# Red Hat Quay 3.11

## 关于 Quay IO

关于 Quay IO

Last Updated: 2024-06-05

# Red Hat Quay 3.11 关于 Quay IO

关于 Quay IO

## 法律通告



## 摘要

关于 Quay IO

# 目录

# 前言

本指南为用户提供了充分利用我们强大且功能丰富的容器 registry 服务 Quay.io 所需的知识和工具。

# 第 1 章 QUAY.IO 概述

Quay.io 是一个 registry，用于存储、构建和分发容器镜像和其他 OCI 工件。这种强大且功能丰富的容器 registry 服务已得到开发人员、企业和企业之间广泛流行，以作为容器化生态系统中的先驱平台之一建立。它为各种用户需求提供免费和付费的层。

在其核心上，Quay.io 充当用于存储、管理和分发容器镜像的集中存储库。Quay.io 的一个主要优点是其灵活性和易用性。它提供了一个直观的 Web 界面，允许用户快速上传和管理其容器镜像。开发人员可以创建私有存储库，确保敏感或专有代码在其机构中保持安全。此外，用户可以设置访问控制和管理团队协作，从而在指定的团队成员中无缝共享容器镜像。

Quay.io 通过其集成镜像扫描程序 Clair 来解决容器安全问题。该服务会自动扫描容器镜像以了解已知漏洞和安全问题，让开发人员可以深入了解潜在的风险并建议补救步骤。

Quay.io 提高了自动化功能，并支持与流行的持续集成/持续部署(CI/CD)工具和平台集成，实现容器构建和部署流程的无缝自动化。因此，开发人员可以简化其工作流，从而显著降低人工干预并提高整体开发效率。

Quay.io 满足大型和小型部署的需求。其强大的架构和对高可用性的支持可确保组织将其用于关键任务应用程序。平台可以处理大量容器镜像流量，并提供高效的复制和分发机制，以向不同的地理位置提供容器镜像。

Quay.io 已建立了自己作为容器爱好者的活跃中心。开发人员可以发现由其他用户共享的一组预构建公共容器镜像，从而更轻松地为其项目查找有用的工具、应用程序和服务。这种开放共享生态系统促进了协作并加快容器社区内的软件开发。

随着容器化在软件开发环境中持续增长，Quay.io 仍然保持在前进，不断改进和扩展其服务。该平台对安全性、易用性、自动化和社区参与的承诺已证明其作为各个开发人员和大型组织的首选容器 registry 服务。

随着技术的演进，通过官方网站或其他可靠的来源验证 Quay.io 平台上的最新功能和更新至关重要。无论您是个人开发人员、团队的一部分还是代表企业，Quay.io 都可以提高容器化体验，简化构建和部署现代应用的旅程。

# 第 2 章 QUAY.IO 支持

技术支持是 Quay.io 容器 registry 服务的一个关键方面，不仅有助于管理容器镜像，同时确保托管平台的功能和可用性。

为了帮助用户遇到与功能相关的问题，红帽可让 Quay.io 客户访问多个资源。红帽知识库 包含有价值的内容，以最大化红帽产品和技术的潜力。用户可以找到文章、产品文档和视频，其中概述了安装、配置和使用红帽产品的最佳实践。它还充当已知问题的解决方案中心，提供简洁的根原因描述和补救步骤。

此外，Quay.io 客户还可以考虑技术支持团队来解决问题、解决问题，并提供解决方案，以优化平台体验。无论是了解特定功能、自定义配置还是解决容器镜像构建问题，支持团队都致力于通过清晰程度和专业知识逐步指导用户。

对于 Quay.io 状态页面上 未列出的服务中断或性能问题的事件，其中包括可用性和功能问题，请支付客户使用 红帽客户门户网站 引发技术支持票据。服务事件被定义为计划中断服务，或降低服务质量，从而影响平台的多个用户。

通过这种全面的技术支持系统，Quay.io 确保用户可以自信地管理其容器镜像，优化其平台体验，并克服可能出现的任何挑战。

其他资源

当前 Red Hat Quay 和 Quay.io 用户可在 Red Hat Quay 故障排除指南 中找到有关故障排除和支持的更多信息。

# 第 3 章 CLAIR 安全扫描程序

Clair v4 (Clair)是一个开源应用，它利用静态代码分析来解析镜像内容并报告影响内容的漏洞。Clair 与 Quay.io 打包，被自动启用，并由 Red Hat Quay 开发团队管理。

对于 Quay.io 用户，镜像会在镜像推送到存储库后自动进行索引。然后，会从 Clair 获取报告，该镜像与其 CVE 的数据库匹配，以报告安全信息。此过程会在 Quay.io 上自动进行，不需要手动重新设置。

## 3.1. 关于 CLAIR

Clair 使用国家漏洞数据库(NVD)中的常见漏洞评分系统(CVSS)数据功能丰富的漏洞数据，这是与安全相关的信息，包括各种软件组件和系统中的已知漏洞和安全问题。使用 NVD 中的分数为 Clair 提供以下优点：

- 数据同步.Clair 可以定期将其漏洞数据库与 NVD 同步。这样可确保它具有最新的漏洞数据。
- 匹配并增强.Clair 将容器镜像中发现的漏洞的元数据和标识符与 NVD 中的数据进行比较。这个过程涉及将唯一标识符（如常见漏洞和暴露(CVE) ID）与 NVD 中的条目匹配。找到匹配项时，Clair 可以通过来自 NVD 的额外详情来增强其漏洞信息，如严重性分数、描述和引用。
- 严重性分数.NVD 为漏洞分配严重性分数，如通用漏洞评分系统(CVSS)分数，以指示与每个漏洞相关的潜在影响和风险。通过整合 NVD 的严重性分数，Clair 可以根据它检测到的漏洞的严重程度提供更多上下文。

如果 Clair 从 NVD 找到漏洞，对容器镜像中检测到的漏洞的严重性和潜在影响的详细说明和潜在影响将报告给 UI 上的用户。CVSS 增强数据提供了以下优点：

- 漏洞优先级.通过使用 CVSS 分数，用户可以根据其严重性对漏洞进行优先级排序，帮助他们首先解决最重要的问题。
- 评估风险。CVSS 分数可帮助 Clair 用户了解对其容器化应用程序造成漏洞的潜在风险。
- 通讯严重性.CVSS 分数为 Clair 用户提供了一种标准化的方法，来跨团队和机构沟通漏洞的严重性。
- 通知修复策略.CVSS 增强数据可在开发适当的补救策略时指导 Quay.io 用户。
- 合规性和报告.将 CVSS 数据集成到 Clair 生成的报告中可帮助组织展示其在解决安全漏洞方面以及符合行业标准和法规的承诺。

### 3.1.1. Clair 漏洞数据库

Clair 使用以下漏洞数据库来报告镜像中的问题：

- Ubuntu Oval 数据库
- Debian 安全跟踪器
- Red Hat Enterprise Linux (RHEL) Oval 数据库
- SUSE Oval 数据库
- Oracle Oval 数据库
- alpine SecDB 数据库

- VMware Photon OS 数据库
- Amazon Web Services (AWS) UpdateInfo
- 开源漏洞(OSV)数据库

### 3.1.2. Clair 支持的依赖项

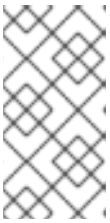
Clair 支持识别和管理以下依赖项：

- Java
- Golang
- Python
- Ruby

这意味着，它可以分析和报告这些语言中项目依赖的第三方库和软件包。

当包含 Clair 不支持的语言的软件包的镜像被推送到您的存储库时，无法在这些软件包上执行漏洞扫描。用户不会收到不支持的依赖项或软件包的分析或安全报告。因此，应该考虑以下结果：

- **安全风险**.没有扫描漏洞的依赖项或软件包可能会给您的组织带来安全风险。
- **合规问题**.如果您的组织具有特定的安全性或合规要求、未扫描或部分扫描，容器镜像可能会导致与某些法规不兼容。



注意

扫描的镜像会被索引，并创建了漏洞报告，但可能会忽略某些不支持的语言中的数据。例如，如果您的容器镜像包含 Lua 应用程序，则不会提供来自 Clair 的反馈，因为 Clair 不会检测到它。它可以检测容器镜像中使用的其他语言，并显示这些语言检测到的 CVE。因此，Clair 镜像 *会根据 Clair 支持的内容进行完全扫描*。

## 3.2. CLAIR 严重性映射

Clair 提供了全面的漏洞评估和管理方法。其基本功能之一是对安全数据库严重性字符串进行规范化。这个过程通过将漏洞严重性映射到预定义的值，从而简化对漏洞严重性的评估。通过此映射，客户端可以有效地响应漏洞严重性，而无需破坏每个安全数据库的唯一严重性字符串。这些映射的严重性字符串与相应安全数据库中发现的字符串一致，确保漏洞评估的一致性和准确性。

### 3.2.1. Clair 严重性字符串

Clair 会警告用户有以下严重性字符串：

- Unknown
- negligible
- 低
- Medium
- High

- Critical

这些严重性字符串与相关安全数据库中找到的字符串类似。

**alpine 映射**
alpine SecDB 数据库不提供严重性信息。所有漏洞严重性都将为 Unknown。

| alpine 严重性 | Clair 严重性 |
|---|---|
| * | Unknown |

**AWS 映射**
AWS UpdateInfo 数据库提供严重性信息。

| AWS 严重性 | Clair 严重性 |
|---|---|
| 低 | 低 |
| 中 | Medium |
| 重要 | High |
| critical | Critical |

**Debian 映射**
Debian Oval 数据库提供严重性信息。

| Debian 严重性 | Clair 严重性 |
|---|---|
| * | Unknown |
| Unimportant | 低 |
| 低 | Medium |
| Medium | High |
| High | Critical |

**Oracle 映射**
Oracle Oval 数据库提供严重性信息。

| Oracle 严重性 | Clair 严重性 |
|---|---|
| N/A | Unknown |

| Oracle 严重性 | Clair 严重性 |
| --- | --- |
| 低 | 低 |
| 中 | Medium |
| 重要 | High |
| CRITICAL | Critical |

**RHEL 映射**
RHEL Oval 数据库提供严重性信息。

| RHEL 严重性 | Clair 严重性 |
| --- | --- |
| None | Unknown |
| 低 | 低 |
| Moderate（中度） | Medium |
| 重要的 | High |
| Critical | Critical |

**SUSE 映射**
SUSE Oval 数据库提供严重性信息。

| 重要性 | Clair 严重性 |
| --- | --- |
| None | Unknown |
| 低 | 低 |
| Moderate（中度） | Medium |
| 重要的 | High |
| Critical | Critical |

**Ubuntu 映射**
Ubuntu Oval 数据库提供严重性信息。

| 重要性 | Clair 严重性 |
| --- | --- |
| Untriaged | Unknown |
| negligible | negligible |
| 低 | 低 |
| Medium | Medium |
| High | High |
| Critical | Critical |

### OSV 映射

表 3.1. CVSSv3

| 基本分数 | Clair 严重性 |
| --- | --- |
| 0.0 | negligible |
| 0.1-3.9 | 低 |
| 4.0-6.9 | Medium |
| 7.0-8.9 | High |
| 9.0-10.0 | Critical |

表 3.2. CVSSv2

| 基本分数 | Clair 严重性 |
| --- | --- |
| 0.0-3.9 | 低 |
| 4.0-6.9 | Medium |
| 7.0-10 | High |

# 第 4 章 QUAY.IO 用户界面概述

Quay.io 的用户界面(UI)是一个基本的组件，充当用户网关，用于管理和与平台生态系统中的容器镜像交互。Quay.io 的 UI 旨在提供直观且用户友好的界面，使用户可以轻松地浏览和利用 Quay.io 的功能和功能。

本文档部分旨在向用户介绍 Quay.io UI 的关键元素和功能。它将涵盖诸如 UI 布局、导航和主要功能等重要方面，为用户提供一个稳定的基础，供用户探索并充分利用 Quay.io 的容器 registry 服务。

在这个文档中，以下主题提供了逐步说明、视觉辅助和实际示例：

- 探索应用程序和软件仓库
- 使用 Quay.io 指南
- 定价和 Quay.io 计划
- 登录和使用 Quay.io 功能

本文档集中确保用户能够快速获取 UI 的细微问题，并通过 Quay.io 成功浏览其容器化之旅。

## 4.1. QUAY.IO 登录页面

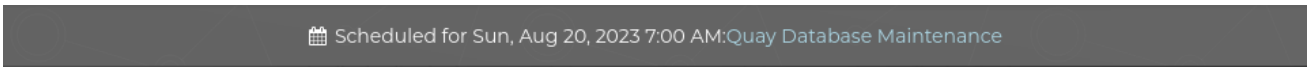
Quay.io 登录页面充当用户访问所提供的容器 registry 服务的中央 hub。此页面提供了基本的信息和链接，可指导用户安全存储、构建和部署容器镜像。

Quay.io 的登录页面包含到以下资源的链接：

- 浏览。在本页中，您可以搜索 Quay.io 数据库以查找各种应用和存储库。
- 教程.在本页中，您可以逐步执行一个步骤，其中演示了如何使用 Quay.io。
- 定价.在本页中，您可以了解为 Quay.io 提供的各种定价层。本页中还解决了各种常见问题。
- 登录。通过单击此链接，您将重新定向到 Quay.io 存储库。


.

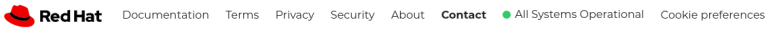
登录页面还包含有关计划维护的信息。在计划的维护期间，Quay.io 以只读模式运行，并以正常方式拉取功能。推送和构建在计划的维护期间无法正常工作。您可以通过进入到 Quay.io **Status** 页面并点 **Subscribe To Updates** 来订阅 Quay.io 维护的更新。




登录页面还包括到以下资源的链接：

- 文档.本页提供了使用 Quay.io 的文档。
- 术语.本页提供有关红帽在线服务的法律信息。
- 隐私.本页提供有关红帽隐私声明的信息。

- 安全. 本页提供有关 Quay.io 安全的信息，包括 SSL/TLS、加密、密码、访问控制、防火墙和数据弹性。
- 关于.本页包含有关使用的软件包和项目以及产品的简短历史记录的信息。
- 联系。本页包含有关支持以及联系红帽支持团队的信息。
- 所有系统操作.此页面包含 Quay.io 状态的信息，以及维护的简短历史记录。
- Cookie。点击此链接时会出现一个弹出窗口，供您设置 Cookie 首选项。




.

您还可以查找有关内部的 Trying Red Hat Quay 的信息，或查找 云上的 Red Hat Quay，这会将您重定向到 定价 页面。每个选项都提供免费试用。

### 4.1.1. 创建 Quay.io 帐户

注册红帽帐户并创建一个 Quay.io 用户名， 需要新用户 Quay.io。这些帐户已关联，有两个不同的区别：

- Quay.io 帐户可用于将容器镜像或开放容器项目镜像推送到 Quay.io 以存储镜像。
- Red Hat 帐户允许用户访问 Quay.io 用户界面。对于付费客户，此帐户还可用于访问 红帽生态系统目录 中的镜像，这些镜像可推送到其 Quay.io 存储库。

用户必须首先注册红帽帐户，然后创建一个 Quay.io 帐户。用户需要两个帐户来正确使用 Quay.io 的所有功能。

#### 4.1.1.1. 注册红帽帐户

使用以下步骤为 Red Hat account for Quay.io 注册。

流程

1. 访问红帽客户门户。
2. 在导航窗格中，单击 **Log In**。
3. 当导航到登录页面时，点 **Register for a Red Hat Account**。
4. 输入红帽登录 ID。
5. 输入密码。
6. 输入以下个人信息：
   - 名
   - 姓
   - 电子邮件地址
   - 电话号码
7. 输入相对于国家/地区或区域的以下联系信息。例如：

- **Country/region**
- **地址**
- **邮政代码**
- **City**
- **County**

8. 选择并同意红帽的条款和条件。
9. 单击 **Create my account**。
10. 导航到 Quay.io 并登录。

#### 4.1.1.2. 创建 Quay.io 用户帐户

使用以下步骤创建 Quay.io 用户帐户。

先决条件

- 您已创建了红帽帐户。

流程

1. 如果需要，通过单击 **I am** 不是机器人 并确认来解析 captcha。您将被重定向到 **Confirm Username** 页面。
2. 在 **Confirm Username** 页面中，输入一个用户名。默认情况下会生成一个用户名。如果同一用户名已存在，则末尾添加一个数字，使其唯一。此用户名用作 Quay Container Registry 中的命名空间。
3. 在决定用户名后，点 **Confirm Username**。您将被重定向到 Quay.io **存储库** 页面，它充当一个专用 hub，用户可以在其中轻松访问和管理其存储库。在此页面中，用户可以高效地组织、浏览和与其容器镜像及相关资源进行交互。

#### 4.1.1.3. Quay.io 单点登录支持

Red Hat Single Sign On (SSO)可与 Quay.io 一起使用。使用以下步骤使用 Quay.io 设置 Red Hat SSO。对于大多数用户，这些帐户已链接。但是，对于某些旧的 Quay.io 用户，可能需要这个过程。

先决条件

- 您已创建了 Quay.io 帐户。

流程

1. 导航到 Quay.io **Recovery** 页面。
2. 输入您的用户名和密码，然后单击 **Sign in to Quay Container Registry**。
3. 在导航窗格中，点击您的 username → **Account Settings**。
4. 在导航窗格中，单击 **External Logins and Applications**。

5. 单击 **Attach to Red Hat**。
6. 如果您已登录到 Red Hat SSO，您的帐户会自动链接。否则，系统会提示您通过输入红帽登录或电子邮件和密码登录 Red Hat SSO。或者，您可能需要首先创建新帐户。
登录 Red Hat SSO 后，您可以从登录页面中选择使用您的红帽帐户对 Quay.io 进行身份验证。

其他资源

- 如需更多信息，请参阅 Quay.io Now 支持 Red Hat Single Sign On 。

### 4.1.2. 探索 Quay.io

Quay.io **Explore** 页面是一个宝贵的 hub，允许用户将大量容器镜像、应用程序和存储库集合，由 Quay.io 社区共享。通过直观且用户友好的设计，**Explore** 页面提供了强大的搜索功能，让用户能够轻松发现容器化的应用程序和资源。

### 4.1.3. 尝试 Quay.io （已弃用）

**注意**

Red Hat Quay 指南当前已弃用，并在 v2 UI 正式发布(GA)时删除。

Quay.io 教程 页面为用户提供 Quay.io 容器 registry 服务。点 **Continue Tutorial** 用户了解如何在 Quay.io 上执行以下功能：

- 通过 Docker CLI 登录 Quay Container Registry
- 启动容器
- 从容器创建镜像
- 将存储库推送到 Quay Container Registry
- 查看软件仓库
- 设置构建触发器
- 更改存储库的权限

### 4.1.4. 关于 Quay.io 定价的信息

除了免费的层外，Quay.io 还提供一些已增强优势的付费计划。

Quay.io 定价 页面提供有关 Quay.io 计划及每个计划的相关价格的信息。每个层的成本可在 定价 页面中找到。所有 Quay.io 计划都包括以下优点：

- 持续集成
- 公共软件仓库
- 机器人帐户
- 团队
- SSL/TLS 加密

- 日志记录和审计
- 发票历史记录

Quay.io 订阅由 Stripe 支付处理平台处理。注册 Quay.io 需要一个有效的信用卡。

若要注册 Quay.io，请使用以下流程：

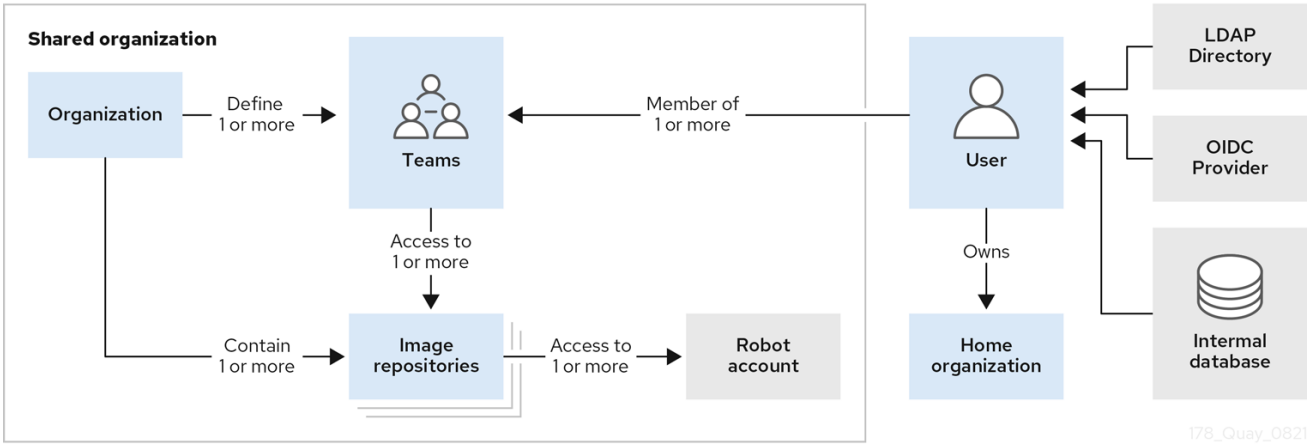
流程

1. 导航到 Quay.io **Pricing** 页面。
2. 决定一个计划，如 **Small**，然后单击 **Buy Now**。您将被重定向到 **Create New Organization** 页面。输入以下信息：
   - 机构名称
   - 机构电子邮件
   - 可选。如果您希望一个大于 **Small** 的计划，您可以选择不同的计划。
3. 解析 captcha，然后选择 **Create Organization**。
4. 您将被重定向到 Stripe。输入以下信息：
   - 卡信息，包括 **MM/YY** 和 **CVC**
   - 卡中的名称
   - 国家或地区或地区
   - **ZIP**（如果适用）
   - 如果要保存您的信息，请选中该框。
   - 电话号码
5. 填写完所有框后，点 **Subscribe**。

# 第 5 章 用户和机构

在创建存储库以在 Quay.io 中包含容器镜像之前，您应该考虑如何构建这些存储库。使用 Quay.io 时，每个存储库都需要一个与组织或用户 的连接。 *这种关系定义了存储库的所有权和权限。*

## 5.1. 租期模型




- ***机构** 提供了一种在不属于单个用户的通用命名空间下共享存储库的方法。相反，这些存储库属于共享设置中的多个用户，如公司。*
- *团队 为组织提供了一种方式来委派权限。可以在全局级别（例如，跨所有存储库）或特定存储库设置权限。它们也可以为特定的集合或用户组设置。*
- ***用户可以通过** Web UI 或使用 Podman 或 Docker 等客户端登录注册表，使用各自的登录命令，例如 **$ podman login**。每个用户都自动获得一个用户命名空间，例如 < **quay-server.example.com>/<user>/<username>** 或 **quay.io/<username>**。*
- ***机器人帐户** 为非人用户（如管道工具）提供对存储库的自动访问。机器人帐户与 OpenShift Container Platform **服务帐户** 类似。通过添加类似您其他用户或团队的该帐户，可以为存储库中授予机器人帐户的权限。*

## 5.2. 登录到 QUAY

*Quay.io 的用户帐户代表个人，具有对平台的功能和功能的验证访问权限。通过此帐户，您可以创建和管理存储库、上传和检索容器镜像，以及控制这些资源的访问权限。此帐户是整理和监督 Quay.io 中的容器镜像管理的基础。*

***注意***

*并非所有 Quay.io 上的功能都要求用户登录。例如，您可以匿名从 Quay.io 拉取镜像，而无需登录，只要您拉取的镜像来自公共存储库。*

*用户有两个用于登录 Quay.io 的选项：*

- *通过 Quay.io 登录。*
  *此选项为用户提供了传统的 UI，以及用户 beta UI 环境的选项，它遵循 PatternFly UI 原则。*
- *通过 Red Hat Hybrid Cloud Console 登录。*

这个选项使用 Red Hat SSO 进行身份验证，且是红帽的公共托管服务。这个选项 始终 要求用户登录。与其他受管服务一样，Red Hat Hybrid Cloud Console 上的 Quay 通过遵循 PatternFly UI 原则来增强用户体验。

在 Red Hat Hybrid Cloud Console 上直接使用 Quay.io 和 Quay 之间的区别，包括免费层上的用户。无论您直接使用 Quay.io，在 Hybrid Cloud Console 上，需要登录的功能（如推送到存储库）都使用您的 Quay.io 用户名规格。

### 5.2.1. 登录到 Quay.io

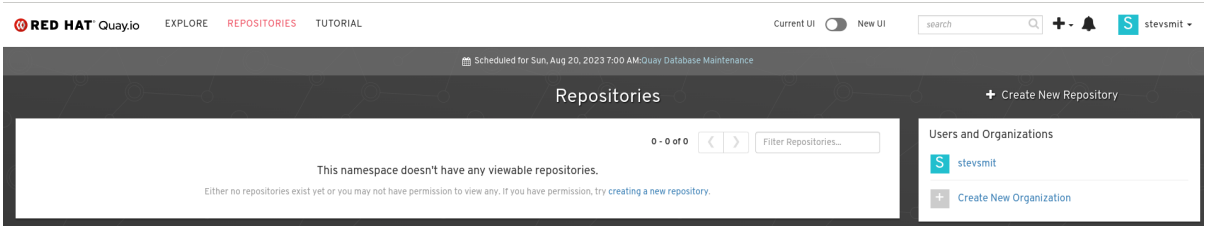
使用以下步骤登录到 Quay.io。

#### 先决条件

- 您已创建了红帽帐户和 Quay.io 帐户。如需更多信息，请参阅"创建 Quay.io 帐户"。

#### 流程

1. 导航到 Quay.io。
2. 在导航窗格中，选择 **Sign In** 并使用您的红帽凭证登录。
3. 如果这是您首次登录，则必须确认自动生成的用户名。单击 **Confirm Username** 以登录。您将被重定向到 Quay.io 存储库登录页面。



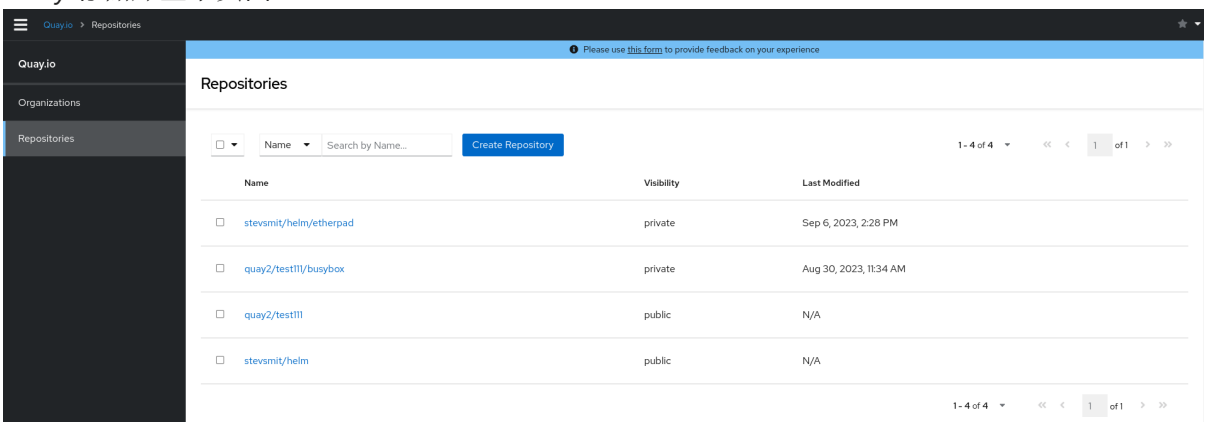

### 5.2.2. 通过混合云控制台登录到 Quay

#### 先决条件

- 您已创建了红帽帐户和 Quay.io 帐户。如需更多信息，请参阅"创建 Quay.io 帐户"。

#### 流程

1. 导航到 Red Hat Hybrid Cloud Console 上的 Quay，并使用您的红帽帐户登录。您将被重定向到 Quay 存储库登录页面：



## 5.3. 创建软件仓库

存储库提供用于存储一组相关容器镜像的中央位置。这些镜像可用于以标准化的格式构建应用程序及其依赖项。

仓库按命名空间组织。每个命名空间可以有多个软件仓库。例如，您可能有一个个人项目的命名空间、一个用于公司的命名空间，或针对您所在机构的特定团队有一个命名空间。

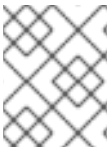
通过付费计划，Quay.io 为用户提供其存储库的访问控制。用户可以公开存储库，这意味着任何人都可以拉取或下载，或者用户可从中拉取或下载镜像，或者用户可以对其进行私有，并限制对授权用户或团队的访问。

**注意**

Quay.io 的空闲层不允许使用私有存储库。您必须升级到 Quay.io 的付费层来创建私有存储库。如需更多信息，请参阅"与 Quay.io 定价相关的信息"。

在 Quay.io 中创建存储库的方法有两种：使用相关 **docker** 或 **podman** 命令推送镜像，或者使用 Quay.io UI。

如果您在不先在 UI 上创建存储库的情况下通过命令行界面(CLI)推送镜像，则创建的存储库将设置为 **Private**，无论您拥有的计划是什么。



**注意**

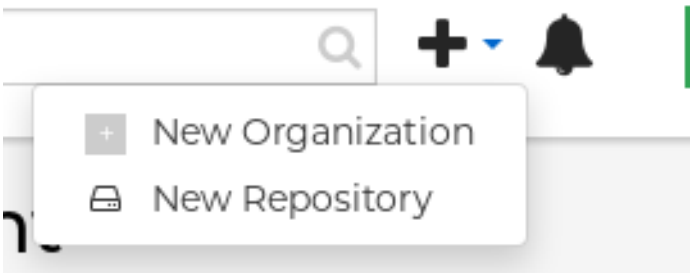
建议您在推送镜像前在 Quay.io UI 上创建存储库。Quay.io 检查计划状态，如果计划未激活，则不允许创建私有存储库。

### 5.3.1. 使用 UI 创建镜像存储库

使用以下步骤使用 Quay.io UI 创建存储库。

**流程**

1. 通过 Web UI 登录您的用户帐户。
2. 在 Quay.io 登录页面上，单击 **Create New Repository**。或者，您可以点 **+** 图标 → **New Repository**。例如：




3. 在 **Create New Repository** 页面中：
   a. 将 **Repository Name** 附加到您的用户名或您要使用的机构中。

**重要**

不要在您的仓库名称中使用以下词语：* **build** * **trigger** * **tag**

当这些词语用于存储库名称时，用户无法访问存储库，且无法永久删除存储库。尝试删除这些软件仓库会返回以下错误： **Failed to delete repository <repository_name>, HTTP404 - Not Found。**

b. 可选。单击 **Click to set repository description**，以添加存储库的描述。

c. 根据您的需要，点 **Public** 或 **Private**。

d. 可选。选择所需的存储库初始化。

4. 单击 **Create Private Repository** 以创建新的空存储库。

### 5.3.2. 使用 CLI 创建镜像存储库

使用正确的凭证时，您可以使用 Quay.io 实例中尚不存在的 Docker 或 Podman 将镜像推送到 存储库。推送镜像指的是将容器镜像从本地系统或开发环境上传到容器 registry （如 Quay.io）的过程。将镜像推送到 Quay.io 后，会创建一个存储库。

如果您在不先在 UI 上创建存储库的情况下通过命令行界面(CLI)推送镜像，则创建的存储库将设置为 **Private**，无论您拥有的计划是什么。

**注意**

建议您在推送镜像前在 Quay.io UI 上创建存储库。Quay.io 检查计划状态，如果计划未激活，则不允许创建私有存储库。

通过推送镜像来创建镜像存储库。

**先决条件**

- 您已下载并安装 **podman** CLI。
- 您已登录到 Quay.io。
- 您已拉取了一个镜像，如 busybox。

**流程**

1. 从示例 registry 中拉取示例页面。例如：

```
$ podman pull busybox
```

**输出示例**

```
Trying to pull docker.io/library/busybox...
Getting image source signatures
Copying blob 4c892f00285e done
Copying config 22667f5368 done
```

```
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2. 使用新存储库和镜像名称标记本地系统上的镜像。例如：

```
$ podman tag docker.io/library/busybox quay.io/quayadmin/busybox:test
```

3. 将镜像推送到 registry。执行此步骤，您可以使用浏览器在存储库中查看标记的镜像。

```
$ podman push --tls-verify=false quay.io/quayadmin/busybox:test
```

**输出示例**

```
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
```

## 5.4. 管理对软件仓库的访问

作为 Quay.io 用户，您可以创建自己的存储库，并使其可以被属于您的实例的其他用户访问。或者，您可以创建特定的组织，以允许根据定义的团队访问存储库。

在 User 和 Organization 存储库中，您可以通过创建与 Robot 帐户关联的凭证来允许访问这些存储库。机器人帐户使得各种容器客户端（如 Docker 或 Podman）很容易访问您的存储库，而无需客户端具有 Quay.io 用户帐户。

### 5.4.1. 允许访问用户软件仓库

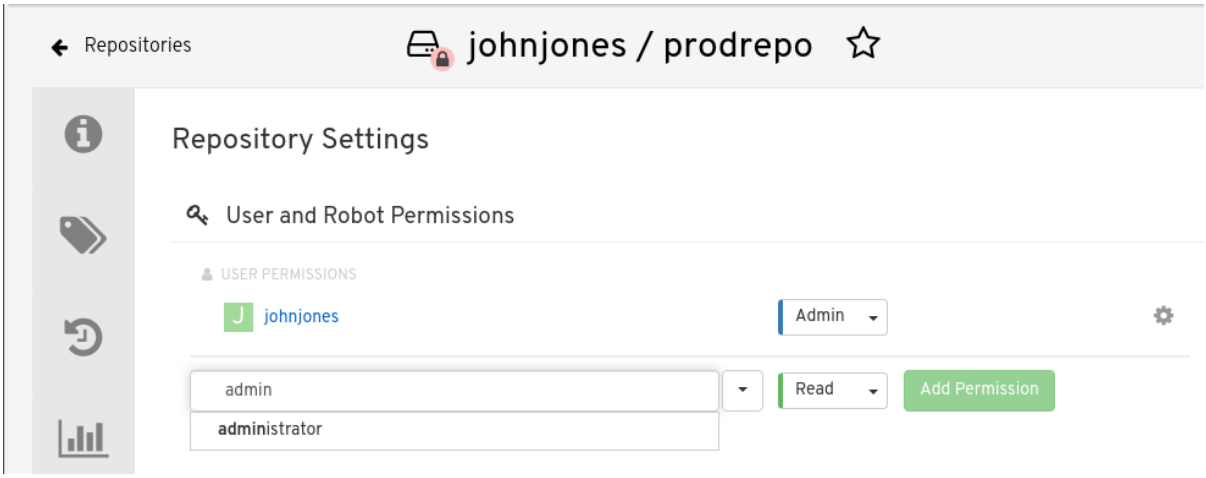
在用户命名空间中创建存储库时，您可以将对该存储库的访问权限添加到用户帐户或通过 Robot Accounts 添加。

#### 5.4.1.1. 允许用户访问用户存储库

使用以下步骤允许访问与用户帐户关联的存储库。

**流程**

1. 使用您的用户帐户登录到 Quay.io。
2. 在用户命名空间下选择一个将在多个用户间共享的存储库。
3. 在导航窗格中选择 **Settings**。
4. 键入您要授予存储库访问权限的用户名。当您键入时，名称应当为。例如：




5. 在权限框中，选择以下之一：
   - **阅读**。允许用户从存储库查看和拉取。
   - **写入**。允许用户查看存储库、从存储库拉取镜像或将镜像推送到存储库。
   - **管理员**.为用户提供存储库的所有管理设置，以及所有 **读和写** 权限。
6. 选择 **Add Permission** 按钮。用户现在具有分配的权限。
7. 可选。您可以选择 **Options** 图标，然后选择 **Delete Permission** 来删除或更改存储库的权限。

#### 5.4.1.2. 允许机器人访问用户存储库

机器人帐户用于设置 Quay.io 注册表中存储库的自动化访问权限。它们与 OpenShift Container Platform 服务帐户类似。

设置 Robot 帐户结果如下：

- 生成与 Robot 帐户关联的凭据。
- 可识别 Robot 帐户可以从中推送和拉取镜像的存储库和镜像。
- 可以复制和粘贴生成的凭据，以用于不同的容器客户端，如 Docker、Podman、Kubernetes、Msos 等，以访问每个定义的存储库。

每个 Robot 帐户都仅限于单个用户命名空间或机构。例如，Robot 帐户可以为用户提供 **jsmith** 的所有存储库的访问权限。但是，它无法提供对不在用户存储库列表中的存储库的访问。

使用以下步骤设置允许访问您的存储库的 Robot 帐户。

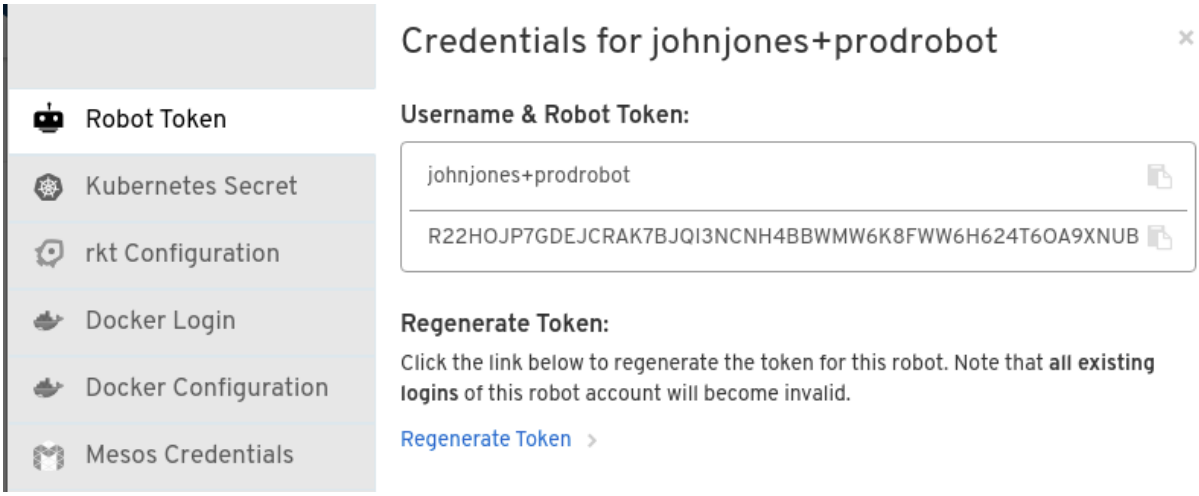
**流程**

1. 在 **Repositories** 登录页面上，单击用户的名称。
2. 在导航窗格上，单击 **Robot Accounts**。
3. 单击 **Create Robot Account**。
4. 为您的 Robot 帐户提供名称。
5. 可选。为您的 Robot 帐户提供描述。

6. 单击 **Create Robot Account**。Robot 帐户的名称成为您的用户名以及机器人的名称的组合，如 **jsmith+robot**
7. 选择您要与 Robot 帐户关联的存储库。
8. 将 Robot 帐户的权限设置为以下之一：
   - **None**。Robot 帐户没有存储库的权限。
   - **阅读**。Robot 帐户可以从存储库查看和拉取。
   - **写入**。Robot 帐户可以从存储库读取（拉取）并写入(push)到存储库。
   - **管理员**.从存储库拉取和推送到存储库的完整访问权限，以及执行与存储库关联的管理任务。
9. 点 **Add permissions** 按钮应用设置。
10. 在 **Robot Accounts** 页面上，选择 Robot Account 以查看该人的凭据信息。
11. 在 **Robot Account** 选项下，单击 **Copy to Clipboard**，为机器人复制生成的令牌。要生成新的令牌，您可以点 **Regenerate Token**。

**注意**

重新生成令牌会使此机器人的任何以前的令牌无效。




12. 使用以下方法获取生成的凭证：
    - **Kubernetes Secret**: 选择此项以 Kubernetes pull secret yaml 文件的形式下载凭证。
    - **rkt Configuration** ：选择此项以 **.json** 文件的形式下载 rkt 容器运行时的凭证。
    - **docker Login** ：选择此项以复制包含凭据的完整 **docker login** 命令行。
    - **docker Configuration** ：选择此项以下载用作 Docker **config.json** 文件的文件，以将凭证永久存储在客户端系统上。
    - **Mesos Credentials**: 选择此项下载一个 tarball，它提供可在 Mesos 配置文件的 URI 字段中标识的凭证。

### 5.4.2. 机构软件仓库

创建机构后，您可以将一组存储库直接关联到该机构。机构的存储库与基本存储库不同，组织旨在通过一组用户设置共享存储库。在 Quay.io 中，用户组可以是 团队，也可以是具有相同权限的用户集合，也可以是 单独的用户。

有关机构的其他有用信息包括：

- 您不能将机构嵌入到另一个机构中。要从属一个机构，您可以使用团队。
- 机构无法直接包含用户。您必须首先添加一个团队，然后为每个团队添加一个或多个用户。

**注意**

单个用户可以添加到机构内部的特定存储库中。因此，这些用户不是 **Repository Settings** 页面中任何团队的成员。**Teams 和 Memberships** 页面上的 **Collaborators View** 显示有权直接访问机构中特定存储库的用户，而无需成为该组织的一部分。

- 团队 可以在机构中设置，就像使用存储库和相关镜像的成员一样，或者作为具有特殊权限的管理员来管理组织的管理员。

#### 5.4.2.1. 创建机构

使用以下步骤创建机构。

**流程**

1. 在 **Repositories** 登录页面上，单击 **Create New Organization**。
2. 在 **Organization Name** 下，输入至少 2 个字符长且小于 225 个字符的名称。
3. 在 **组织电子邮件** 下，输入与您帐户电子邮件不同的电子邮件。
4. 为您的组织选择一个计划，选择免费计划，或者选择付费计划之一。
5. 点 **Create Organization** 以完成创建。

##### 5.4.2.1.1. 使用 API 创建另一个机构

您可以使用 API 创建另一个机构。为此，您必须使用 UI 创建了第一个机构。您还必须已生成 OAuth 访问令牌。

使用以下步骤使用 Red Hat Quay API 端点创建另一个机构。

**先决条件**

- 您至少已使用 UI 创建了一个机构。
- 您已生成了 OAuth 访问令牌。如需更多信息，请参阅"创建 OAuth 访问令牌"。

**流程**

1. 输入以下命令创建一个名为 **data.json** 的文件：

```
$ touch data.json
```

2. 在文件中添加以下内容，这将是新机构的名称：

```
{"name":"testorg1"}
```

3. 输入以下命令使用 API 端点创建新机构，传递您的 OAuth Access Token 和 Red Hat Quay registry 端点：

```
$ curl -X POST -k -d @data.json -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" http://<quay-server.example.com>/api/v1/organization/
```

输出示例

```
"Created"
```

#### 5.4.2.2. 将团队添加到机构

为您的机构创建团队时，您可以选择团队名称，选择哪些存储库可供团队使用，并决定对团队的访问权限级别。

使用以下步骤为您的机构创建团队。

**先决条件**

- 您已创建了一个机构。

**流程**

1. 在 Repositories 登录页面上，选择要向其添加团队的组织。
2. 在导航窗格中，选择 Teams 和 Membership。默认情况下，**所有者** 团队存在，具有创建该组织的用户的 Admin 特权。
3. 单击 Create New Team。
4. 输入新团队的名称。请注意，团队必须以小写开头。它还可以使用小写字母和数字。不允许大写字母或特殊字符。
5. 点 Create team。
6. 点要重定向到团队页面的 **团队** 名称。在这里，您可以添加团队的描述，并添加团队成员，如注册的用户、机器人或电子邮件地址。如需更多信息，请参阅"将用户添加到团队"。
7. 单击 No repositories 文本，以调出可用存储库的列表。选择您要提供团队访问权限的每个存储库的方框。
8. 选择您希望团队具有的适当权限：
   - **None**。团队成员没有对存储库的权限。
   - **阅读**。团队成员可以从存储库查看和拉取。
   - **写入**。团队成员可以从存储库读取（拉取）并写入（推送）到存储库。
   - **管理员**.从存储库拉取和推送到存储库的完整访问权限，以及执行与存储库关联的管理任务。

9. 点 **Add permissions** 保存团队的存储库权限。

#### 5.4.2.3. 设置团队角色

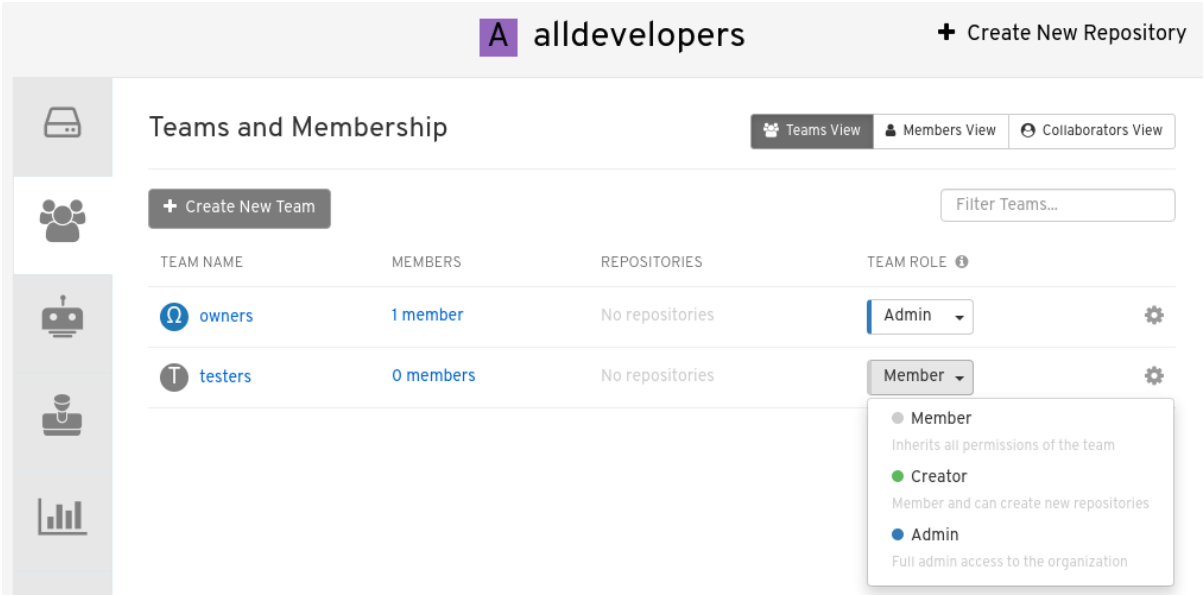
添加团队后，您可以在机构中设置该团队的角色。

**先决条件**

- 您已创建了团队。

**流程**

1. 在 **Repository** 登录页面上，点您的机构名称。
2. 在导航窗格中，单击 **Teams 和 Membership**。
3. 选择 **TEAM ROLE** 下拉菜单，如下图所示：




4. 对于所选团队，请选择以下角色之一：
   - **成员**.继承为团队设置的所有权限。
   - **创建者**.所有成员权限，以及创建新存储库的能力。
   - **管理员**.对机构的完整管理访问权限，包括创建团队、添加成员和设置权限的能力。

#### 5.4.2.4. 将用户添加到团队

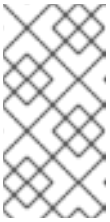
使用组织的管理特权，您可以将用户和机器人帐户添加到团队中。当您添加用户时，Quay.io 会向该用户发送电子邮件。用户会一直处于待处理状态，直到他们接受邀请。

使用以下步骤将用户或机器人帐户添加到团队中。

**流程**

1. 在 **Repository** 登录页面上，点您的机构名称。
2. 在导航窗格中，单击 **Teams 和 Membership**。

3. 选择您要将用户添加到的团队或机器人帐户。
4. 在 **Team Members** 框中，为以下之一输入信息：
   - 来自 registry 上帐户的用户名。
   - registry 上用户帐户的电子邮件地址。
   - 机器人帐户的名称。名称必须采用 <organization_name>+<robot_name> 的形式。



**注意**

机器人帐户会立即添加到团队。对于用户帐户，加入的邀请将发送给用户。在用户接受该邀请前，用户仍然处于 **INVITED TO JOIN** 状态。用户接受电子邮件邀请加入团队后，他们会从 **INVITED TO JOIN** 列表移到机构的 **MEMBERS** 列表。

## 5.5. 用户设置

**User Settings** 页面为用户提供了一种设置其电子邮件地址、密码、帐户类型、设置桌面通知、选择 avatar、删除帐户、调整 时间机器设置 以及查看计费信息的方法。

### 5.5.1. 导航到 User Settings 页面

使用以下步骤进入到 **User Settings** 页面。

**流程**

1. 在 Quay.io 上，点标头中的用户名。
2. 选择 **Account Settings**。您将被重定向到 **User Settings** 页面。

### 5.5.2. 调整用户设置

使用以下步骤调整用户设置。

**流程**

- 要更改您的电子邮件地址，请选择电子邮件地址的当前 **电子邮件地址**。在弹出窗口中，输入新的电子邮件地址，然后单击 **更改电子邮件**。在应用更改之前，将发送验证电子邮件。
- 要更改您的密码，请点 **Change password**。在两个框中输入新密码，然后单击 **Change Password**。
- 单击单个帐户或 **帐户类型** 旁边的选项，更改 **帐户类型**。在某些情况下，您可能需要在更改帐户类型前保留机构。
- 单击桌面通知旁边的选项，来调整您的桌面 **通知**。用户可以启用或禁用此功能。
- 您可以单击 **Begin deletion** 来删除帐户。如果您有一个活跃的计划，或者是您是唯一管理员的机构的成员，则无法删除帐户。您必须通过输入命名空间来确认删除。

**重要**

删除帐户不可逆，将删除所有帐户的数据，包括存储库、创建的构建触发器和通知。

- 您可以通过单击 **Time Machine** 旁边的 drop-box 来设置 时间机器 功能。此功能指定标签在收集垃圾回收前可在机器访问的时间。选择时间后，单击 **Save Expiration Time**。

### 5.5.3. 账单信息

您可以查看用户设置 上的计费信息。**在本节中，有以下信息：**

- **当前计划.本节表示您注册的当前计划 Quay.io 计划。它还显示您拥有的私有存储库量。**
- **发票.如果您正处于付费计划，您可以单击 View Invoices 以查看发票列表。**
- **接收.如果您正参与付费计划，可以选择是否收到付款电子邮件、其他用户或选择不满收据。**

# 第 6 章 使用标签

镜像标签指的是分配给特定版本或容器镜像变体的标签或标识符。容器镜像通常由代表镜像不同部分的多个层组成。镜像标签用于区分镜像的不同版本或提供有关镜像的附加信息。

镜像标签具有以下优点：

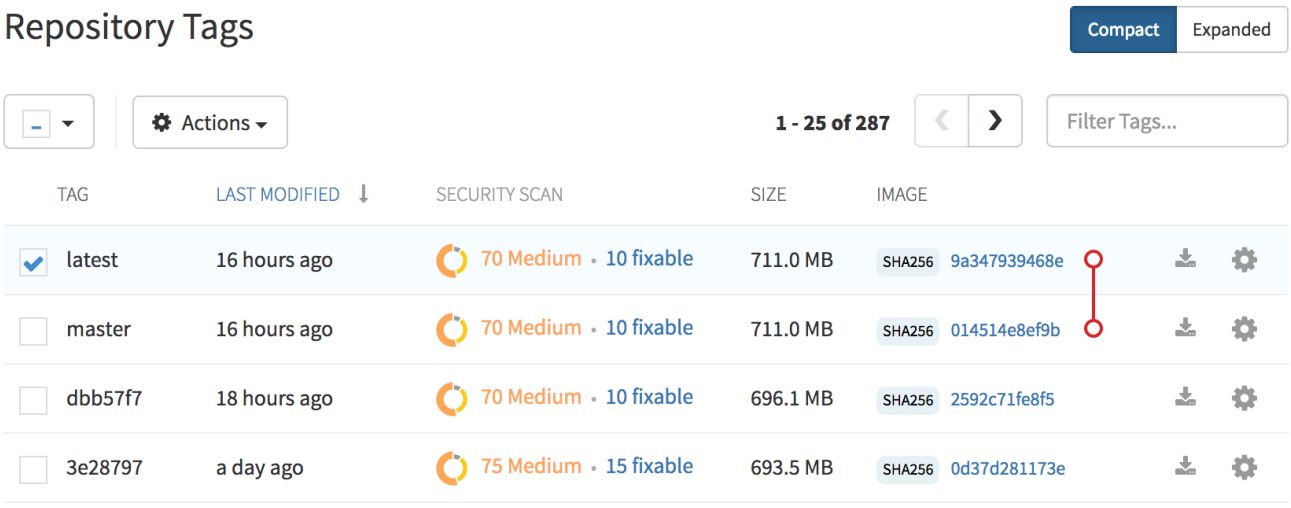
- 版本和发布：通过镜像标签，您可以表示应用程序或软件的不同版本或版本。例如，您可能有一个标记为 v1.0 的镜像，以代表更新版本的初始发行版本和 v1.1。这有助于维护镜像版本的清晰记录。
- 回滚和测试：如果您遇到新镜像版本的问题，您可以通过指定标签来轻松恢复到以前的版本。这在调试和测试阶段特别有用。
- 开发环境：在使用不同环境时镜像标签很有用。您可以使用 dev 标签进行开发版本，qa 用于质量保证测试，以及生产环境的 prod，每个标签都有自己的功能和配置。
- 持续集成/持续部署(CI/CD)：CI/CD 管道通常使用镜像标签来自动化部署过程。新的代码更改可触发使用特定标签创建新镜像，从而实现无缝更新。
- 功能分支：当多个开发人员处理不同的功能或程序错误修复时，他们可以为更改创建不同的镜像标签。这有助于隔离和测试各个功能。
- 自定义：您可以使用镜像标签来自定义具有不同配置、依赖项或优化的镜像，同时跟踪每个变体。
- 安全和补丁：发现安全漏洞时，您可以使用更新标签创建镜像的补丁版本，以确保您的系统使用最新的安全版本。
- Dockerfile 更改：如果修改 Dockerfile 或构建过程，您可以使用镜像标签来区分从之前和更新的 Dockerfile 构建的镜像。

总体而言，镜像标签提供了一种结构化的方式来管理和组织容器镜像，实现高效开发、部署和维护工作流。

## 6.1. 查看和修改标签

要查看 Quay.io 上的镜像标签，请导航到存储库，再单击 Tags 选项卡。例如：

查看和修改存储库中的标签



### 6.1.1. 向镜像添加新镜像标签

您可以向 Quay.io 中的镜像添加新标签。

流程

1. 单击标签旁边的 Settings 或 gear，再单击 Add New Tag。
2. 输入标签的名称，然后单击 Create Tag。
   新标签现在列在 Repository Tags 页面上。

### 6.1.2. 移动镜像标签

如果需要，您可以将标签移到不同的镜像。

流程

- 单击标签旁边的 Settings 或 gear 图标，再单击 Add New Tag 并输入现有标签名称。Quay.io 确认您想要移动标签，而不是添加标签。

### 6.1.3. 删除镜像标签

删除镜像标签可有效地从 registry 中删除镜像的特定版本。

要删除镜像标签，请使用以下步骤。

流程

1. 导航到存储库的 Tags 页面。
2. 单击 Delete Tag。这将删除标签及其唯一的任何镜像。

注意

根据分配给时间机器功能分配的时间，可以恢复删除镜像标签。如需更多信息，请参阅"恢复标签更改"。

#### 6.1.3.1. 查看标签历史记录

Quay.io 提供镜像及其对应镜像标签的全面历史记录。

流程

- 导航到存储库的 Tag History 页面，以查看镜像标签历史记录。

#### 6.1.3.2. 恢复标签更改

Quay.io 提供了全面的时间机器功能，允许旧镜像标签在存储库中保留，以便可以恢复对标签所做的更改。此功能允许用户恢复标签更改，如标签删除。

流程

1. 导航到存储库的 Tag History 页面。

2. 在镜像标签被更改或删除的时间表中找到点。接下来，单击 Revert 下的选项，将标签恢复到其镜像，或者单击 Permanently Delete 下的选项来永久删除镜像标签。

### 6.1.4. 通过标签或摘要获取镜像

Quay.io 提供多种使用 Docker 和 Podman 客户端拉取镜像的方法。

流程

1. 导航到存储库的 Tags 页面。
2. 在 清单 下，单击Fetch Tag 图标。
3. 当弹出框出现时，用户会看到以下选项：
   - Podman Pull （通过标签）
   - Docker Pull （通过标签）
   - Podman Pull （按摘要）
   - Docker Pull （按摘要）
     选择任何四个选项可返回相应客户端的命令，供用户拉取(pull)镜像。
4. 点 Copy Command 复制命令，该命令可用于命令行界面(CLI)。例如：

```
$ podman pull quay.io/quayadmin/busybox:test2
```

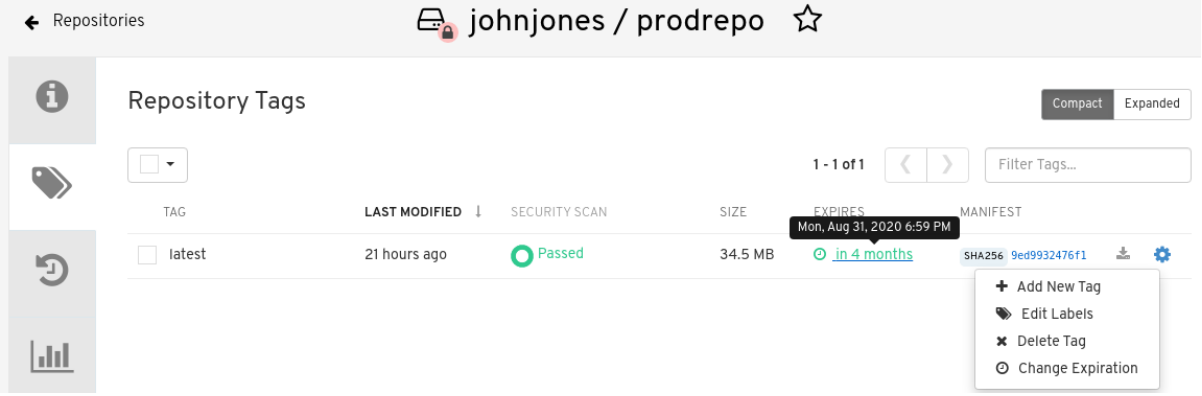
## 6.2. 标签过期

可以使用 标签到期功能，将镜像设置为在所选日期和时间时从 Quay.io 存储库过期。此功能包括以下特征：

- 当镜像标签过期时，它将从存储库中删除。如果这是特定镜像的最后一个标签，则镜像也会设置为被删除。
- 过期会根据每个标签设置。对于整个仓库，没有设置它。
- 标签过期或删除后，它不会立即从 registry 中删除。这取决于在时间 机器功能中设计的分配时间，该功能定义何时永久删除标签或垃圾收集。默认情况下，这个值设置为14 天，但管理员可以将此时间调整为多个选项之一。在进行垃圾回收前，可以恢复标签更改。

可以通过以下两种方式之一设置标签过期：

- 在创建镜像时，通过在 Dockerfile 中设置 quay.expires-after= LABEL。这会将一个时间设置为在构建镜像时过期。
- 通过选择 Quay.io UI 上的过期日期。例如：




### 6.2.1. 从 Dockerfile 设置标签过期

**使用 docker label 命令添加标签，例如 quay.expires-after=20h 会导致标签在指示的时间后自动过期。可接受小时、天或周的以下值：**

- **1h**
- **2d**
- **3w**

**过期时间从镜像推送到 registry 的时间开始。**

### 6.2.2. 从存储库设置标签过期

**可以在 Quay.io UI 中设置标签过期。**

**流程**

1. **导航到存储库，再单击导航窗格中的 Tags。**
2. **点镜像标签的 Settings 或 gear 图标，然后选择 Change Expiration。**
3. **选择提示时的日期和时间，然后选择 Change Expiration。当达到过期时间时，标签设置为从存储库中删除。**

## 6.3. 查看 CLAIR 安全扫描

**Quay.io 附带 Clair 安全扫描程序。有关 Quay.io 上的 Clair 的更多信息，请参阅"Clair 安全扫描程序"。**

**流程**

1. **导航到存储库，再单击导航窗格中的 Tags。此页面显示安全扫描的结果。**
2. **要显示有关多架构镜像的更多信息，请点 See Child Manifests 查看扩展视图中的清单列表。**
3. **点 See Child Manifests 下的相关链接，例如，1 Unknown 被重定向到 Security Scanner 页面。**
4. **Security Scanner 页面提供了标签的信息，如镜像易受哪些 CVE，以及您可能可用的补救选项。**

*注意*

*镜像扫描仅列出 Clair 安全扫描程序发现的漏洞。用户对漏洞的做了哪些操作是用户所说的。*

# 第 7 章 查看和导出日志

为 Quay.io 中的所有存储库和命名空间收集活动日志。

查看 Quay.io 的使用日志可为操作和安全目的提供宝贵见解和优势。使用日志可能会显示以下信息：

- 资源规划 ：使用日志可在镜像拉取、推送和您的 registry 整个流量数量上提供数据。
- 用户活动 ：日志可帮助您跟踪用户活动，显示哪些用户正在访问并与 registry 中的镜像交互。这对审计、了解用户行为和管理访问控制非常有用。
- 使用模式 ：通过研究使用情况模式，您可以深入了解哪些镜像被常见，使用哪些版本，以及很少访问哪些镜像。这些信息可帮助对镜像维护和清理工作进行优先排序。
- 安全审计 ：使用日志可让您跟踪谁正在访问镜像以及时间。这对安全审计、合规性以及调查任何未授权或可疑活动至关重要。
- Image Lifecycle Management: Logs 可以显示哪些镜像被拉取、推送和删除。此信息对于管理镜像生命周期至关重要，包括弃用旧镜像并确保只使用授权的镜像。
- 合规性 和规范要求 ：许多行业具有规定跟踪和审核对敏感资源的访问权限的合规性要求。使用日志可帮助您演示遵守此类法规。
- 识别异常行为 ：使用 日志中异常或异常模式可以指示潜在的安全漏洞或恶意活动。监控这些日志可帮助您更有效地检测和响应安全事件。
- 趋势分析 ：随着时间的推移，使用日志可以提供趋势并深入了解您的 registry 的使用方式。这有助于您就资源分配、访问控制和镜像管理策略做出明智的决策。

访问日志文件的方法有多种：

- 通过 Web UI 查看日志.
- 导出日志以便可以在外部保存它们。
- 使用 API 访问日志条目。

要访问日志，您必须具有所选存储库或命名空间的管理权限。

**注意**

通过 API 最多提供 100 个日志结果。要收集更多结果，您必须使用本章中描述的日志导出器功能。

## 7.1. 使用 UI 查看日志

使用以下步骤使用 Web UI 查看存储库或命名空间的日志条目。

**流程**

1. 导航到您作为管理员的仓库或命名空间。
2. 在导航窗格中，选择 Usage Logs。


3. *可选。在 usage 日志页面中：*

   a. *通过将日期添加到 From 和 to 框来设置用于查看日志条目的日期范围。默认情况下，UI 会显示日志条目的最新周。*

   b. *在 Filter Logs 框中输入字符串，以显示指定关键字的日志条目。例如，您可以键入 delete 来过滤日志来显示已删除的标签。*

   c. *在 Description 下，将日志条目的箭头切换为查看更多或更小的文本，与特定日志条目相关联。*

## *7.2. 导出存储库日志*

*您可以使用 Export Logs 功能获取大量日志文件，并将它们保存到 Quay.io 之外。这个功能有以下优点和限制：*

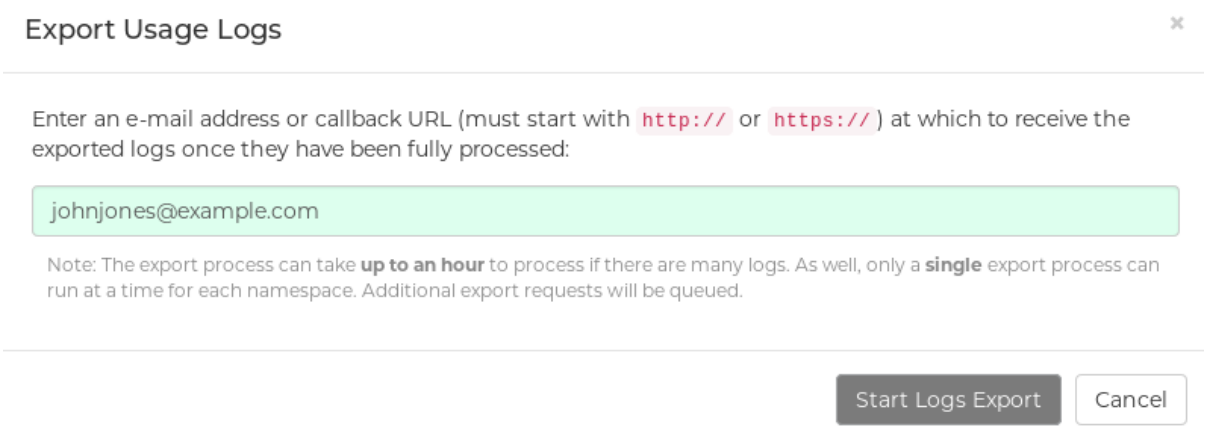
- *您可以为您要从存储库收集的日志选择日期范围。*
- *您可以通过电子邮件附加或定向到回调 URL 来请求日志发送到您。*
- *要导出日志，您必须是存储库或命名空间的管理员。*
- *为所有用户保留 30 天的日志。*
- *导出日志仅收集之前生成的日志数据。它不流传输日志记录数据。*
- *当日志被收集并供您使用时，如果想要保存该数据，应该立即复制这些数据。默认情况下，数据在一小时后过期。*

*使用以下步骤导出日志。*

*流程*

1. *选择具有管理员特权的存储库。*

2. 在导航窗格中，选择 Usage Logs。
3. 可选。如果要指定特定的日期，请在 From 和 to 框中输入范围。
4. 点 Export Logs 按钮。此时会出现 Export Usage Logs 弹出窗口，如下所示




5. 输入电子邮件地址或回调 URL 以接收导出的日志。对于回调 URL，您可以使用一个指定域的 URL，例如 <webhook.site>。
6. 选择 Start Logs Export 以开始进程来收集所选日志条目。根据要收集的日志记录数据量，这可能需要任何几分钟到几分钟才能完成。
7. 日志导出完成后，会出现以下两个事件之一：
   - 会收到一封电子邮件，提醒您请求的导出日志条目可用。
   - 从 Webhook URL 返回日志导出请求的成功状态。另外，还会提供一个到导出的数据的链接，供您删除以下载日志。

# 第 8 章 构建容器镜像

构建容器镜像涉及为容器化应用程序创建蓝图。蓝图依赖于其他公共存储库中的基础镜像，以定义如何安装和配置应用程序。

**注意**

因为蓝图依赖于其他公共存储库中的镜像，所以它们可能会受速率限制。因此，您的构建可能会失败。

Quay.io 支持构建 Docker 和 Podman 容器镜像的功能。此功能对于依赖容器和容器编配的开发人员和组织而言非常宝贵。

在 Quay.io 上，此功能可在免费和付费的层计划中正常工作。

**注意**

Quay.io 限制一个用户可以一次提交的同步构建数量。

## 8.1. 构建上下文

使用 Docker 或 Podman 构建镜像时，将指定一个目录成为 构建上下文。对于手动构建和构建触发器，这也是如此，因为 Quay.io 创建的 Build 与在本地机器上运行 **docker build** 或 **podman build** 不同。

Quay.io 构建上下文始终在构建设置 的子目录 中指定，如果未指定目录，则回退到 Build 源的根目录。

触发构建时，Quay.io 构建 worker 将 Git 存储库克隆到 worker 机器，然后在执行构建前输入 Build 上下文。

对于基于 **.tar** 归档的构建，构建 worker 会提取存档并输入构建上下文。例如：

提取的构建存档

```
example
├── .git
├── Dockerfile
├── file
└── subdir
    └── Dockerfile
```

假设您 Extracted Build 存档 是目录结构获取名为 example 的 Github 存储库。如果在 Build 触发器设置中没有指定子目录，或者在手动启动构建时，构建将在示例目录中运行。

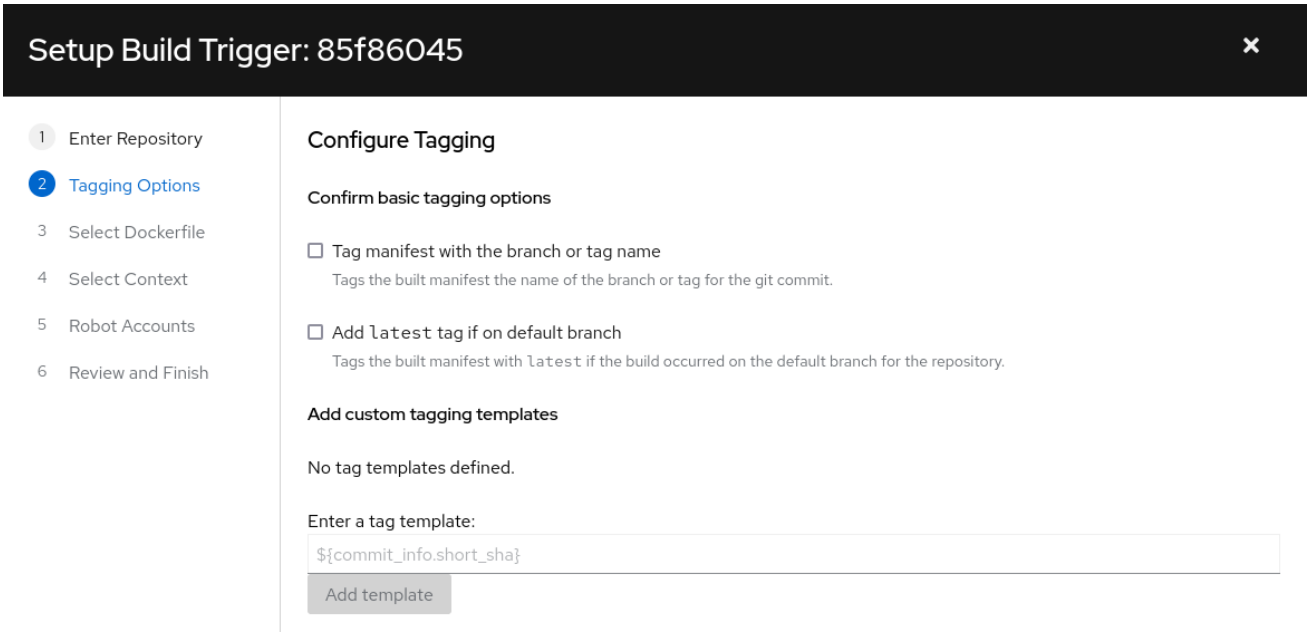
如果在 Build 触发器设置中指定子目录，如 **subdir**，则只有其中的 Dockerfile 可用于构建。这意味着您无法使用 Dockerfile 中的 **ADD** 命令来添加文件，因为它位于构建上下文之外。

与 **Docker Hub** 不同，**Dockerfile** 是 **Quay.io** 上构建上下文的一部分。因此，它不能出现在 **.dockerignore** 文件中。

## 8.2. 构建触发器的标签命名

自定义标签可用于 Quay.io。

一个选项是包含作为每个构建的镜像标签分配的任何字符串字符。或者，您可以在构建触发器的 Configure Tagging 部分中使用以下标签模板，以使用每个提交中的信息标记镜像：




- ${commit}: 发布的提交的完全 SHA
- ${parsed_ref.branch}: Branch 信息（如果可用）
- ${parsed_ref.tag}: Tag 信息（如果可用）
- ${parsed_ref.remote}: 远程名称
- ${commit_info.date}: 提交时的日期
- ${commit_info.author.username}: Username of the author of the commit

- ${commit_info.short_sha}: 提交 SHA 的第一个 7 个字符
- ${committer.properties.username}: committer 的 Username

此列表未完成，但 包含了用于标记目的的最有用的选项。您可以在此页面中找到完整的标签模板模式。https://github.com/quay/quay/blob/abfde5b9d2cf7d7145e68a00c9274011b4fe0661/buildtrigger/basehandler.py#L96-L195

如需更多信息，请参阅为 Red Hat Quay 和 Quay.io 构建触发器中设置自定义标签模板

### 8.3. 跳过源控制触发的构建

要指定 Quay.io 构建系统应忽略提交，请在提交消息的任何位置添加文本 [skip build] 或 [build skip]。

### 8.4. 查看和管理构建

存储库构建可以在 Quay.io UI 上查看和管理。

**流程**

1. 导航到 Quay.io 并选择存储库。
2. 在导航窗格中，选择 Builds。

### 8.5. 创建新构建

默认情况下，Quay.io 用户可以开箱即用创建新的构建。

**先决条件**

- 您已导航到存储库的 Builds 页面。

**流程**

1. 在 Builds 页面上，单击 Start New Build。

2. 出现提示时，单击 Upload Dockerfile 以上传 Dockerfile 或包含根目录中的 Dockerfile 的存档。

3. 单击 Start Build。

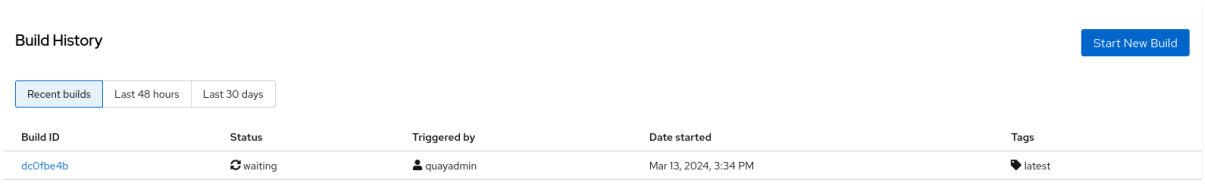
注意

- 目前，在手动启动构建时，用户无法指定 Docker 构建上下文。
- 目前，Red Hat Quay v2 UI 不支持 BitBucket。

4. 您将被重定向到 Build，可以实时查看。等待 Dockerfile 构建完成并推送。

5. 可选。您可以点 Download Logs 下载日志，或 Copy Logs 来复制日志。

6. 点 back 按钮返回到 Repository Builds 页面，您可以在其中查看 Build History。




## 8.6. 构建触发器

构建触发器会在满足触发的条件时调用构建，如源控制推送、创建 webhook 调用 等等。

### 8.6.1. 创建构建触发器

使用以下步骤使用自定义 Git 存储库创建构建触发器。

注意

以下流程假设您没有在 config.yaml 文件中包含 Github 凭证。

先决条件

- 您已导航到存储库的 Builds 页面。

流程

1. 在 Builds 页面中，点 Create Build Trigger。

2. 选择所需的平台，如 Github、BitBucket、Gitlab 或使用自定义 Git 存储库。在本例中，我们使用来自 Github 的自定义 Git 存储库。

3. 输入自定义 Git 存储库名称，例如 git@github.com:<username>/<repo>.git。然后，单击 Next。

4. 提示时，通过选择其中一个或两者都选项来配置标记选项：

   - 使用分支或标签名称 标记清单。选择此选项时，构建的清单会标记 git 提交的分支或标签的名称。

   - 如果在默认分支 上，添加 latest 标签。在选择此选项时，如果构建发生在存储库的默认分支上，则构建带有 latest 的清单会被标记。

   另外，您可以添加自定义标记模板。您可以在此处输入多个标签模板，包括将提交中的短 SHA ID、时间戳、作者名称、提交者和分支名称用作标签。如需更多信息，请参阅"构建触发器的标签命名"。

   配置标记后，单击 下一步。

5.

出现提示时，选择在调用触发器时要构建的 Dockerfile 的位置。如果 Dockerfile 位于 git 存储库的根目录并命名 Dockerfile，请输入 /Dockerfile 作为 Dockerfile 路径。然后，单击 Next。

6. 出现提示时，选择 Docker 构建的上下文。如果 Dockerfile 位于 Git 存储库的根目录，请输入 / 作为构建上下文目录。然后，单击 Next。

7. 可选。选择可选的机器人帐户。这可让您在构建过程中拉取私有基础镜像。如果您知道没有使用私有基础镜像，您可以跳过这一步。

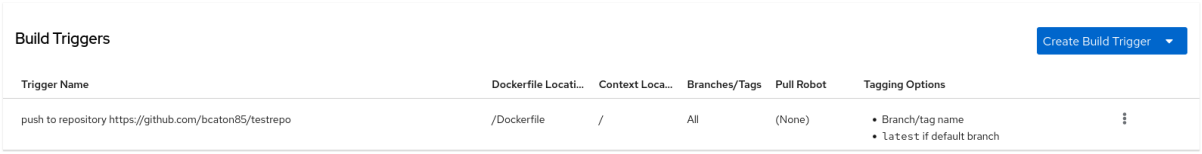
8. 点击 Next。检查任何验证警告。如有必要，请在单击 Finish 前修复问题。

9. 您收到了触发器已成功激活的警报。请注意，使用这个触发器需要以下操作：

   - 您必须为以下公钥授予 git 存储库读取访问权限。
   - 您必须将存储库设置为 POST 到以下 URL，以触发构建。

   保存 SSH 公钥，然后点 return to <organization_name>/<repository_name>。您将被重定向到存储库的 Builds 页面。

10. 在 Builds 页面中，您现在有一个 Build 触发器。例如：




### 8.6.2. 手动触发构建

可以按照以下流程手动触发构建。

**流程**

1. 在 Builds 页面中，启动新构建。

2. 出现提示时，选择 Invoke Build Trigger。

3. 点 Run Trigger Now 手动启动进程。

   构建启动后，您可以看到 Repository Builds 页面上的 Build ID。

### 8.7. 设置自定义 GIT 触发器

自定义 Git 触发器是任何 Git 服务器充当构建触发器的通用方法。它只依赖于 SSH 密钥和 Webhook 端点。其他所有都保留供用户实施。

#### 8.7.1. 创建触发器

创建自定义 Git 触发器与创建任何其他触发器类似，但以下除外：

- Quay.io 无法自动检测要用于触发器的正确 Robot 帐户。这必须在创建过程中手动完成。
- 创建触发器后还有额外的步骤。这些步骤在以下部分详细介绍。

#### 8.7.2. 自定义触发器创建设置

在创建自定义 Git 触发器时，需要额外的步骤：

1. 您必须提供创建触发器时生成的 SSH 公钥的读访问权限。

2. 您必须设置一个 webhook，它将 POST 到 Quay.io 端点来触发构建。

可以通过从 Settings 或 gear 图标选择 View Credentials 来使用密钥和 URL。

查看和修改存储库中的标签


#### *8.7.2.1. SSH 公钥访问*

***根据 Git 服务器配置，可以通过多种方式安装 Quay.io 为自定义 Git 触发器生成的 SSH 公钥。***

***例如，Git 文档描述了一个小型服务器设置，在其中将密钥添加到 $HOME/.ssh/authorize_keys 将提供对 Builders 的访问权限，以克隆存储库。对于任何未正式支持的 git 存储库管理软件，通常有一个输入密钥的位置，通常被标记为 Deploy Keys。***

#### *8.7.2.2. Webhook*

***要自动触发构建，一个必须使用以下格式将 .json 有效负载 POST 到 Webhook URL：***

***这可以通过各种不同的方法来实现，具体取决于服务器设置，但大多数情况下，可以使用 post-receive Git Hook 来完成。***

***注意***

***此请求需要一个包含 application/json 的 Content-Type 标头才能有效。***

***Webhook 示例***

```
{
  "commit": "1c002dd",                                  // required
  "ref": "refs/heads/master",                           // required
  "default_branch": "master",                           // required
  "commit_info": {                                      // optional
    "url": "gitsoftware.com/repository/commits/1234567", // required
    "message": "initial commit",                        // required
    "date": "timestamp",                                // required
    "author": {                                         // optional
      "username": "user",                               // required
      "avatar_url": "gravatar.com/user.png",            // required
      "url": "gitsoftware.com/users/user"               // required
    },
    "committer": {                                      // optional
      "username": "user",                               // required
      "avatar_url": "gravatar.com/user.png",            // required
      "url": "gitsoftware.com/users/user"               // required
    }
  }
}
```

# 第 9 章 仓库通知

Quay.io 支持在存储库中为存储库的生命周期中发生的各种事件添加通知。

## 9.1. 创建通知

使用以下步骤添加通知。

### 先决条件

- 您已创建了一个存储库。
- 您有对该存储库的管理特权。

### 流程

1. 导航到 Quay.io 上的存储库。
2. 在导航窗格中，单击 Settings。
3. 在 Events 和 Notifications 类别中，点 Create Notification 为存储库事件添加新通知。您将被重定向到 Create repository 通知 页面。
4. 在 Create repository notification 页面上，选择下拉菜单以显示事件列表。您可以为以下类型的事件选择通知：
    - 推送到存储库
    - Dockerfile 构建队列
    - Dockerfile 构建已启动

- *Dockerfile 构建 Successfully Completed*
- *Docker 构建取消*
- *发现软件包漏洞*

5. *选择事件类型后，选择通知方法。支持以下方法：*

- *Quay 通知*
- *电子邮件*
- *Webhook POST*
- *Flowdock 团队通知*
- *HipChat Room 通知*
- *Slack Room 通知*

*根据您选择的方法，必须包含其他信息。例如，如果您选择了 E-mail，则需要包含电子邮件地址和可选通知标题。*

6. *选择事件和通知方法后，单击 Create Notification。*

### *9.2. 仓库事件描述*

*以下小节详细介绍了存储库事件。*

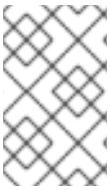
#### *9.2.1. 仓库 Push*

*将一个或多个镜像成功推送到存储库：*

```
{
  "name": "repository",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "homepage": "https://quay.io/repository/dgangaia/repository",
  "updated_tags": [
    "latest"
  ]
}
```

### 9.2.2. Dockerfile 构建队列

*以下示例是 Dockerfile 构建的响应，该构建已排队到 Build 系统。*



**注意**

响应可能会因使用可选属性的不同而有所不同。

```
{
  "build_id": "296ec063-5f86-4706-a469-f0a400bf9df2",
  "trigger_kind": "github",                                    //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                  //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "repo": "test",
  "trigger_metadata": {
    "default_branch": "master",
    "commit": "b7f7d2b948aacbe844ee465122a85a9368b2b735",
    "ref": "refs/heads/master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                           //Optional
      "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
      "date": "2019-03-06T12:48:24+11:00",
      "message": "adding 5",
      "author": {                                          //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                        //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"    //Optional
      },
```

```
      "committer": {
        "username": "web-flow",
        "url": "https://github.com/web-flow",
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
      }
    }
  },
  "is_manual": false,
  "manual_user": null,
  "homepage": "https://quay.io/repository/dgangaia/test/build/296ec063-5f86-4706-a469-
f0a400bf9df2"
}
```

#### 9.2.3. Dockerfile 构建已启动

以下示例是 **Dockerfile** 构建的响应，该构建已排队到 **Build** 系统。

**注意**

响应可能会因使用可选属性的不同而有所不同。

```
{
  "build_id": "a8cc247a-a662-4fee-8dcb-7d7e822b71ba",
  "trigger_kind": "github",                                    //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                  //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "50bc599",
  "trigger_metadata": {                                        //Optional
    "commit": "50bc5996d4587fd4b2d8edc4af652d4cec293c42",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                           //Optional
      "url": "https://github.com/dgangaia/test/commit/50bc5996d4587fd4b2d8edc4af652d4cec293c42",
      "date": "2019-03-06T14:10:14+11:00",
      "message": "test build",
      "committer": {                                           //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                          //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"   //Optional
      },
      "author": {                                              //Optional
        "username": "dgangaia",
```

```
      "url": "https://github.com/dgangaia",                    //Optional
      "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"   //Optional
     }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/a8cc247a-a662-4fee-8dcb-
7d7e822b71ba"
}
```

#### 9.2.4. Dockerfile 构建成功完成

以下示例是来自由 Build 系统成功完成的 Dockerfile 构建的响应。

注意

此事件与构建镜像或镜像的 Repository Push 事件同时发生。

```
{
  "build_id": "296ec063-5f86-4706-a469-f0a400bf9df2",
  "trigger_kind": "github",                                    //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                  //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "b7f7d2b",
  "image_id": "sha256:0339f178f26ae24930e9ad32751d6839015109eabdf1c25b3b0f2abf8934f6cb",
  "trigger_metadata": {
    "commit": "b7f7d2b948aacbe844ee465122a85a9368b2b735",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                           //Optional
      "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
      "date": "2019-03-06T12:48:24+11:00",
      "message": "adding 5",
      "committer": {                                           //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                  //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
//Optional
      },
      "author": {                                              //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                  //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
```

```
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/296ec063-5f86-4706-a469-
f0a400bf9df2",
  "manifest_digests": [

"quay.io/dgangaia/test@sha256:2a7af5265344cc3704d5d47c4604b1efcbd227a7a6a6ff73d6e4e08a27f
d7d99",

"quay.io/dgangaia/test@sha256:569e7db1a867069835e8e97d50c96eccafde65f08ea3e0d5debaf16e25
45d9d1"
  ]
}
```

#### 9.2.5. Dockerfile 构建失败

以下示例是 **Dockerfile** 构建失败的响应。

```
{
  "build_id": "5346a21d-3434-4764-85be-5be1296f293c",
  "trigger_kind": "github",                                    //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "docker_url": "quay.io/dgangaia/test",
  "error_message": "Could not find or parse Dockerfile: unknown instruction: GIT",
  "namespace": "dgangaia",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                    //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "6ae9a86",
  "trigger_metadata": {                                        //Optional
    "commit": "6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                           //Optional
      "url": "https://github.com/dgangaia/test/commit/6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
      "date": "2019-03-06T14:18:16+11:00",
      "message": "failed build test",
      "committer": {                                           //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                          //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"     //Optional
      },
      "author": {                                              //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                          //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
      }
    }
```

```
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/5346a21d-3434-4764-85be-
5be1296f293c"
}
```

#### 9.2.6. Dockerfile 构建已取消

以下示例是已取消的 Dockerfile 构建的响应。

```
{
  "build_id": "cbd534c5-f1c0-4816-b4e3-55446b851e70",
  "trigger_kind": "github",
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "cbce83c",
  "trigger_metadata": {
    "commit": "cbce83c04bfb59734fc42a83aab738704ba7ec41",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {
      "url": "https://github.com/dgangaia/test/commit/cbce83c04bfb59734fc42a83aab738704ba7ec41",
      "date": "2019-03-06T14:27:53+11:00",
      "message": "testing cancel build",
      "committer": {
        "username": "web-flow",
        "url": "https://github.com/web-flow",
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
      },
      "author": {
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/cbd534c5-f1c0-4816-b4e3-
55446b851e70"
}
```

#### 9.2.7. 检测到漏洞

以下示例是 Dockerfile 构建的响应，已检测到存储库中的漏洞。

```
{
  "repository": "dgangaia/repository",
  "namespace": "dgangaia",
  "name": "repository",
  "docker_url": "quay.io/dgangaia/repository",
  "homepage": "https://quay.io/repository/dgangaia/repository",

  "tags": ["latest", "othertag"],

  "vulnerability": {
    "id": "CVE-1234-5678",
    "description": "This is a bad vulnerability",
    "link": "http://url/to/vuln/info",
    "priority": "Critical",
    "has_fix": true
  }
}
```

### 9.3. 通知操作

#### 9.3.1. 添加了通知

通知添加到 Repository Settings 页面 的事件和通知 部分中。它们也添加到 通知 窗口中，可通过单击 Quay.io 导航窗格中的 bell 图标找到。

Quay.io 通知可以设置为作为一个整体发送 User, Team, 或 organization。

#### 9.3.2. 电子邮件通知

电子邮件将发送到描述指定事件的指定地址。电子邮件地址必须 基于每个软件仓库 进行验证。

#### 9.3.3. Webhook POST 通知

使用事件数据向指定的 URL 发出 HTTP POST 调用。有关事件数据的更多信息，请参阅"存储库事件描述"。

当 URL 为 HTTPS 时，调用从 Quay.io 设置了一个 SSL 客户端证书。此证书的验证证明调用源自 Quay.io。带有 2xx 范围内的状态代码的响应被视为成功。对任何其他状态代码的响应被视为失败，并导致重试 Webhook 通知。

#### 9.3.4. Flowdock 通知

*将消息发送到 Flowdock。*

### *9.3.5. HipChat 通知*

*将消息发布到 HipChat。*

### *9.3.6. Slack 通知*

*将消息发送到 Slack。*

# 第 10 章 开放容器项目支持

容器 registry 最初设计为支持 Docker 镜像格式中的容器镜像。为了促进除 Docker 外的其他运行时，还创建了开放容器项目(OCI)，以提供与容器运行时和镜像格式相关的标准化。大多数容器注册表都支持 OCI 标准化，因为它基于 Docker 镜像清单 V2、Schema 2 格式。

除了容器镜像外，还出现各种工件，它们不仅支持单独的应用程序，还存在 Kubernetes 平台作为一个整体的支持。这些范围包括用于安全性和监管的 Open Policy Agent (OPA)策略到 Helm chart 和 Operator，以帮助应用程序部署。

Quay.io 是一个私有容器 registry，它不仅存储容器镜像，还支持整个工具生态系统，以帮助管理容器。Quay.io 努力与 OCI 1.0 镜像和分发 规格 兼容，并支持像 Helm chart 等常用介质类型（只要它们推送了支持 OCI 的 Helm 版本），以及容器镜像的清单或层组件中的各种任意介质类型。当 registry 更加严格时，对此类 novel 介质类型的支持与以前的 Quay.io 迭代不同。由于 Quay.io 现在与更广泛的介质类型一起工作，包括之前不在其支持范围范围之外的介质类型，所以现在它比标准容器镜像格式更广泛，也只是标准容器镜像格式的修改。

除了其对 novel 介质类型的支持外，Quay.io 还可确保与 Docker 镜像的兼容性，包括 V2_2 和 V2_1 格式。与 Docker V2_2 和 V2_1 镜像的兼容性演示了 Quay.io 针对 Docker 用户提供无缝体验的承诺。此外，Quay.io 继续扩展对 Docker V1 拉取的支持，并满足可能仍然依赖此早期版本的 Docker 镜像的用户。

对 OCI 工件的支持会被默认启用。

## 10.1. HELM 和 OCI 的先决条件

Helm 简化了应用程序的打包和部署方式。Helm 使用名为 Charts 的打包格式，其中包含代表应用程序的 Kubernetes 资源。Quay.io 支持 Helm chart，只要它们是 OCI 支持的版本。

使用以下步骤预配置您的系统以使用 Helm 和其他 OCI 介质类型。

### 10.1.1. 安装 Helm

使用以下步骤安装 Helm 客户端。

流程

1. 从 Helm 发行版本页面下载最新版本的 Helm。

2. 输入以下命令解包 Helm 二进制文件：

```
$ tar -zxvf helm-v3.8.2-linux-amd64.tar.gz
```

3. 将 Helm 二进制文件移到所需位置：

```
$ mv linux-amd64/helm /usr/local/bin/helm
```

有关安装 Helm 的更多信息，请参阅安装 Helm 文档。

### 10.1.2. 升级到 Helm 3.8

对 OCI registry chart 的支持要求 Helm 已升级到至少 3.8。如果您已经下载了 Helm，且需要升级到 Helm 3.8，请参阅 Helm 升级 文档。

## 10.2. 使用 HELM CHART

使用以下示例从 Red Hat Community of practice (CoP)存储库下载并推送 etherpad 图表。

**先决条件**

- 您已登录到 Quay.io。

**流程**

1. 输入以下命令添加 chart 存储库：

```
$ helm repo add redhat-cop https://redhat-cop.github.io/helm-charts
```

2. 输入以下命令在 Chart 仓库中本地更新可用 chart 的信息：

```
$ helm repo update
```

3. 输入以下命令从存储库拉取 chart：

```
$ helm pull redhat-cop/etherpad --version=0.0.4 --untar
```

4. 输入以下命令将 chart 打包到 chart 归档中：

```
$ helm package ./etherpad
```

输出示例

```
Successfully packaged chart and saved it to: /home/user/linux-amd64/etherpad-0.0.4.tgz
```

5. 使用 helm registry 登录 Quay.io：

```
$ helm registry login quay.io
```

6. 使用 helm push 命令将 chart 推送到您的存储库：

```
helm push etherpad-0.0.4.tgz oci://quay.io/<organization_name>/helm
```

输出示例：

```
Pushed: quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad:0.0.4
Digest: sha256:a6667ff2a0e2bd7aa4813db9ac854b5124ff1c458d170b70c2d2375325f2451b
```

7. 通过删除本地副本来确保推送正常工作，然后从存储库拉取 chart：

```
$ rm -rf etherpad-0.0.4.tgz
```

```
$ helm pull oci://quay.io/<organization_name>/helm/etherpad --version 0.0.4
```

输出示例：

```
Pulled: quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad:0.0.4
Digest:
sha256:4f627399685880daf30cf77b6026dc129034d68c7676c7e07020b70cf7130902
```

### 10.3. COSIGN OCI 支持

Cosign 是一个可用于签署和验证容器镜像的工具。它使用 ECDSA-P256 签名算法和 Red Hat 简单签名有效负载格式来创建存储在 PKIX 文件中的公钥。私钥存储为加密的 PEM 文件。

Cosign 目前支持以下内容：

- 硬件和 KMS 签名
- bring-your-own PKI
- OIDC PKI
- 内置二进制透明度和时间戳服务

使用以下步骤直接安装 Cosign。

**先决条件**

- 已安装 Go 版本 1.16 或更高版本。

**流程**

1. 输入以下 go 命令直接安装 Cosign：

```
$ go install github.com/sigstore/cosign/cmd/cosign@v1.0.0
```

输出示例

```
go: downloading github.com/sigstore/cosign v1.0.0
go: downloading github.com/peterbourgon/ff/v3 v3.1.0
```

2. 输入以下命令为 Cosign 生成键值对：

```
$ cosign generate-key-pair
```

输出示例

```
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

3. 输入以下命令为键值对签名：

```
$ cosign sign -key cosign.key quay.io/user1/busybox:test
```

输出示例

```
Enter password for private key:
Pushing signature to: quay-server.example.com/user1/busybox:sha256-
ff13b8f6f289b92ec2913fa57c5dd0a874c3a7f8f149aabee50e3d01546473e3.sig
```

如果您遇到 错误： signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[] error，因为 Cosign 依赖于 ~./docker/config.json 用于授权，您可能需要执行以下命令：

```
$ podman login --authfile ~/.docker/config.json quay.io
```

输出示例

```
Username:
Password:
Login Succeeded!
```

4. 输入以下命令查看更新的授权配置：

```
$ cat ~/.docker/config.json
{
 "auths": {
  "quay-server.example.com": {
   "auth": "cXVheWFkbWluOnBhc3N3b3Jk"
  }
 }
```

## 10.4. 安装和使用 COSIGN

使用以下步骤直接安装 Cosign。

### 先决条件

- 已安装 Go 版本 1.16 或更高版本。
- 您已在 config.yaml 文件中将 FEATURE_GENERAL_OCI_SUPPORT 设置为 true。

### 流程

1. 输入以下 go 命令直接安装 Cosign：

```
$ go install github.com/sigstore/cosign/cmd/cosign@v1.0.0
```

输出示例

```
go: downloading github.com/sigstore/cosign v1.0.0
go: downloading github.com/peterbourgon/ff/v3 v3.1.0
```

2. 输入以下命令为 Cosign 生成键值对：

```
$ cosign generate-key-pair
```

输出示例

```
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

3. 输入以下命令为键值对签名：

```
$ cosign sign -key cosign.key quay.io/user1/busybox:test
```

输出示例

```
Enter password for private key:
Pushing signature to: quay-server.example.com/user1/busybox:sha256-
ff13b8f6f289b92ec2913fa57c5dd0a874c3a7f8f149aabee50e3d01546473e3.sig
```

如果您遇到 错误： signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[] error，因为 Cosign 依赖于 ~./docker/config.json 用于授权，您可能需要执行以下命令：

```
$ podman login --authfile ~/.docker/config.json quay.io
```

*输出示例*

```
Username:
Password:
Login Succeeded!
```

4. *输入以下命令查看更新的授权配置：*

```
$ cat ~/.docker/config.json
{
 "auths": {
  "quay-server.example.com": {
   "auth": "cXVheWFkbWluOnBhc3N3b3Jk"
  }
 }
```