **DESIRED** 计数，这可能表示永久故障条件。在这种情况下，可能适用以下条件：

- Tang 服务器 URL 或 **NEW_TANG_PIN** 定义中的排字错误。
- Tang 服务器已停用，或者密钥永久丢失。

先决条件

- 此流程中显示的命令可以在 Tang 服务器或可通过网络访问 Tang 服务器的任何 Linux 系统上运行。

流程

1. 对守护进程集中定义的每个 Tang 服务器配置执行简单的加密和解密操作，以验证 Tang 服务器配置。
   这是使用错误的指纹进行加密和解密尝试的示例：

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver02:7500","thp":"badthumbprint"}' | \
  clevis decrypt
```

输**出示例**

```
Unable to fetch advertisement: 'http://tangserver02:7500/adv/badthumbprint'!
```

这是使用很好的指纹加密和解密尝试的示例：

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver03:7500","thp":"goodthumbprint"}' | \
  clevis decrypt
```

输**出示例**

```
okay
```

2. 在确定根本原因后，纠正根本情况：
   a. 删除无法正常工作的守护进程集。
   b. 编辑守护进程集定义，以修复根本问题。这可能包括以下操作：
      - 编辑 Tang 服务器条目，以更正 URL 和 thumbprint。
      - 删除不再处于服务中的 Tang 服务器。
      - 添加新 Tang 服务器，该服务器替代已停用的服务器。
3. 再次分发更新的守护进程集。

注意

当从配置替换、删除或添加 Tang 服务器时，只要至少一台原始服务器仍在正常工作，包括当前被重新密钥的服务器，就可以成功执行重新密钥操作。如果没有原始 Tang 服务器正常工作或可以恢复，则无法恢复系统，您必须重新部署受影响的节点。

验证

检查守护进程集中各个容器集的日志，以确定重新密钥是否成功完成。如果重新密钥不成功，日志可能会指示故障条件。

1. 找到由守护进程集创建的容器的名称：

```
$ oc get pods -A | grep tang-rekey
```

输**出示例**

```
openshift-machine-config-operator  tang-rekey-7ks6h  1/1  Running   20 (8m39s ago)  89m
```

2. 显示容器中的日志。以下日志来自一个成功完成的重新密钥操作：

```
$ oc logs tang-rekey-7ks6h
```

输**出示例**

```
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. 删除旧的 Tang 服务器密钥

先决条件

- 运行 Tang 服务器的 Linux 计算机上的 root shell。

流程

1. 找到并访问存储 Tang 服务器密钥的目录。这通常是 **/var/db/tang** 目录：

```
# cd /var/db/tang/
```

2. 列出当前的 Tang 服务器密钥，显示公告和未广播的密钥：

```
# ls -A1
```

输**出示例**

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

3. 删除旧密钥：

```
# rm .*.jwk
```

4. 列出当前的 Tang 服务器密钥以验证未归档的密钥不再存在：

```
# ls -A1
```

输**出示例**

```
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

验证

此时，服务器仍然会公告新密钥，但尝试根据旧密钥解密将失败。

1. 查询 Tang 服务器以获取当前公告的键 thumbprints:

```
# tang-show-keys 7500
```

输**出示例**

```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. 解密之前创建的测试文件以验证旧密钥的解密失败：

```
# clevis decrypt </tmp/encryptValidation
```

输**出示例**

```
Error communicating with the server!
```

如果您在共享相同关键资料的负载均衡器后面运行多个 Tang 服务器，请确保在继续之前，在此进行的更改会正确同步整个服务器集。

17.4. 灾难恢复注意事项

本节描述了几个潜在的灾难情况，以及应对每个灾难的步骤。在发现或假定可能发生其他情况时，将在此处添加其他情况。

17.4.1. 客户端机器丢失

当丢失了使用 Tang 服务器来对磁盘进行解密的集群节点时*并不会*造成灾难。无论计算机被盗、或出现硬件故障或其他丢失情景都不重要：磁盘会被加密并被视为不可恢复。

但是，如果被盗，Tang 服务器的密钥轮转和所有剩余节点的密钥重新密钥会明智地进行，从而确保磁盘即使在随后获得 Tang 服务器访问权限的情况下仍无法恢复。

要从这一情形中恢复，请重新安装或替换节点。

17.4.2. 计划丢失客户端网络连接

单个节点的网络连接丢失将导致其无法以无人值守的方式引导。

如果您计划做可能导致网络连接丢失的工作，您可以显示现场技术人员要手动使用的密码短语，然后在之后轮转密钥使其无效：

流程

1. 在网络不可用前，使用这个命令显示第一个插槽中使用的密码 **-s 1** 的设备 **/dev/vda** 2:

```
$ sudo clevis luks pass -d /dev/vda2 -s 1
```

2. 无效该值并使用这个命令重新生成一个新的随机引导时密语：

```
$ sudo clevis luks regen -d /dev/vda2 -s 1
```

17.4.3. 网络连接意外丢失

如果网络中断意外且节点重启，请考虑以下情况：

- 如果任何节点仍处于在线状态，请确保它们不会重启，直到恢复网络连接为止。这不适用于单节点集群。
- 节点将保持离线状态，直到恢复网络连接或在控制台中手动输入预先建立的密码短语。在特殊情况下，网络管理员可能能够重新配置网络段以重新建立访问权限，但这与 NBDE 的意图相反，即缺乏网络访问权限意味着缺乏启动能力。
- 节点中缺少网络访问可合理影响该节点正常工作的能力以及启动能力。即使该节点要通过手动干预引导，缺少网络访问也会使其有效无法使用。

17.4.4. 手动恢复网络连接

对于网络恢复而言，现场技术人员也可以使用稍微复杂且手动密集型的流程。

流程

1. 现场技术人员从硬盘中提取 Clevis 标头。根据 BIOS 锁定，这可能会涉及到删除磁盘并在实验室计算机中安装它们。
2. 现场技术人员将 Clevis 标头传输到具有合法访问权限的 Tang 网络的同事，然后执行解密。
3. 由于需要有限地访问 Tang 网络，技术人员应该无法通过 VPN 或其他远程连接访问该网络。同样，为了自动解密磁盘，技术人员无法通过此网络修补远程服务器。
4. 技术人员重新安装磁盘并手动输入其同事提供的纯文本密码短语。
5. 机器即使没有直接访问 Tang 服务器也成功启动。请注意，关键资料从安装站点传输到具有网络访问的另一个站点必须小心进行。
6. 恢复网络连接后，技术人员会轮转加密密钥。

17.4.5. 紧急恢复网络连接

如果您无法手动恢复网络连接，请考虑以下步骤。请注意，如果还有其他方法可以恢复网络连接，则不建议采用这些步骤。

- 这个方法只能通过高度信任的技术人员执行。
- 将 Tang 服务器的关键资料带到远程站点将被视为关键材料的破坏，而且所有服务器都必须更新密钥并重新加密。
- 这种方法必须仅在极端情况下使用，或者作为概念恢复方法验证来证明其可行性。
- 同样极端，但在理论上可行，是通过不可中断电源 (UPS) 为服务器提供动力，将服务器传输到具有网络连接的位置，以引导和解密磁盘，然后在攻击机原始位置恢复服务器，以继续操作。
- 如果要使用备份手动密码短语，您必须在出现失败前创建它。

- 正如在 TPM 和 Tang 与独立 Tang 安装相比，攻击场景变得更加复杂，因此，如果使用相同的方法，紧急灾难恢复过程也会变得更加复杂。

17.4.6. 网络片段丢失

如果网络片段丢失，导致 Tang 服务器暂时不可用，这会导致以下结果：

- 如果还有其他服务器可用，OpenShift Container Platform 节点将继续正常引导。
- 在恢复网络段前，新节点无法建立它们的加密密钥。在这种情况下，确保与远程地理位置的连接，以实现高可用性和冗余性。这是因为，当您安装新节点或重新打包现有节点的密钥时，您在该操作中引用的所有 Tang 服务器都必须可用。

对于高度多样化的网络，例如五个地理区域，每个客户端连接到最接近的三个客户端的混合模式值得调查。

在这种情况下，新客户端可以通过可访问的服务器子集建立其加密密钥。例如，在 **tang1**、**tang2** 和 **tang3** 服务器的集合中，如果 **tang2** 变为不可访问的客户端，仍然可以使用 **tang1** 和 **tang3** 建立其加密密钥，稍后使用全集重新建立其加密密钥。这可能涉及人工干预或更复杂的自动化。

17.4.7. 丢失 Tang 服务器

对客户端而言，丢失具有相同关键材料的负载平衡服务器中的单个 Tang 服务器完全透明。

与同一 URL 关联的所有 Tang 服务器的临时故障（即整个负载平衡集）可被视为与网络段的丢失相同。现有客户端能够解密其磁盘分区，只要有其他预配置的 Tang 服务器可用。只有其中一台服务器重新上线后，新客户端才能注册。

您可以通过重新安装服务器或从备份中恢复服务器来缓解 Tang 服务器的物理丢失。确保密钥材料的备份和恢复进程受到未授权访问的充分保护。

17.4.8. 重新密钥密钥

如果关键资料可能会暴露给未经授权的第三方，例如通过 Tang 服务器的物理偏移或相关数据，则立即轮转密钥。

流程

1. 为包含受影响材料的任何 Tang 服务器更新密钥。
2. 使用 Tang 服务器更新所有客户端的密钥。
3. 销毁原始密钥材料。
4. 检查导致意外公开主加密密钥的任何事件。如果可能，请脱机使受入侵的节点脱机并重新加密其磁盘。

提示

在同一物理硬件上重新格式化和重新安装（虽然速度较慢）很容易自动和测试。