OpenShift Container Platform 3.11

容器安全性指南

OpenShift Container Platform 3.11 容器安全性指南

Last Updated: 2020-10-01

OpenShift Container Platform 3.11 容器安全性指南

OpenShift Container Platform 3.11 容器安全性指南

法律通告

摘要

使用这些建议保护集群

目录

第 1 章 简介 3
1.1. 关于本指南 3
1.2. 什么是容器？ 3
1.3. OPENSHIFT CONTAINER PLATFORM 中的容器安全性 3

第 2 章 容器主机和多租户 5
2.1. 如何在 RHEL 中保护容器 5
2.2. 多租户：虚拟化和容器的比较 5

第 3 章 容器内容 7
3.1. 容器内的安全 7
3.2. 容器内容扫描 7
3.3. 将外部扫描工具与 OPENSHIFT 集成 7

第 4 章 REGISTRY 12
4.1. 在哪里可以找到您的容器？ 12
4.2. 不可变和已认证的容器 12
4.3. RED HAT REGISTRY 和 RED HAT CONTAINER CATALOG 12
4.4. OPENSHIFT CONTAINER REGISTRY 13

第 5 章 构建过程 14
5.1. 构建一次，多处部署 14
5.2. 构建管理和安全 14
5.3. 在构建期间保护输入 15
5.4. 设计构建过程 15

第 6 章 DEPLOYMENT 17
6.1. 控制容器中可部署的哪些内容 17
6.2. 控制可以部署的镜像源 18
6.3. SECRET 和 CONFIGMAP 20
6.4. 安全性上下文约束 (SCC) 20
6.5. 持续部署工具 21

第 7 章 保护容器平台 22
7.1. 容器编配 22
7.2. 认证和授权 22
7.3. 为平台管理证书 24

第 8 章 网络安全性 25
8.1. 网络命名空间 25
8.2. 隔离应用程序 25

第 9 章 附加存储 26
9.1. 持久性卷插件 26
9.2. 共享存储 26
9.3. 块存储 26

第 10 章 监控集群事件和日志 27
10.1. 简介 27
10.2. 集群事件 27
10.3. 集群日志 28

第 1 章 简介

1.1. 关于本指南

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。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（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。

1.2. 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 OS 内核中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

深入阅读

- *OpenShift Container Platform 架构*: Core Concepts → Containers and Images
- *Red Hat Enterprise Linux Atomic Host 容器安全指南*

1.3. OPENSHIFT CONTAINER PLATFORM 中的容器安全性

本指南描述了容器解决方案堆栈中每个层的安全性的关键元素，同时还介绍了如何使用 OpenShift Container Platform 在不同阶段和每一层都使用 OpenShift Container Platform 来大规模创建、部署和管理容器。

深入阅读

- *红帽系统中 Red Hat Enterprise Linux Atomic Host 容器的概述*
- *Red Hat Enterprise Linux Atomic Host 容器安全指南*

第 2 章 容器主机和多租户

2.1. 如何在 RHEL 中保护容器

容器允许您通过在单一主机上部署多个应用程序，使用内核和 docker 运行时来为每个容器启动，从而简化多租户部署。

您必须有一个可保护主机内核的操作系统（OS）并且相互保护容器的安全。在 Linux 中，容器只是一种特殊的进程，因此保护容器与保护任何运行的进程是一样的。容器应该以非 root 用户身份运行。建议降低权限级别或创建具有最少权限的容器。

由于 OpenShift Container Platform 在 Red Hat Enterprise Linux（RHEL）和 RHEL Atomic Host 上运行，以下概念默认应用于任何已部署的 OpenShift Container Platform 集群，并且是平台上确保容器安全的核心所在。

- *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同一资源，而不会造成冲突。如需了解有关命名空间类型（如 mount、PID 和 network）的详细信息，请参阅 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。
- 使用 *RHEL Atomic Host* 部署容器可尽量减小主机环境并根据容器进行调整，从而减少攻击面。

深入阅读

- Linux man page: *namespaces(7)*
- *Red Hat Enterprise Linux Atomic Host 概述：使用* SELinux 的安全容器
- *Red Hat Enterprise Linux 资源管理指南*: 控制组群(CGroups)简介
- *Red Hat Enterprise Linux Atomic Host Container 安全指南*: Linux 漏洞和 seccomp
- 内核文档： *seccomp*

2.2. 多租户：虚拟化和容器的比较

传统虚拟化也支持多租户，但方式与容器非常不同。虚拟化依赖于虚拟机监控程序启动虚拟机（VM），每个虚拟机都有自己的操作系统（OS），以及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，仍必须对安全性进行监控：一个虚拟客户机可能使用虚拟机监控程序错误来访问另一个虚拟机或主机内核。当操作系统需要补丁时，它必须对所有使用该操作系统的虚拟机进行补丁。

容器可以在客户机虚拟机中运行，在有些用例中可能是需要的。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，单一主机上的容器多租户提供了一个更加轻便、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。

深入阅读

- *Red Hat 系统中的容器的 Red Hat Enterprise Linux Atomic Host 概述* : Linux 容器与 KVM 虚拟化的比较

第 3 章 容器内容

3.1. 容器内的安全

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，是谁构建的，以及软件包中是否有恶意代码。

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最新的？

深入阅读

- *OpenShift Container Platform 使用镜像*
  - 红帽提供的用于 OpenShift Container Platform 的框架、数据库和服务容器镜像的参考文档

3.2. 容器内容扫描

容器扫描工具可利用不断更新的漏洞数据库，以确保始终了解容器内容的已知漏洞的最新信息。已知漏洞列表不断变化；您必须在首次下载容器镜像时检查容器镜像的内容，并持续跟踪所有批准和部署的镜像的漏洞状态。

RHEL 提供了一个可插入的 API，以支持多个扫描仪。您还可以使用带有 OpenSCAP 的 Red Hat CloudForms 扫描容器镜像以了解安全问题。有关 RHEL 中 OpenSCAP 的一般信息，请参阅 Red Hat Enterprise Linux 安全指南，以及 Red Hat CloudForms 策略及侧写指南 中有关 OpenSCAP 集成的具体内容。

OpenShift Container Platform 可让您在 CI/CD 过程中利用这些扫描程序。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。这在构建过程中进行了更详细的介绍。

3.3. 将外部扫描工具与 OPENSHIFT 集成

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）可能会使用元数据为镜像对象添加注解，以汇总结果并控制 Pod 执行。本节描述了该注解的可识别格式，以便在控制台中可靠使用它来为用户显示有用的数据。

3.3.1. 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 3.1. 注解键格式

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

3.3.1.1. 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 3.2. 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 和/或更多详情。必需，以便用户可以验证数据。	字符串
scannerVersion	否	扫描程序版本	字符串
compliant	否	合规性通过/失败	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 3.3. Summary 字段值格式

字段	描述	类型

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 和/或更多详情。可选。	字符串

3.3.1.2. 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 }
  ]
}
```

3.3.2. 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。

3.3.2.1. 注解 CLI 命令示例

将 **<image>** 替换为一个镜像摘要（digest）。例如 **sha256:fec8a395afe3e804b3db5cb277869142d2b5c561ebb517585566e160ff321988**：

```
$ oc annotate image <image> \
    quality.images.openshift.io/vulnerability.redhatcatalog='{ \
    "name": "Red Hat Container Catalog", \
    "description": "Container health index", \
    "timestamp": "2016-09-08T05:04:46Z", \
    "compliant": null, \
    "scannerVersion": "1.2", \
    "reference": "https://access.redhat.com/errata/RHBA-2016:1566", \
    "summary": "[ \
      { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null } ]" }'
```

3.3.3. 控制 Pod 执行

为了通过编程来控制镜像是否可以运行，可以使用 **images.openshift.io/deny-execution** 镜像策略。如需更多信息，请参阅镜像策略。

3.3.3.1. 注解示例

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

3.3.4. 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 REST API 来编写注解。有关 REST API 和使用 **PATCH** 更新镜像的信息，请参阅 REST API 参考。

3.3.4.1. 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。

Patch 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：

Patch 调用数据

```
{
"metadata": {
  "annotations": {
    "quality.images.openshift.io/vulnerability.redhatcatalog":
       "{ 'name': 'Red Hat Container Catalog', 'description': 'Container health index', 'timestamp': '2016-
09-08T05:04:46Z', 'compliant': null, 'reference': 'https://access.redhat.com/errata/RHBA-2016:1566',
'summary': [{'label': 'Health index', 'data': '4', 'severityIndex': 1, 'reference': null}] }"
    }
  }
}
```

第 4 章 REGISTRY

4.1. 在哪里可以找到您的容器？

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。

4.2. 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。部署不可变容器时，,您不需要介入正在运行的容器来替换一个或多个二进制文件；您可以重建并重新部署更新的容器镜像。

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。

深入阅读

- 更多 OpenShift Container Platform 中不可变容器的信息：
  - *OpenShift Container Platform 架构*: 镜像流
  - *OpenShift Container Platform 开发者指南*: 引用镜像流中的镜像

4.3. RED HAT REGISTRY 和 RED HAT CONTAINER CATALOG

红帽通过 *Red Hat Registry* （由红帽在 registry.redhat.io 托管的公共容器 registry）为红帽产品和合作伙伴提供经过认证的容器。*Red Hat Container Catalog* 可让您识别与 Red Hat Registry 提供的容器镜像关联的程序错误修复或安全公告。

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"健康索引"处理通过红帽容器目录提供的容器的安全风险。这些容器使用红帽提供的软件和勘误，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。

深入阅读

- Red Hat Container Catalog FAQ
- 红帽产品安全中心
- 红帽安全公告

4.4. OPENSHIFT CONTAINER REGISTRY

OpenShift Container Platform 包含 *OpenShift Container Registry*，它是一个与可用于管理容器镜像的平台集成的私有 registry。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。

深入阅读

- *OpenShift Container Platform 架构*: Infrastructure Components → Image Registry

第 5 章 构建过程

5.1. 构建一次，多处部署

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。

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。

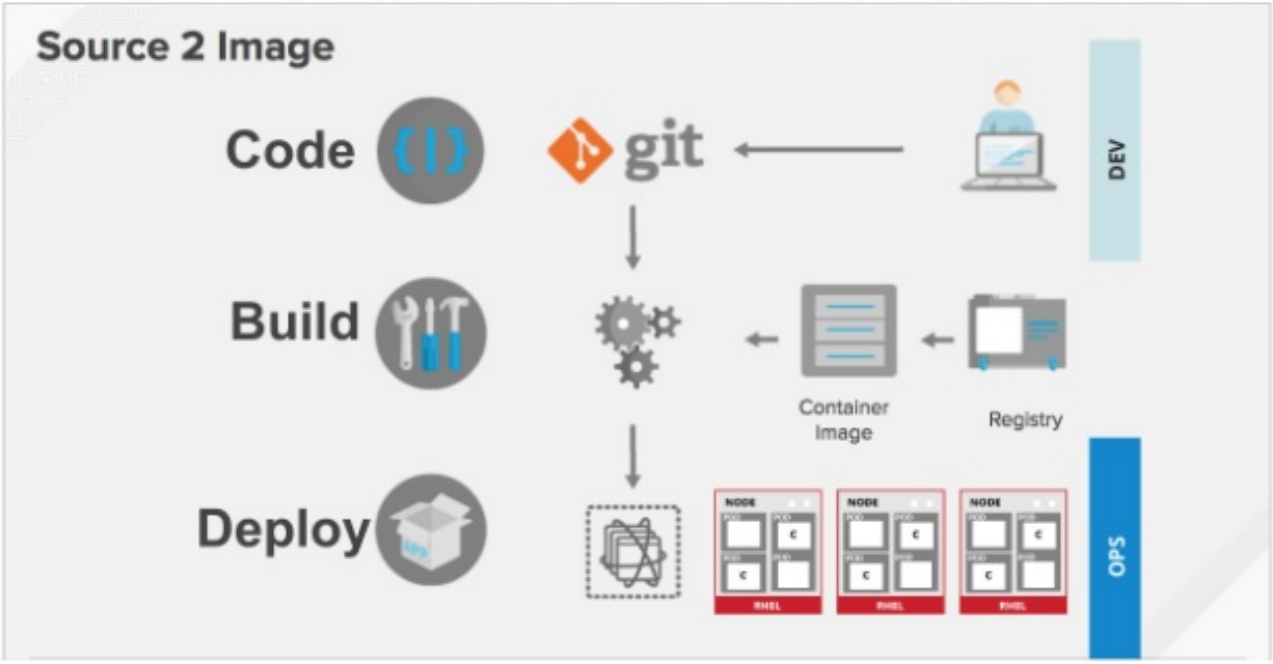
保持容器的不可变性也是很重要的。您不应该修补运行中的容器，而应该重建并重新部署这些容器。

5.2. 构建管理和安全

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。

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 OpenShift Container Registry 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。

深入阅读

- *OpenShift Container Platform 开发人员指南*
  - 构建如何工作
  - 触发构建
- *OpenShift Container Platform 架构*: Source-to-Image(S2I)构建
- *使用镜像的 OpenShift Container Platform*: Other Images → Jenkins

5.3. 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 ***.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** 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** 的 **source** 部分中：

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

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

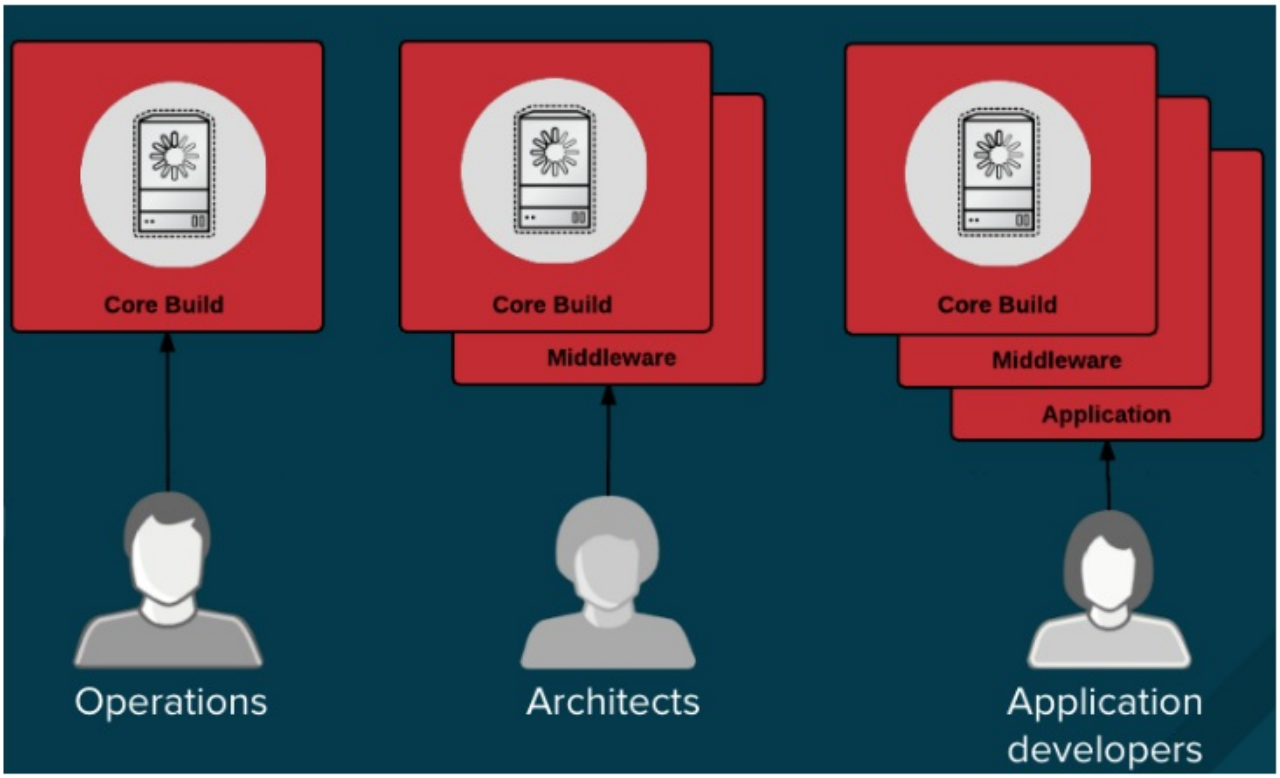
```
$ oc new-build \
    openshift/nodejs-010-centos7~https://github.com/sclorg/nodejs-ex.git \
    --build-secret secret-npmrc
```

深入阅读

- *OpenShift Container Platform 开发者指南*: 输入 Secret

5.4. 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及编写代码。

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。

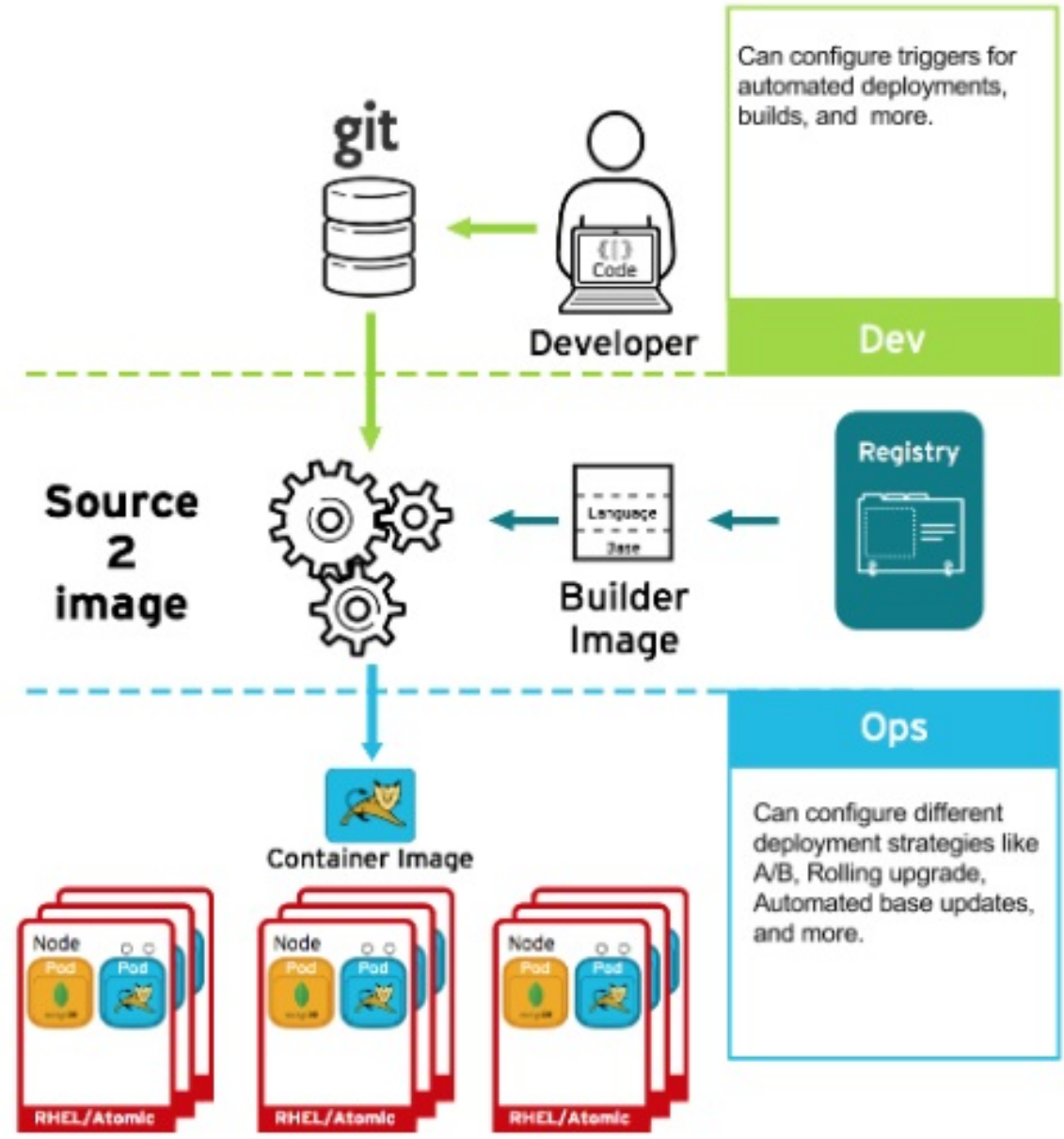
深入阅读

- *Red Hat Enterprise Linux Atomic Host Managing Containers* : 签名容器镜像

第 6 章 DEPLOYMENT

6.1. 控制容器中可部署的哪些内容

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，而不是修补正在运行的容器，我们不建议这样做。



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，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。

oc set triggers 命令可以用来为部署配置设置部署触发器。例如，在名为 **frontend** 的部署配置中设置 **ImageChangeTrigger**:

```
$ oc set triggers dc/frontend \
    --from-image=myproject/origin-ruby-sample:latest \
    -c helloworld
```

深入阅读

- *OpenShift Container Platform 开发人员指南*
  - 部署如何工作
  - 设置部署触发器
  - 应用程序生命周期管理 →跨环境的提升应用程序

6.2. 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，确保它们来自可信源，且未被更改。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（接受、拒绝或要求公钥）

使用这些策略参数，可以将 registry 或 registry 的部分（甚至单独的镜像）列入白名单（接受）、列入黑名单（拒绝）或者使用可信公钥定义信任关系，以确保以加密方式验证源。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": "insecureAcceptAnything"}]
```

```
        }
    }
}
```

该策略可以在节点上保存为 */etc/containers/policy.json*。这个示例强制执行以下规则：

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

有关配置主机的具体步骤，请参阅启用镜像签名支持。更多信息，请参见以下部分签名传输。有关镜像签名策略的详情，请参阅 Signature 验证策略文件格式源代码文档。

6.2.1. 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。根据 Signature 访问协议，您可以直接访问每个镜像的签名；服务器目录的根用户不会显示其文件结构。

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 Ansible 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。

如需更多详细信息，请参阅 注册配置目录 或者 签名访问协议 源代码文档。

深入阅读

- *OpenShift Container Platform 集群管理指南*
  - 默认调度

- *红帽知识库*
  - 容器镜像签名集成指南
- *源代码参考*
  - 镜像签名策略
  - 签名传输
  - 签名格式

6.3. SECRET 和 CONFIGMAP

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，使用 web 控制台将 secret 添加到部署配置中，以便其能够访问私有镜像存储库：

1. 创建新项目。
2. 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。
3. 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。

ConfigMap 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。

深入阅读

- *OpenShift Container Platform 开发人员指南*
  - Secrets
  - ConfigMaps

6.4. 安全性上下文约束 (SCC)

您可以使用 *安全性上下文约束* (SCC)来定义 Pod 运行（容器集合）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。

如果您有所需的权限，您可以将默认 SCC 策略调整为更宽松。

深入阅读

- *OpenShift Container Platform 架构*: 安全性上下文约束
- *OpenShift Container Platform 安装集群*: 安全警告
  - 讨论特权容器

6.5. 持续部署工具

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。

第 7 章 保护容器平台

7.1. 容器编配

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。

深入阅读

- OpenShift Container Platform *架构*: 如何保护 OpenShift Container Platform?

7.2. 认证和授权

7.2.1. 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。Deny All 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。如需完整的身份提供程序列表，请参阅配置身份验证和用户代理。

例如，在安装后配置 GitHub 身份提供程序：

1. 编辑 */etc/origin/master-config.yaml* 中的 master 配置文件。
2. 修改 **oauthConfig** 小节：

```
oauthConfig:
  ...
  identityProviders:
  - name: github
    challenge: false
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: GitHubIdentityProvider
      clientID: ...
      clientSecret: ...
      organizations:
      - myorganization1
      - myorganization2
      teams:
      - myorganization1/team-a
      - myorganization2/team-b
```

注意

如需更多详细信息和使用情况，请参阅配置身份验证中的 GitHub 部分。

3. 保存更改后，重启 master 服务以使更改生效：

```
# master-restart api
# master-restart controllers
```

深入阅读

- *OpenShift Container Platform 架构*
  - 额外概念 → 身份验证
  - 额外概念 → 授权
- *OpenShift Container Platform CLI 参考*
- *OpenShift Container Platform 开发者指南*：CLI 身份验证

7.2.2. 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，请参阅在 Red Hat OpenShift 上运行 APIcast 。

7.2.3. Red Hat SSO

Red Hat Single Sign-On（RH-SSO）服务器通过根据标准提供 Web SSO 功能来保护应用程序，包括 SAML 2.0、OpenID Connect 和 OAuth 2.0。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 Red Hat SSO 与基于 LDAP 的目录服务集成，包括 Microsoft Active Directory 和 Red Hat Enterprise Linux Identity Management。

有关使用指南，请参阅 Red Hat JBoss SSO 。

7.2.4. 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 不直接向集群公开

深入阅读

- *OpenShift Container Platform 架构*: Infrastructure Components → Web Console

- *OpenShift Container Platform Developer Guide*: Web Console Authentication

7.3. 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 基于 Ansible 的安装程序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

7.3.1. 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。

在使用 Ansible playbook 的初始集群安装过程中，可使用 **openshift_master_overwrite_named_certificates** Ansible 变量配置自定义证书，这些变量可在清单文件中配置。例如：

```
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/custom1.crt", "keyfile":
"/path/on/host/to/custom1.key", "cafile": "/path/on/host/to/custom-ca1.crt"}]
```

如需了解更多选项以及如何运行安装 playbook 的说明，请参阅配置自定义证书部分。

安装程序提供 Ansible playbook 以检查所有集群证书的过期日期。额外的 playbook 可以使用当前的 CA 自动重新部署所有证书，仅重新部署特定证书，或者自行重新部署新生成的或自定义 CA。请参阅在这些 playbook 上 重新部署证书 以了解更多证书。

深入阅读

- *OpenShift Container Platform 配置集群*
  - 配置自定义证书
  - 检查证书过期
  - 重新部署证书

第 8 章 网络安全性

8.1. 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。

使用网络命名空间可以隔离 Pod 网络。每个 pod 都有自己的 IP 和端口范围来绑定，从而将 pod 网络相互隔离。来自不同项目的 Pod 不能与不同项目的 Pod 和服务互相发送或接收数据包。您可以使用它来隔离集群中的开发人员、测试和生产环境。

OpenShift Container Platform 还提供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。

深入阅读

- *OpenShift Container Platform 架构*: Networking
- *OpenShift Container Platform 集群* 管理网络
- *Red Hat Enterprise Linux Atomic Host Managing Containers* : Running Super-Privileged Containers

8.2. 隔离应用程序

OpenShift Container Platform 可让您将单一集群上的网络流量分段，以创建多租户集群将用户、团队、应用程序和环境隔离。

例如，若要在集群中隔离项目网络，或反之，请运行：

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

在上例中，**<project1>** 和 **<project2>** 中的所有 pod 和服务都无法从集群中的其他非全局项目访问任何 pod 和服务，反之亦然。

第 9 章 附加存储

9.1. 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。

OpenShift Container Platform 为多种存储提供插件，包括 NFS、AWS Elastic Block Stores(EBS)、GCE Persistent Disks、GlusterFS、iSCSI、RADOS(Ceph)和 Cinder。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。

您可以通过存储类型所支持的任何方式在主机上挂载 **PersistentVolume** (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**。

深入阅读

- *OpenShift Container Platform 架构*: Additional Concepts → Storage
- *OpenShift Container Platform 配置集群*: 配置存储性存储 → 卷安全性

9.2. 共享存储

对于 NFS、Ceph 和 Gluster 等共享存储供应商，PV 将组 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。

深入阅读

- *OpenShift Container Platform 配置集群*
  - 使用 NFS 的持久性存储
  - 使用 Ceph RBD 的持久性存储
  - 使用 GlusterFS 的持久性存储

9.3. 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。

深入阅读

- *OpenShift Container Platform 配置集群*
  - 使用 AWS Elastic Block Storage 的持久性存储
  - 使用 GCE Persistent Disk 的持久性存储
  - 使用 iSCSI 的持久性存储

第 10 章 监控集群事件和日志

10.1. 简介

除了本指南其他部分提到的安全措施外，监控和审核 OpenShift Container Platform 集群的能力是防止集群及其用户遭到不当使用的重要部分。

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。

10.2. 集群事件

建议集群管理员熟悉 *Event* 资源类型，并查看事件列表以确定值得关注的事件。根据 master 控制器和插件配置，通常会比这里列出的事件类型更多。

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 *Node* 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：

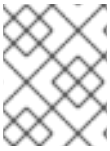
```
$ oc get events -n default -o json \
  | jq '.items[] | select(.involvedObject.kind == "Node" and .reason == "NodeHasDiskPressure")'

{
  "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。

10.3. 集群日志

本节论述了集群中生成的操作日志的类型。

10.3.1. 服务日志

OpenShift Container Platform 为在集群中静态 pod 上运行的服务生成日志：

- API（使用 **master-logs api api**）
- 控制器（使用 **master-logs controllers controllers**）
- etcd（使用 **master-logs etcd etcd**）
- atomic-openshift-node（使用 **journalctl -u atomic-openshift-node.service**）

这些日志主要用于调试目的，而不是安全审核。您可以使用 **master-logs api api**、**master-logs controllers controllers** 或 **master-logs etcd etcd** 命令来检索各个服务的日志。如果您的集群运行一个聚合的日志记录堆栈（如 Ops 集群 ）,集群管理员可以从 logging *.operations* 索引中检索日志。

注意

API 服务器、控制器和 etcd 静态 Pod 在 kube-system 命名空间中运行。

10.3.2. Master API 审计日志

要记录用户、管理员或系统组件对 master API 请求的日志，请启用 主 API 审计日志记录。这将在每个 master 主机上创建一个文件，如果没有配置文件，则将包含在服务日志中。日志中的条目可以通过搜索 "AUDIT" 找到。

审计日志条目 包括一行，在收到每个 REST 请求时记录每个 REST 请求，在完成时使用 HTTP 响应代码记录一行。例如，这里是系统管理员请求列出节点列表的记录：

```
2017-10-17T13:12:17.635085787Z AUDIT: id="410eda6b-88d4-4491-87ff-394804ca69a1"
ip="192.168.122.156" method="GET" user="system:admin" groups="\"system:cluster-
admins\",\"system:authenticated\"" as="<self>" asgroups="<lookup>" namespace="<none>"
uri="/api/v1/nodes"
2017-10-17T13:12:17.636081056Z AUDIT: id="410eda6b-88d4-4491-87ff-394804ca69a1"
response="200"
```

定期轮询每个响应代码的最新请求数量可能会有帮助，如下例所示：

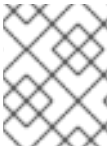
```
$ tail -5000 /var/lib/origin/audit-ocp.log \
  | grep -Po 'response="..."' \
  | sort | uniq -c | sort -rn

   3288 response="200"
      8 response="404"
      6 response="201"
```

以下列表更详细地描述了一些响应代码：

- **200** 或 **201** 响应代码代表请求成功。
- **400** 响应代码可能值得关注，因为它们表示请求格式不正确，大多数客户端不应该发生这种请求。
- **404** 响应代码通常是对尚不存在的资源的异步请求。
- **500** - **599** 响应代码显示服务器错误，这可能是 bug、系统故障甚至恶意活动所致。

如果出现异常的错误响代码，就可以检索相应请求的审计日志条目以供进一步调查。



注意

请求的 IP 地址通常是集群主机或 API 负载均衡器，且没有负载均衡器代理请求后面的 IP 地址记录（但负载均衡器日志对确定请求来源很有用）。

查找特定用户或组群的异常请求数量会很有用。

以下示例根据审计日志的最后 5000 行中的请求数列出了前 10 个用户：

```
$ tail -5000 /var/lib/origin/audit-ocp.log \
  | grep -Po ' user="(.*?)(?<!\\)"' \
  | sort | uniq -c | sort -rn | head -10

  976  user="system:openshift-master"
  270  user="system:node:origin-node-1.example.local"
  270  user="system:node:origin-master.example.local"
   66  user="system:anonymous"
   32  user="system:serviceaccount:kube-system:cronjob-controller"
   24  user="system:serviceaccount:kube-system:pod-garbage-collector"
   18  user="system:serviceaccount:kube-system:endpoint-controller"
   14  user="system:serviceaccount:openshift-infra:serviceaccount-pull-secrets-controller"
   11  user="test user"
    4  user="test \" user"
```

更高级的查询通常需要使用额外的日志分析工具。审核员需要详细了解 OpenShift v1 API 和 Kubernetes v1 API，以汇总审计日志中涉及的资源类型（**uri** 字段）的请求摘要。详情请查看 REST API 参考。

更多高级审计日志记录功能 可用。此功能支持提供审计策略文件来控制哪些请求的日志记录和日志的详细程度。高级审计日志条目以 JSON 格式提供更详细的信息，可以通过 webhook 而不是文件或系统 journal 记录。