OpenShift Container Platform 4.6

安全性与合规性

了解和管理 OpenShift Container Platform 的安全性

Last Updated: 2023-05-01

OpenShift Container Platform 4.6 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
- 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。

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

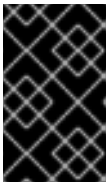
- 了解身份验证
- 配置内部 OAuth 服务器
- 了解身份提供程序配置
- 使用 RBAC 定义和应用权限
- 管理安全性上下文约束

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 4.6 在 RHCOS 主机上运行，同时可选择使用 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 安全功能是确保以更加安全的方式在 OpenShift 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 7 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 节点上禁用 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 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 适用于高效运行容器工作负载，以及 Kubernetes 和 OpenShift 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
- 适用的平台
- 具有用户置备基础架构的集群的机器要求
- 选择如何配置 RHCOS
- Ignition

- 内核参数
- 内核模块
- FIPS 加密
- 磁盘加密
- Chrony 时间服务
- OpenShift Container Platform 集群更新

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安全相关配置，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。

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（也称为主节点）。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。

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。

警告

不支持在 RHCOS 节点上禁用 SELinux。

2.3.2.2. 安装过程中强化

您可以中断 OpenShift 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 镜像创建 Red Hat Enterprise Linux CoreOS (RHCOS) 机器
- 自定义节点
- 为节点添加内核参数
- 安装配置参数 - 请参阅 **fips**
- 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

启用容器签名验证需要将 registry URL 链接到 sigstore 的文件，然后指定验证镜像的密钥。

流程

1. 创建将 registry URL 链接到 sigstore 并指定要验证镜像的密钥的文件。
   - 创建 **policy.json** 文件：

```
$ cat > policy.json <<EOF
{
  "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"
        }
      ]
    }
  }
}
EOF
```

   - 创建 **registry.access.redhat.com.yaml** 文件：

```
$ cat <<EOF > registry.access.redhat.com.yaml
```

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

- 创建 **registry.redhat.io.yaml** 文件：

```
$ cat <<EOF > registry.redhat.io.yaml
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
EOF
```

2. 使用 **base64** 编码格式设置用于机器配置模板的文件：

```
$ export ARC_REG=$( cat registry.access.redhat.com.yaml | base64 -w0 )
$ export RIO_REG=$( cat registry.redhat.io.yaml | base64 -w0 )
$ export POLICY_CONFIG=$( cat policy.json | base64 -w0 )
```

3. 创建将导出的文件写入 worker 节点上磁盘的机器配置：

```
$ cat > 51-worker-rh-registry-trust.yaml <<EOF
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 51-worker-rh-registry-trust
spec:
  config:
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
      version: 2.2.0
    networkd: {}
    passwd: {}
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,${ARC_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.access.redhat.com.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${RIO_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.redhat.io.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${POLICY_CONFIG}
          verification: {}
```

```
        filesystem: root
        mode: 420
        path: /etc/containers/policy.json
  osImageURL: ""
EOF
```

4. 应用创建的机器配置：

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

5. 创建机器配置，将导出的文件写入 master 节点上的磁盘：

```
$ cat > 51-master-rh-registry-trust.yaml <<EOF
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 51-master-rh-registry-trust
spec:
  config:
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
      version: 2.2.0
    networkd: {}
    passwd: {}
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,${ARC_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.access.redhat.com.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${RIO_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.redhat.io.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${POLICY_CONFIG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/policy.json
  osImageURL: ""
EOF
```

6. 将 master 机器配置更改应用到集群：

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

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. 其他资源

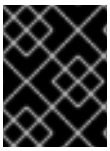
- 机器配置概述

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

只有在 **x86_64** 架构中的 OpenShift Container Platform 部署支持 FIPS 验证的/Modules in Process 加密库。

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 和 8 有标准的 UBI 镜像（**ubi7/ubi** 和 **ubi8/ubi**），以及基于这些系统的最小镜像（**ubi7/ubi-minimal** 和 **ubi8/ubi-mimimal**）。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 安全扫描程序执行。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

字段	必需？	描述	类型
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 4.6 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。

如需详细了解与红帽软件相关的安全更新和漏洞，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 Container Platform 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，以便稍后进行搜索和分析。
- **Clair 安全扫描**：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 OpenShift Container Platform 内部 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

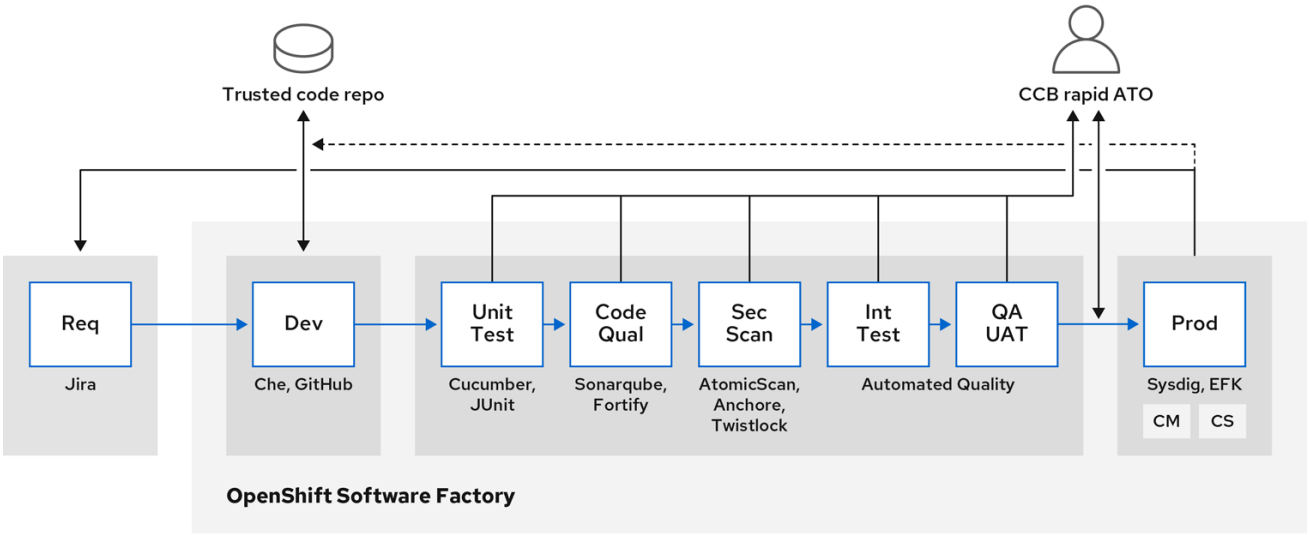
在容器环境中，软件构建过程是生命周期中应用程序代码与所需的运行时库集成的阶段。管理此构建过程是保护软件堆栈的关键。

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：



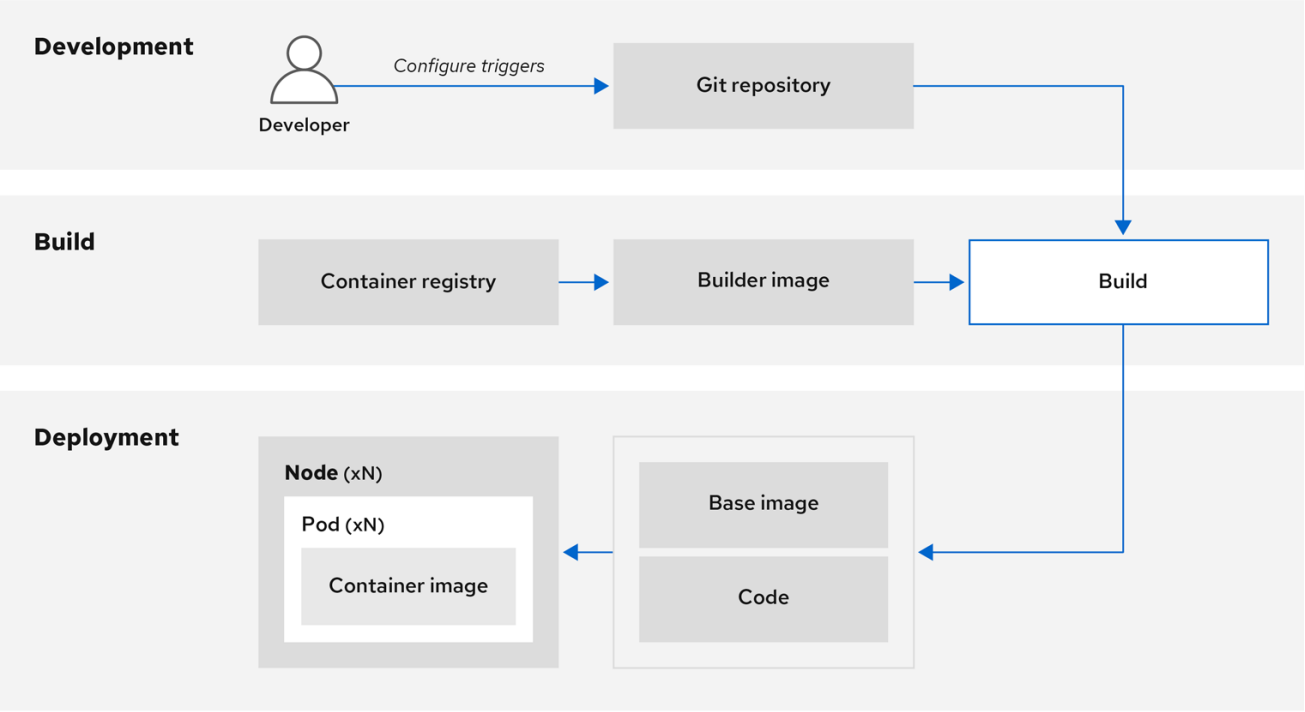
OpenShift Container Platform 可以与可信代码存储库（如 GitHub）和开发平台（如 Che）集成，用于创建和管理安全代码。单元测试可以依赖 Cucumber 和 JUnit。您可以通过 Anchore 或 Twistlock 检查容器中的漏洞和合规问题，并使用镜像扫描工具，如 AtomicScan 或 Clair。Sysdig 等工具可以提供对容器化应用程序的持续监控。

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：

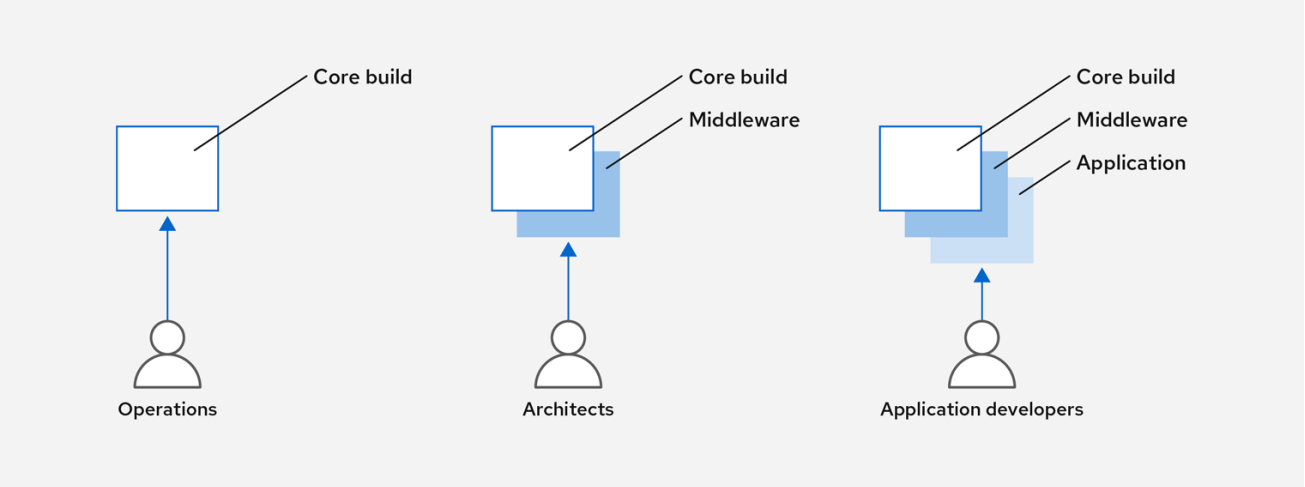
```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
  - destinationDir: .
    secret:
      name: secret-npmrc
```

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

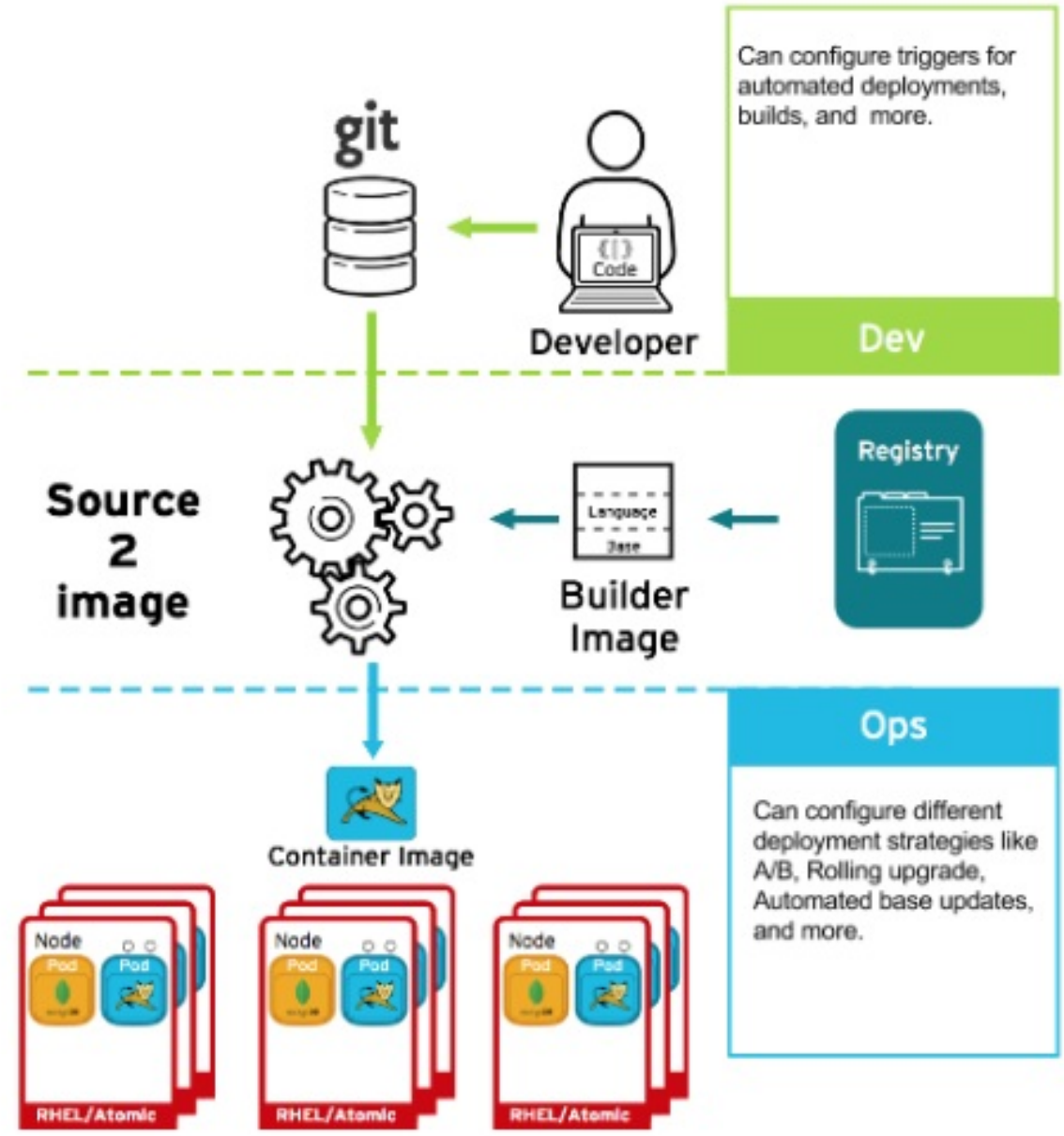
- 理解镜像构建
- 触发和修改构建
- 创建构建输入
- 输入 secret 和配置映射
- CI/CD 方法和实践
- 关于 OpenShift Serverless
- 使用 Topology 视图查看应用程序组成情况

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 Container Storage *
- 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**

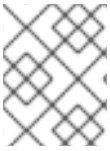
```
{
  "apiVersion": "v1",
  "count": 3,
  "involvedObject": {
    "kind": "Node",
    "name": "origin-node-1.example.local",
    "uid": "origin-node-1.example.local"
  },
  "kind": "Event",
  "reason": "NodeHasDiskPressure",
  ...
}
```

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 Cluster Logging Operator 部署、管理和升级集群日志记录。

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
- 了解集群日志记录
- 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。

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   4.6.0     True        False         False      145m
```

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

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。

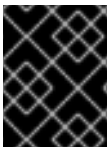
注意

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。



重要

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。

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

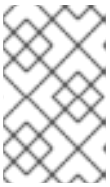
将这个注解添加到配置映射后，会删除其中的所有现有数据。建议您使用单独的配置映射来包含 **service-ca.crt**，而不是使用存储您的 Pod 配置的同一配置映射。

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：

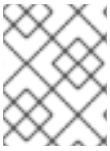
```
$ oc get crd <crd_name> -o yaml
```

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 Bundle 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，节点将重启。在下次引导过程中，服务 **coreos-update-ca-trust.service** 在 RHCOS 节点上运行，该节点上使用新的 CA 证书自动更新信任捆绑包。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
- machine-config-operator
- console-operator
- insights-operator
- machine-api-operator
- operator-lifecycle-manager

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; 证书来自由 bootstrap 过程生成的证书颁发机构 (CA)。安装集群后，节点证书会被自动轮转。

4.4.2. 管理

这些证书由系统而不是用户管理。

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 CA 有效时间为 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。

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. 管理

这些证书由系统而不是用户管理。

4.6.4. 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

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

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. 用户提供的默认入口证书

4.8.1. 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。

4.8.2. 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。

4.8.3. 管理

用户提供的证书由用户管理。

4.8.4. 过期

用户提供的证书由用户管理。

4.8.5. 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。

4.8.6. 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

4.9. 入口证书（INGRESS CERTIFICATE）

4.9.1. 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。

4.9.2. 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。

4.9.3. 工作流

图 4.1. **自定义证书工作流**

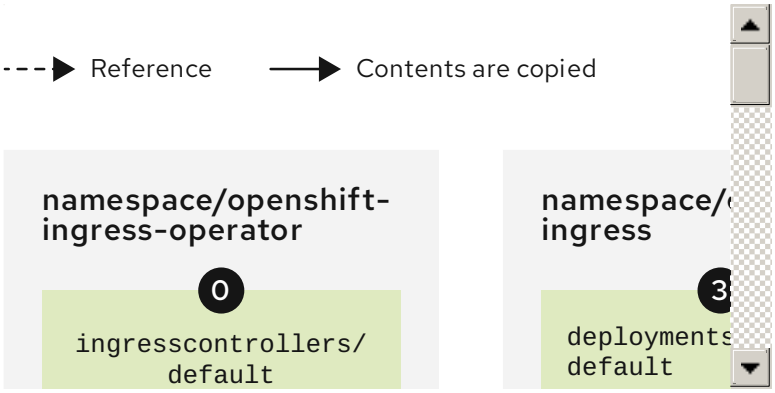
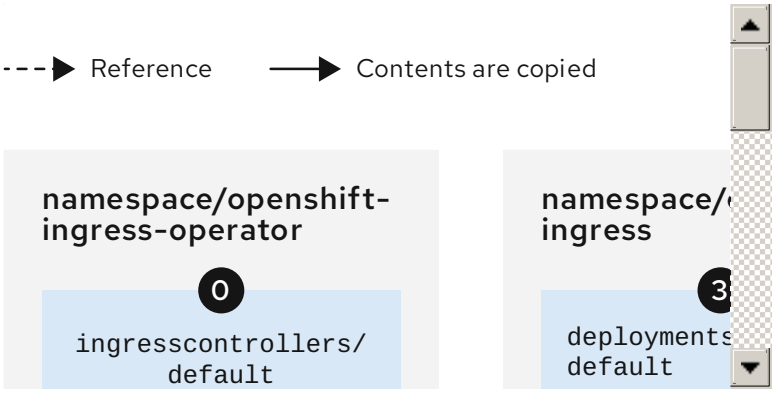


图 4.2. **默认证书工作流**



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 4.6 及更新版本使用 **default-ingress-cert**。

4.9.4. 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。

4.9.5. 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。

4.9.6. 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。

4.9.7. 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。

4.10. 监控和集群日志记录 OPERATOR 组件证书

4.10.1. 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。

4.10.2. 管理

这些证书由系统而不是用户管理。

4.11. CONTROL PLANE 证书

4.11.1. 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

4.11.2. 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 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。

5.1.1. 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

5.1.1.1. 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)

5.1.1.2. 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)

5.1.2. 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

5.1.2.1. 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。

5.1.2.2. 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)

5.1.2.3. 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)

5.1.3. 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

5.1.3.1. 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 {product-name} 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)

5.1.4. 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

5.1.4.1. 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)

5.1.5. 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

5.1.5.1. 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。

5.1.5.2. 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)

5.1.6. 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

5.1.6.1. 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。

5.1.6.2. 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。

5.1.6.3. 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。

5.1.7. 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

5.1.7.1. 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 配置集附带的合规性内容。
- 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。

5.1.8. 其他资源

- 了解 Compliance Operator

5.2. 支持的合规性配置集

有多个配置集可用于安装 Compliance Operator(CO)。

5.2.1. 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	Compliance Operator 版本	行业标准基准	支持的构架
ocp4-cis	CIS Red Hat OpenShift Container Platform 4 Benchmark	0.1.39+	CIS Benchmarks ™ footnote:cisbenchmark[To locate the CIS RedHat OpenShift Container Platform v4 Benchmarks，进入 CIS Benchmarks 并在搜索框中输入 **Kubernetes**。点 Kubernetes，然后 **Download Latest CIS Benchmark**，然后注册以下载基准。]	**x86_64 ppc64le s390x**

profile	配置集标题	Compliance Operator 版本	行业标准基准	支持的构架
ocp4-cis-node	CIS Red Hat OpenShift Container Platform 4 Benchmark	0.1.39+	CIS Benchmarks ™ footnote:cisbenchmark[]	**x86_64 ppc64le s390x**
ocp4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	0.1.39+	ACSC 强化 Linux 工作站和服务器	**x86_64**
ocp4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Platform 级别	0.1.39+	NIST SP-800-53 Release Search	**x86_64**
rhcos4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	0.1.39+	ACSC 强化 Linux 工作站和服务器	**x86_64**
rhcos4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	0.1.39+	NIST SP-800-53 Release Search	**x86_64**
ocp4-moderate-node	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - 节点级别	0.1.44+	NIST SP-800-53 Release Search	**x86_64**
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	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	0.1.44+	NERC CIP 标准	**x86_64**

profile	配置集标题	Compliance Operator 版本	行业标准基准	支持的构架
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	0.1.44+	NERC CIP 标准	**x86_64**
ocp4-pci-dss	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	0.1.47+	PCI 安全标准 ® Council 文档库	**x86_64**
ocp4-pci-dss-node	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	0.1.47+	PCI 安全标准 ® Council 文档库	**x86_64**
ocp4-high	NIST 800-53 HighImpact Baseline for Red Hat OpenShift - Platform 级别	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	0.1.52+	NIST SP-800-53 Release Search	**x86_64**
rhcos4-high	NIST 800-53 high-Impact Baseline for Red Hat Enterprise Linux CoreOS	0.1.52+	NIST SP-800-53 Release Search	**x86_64**

5.2.2. 其他资源

- 有关查看系统中可用的合规配置集的更多信息，请参阅了解 Compliance Operator 中的 Compliance Operator 配置集。

5.3. 安装 COMPLIANCE OPERATOR

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

5.3.1. 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。

5.3.2. 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"
     name: openshift-compliance
   ```

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
```

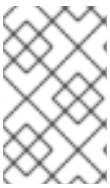
5. 定义一个 **Subscription** 对象：

subscription-object.yaml示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "release-0.1"
  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
```

5.3.3. 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. 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
```

5.4.1. 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**

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  pvAccessModes:
  - ReadWriteOnce 1
  rotation: 3 2
  size: 1Gi 3
roles:
- worker 4
- master 5
scanTolerations: 6
  default:
  - operator: Exists
  schedule: 0 1 * * * 7
```

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 默认扫描设置对象还会扫描所有节点。

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：

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default-auto-apply
  namespace: openshift-compliance
autoUpdateRemediations: true 1
autoApplyRemediations: true 2
rawResultStorage:
  pvAccessModes:
    - ReadWriteOnce
  rotation: 3
  size: 1Gi
schedule: 0 1 * * *
roles:
  - worker
  - master
scanTolerations:
  default:
  - operator: Exists
```

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。

5.4.2. 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：

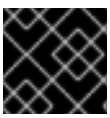
```
$ oc get scansettings rs-on-workers -n openshift-compliance -o yaml
```

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. 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。

5.5.1. 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 <namespace> profiles.compliance
```

本例在默认 **openshift-compliance** 命名空间中显示配置集：

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

输**出示例**

```
NAME                 AGE
ocp4-cis             32m
ocp4-cis-node        32m
ocp4-e8              32m
ocp4-moderate        32m
ocp4-moderate-node   32m
ocp4-nerc-cip        32m
ocp4-nerc-cip-node   32m
ocp4-pci-dss         32m
ocp4-pci-dss-node    32m
rhcos4-e8            32m
rhcos4-moderate      32m
rhcos4-nerc-cip      32m
```

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。

- 查看配置集的详细信息：

```
$ oc get -n <namespace> -oyaml profiles.compliance <profile name>
```

本例显示 **rhcos4-e8** 配置集的详情：

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

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: ...
  id: xccdf_org.ssgproject.content_profile_e8
  kind: Profile
  metadata:
    annotations:
      compliance.openshift.io/image-digest: pb-rhcos426smj
      compliance.openshift.io/product: redhat_enterprise_linux_coreos_4
      compliance.openshift.io/product-type: Node
    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
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
```

- 查看所需配置集中的规则：

```
$ oc get -n <namespace> -oyaml rules.compliance <rule_name>
```

本例在 **rhcos4** 配置集中显示了 **rhcos4-audit-rules-login-events** 规则：

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

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-rhcos426smj
      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
    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
  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.
```

5.6. 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。

5.6.1. 更新安全性内容

安全性内容作为 **ProfileBundle** 对象引用的容器镜像提供。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：

输**出示例**

```
  apiVersion: compliance.openshift.io/v1alpha1
  kind: ProfileBundle
  metadata:
    name: rhcos4
  spec:
    contentImage: quay.io/user/ocp4-openscap-
content@sha256:a1749f5150b19a9560a5732fe48a89f07bffc79c0832aa8c49ee5504590ae687 (1)
    contentFile: ssg-rhcos4-ds.xml
```

(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。

5.6.2. 使用镜像流

contentImage 引用指向一个有效的 **ImageStreamTag**，而 Compliance Operator 可确保内容自动更新。

注意

ProfileBundle 对象还接受 **ImageStream** 引用。

镜**像流示例**

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

输**出示例**

```
NAME            IMAGE REPOSITORY                                                TAGS
```

```
UPDATED
openscap-ocp4-ds   image-registry.openshift-image-registry.svc:5000/openshift-
compliance/openscap-ocp4-ds   latest   32 seconds ago
```

流程

1. 确保将查询策略设置为 local：

```
$ oc patch is openscap-ocp4-ds \
    -p '{"spec":{"lookupPolicy":{"local":true}}}' \
    --type=merge
    imagestream.image.openshift.io/openscap-ocp4-ds patched
    -n openshift-compliance
```

2. 通过检索 **istag** 名称，对 **ProfileBundle** 使用 **ImageStreamTag** 的名称：

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

输**出示例**

```
NAME                  IMAGE REFERENCE
UPDATED
openscap-ocp4-ds:latest   image-registry.openshift-image-registry.svc:5000/openshift-
compliance/openscap-ocp4-
ds@sha256:46d7ca9b7055fe56ade818ec3e62882cfcc2d27b9bf0d1cbae9f4b6df2710c96   3
minutes ago
```

3. 创建 **ProfileBundle**：

```
$ cat << EOF | oc create -f -
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  name: mybundle
   spec:
     contentImage: openscap-ocp4-ds:latest
     contentFile: ssg-rhcos4-ds.xml
EOF
```

这个 **ProfileBundle** 将跟踪镜像，对其应用的所有更改，如更新标签以指向其他散列，将立即反映在 **ProfileBundle** 中。

5.6.3. ProfileBundle CR 示例

捆绑包对象需要两条信息：包含 **contentImage** 的容器镜像的 URL 和包含合规性内容的文件。**contentFile** 参数相对于文件系统的根目录。内置的 **rhcos4 ProfileBundle** 对象可以在以下示例中定义：

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  name: rhcos4
```

```
spec:
  contentImage: quay.io/complianceascode/ocp4:latest (1)
  contentFile: ssg-rhcos4-ds.xml (2)
```

(1) 内容镜像位置。

(2) 包含合规性内容的文件位置。

重要

用于内容镜像的基础镜像必须包含 **coreutils**。

5.6.4. 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。

5.7. 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 提供一个可轻松定制配置集的对象，称为 **TailoredProfile**。它假设您要扩展预先存在的配置集，并允许您启用和禁用来自 **ProfileBundle** 的规则和值。

注意

您将只能使用可作为您要扩展的配置集所属的 **ProfileBundle** 一部分的规则和变量。

5.7.1. 使用定制配置集

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. 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 5.2. 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。
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. 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
```

5.8. 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。

5.8.1. 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 \
    | jq '.status.scanStatuses[].resultsStorage'
    {
      "name": "rhcos4-moderate-worker",
      "namespace": "openshift-compliance"
    }
    {
      "name": "rhcos4-moderate-master",
      "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：

Pod 示例

```
apiVersion: "v1"
kind: Pod
metadata:
  name: pv-extract
spec:
  containers:
    - name: pv-extract-pod
      image: registry.access.redhat.com/ubi8/ubi
      command: ["sleep", "3000"]
      volumeMounts:
      - mountPath: "/workers-scan-results"
        name: workers-scan-vol
  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 .
```

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
```

5.9. 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。

5.9.1. 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 compliancecheckresults -l compliance.openshift.io/suite=example-compliancesuite
```

列出属于特定扫描的检查：

```
$ oc get compliancecheckresults -l compliance.openshift.io/scan=example-compliancescan
```

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 compliancecheckresults -l 'compliance.openshift.io/check-
status=FAIL,compliance.openshift.io/automated-remediation'
```

列出必须手动修复的所有失败检查：

```
$ oc get 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 5.3. 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。
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。

ComplianceCheckResult 状态	描述
NOT-APPLICABLE	合规检查未运行，因为它不适用或未选中。

5.9.2. 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: 3.1.0
        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
```

5.9.3. 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
```

输**出示例**

```
NAME                                         STATUS  ROLES  AGE    VERSION
ip-10-0-128-92.us-east-2.compute.internal  Ready   master 5h21m  v1.23.3+d99c04f
ip-10-0-158-32.us-east-2.compute.internal  Ready   worker 5h17m  v1.23.3+d99c04f
ip-10-0-166-81.us-east-2.compute.internal  Ready   worker 5h17m  v1.23.3+d99c04f
ip-10-0-171-170.us-east-2.compute.internal Ready   master 5h21m  v1.23.3+d99c04f
ip-10-0-197-35.us-east-2.compute.internal  Ready   master 5h22m  v1.23.3+d99c04f
```

2. 为节点添加标签。

```
$ oc 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
```

5.9.4. 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 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。

5.9.5. 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 annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

5.9.6. 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 get complianceremediations -lcomplianceoperator.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 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 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。

5.9.7. 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 patch complianceremediations/<scan_name>-sysctl-net-ipv4-conf-all-accept-redirects -p
'{"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。

5.9.8. 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 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 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 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 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。

5.9.9. 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 annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

5.9.10. 其他资源

- 修改节点。

5.10. 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。

5.10.1. 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: quay.io/complianceascode/ocp4:latest
      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。

5.10.2. 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 create configmap <scan_name> --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: quay.io/complianceascode/ocp4:latest
      debug: true
  tailoringConfigMap:
      name: <scan_name>
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

5.10.3. 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 annotate compliancescans/<scan_name> 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。

5.10.4. 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。

5.10.4.1. 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 * * *'
```

5.10.5. 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 annotate compliancesuites/<suite-_name> compliance.openshift.io/apply-remediations=
```

5.10.6. 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 annotate compliancesuites/<suite_name> 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。

5.11. 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 compliancescan/<scan_name>
  ```

- 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 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。

5.11.1. 扫描剖析

以下各节概述了 Compliance Operator 扫描的组件和阶段。

5.11.1.1. 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 profilebundle.compliance
```

```
$ oc get 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 -lprofile-bundle=ocp4 -c profileparser
```

```
$ oc get deployments,pods -lprofile-bundle=ocp4
```

```
$ oc logs pods/<pod-name>
```

```
$ oc describe pod/<pod-name> -c profileparser
```

5.11.1.2. 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**。

5.11.1.3. 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。

5.11.1.4. 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：

5.11.1.4.1. 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。

5.11.1.4.2. 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 get cm -lcompliance.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 -lcompliance.openshift.io/scan-name=<scan_name>
```

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
At this point, the scan proceeds to the Running phase.
```

5.11.1.4.3. 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=<scan_name>**）标签：

  ```
  $ 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。

5.11.1.4.4. 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。

5.11.1.4.5. 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 annotate compliancescans/<scan_name> 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。

5.11.1.5. 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                           2.2.0
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 annotate compliancescans/<scan_name> compliance.openshift.io/rescan=
```

扫描将运行并完成。检查补救是否通过：

```
$ oc 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
```

5.11.1.6. 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>
```

5.11.2. 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。

5.12. 卸载 COMPLIANCE OPERATOR

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

5.12.1. 从 OpenShift Container Platform 卸载 OpenShift Compliance Operator

要删除 Compliance Operator，首先需要删除 Compliance Operator 自定义资源定义(CRD)。删除 CRD 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. 删除 Compliance Operator 安装的 CRD：
   a. 切换到 **Administration → Custom Resource Definitions** 页面。
   b. 在 **Name** 字段中搜索 **compliance.openshift.io**。
   c. 点击以下 CRD  旁边的 Options 菜单，然后选择 **Delete Custom Resource Definition**
      - **ComplianceCheckResult**
      - **ComplianceRemediation**
      - **ComplianceScan**
      - **ComplianceSuite**
      - **ProfileBundle**
      - **profile**
      - **规则**
      - **ScanSettingBinding**
      - **ScanSetting**
      - **TailoredProfile**
      - **变量**
2. 删除 OpenShift Compliance 项目：
   a. 切换到 **Home → Projects** 页面。
   b. 点 **openshift-compliance** 项目  旁边的 Options 菜单，然后选择 **Delete Project**。

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

5.13. 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。

5.13.1. 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

5.13.2. 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。

5.13.2.1. ProfileBundle 对象

安装 Compliance Operator 时，它包含了 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。

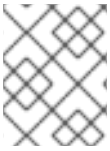
ProfileBundle 对象示例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
  name: <profile bundle name>
  namespace: openshift-compliance
spec:
  contentFile: ssg-ocp4-ds.xml (1)
  contentImage: quay.io/complianceascode/ocp4:latest (2)
status:
  dataStreamStatus: VALID (3)
```

(1) 指定配置文件所在的根目录(/)的路径。

(2) 指定封装配置集文件的容器镜像。

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



注意

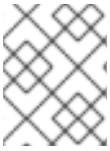
当 **contentFile** 失败时，会出现一个 **errorMessage** 属性，它提供所发生错误的详细信息。

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。

5.13.2.2. 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。

5.13.2.3. 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。

5.13.2.4. 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: <scan type>
  ```

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。

5.13.3. 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。

5.13.3.1. ScanSetting 对象

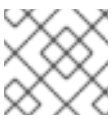
使用 **ScanSetting** 对象定义并重用操作策略来运行扫描。默认情况下，Compliance Operator 会创建以下 **ScanSetting** 对象：

- **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
kind: ScanSetting
metadata:
  name: <name of the scan>
autoApplyRemediations: false 1
autoUpdateRemediations: false 2
schedule: "0 1 * * *" 3
rawResultStorage:
  size: "2Gi" 4
  rotation: 10 5
roles: 6
  - worker
  - master
```

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. 指定扫描应以 cron 格式运行的频率。
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**
5. 指定存储原始结果的扫描量。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。



注意

要禁用轮转策略，请将值设为 **0**。

6 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。

5.13.4. 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。

5.13.4.1. 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。

5.13.5. 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。

5.13.5.1. 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。

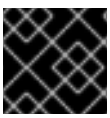
ComplianceSuite 对象示例

```
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: quay.io/complianceascode/ocp4:latest
      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。

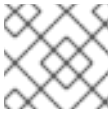
5.13.5.2. 高级 ComplianceScan 对象

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: quay.io/complianceascode/ocp4:latest 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
```

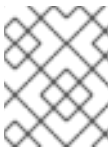
1. 指定 **Node** 或 **Platform**。节点配置集扫描集群节点和平台配置集扫描 Kubernetes 平台。
2. 指定您要运行的配置集的 XCCDF 标识符。
3. 指定封装配置集文件的容器镜像。
4. 它是可选的。指定要运行单个规则的扫描。该规则必须使用 XCCDF ID 标识，并且必须属于指定的配置集。



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。

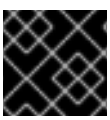
5. 如果您在 OpenShift Container Platform 上，并希望生成补救，则 nodeSelector 标签必须与 **MachineConfigPool** 标签匹配。



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>
```

5.13.6. 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。

5.13.6.1. 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=<suit name>
```

5.13.6.2. 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=<suite name>
```

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'
```

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。

6.1.1. 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

6.1.1.1. 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)

6.1.2. 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

6.1.2.1. 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。

6.1.2.2. 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)

6.1.3. 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

6.1.3.1. 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)

6.1.4. 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

6.1.4.1. 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。

6.1.4.2. 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。

6.1.5. 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"
  name: openshift-file-integrity
```

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: "release-0.1"
  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. 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。

6.3.1. 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:
     node-role.kubernetes.io/worker: ""
  config: {}
```

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
```

6.3.2. 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
```

6.3.3. 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。

6.3.4. 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

可能需要过些时间，**FileIntegrityNodeStatus** 对象的结果才会可用。

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
```

6.3.5. 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。

6.3.5.1. 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。

6.3.5.2. 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+a8yTgHPnIHMaRlS+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。

6.3.6. 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
```

6.4. 配置自定义 FILE INTEGRITY OPERATOR

6.4.1. 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
```

6.4.2. 重要属性

表 6.1. 重要的 spec 和 spec.config 属性

属性	描述

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，则会应用默认的容限，允许容限在 control plane 节点上运行（也称为 master 节点）。
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 **重新init** 进程中保留 AIDE 数据库和日志备份的最大数量，以保持在节点上。除了这个数量外，旧的备份会被守护进程自动修剪。
spec.config.name	包含自定义 AIDE 配置的 configMap 名称。如果省略，则创建一个默认配置。
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 **名称和命名空间 指定的配置映射中包含实际 AIDE 配置的密钥**。默认值为 **aide.conf**。

6.4.3. 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
```

6.4.4. 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 的排除项

6.4.5. 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。

6.4.6. 定义自定义 File Integrity Operator 配置

本例重点根据为 **worker-fileintegrity** CR 提供的默认配置，为 control plane 节点（也称为 master 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
```

6.4.7. 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。

6.5. 执行高级自定义 FILE INTEGRITY OPERATOR 任务

6.5.1. 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。

6.5.2. 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。

6.5.3. 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。

6.6. 对 FILE INTEGRITY OPERATOR 进行故障排除

6.6.1. 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。

6.6.2. 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**。

6.6.3. 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。

6.6.4. 确定守护进程集的 Pod 是否在预期节点上运行

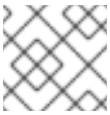
问题

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 pod 正在运行的节点的 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 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。

7.1. 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 7.1. 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\""}}

7.2. 查看审计日志

您可以查看每个 control plane 节点（也称为主节点）的 OpenShift API 服务器、Kubernetes API 服务器和 OpenShift OAuth API 服务器的日志。

流程

查看审计日志：

- 查看 OpenShift API 服务器日志：
  a. 列出每个 control plane 节点可用的 OpenShift API 服务器日志：

```
$ 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 服务器日志：

a. 列出每个 control plane 节点可用的 Kubernetes API 服务器日志：

```
$ 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 服务器日志：

  a. 列出每个 control plane 节点可用的 OpenShift OAuth API 服务器日志：

  ```
  $ 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\""}}
```

7.3. 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")'
```

7.4. 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** 标志运行 **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. 在红帽客户门户中为您的问题单附上压缩文件。

7.5. 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
- 将日志转发到第三方系统

第 8 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。

8.1. 关于审计日志策略配置集

审计日志配置集定义了如何记录 OpenShift API 服务器、Kubernetes API 服务器和 OAuth API 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
默认	仅用于读取和写入请求的日志元数据；除了 OAuth 访问令牌创建（login）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，同时记录对 API 服务器的写入请求（**create**、**update**、**patch**）的具体数据（body）。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]

1. 敏感资源，如 **Secret**、**Route** 和 **OAuthClient** 对象，永远不会记录在元数据级别上。如果您的集群是从 OpenShift Container Platform 4.5 升级的，则 OAuth 令牌不会被记录，因为它们的对象名称可能包含保密信息。

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（CPU、内存和 I/O）。

8.2. 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** 或 **AllRequestBodies**。默认配置集为 **Default**。

3. 保存文件以使改变生效。

4. 验证是否已推出 Kubernetes API 服务器 pod 的新修订版本。这需要几分钟时间。

```
$ 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如下，这意味着更新仍在进行中。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 9 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、OpenShift API 服务器、OpenShift OAuth API 服务器和 OpenShift OAuth 服务器。

9.1. 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 9.1. 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 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。 **注意** 在 OpenShift Container Platform 4.6、4.7 和 4.8 中，**Modern** 配置集不被支持。如果选择，则启用 **Intermediate** 配置集。 **重要** **Modern** 配置集当前不受支持。

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** OpenShift Container Platform router 会启用由红帽提供的 OpenSSL 默认 TLS **1.3** 加密套件。您的集群可能会接受 TLS **1.3** 连接和密码套件，即使 OpenShift Container Platform 4.6、4.7 和 4.8 不支持 TLS 1.3。

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。

9.2. 查看 TLS 安全配置集详情

您可以查看以下每个组件的预定义 TLS 安全配置集的最低 TLS 版本和密码： Ingress Controller 和 control plane。

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>**，请指定 **ingresscontroller** 或 **apiserver**。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>**，请指定 **ingresscontroller** 或 **apiserver**。

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。

9.3. 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。

重要

HAProxy Ingress Controller 镜像不支持 TLS **1.3**，因为 **Modern** 配置集需要 TLS **1.3**，因此不支持它。Ingress Operator 会将 **Modern** 配置集转换为 **Intermediate**。Ingress Operator 还会将 **Old** 或 **Custom** 配置集的 TLS **1.0** 转换为 **1.1**，将**Custom** 配置集的 TLS **1.3** 转换为 **1.2**。

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
 ...
```

9.4. 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
- OpenShift API 服务器
- OpenShift OAuth API 服务器
- OpenShift OAuth 服务器

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
 ...
```

第 10 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。seccomp、安全计算模式是一种 Linux 内核功能，可用于将容器中运行的进程限制为仅调用可用系统调用的一个子集。可以通过创建应用到容器或 pod 的配置集来配置这些系统调用。seccomp 配置集作为 JSON 文件存储在磁盘上。

重要

OpenShift 工作负载默认运行 unconfined，不应用任何 seccomp 配置集。

重要

seccomp 配置集不能应用到特权容器。

10.1. 为所有 POD 启用默认 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**。您可以通过创建自定义安全性上下文约束(SCC)，为 pod 或容器工作负载启用默认的 seccomp 配置集。

注意

需要创建自定义 SCC。不要编辑默认 SCC。编辑默认 SCC 可能会导致升级某些平台 Pod 或 OpenShift Container Platform 时出现问题。如需更多信息，请参阅标题为"默认安全性上下文约束"部分。

按照以下步骤为所有 pod 启用默认 seccomp 配置集：

1. 将可用的 **受限** SCC 导出至 yaml 文件中：

```
$ oc get scc restricted -o yaml > restricted-seccomp.yaml
```

2. 编辑创建的 **restricted** SCC yaml 文件：

```
$ vi restricted-seccomp.yaml
```

3. 如本例所示更新：

```
kind: SecurityContextConstraints
metadata:
  name: restricted 1
<..snip..>
seccompProfiles:  2
- runtime/default  3
```

1. 进入 **restricted-seccomp**
2. 添加 **seccompProfile：**
3. 添加 **- 运行时/默认**

4. 创建自定义 SCC：

```
$ oc create -f restricted-seccomp.yaml
```

预**期**输**出**

```
securitycontextconstraints.security.openshift.io/restricted-seccomp created
```

5. 将自定义 SCC 添加到 ServiceAccount：

```
$ oc adm policy add-scc-to-user restricted-seccomp -z default
```

注意

默认服务帐户是应用的 ServiceAccount，除非用户配置不同的对象。OpenShift Container Platform 根据 SCC 中的信息配置 pod 的 seccomp 配置集。

预**期**输**出**

```
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:restricted-seccomp added: "default"
```

在 OpenShift Container Platform 4.6 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。

10.2. 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。

10.2.1. 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。

10.2.2. 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 4.6 中已弃用。

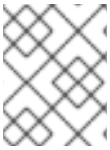
在部署过程中，准入控制器会验证以下内容：

- 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。

10.3. 其他资源

- 管理安全性上下文约束
- 安装后机器配置任务

第 11 章 允许从其他主机对 API 服务器进行基于 JAVASCRIPT 的访问

11.1. 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 配置仅允许 OpenShift 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 12 章 加密 ETCD 数据

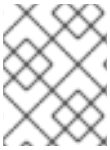
12.1. 关于 ETCD 加密

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。这些密钥会每周进行轮转。您必须有这些密钥才能从 etcd 备份中恢复数据。



注意

请记住，etcd 仅对值进行加密，而不对键进行加密。这意味着资源类型、命名空间和对象名称是不加密的。

12.2. 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

不建议在初始加密过程完成前备份 etcd。如果加密过程还没有完成，则备份可能只被部分加密。

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. 把 **encryption** 项类型设置为 **aescbc**：

```
spec:
  encryption:
    type: aescbc (1)
```

(1) **aescbc** 类型表示 AES-CBC 使用 PKCS#7 padding 和 32 字节密钥来执行加密。

3. 保存文件以使改变生效。
   加密过程开始。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 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, oauthaccesstokens.oauth.openshift.io,
   oauthauthorizetokens.oauth.openshift.io
   ```

   如果输出显示 **EncryptionInProgress**，这意味着加密仍在进行中。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**，这意味着加密仍在进行中。等待几分钟后重试。

12.3. 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**，这意味着解密仍在进行中。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**，这意味着解密仍在进行中。等待几分钟后重试。

第 13 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 集群中的所有命名空间都可以使用它。

13.1. 运行 RED HAT QUAY CONTAINER SECURITY OPERATOR

您可以通过从 Operator Hub 选择并安装该 Operator，从 OpenShift Container Platform Web 控制台启动 Red Hat Quay Container Security Operator，如下所述。

先决条件

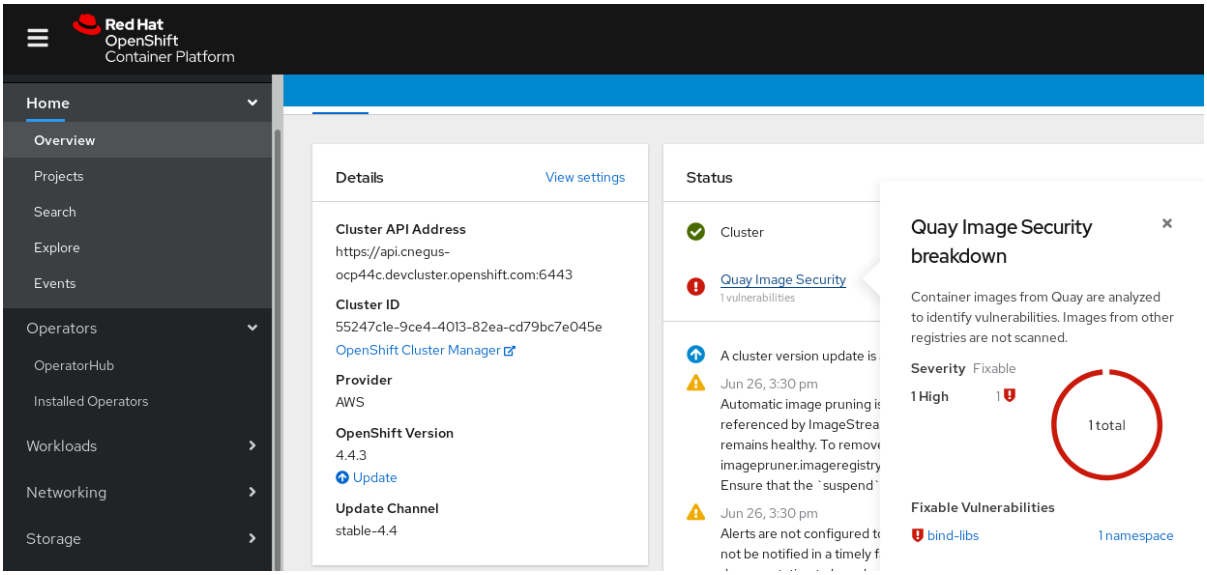
- 具有 OpenShift Container Platform 集群的管理员特权
- 来自集群中运行的 Red Hat Quay 或 Quay.io registry 的容器

流程

1. 导航到 **Operators → OperatorHub** 并选择 **Security**。
2. 选择 **Container Security** Operator，然后选择 **Install** 进入 Create Operator Subscription 页面。
3. 检查设置。所有命名空间和自动批准策略都被默认选择。
4. 选择 **Install**。在 **Installed Operators** 屏幕中几分钟后会出现 **Container Security** Operator。
5. 可选：您可以将自定义证书添加到 Red Hat Quay Container Security Operator。在本例中，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：

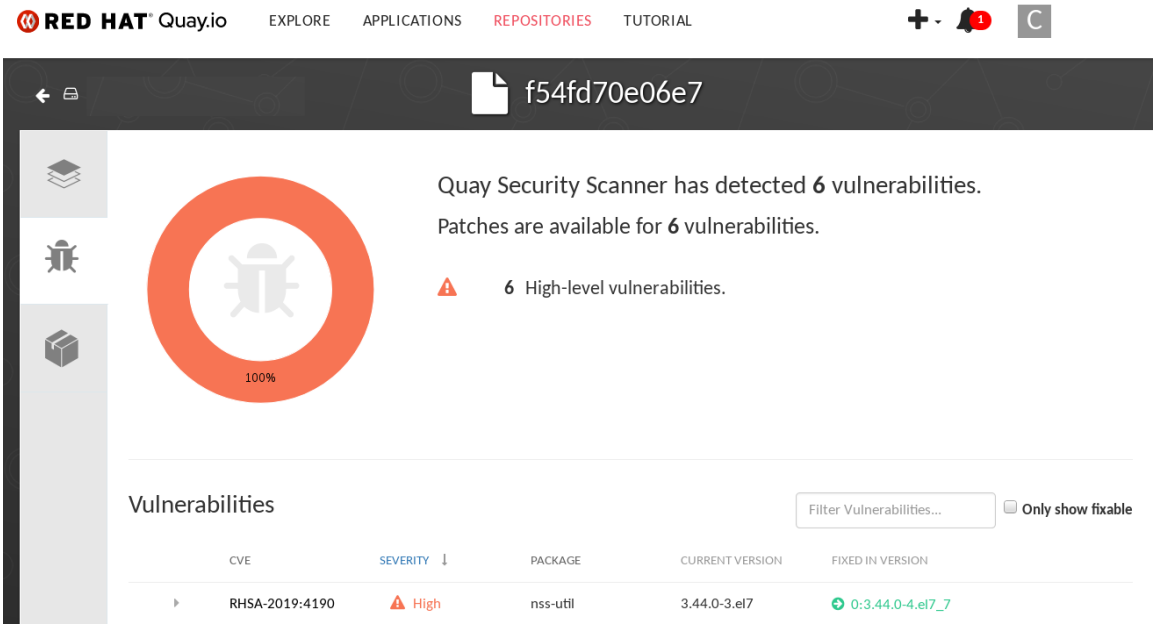
   ```
   $ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n openshift-operators
   ```

6. 如果添加了自定义证书，请重启 Operator Pod 以使新证书生效。
7. 打开 OpenShift Dashboard（**Home → Overview**）。至 **Quay Image Security** 的链接会出现在 status 部分，其中列出了目前发现的漏洞数量。选择该链接以查看 **Quay 镜像安全分**类，如下图所示：

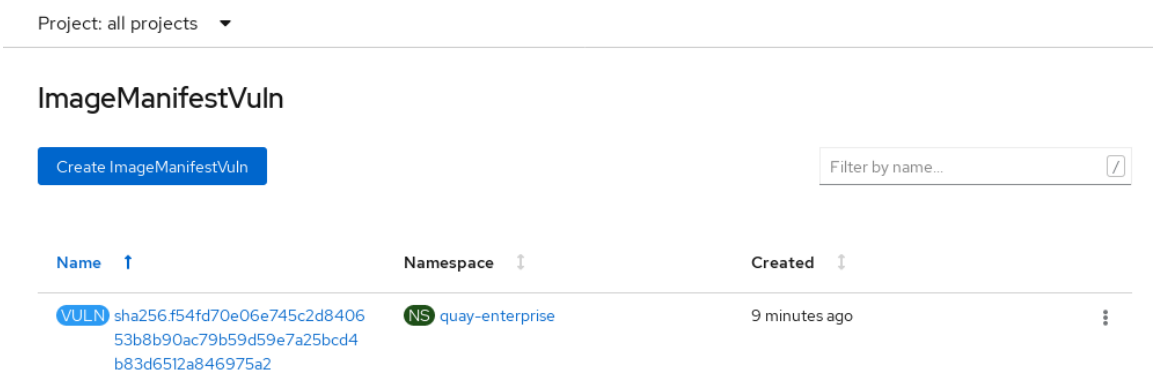


8. 对于任何检测到的安全漏洞，您可以在此时进行两个操作之一：

- 选择到这个漏洞的链接。您会进入容器来自的容器 registry，在那里查看有关该漏洞的信息。下图显示了从 Quay.io registry 中检测到的漏洞示例：



- 选择 namespaces 链接以进入 **ImageManifestVuln** 界面，您可以在其中查看所选镜像的名称以及该镜像正在运行的所有命名空间。下图表示，一个存在特定漏洞的镜像正在 **quay-enterprise** 命名空间中运行：



此时，您知道哪些镜像存在这个安全漏洞，需要做什么来修复这些漏洞，以及镜像在中运行的每个命名空间。您可以：

- 警告运行镜像的用户需要修正这个漏洞
- 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

请注意，如果您删除 pod，可能需要几分钟时间才能在仪表板上重置漏洞。

13.2. 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

- 在 OpenShift Container Platform 实例上运行 Red Hat Quay Container Security Operator

流程

- 要查询检测到的容器镜像漏洞，请输入：

```
$ oc get vuln --all-namespaces
```

输**出示例**

```
NAMESPACE     NAME             AGE
default       sha256.ca90...   6m56s
skynet        sha256.ca90...   9m37s
```

- 要显示特定漏洞的详情，在 **oc describe** 命令中提供漏洞名称及其命名空间。本例演示了一个活跃的容器，其镜像包含存在漏洞的 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...
```