OpenShift Container Platform 4.4

安全性

了解和管理 OpenShift Container Platform 的安全性

Last Updated: 2021-03-11

OpenShift Container Platform 4.4 安全性

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 容器安全性 3
1.1. 了解容器安全性 3
1.2. 了解主机和虚拟机安全性 4
1.3. 强化 RHCOS 6
1.4. 了解合规性 8
1.5. 保护容器内容 8
1.6. 安全地使用容器 REGISTRY 13
1.7. 保护构建过程 15
1.8. 部署容器 19
1.9. 保护容器平台 23
1.10. 保护网络 26
1.11. 保护附加存储 27
1.12. 监控集群事件和日志 28

第 2 章 配置证书 31
2.1. 替换默认入口证书 31
2.2. 添加 API 服务器证书 32
2.3. 使用服务提供的证书 SECRET 保护服务流量 33

第 3 章 证书类型和描述 41
3.1. API 服务器的用户提供的证书 41
3.2. 代理证书 41
3.3. 服务 CA 证书 44
3.4. 节点证书 46
3.5. BOOTSTRAP 证书 46
3.6. ETCD 证书 46
3.7. OLM 证书 47
3.8. 用户提供的默认入口证书 47
3.9. 入口证书（INGRESS CERTIFICATE） 48
3.10. 监控和集群日志记录 OPERATOR 组件证书 51
3.11. CONTROL PLANE 证书 51

第 4 章 查看审计日志 52
4.1. 关于 API 审计日志 52
4.2. 查看审计日志 53

第 5 章 允许从其他主机对 API 服务器进行基于 JAVASCRIPT 的访问 56
5.1. 允许从其他主机对 API 服务器进行基于 JAVASCRIPT 的访问 56

第 6 章 加密 ETCD 数据 58
6.1. 关于 ETCD 加密 58
6.2. 启用 ETCD 加密 58
6.3. 禁用 ETCD 加密 59

第 7 章 对 POD 进行安全漏洞扫描 61
7.1. 运行 CONTAINER SECURITY OPERATOR 61
7.2. 通过 CLI 查询镜像安全漏洞 63

第 1 章 容器安全性

1.1. 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。
- 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。

1.1.1. 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。

1.1.2. 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

1.2. 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。

1.2.1. 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 4.4 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)。

- *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

1.2.2. 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。

1.2.3. 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。

1.3. 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。

1.3.1. 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。

1.3.2. 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 master 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。

1.3.2.1. 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 SELinux 或各种低级别设置，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。

1.3.2.2. 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。

1.3.2.3. 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

1.4. 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。

1.4.1. 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。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

1.5. 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。

1.5.1. 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 Container Security Operator 来检查 OpenShift Container Platform 中使用的容器镜像。

1.5.2. 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。

1.5.3. 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。

1.5.3.1. 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。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。

1.5.4. 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。

1.5.4.1. 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 1.1. 注解键格式

组件	描述	可接受值
qualityType	元数据类型	**vulnerability** **license** **operations** **policy**
providerId	供应商 ID 字符串	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

1.5.4.1.1. 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 1.2. 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 1.3. 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

1.5.4.1.2. 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 }
  ]
}
```

本例演示了一个镜像的 Red Hat Container Catalog 注解，带有健康状态索引数据以及指向更多详情的一个外部 URL：

Red Hat Container Catalog 注解

```
{
  "name": "Red Hat Container 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 }
  ]
}
```

1.5.4.2. 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。

1.5.4.2.1. 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 Container 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 } ]" }'
```

1.5.4.3. 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。

1.5.4.3.1. 注解示例

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

1.5.4.4. 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 4.4 REST API 来编写注解。有关 REST API 的常规信息，请参阅 OpenShift Container Platform REST API。

1.5.4.4.1. 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>:8443/oapi/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 Container 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}] }"
    }
  }
}
```

其它资源

- 镜像流对象

1.6. 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。

1.6.1. 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。

1.6.2. 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。

1.6.3. 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，红帽容器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 Container 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。

1.6.4. 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

1.6.5. 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。Quay Container Security Operator 可让您检查在 OpenShift Container Platform 中运行并从 Red Hat Quay registry 拉取的镜像的漏洞。

1.7. 保护构建过程

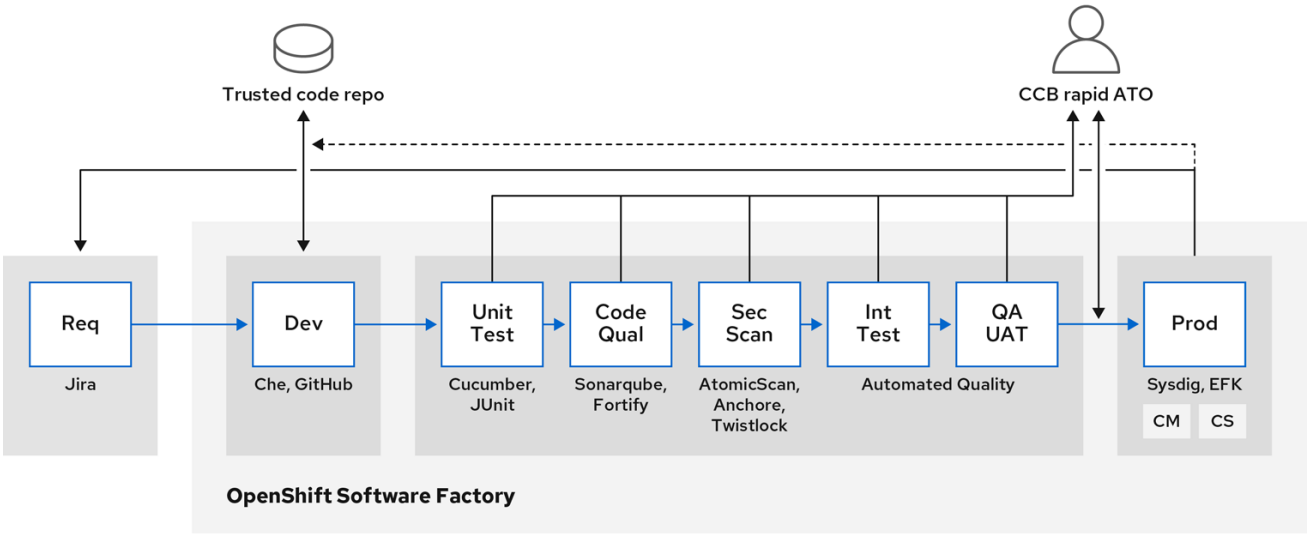
在容器环境中，软件构建过程是生命周期中应用程序代码与所需的运行时库集成的阶段。管理此构建过程是保护软件堆栈的关键。

1.7.1. 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：



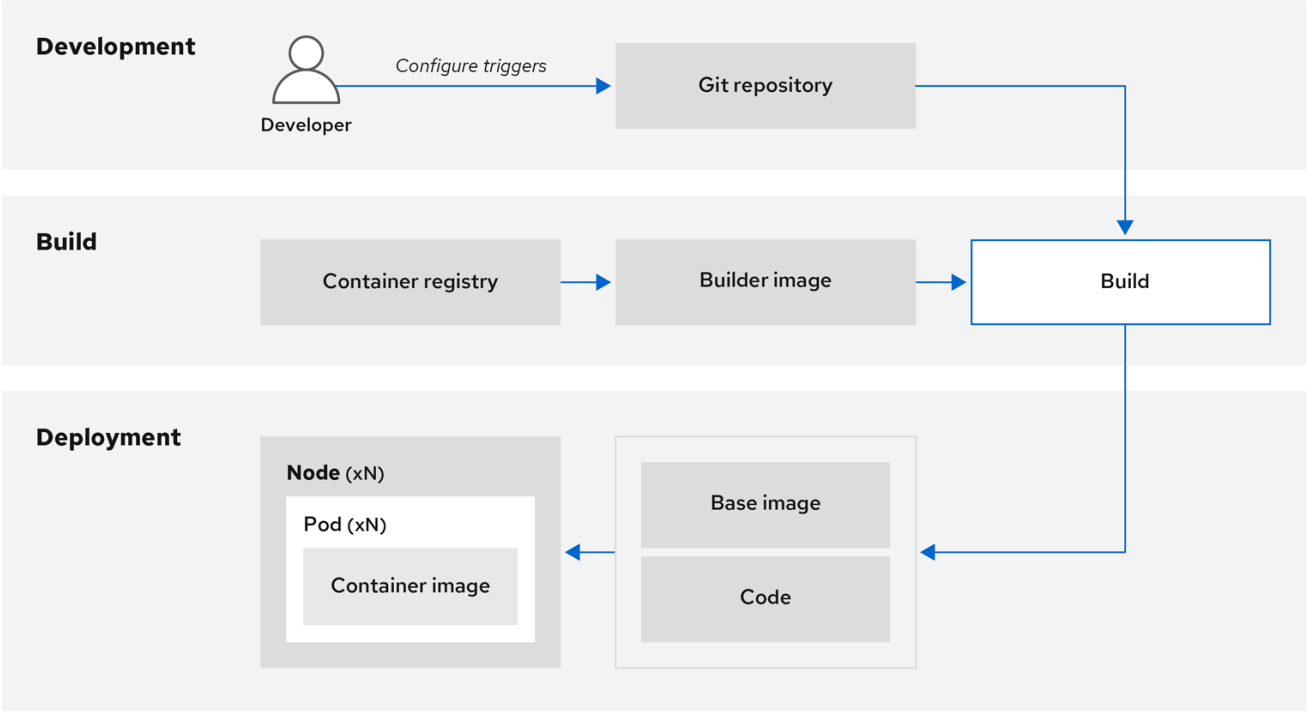
OpenShift Container Platform 可以与可信代码存储库（如 GitHub）和开发平台（如 Che）集成，用于创建和管理安全代码。单元测试可以依赖 Cucumber 和 JUnit。您可以通过 Anchore 或 Twistlock 检查容器中的漏洞和合规问题，并使用镜像扫描工具，如 AtomicScan 或 Clair。Sysdig 等工具可以提供对容器化应用程序的持续监控。

1.7.2. 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。

1.7.3. 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：

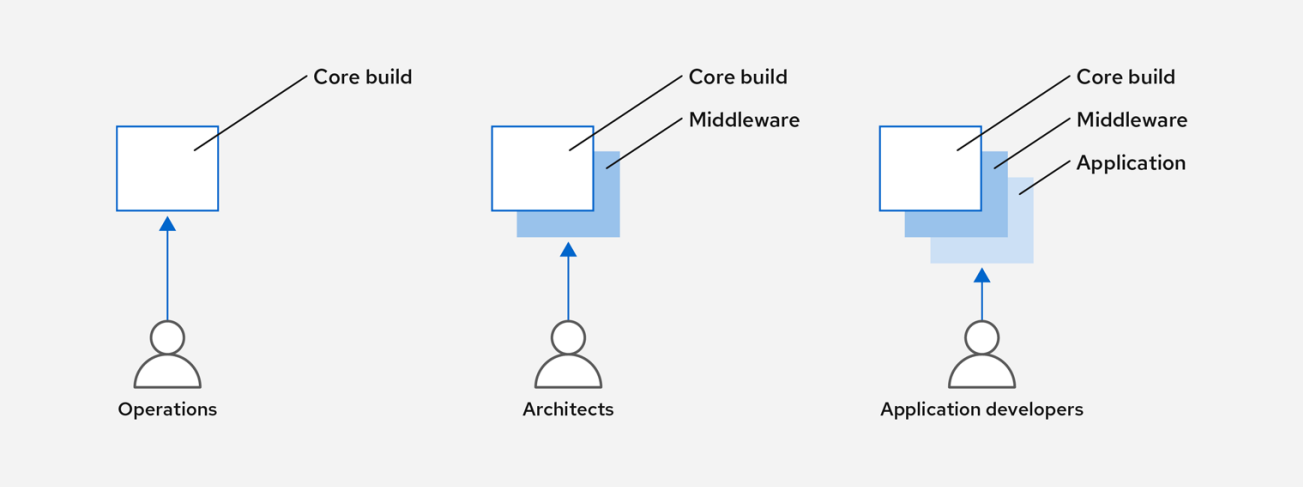
```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
  - destinationDir: .
    secret:
      name: secret-npmrc
```

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
```

1.7.4. 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。

1.7.5. 构建 Knative 无服务器应用程序

借助 Kubernetes 和 Kourier，您可以在 OpenShift Container Platform 中使用 Knative 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。

其他资源

- 理解镜像构建
- 触发和修改构建

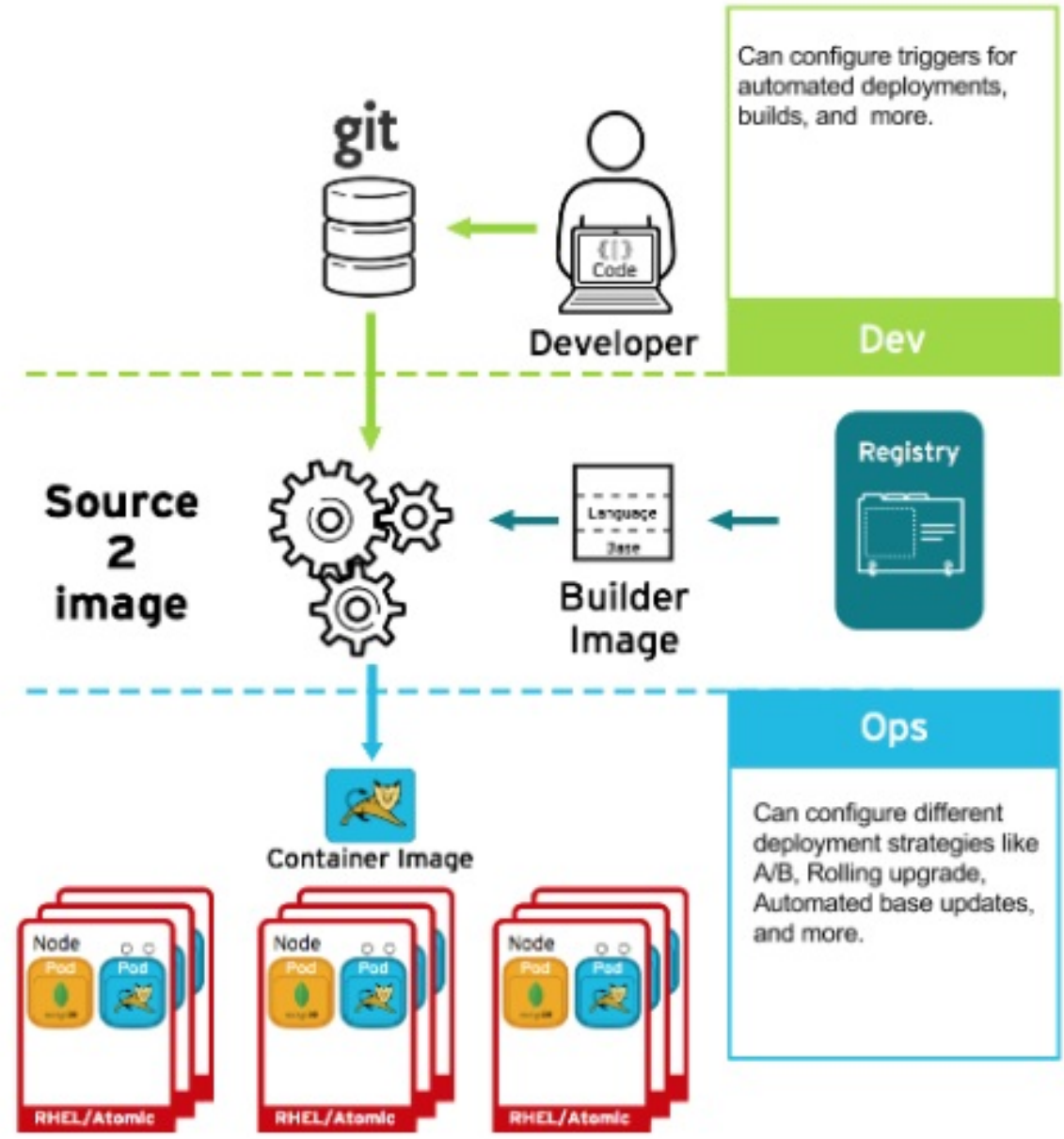
- 创建构建输入
- 输入 secret 和配置映射
- CI/CD 方法和实践
- Knative Serving 架构
- 使用 Topology 视图查看应用程序组成情况

1.8. 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。

1.8.1. 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
```

1.8.2. 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。

1.8.3. 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。

1.8.4. 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。

1.8.5. 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

1.9. 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。

1.9.1. 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。

1.9.2. 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。

1.9.2.1. 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。

1.9.2.2. 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。

1.9.3. 认证和授权

1.9.3.1. 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。

1.9.3.2. 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。

有关使用容器化的 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。

1.9.3.3. 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。

1.9.3.4. 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

1.9.4. 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

1.9.4.1. 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

1.10. 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。

1.10.1. 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。

1.10.2. 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

1.10.3. 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

1.10.4. 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

1.10.5. 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

1.10.6. 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

1.11. 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。

1.11.1. 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**。

1.11.2. 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。

1.11.3. 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

1.12. 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。

1.12.1. 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
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")'
{
  "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'
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。

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

1.12.3. 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 2 章 配置证书

2.1. 替换默认入口证书

2.1.1. 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。

2.1.2. 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。

2.2. 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。

2.2.1. 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. 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。

2. 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。

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

```
$ oc get apiserver cluster -o yaml
...
spec:
  servingCerts:
    namedCertificates:
    - names:
      - <FQDN>
      servingCertificate:
        name: <secret>
...
```

2.3. 使用服务提供的证书 SECRET 保护服务流量

2.3.1. 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 6 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。

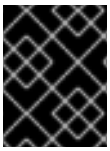
注意

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
```

2.3.2. 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>
```

验证输出中列出的注解：

```
...
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。

2.3.3. 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-----
...
```

2.3.4. 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：

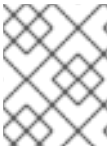
```
$ oc get apiservice <api_service_name> -o yaml
```

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>
...
```

2.3.5. 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>
...
```

2.3.6. 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>
...
```

2.3.7. 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>
...
```

2.3.8. 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>
...
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>

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

2.3.9. 手动轮转服务 CA 证书

服务 CA 证书在 26 个月内有效，并在有效期少于 6 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 -d \
    | 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。

第 3 章 证书类型和描述

3.1. API 服务器的用户提供的证书

3.1.1. 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。

3.1.2. 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。

3.1.3. 管理

用户提供的证书由用户管理。

3.1.4. 过期

用户提供的证书由用户管理。

3.1.5. 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

3.2. 代理证书

3.2.1. 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

3.2.2. 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
. . .
proxy:
  httpProxy: http://<HTTP_PROXY>
  httpsProxy: https://<HTTPS_PROXY>
additionalTrustBundle: |
    -----BEGIN CERTIFICATE-----
   <MY_HTTPS_PROXY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
. . .
```

3.2.3. 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。

3.2.4. 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。

3.2.5. 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。

3.2.6. 管理

这些证书由系统而不是用户管理。

3.2.7. 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: 2.2.0
    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=
```
      filesystem: root
      mode: 0644
      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。

3.2.8. 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。

3.3. 服务 CA 证书

3.3.1. 用途

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

3.3.2. 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 **APIService** 或 **ConfigMap** 资源，以支持通过服务 CA 生成的证书。作为响应，Operator 将其当前 CA 捆绑包写入 **APIService** 资源的 **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。

3.3.3. 管理

这些证书由系统而不是用户管理。

3.3.4. 服务

使用服务 CA 证书的服务包括：

- cluster-autoscaler-operator
- cluster-monitoring-operator
- cluster-authentication-operator
- cluster-image-registry-operator
- cluster-ingress-operator
- cluster-kube-apiserver-operator
- cluster-kube-controller-manager-operator
- cluster-kube-scheduler-operator
- cluster-networking-operator
- cluster-openshift-apiserver-operator
- cluster-openshift-controller-manager-operator
- cluster-samples-operator
- cluster-svcat-apiserver-operator
- cluster-svcat-controller-manager-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

3.4. 节点证书

3.4.1. 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。

3.4.2. 管理

这些证书由系统而不是用户管理。

其他资源

- 操作节点

3.5. BOOTSTRAP 证书

3.5.1. 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。

3.5.2. 管理

这些证书由系统而不是用户管理。

3.5.3. 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。

3.5.4. 自定义

您无法自定义 bootstrap 证书。

3.6. ETCD 证书

3.6.1. 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)。

3.6.2. 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。

3.6.3. 管理

这些证书由系统而不是用户管理。

3.6.4. 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

- 恢复丢失的 master 主机

3.7. OLM 证书

3.7.1. 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。

3.8. 用户提供的默认入口证书

3.8.1. 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。

3.8.2. 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。

3.8.3. 管理

用户提供的证书由用户管理。

3.8.4. 过期

用户提供的证书由用户管理。

3.8.5. 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。

3.8.6. 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

3.9. 入口证书（INGRESS CERTIFICATE）

3.9.1. 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。

3.9.2. 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。

3.9.3. 工作流

图 3.1. **自定**义证书**工作流**

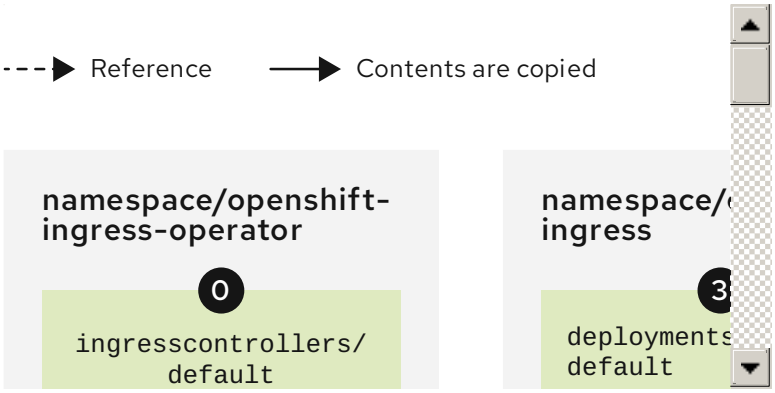
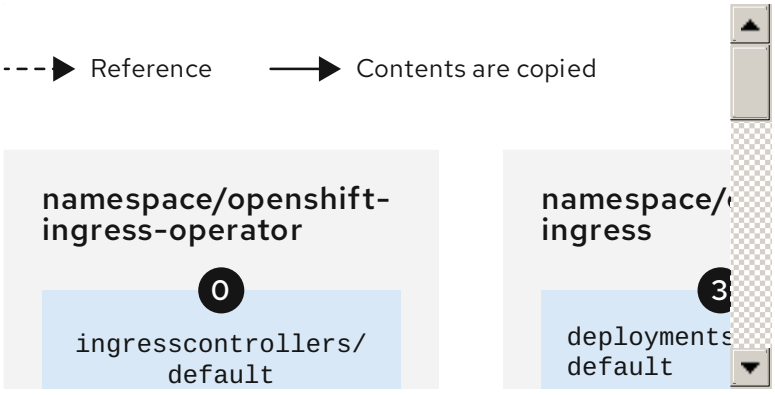


图 3.2. **默**认证书**工作流**



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 包含 Operator 生成的默认 CA 证书（公共部分）的资源; 由 OAuth 和 Web 控制台读取以建立信任。这个对象将在以后的发行版本中被删除。

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

7 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。

8 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。

9 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**。

10 **trustedCA** 字段用来引用用户提供的 CA 捆绑包。

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

12 在 OpenShift Container Platform 4.4 中,有些组件正在从使用 **router-ca** 转换为使用 **default-ingress-cert**。

3.9.4. 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。

3.9.5. 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。

3.9.6. 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。

3.9.7. 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。

3.10. 监控和集群日志记录 OPERATOR 组件证书

3.10.1. 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。

3.10.2. 管理

这些证书由系统而不是用户管理。

3.11. CONTROL PLANE 证书

3.11.1. 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

3.11.2. 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 4 章 查看审计日志

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。

4.1. 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 4.1. 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\""}}
```


4.2. 查看审计日志

您可以查看 OpenShift Container Platform API 服务器或每个 master 节点的 Kubernetes API 服务器的日志。

流程

查看审计日志：

1. 查看 OpenShift Container Platform API 服务器日志
   a. 如果需要，获取您要查看日志的节点名称：

```
$ oc adm node-logs --role=master --path=openshift-apiserver/

ip-10-0-140-97.ec2.internal audit-2019-04-09T00-12-19.834.log
ip-10-0-140-97.ec2.internal audit-2019-04-09T11-13-00.469.log
ip-10-0-140-97.ec2.internal audit.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T00-11-49.835.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T11-08-30.469.log
ip-10-0-153-35.ec2.internal audit.log
```

```
ip-10-0-170-165.ec2.internal audit-2019-04-09T00-13-00.128.log
ip-10-0-170-165.ec2.internal audit-2019-04-09T11-10-04.082.log
ip-10-0-170-165.ec2.internal audit.log
```

b. 查看特定 master 节点和时间戳的 OpenShift Container Platform API 服务器日志，或者查看该 master 的所有日志：

```
$ oc adm node-logs <node-name> --path=openshift-apiserver/<log-name>
```

例如：

```
$ oc adm node-logs ip-10-0-140-97.ec2.internal --path=openshift-apiserver/audit-2019-04-08T13-09-01.227.log
$ oc adm node-logs ip-10-0-140-97.ec2.internal --path=openshift-apiserver/audit.log
```

输出内容类似以下示例：

```
{"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\""}}
```

2. 查看 Kubernetes API 服务器日志：

a. 如果需要，获取您要查看日志的节点名称：

```
$ oc adm node-logs --role=master --path=kube-apiserver/

ip-10-0-140-97.ec2.internal audit-2019-04-09T14-07-27.129.log
ip-10-0-140-97.ec2.internal audit-2019-04-09T19-18-32.542.log
ip-10-0-140-97.ec2.internal audit.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T19-24-22.620.log
ip-10-0-153-35.ec2.internal audit-2019-04-09T19-51-30.905.log
ip-10-0-153-35.ec2.internal audit.log
ip-10-0-170-165.ec2.internal audit-2019-04-09T18-37-07.511.log
ip-10-0-170-165.ec2.internal audit-2019-04-09T19-21-14.371.log
ip-10-0-170-165.ec2.internal audit.log
```

b. 查看特定 master 节点和时间戳的 Kubernetes API 服务器日志，或者查看该 master 的所有日志：

```
$ oc adm node-logs <node-name> --path=kube-apiserver/<log-name>
```

例如：

```
$ oc adm node-logs ip-10-0-140-97.ec2.internal --path=kube-apiserver/audit-2019-04-09T14-07-27.129.log
$ oc adm node-logs ip-10-0-170-165.ec2.internal --path=kube-apiserver/audit.log
```

输出内容类似以下示例：

{"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\""}}

第 5 章 允许从其他主机对 API 服务器进行基于 JAVASCRIPT 的访问

5.1. 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。

如果需要使用其他主机名从 JavaScript 应用程序访问 API 服务器或 OAuth 服务器，可以配置额外的主机名来允许这么做。

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，并且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 6 章 加密 ETCD 数据

6.1. 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。

6.2. 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。

6.3. 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 7 章 对 POD 进行安全漏洞扫描

通过使用 Container Security Operator（CSO），您可以访问 OpenShift Container Platform Web 控制台中用于集群上活跃 pod 的容器镜像，访问 OpenShift Container Platform Web 控制台中的安全漏洞扫描结果。CSO：

- 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，CSO 安装在 **openshift-operators** 命名空间中，因此 OpenShift 集群上的所有命名空间都可以使用它。

7.1. 运行 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 Container Security Operator。如下所述。

先决条件

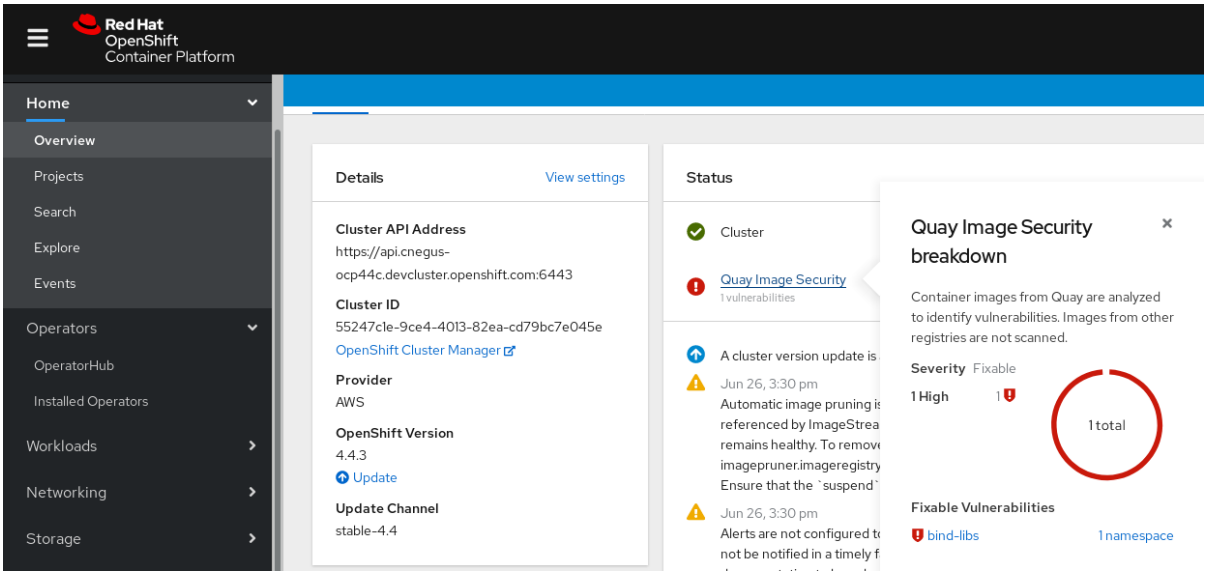
- 具有 OpenShift Container Platform 集群的管理员特权
- 来自集群中运行的 Red Hat Quay 或 Quay.io registry 的容器

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 **Subscribe**。在 **Installed Operators** 屏幕中几分钟后会出现 **Container Security** Operator。
5. 另外，您可以选择在 CSO 中添加自定义证书。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 CSO 中：

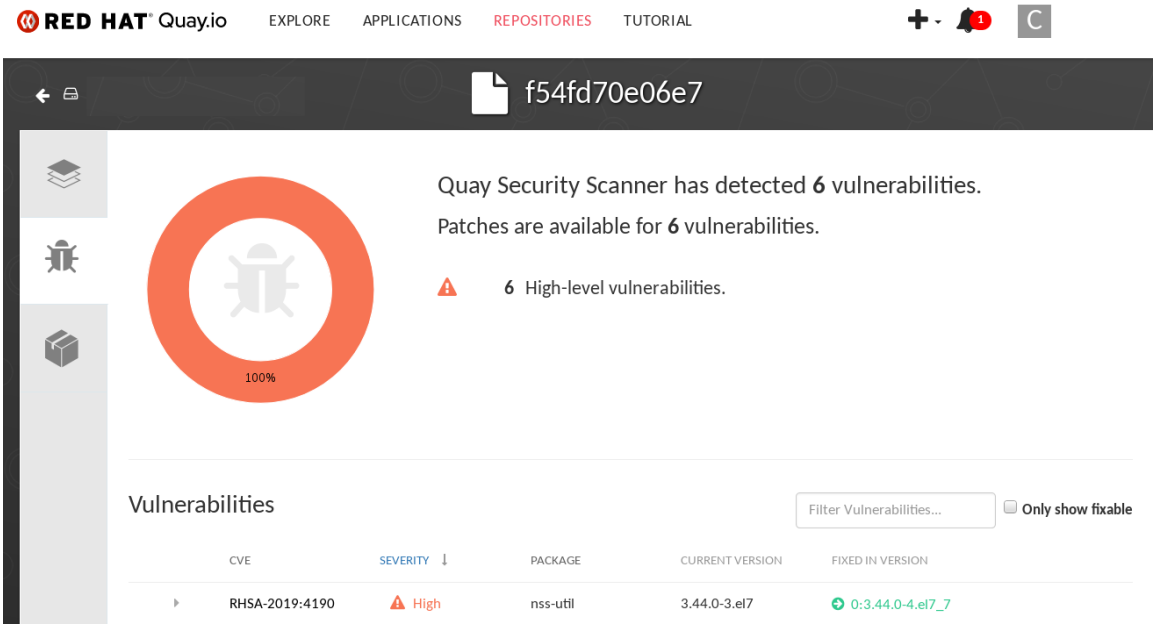
```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n
openshift-operators
```

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：

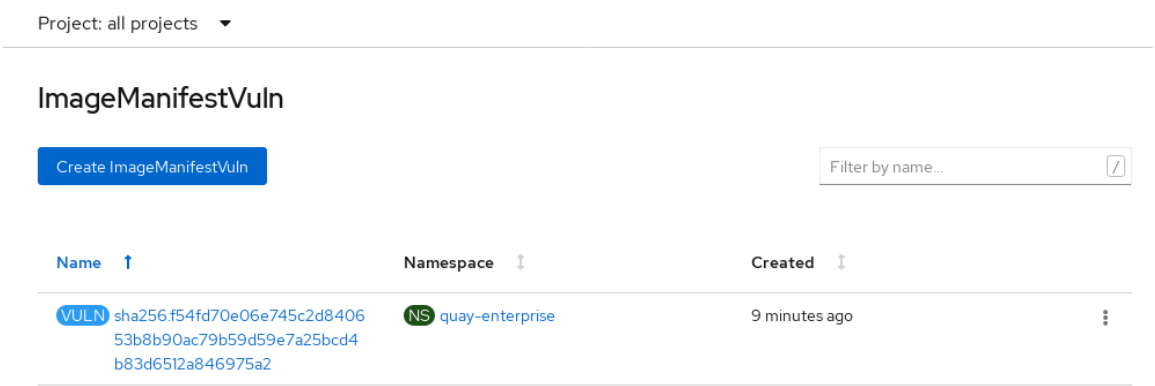


8. 对于任何检测到的安全漏洞，您可以在此时进行两个操作之一：

- 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。

7.2. 通过 CLI 查询镜像安全漏洞

使用 **oc** 命令，可以显示 Container Security Operator 检测到的安全漏洞信息。

先决条件

- 在 OpenShift Container Platform 实例上运行 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
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...
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...
```