Red Hat Quay 3.6

使用 Red Hat Quay

使用 Red Hat Quay

Last Updated: 2023-08-05

Red Hat Quay 3.6 使用 Red Hat Quay

使用 Red Hat Quay

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

了解如何使用 Red Hat Quay

目录

前言 5

第 1 章 RED HAT QUAY 中的用户和机构 6
1.1. RED HAT QUAY TENANCY 模型 6
1.2. 创建用户帐户 6
1.3. 创建机构帐户 7

第 2 章 创建软件仓库 8
2.1. 通过 UI 创建镜像存储库 8
2.2. 通过 DOCKER 或 PODMAN 创建镜像存储库 8

第 3 章 管理对软件仓库的访问 10
3.1. 允许访问用户仓库 10
3.1.1. 允许用户访问用户存储库 10
3.2. 允许访问用户存储库的机器机器 10
3.3. 允许访问机构仓库 12
3.3.1. 将团队添加到机构 12
3.3.2. 设置团队角色 13
3.3.3. 将用户添加到团队 13

第 4 章 使用标签 15
4.1. 查看和修改标签 15
4.1.1. 添加新标签到带标记的镜像 15
4.1.2. 移动标签 15
4.1.3. 删除标签 15
4.1.4. 查看标签历史记录并返回到时间 15
4.1.4.1. 查看标签历史记录 15
4.1.4.2. 返回时间 15
4.1.5. 通过标签或摘要获取镜像 16
4.2. 标签过期 16
4.2.1. 从 Dockerfile 设置标签过期 17
4.2.2. 从仓库设置标签过期 17
4.3. 安全扫描 17

第 5 章 查看和导出日志 18
5.1. 查看日志 18
5.2. 导出存储库日志 19

第 6 章 使用构建 WORKER 自动构建 DOCKERFILE 20
6.1. 架构概述 20
6.1.1. 构建管理器 20
6.1.2. 构建 worker 的 control plane 20
6.1.3. 编配器 20
6.2. OPENSHIFT 要求 20
6.3. 编配要求 20
6.4. 使用 OPENSHIFT 设置 RED HAT QUAY BUILDER 21
6.4.1. OpenShift TLS 组件 21
6.4.2. 准备 OpenShift 进行 Red Hat Quay 构建 21
6.4.3. 启用构建器，并将构建配置添加到 Red Hat Quay Configuration Bundle 23
6.5. OPENSHIFT 路由限制 25
6.6. 构建故障排除 26
6.6.1. DEBUG 配置标志 26
6.7. 设置 GITHUB 构建（可选） 27

第 7 章 构建 DOCKERFILE 28
7.1. 查看和管理构建 28
7.2. 手动启动构建 28
7.3. 构建触发器 28
7.3.1. 创建新构建触发器 28
7.3.2. 手动触发构建触发器 28
7.3.3. 构建上下文 28

第 8 章 设置自定义 GIT TRIGGER 30
8.1. 创建触发器 30
8.2. 触发器(TRIGGER-CREATION)设置 30
8.2.1. SSH 公钥访问 30
8.2.2. Webhook 31

第 9 章 跳过源控制触发的构建 32

第 10 章 设置 GITHUB 构建触发器标签 33
10.1. 了解构建触发器的标签命名 33
10.2. 为构建触发器设置标签名称 33

第 11 章 在 GITHUB 中创建 OAUTH 应用程序 35
11.1. 创建新的 GITHUB 应用程序 35

第 12 章 存储库通知 36
12.1. 仓库事件 36
12.1.1. repository Push 36
12.1.2. Dockerfile 构建队列 36
12.1.3. Dockerfile 构建已启动 37
12.1.4. Dockerfile 构建成功完成 38
12.1.5. Dockerfile 构建失败 39
12.1.6. Dockerfile 构建取消 40
12.1.7. 安全漏洞被检测到 41
12.2. 通知操作 41
12.2.1. Quay 通知 41
12.2.2. 电子邮件 41
12.2.3. Webhook POST 41
12.2.4. Flowdock 通知 41
12.2.5. HipChat 通知 41
12.2.6. Slack 通知 42

第 13 章 OCI 支持和 RED HAT QUAY 43
13.1. HELM 和 OCI 的先决条件 43
13.2. 使用 QUAY 的 HELM CHART 43
13.3. OCI 和 HELM 配置 45
13.4. COSIGN OCI 支持 RED HAT QUAY 46
13.5. 使用带有 QUAY 的 COSIGN 46
13.6. 向 QUAY 添加其他 OCI 介质类型 47
13.7. 在 QUAY 中禁用 OCI 工件 47

第 14 章 使用 RED HAT QUAY API 49
14.1. 从 QUAY.IO 访问 QUAY API 49
14.2. 创建 OAUTH 访问令牌 49
14.3. 从网页浏览器访问 QUAY API 50
14.4. 从命令行访问 RED HAT QUAY API 50
14.4.1. 获取超级用户信息 50

14.4.2. 使用 API 创建超级用户 51
14.4.3. 列出用量日志 52
14.4.3.1. 分页示例 52
14.4.4. 目录同步 55
14.4.5. 通过 API 创建存储库构建 55
14.4.6. 创建机构机器人 56
14.4.7. 触发构建 56
14.4.8. 创建私有存储库 56

前言

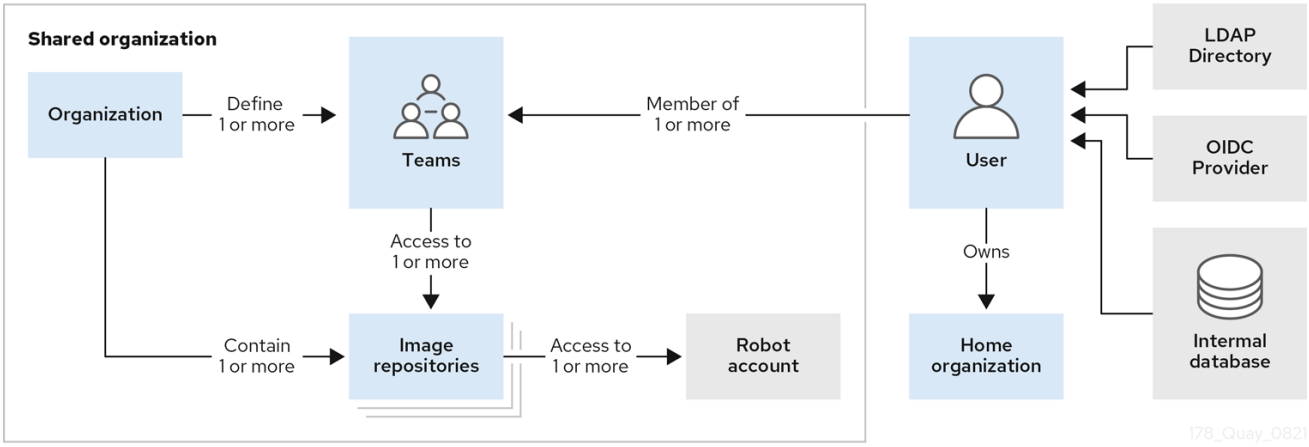
Red Hat Quay 容器镜像 registry 可让您将容器镜像存储在中央位置。作为 Red Hat Quay registry 的普通用户，您可以创建存储库来组织镜像，并选择性地添加对您控制的存储库的读取(pull)和写入(push)访问权限。具有管理特权的用户可以执行更广泛的任务，如添加用户和控制默认设置的能力。

本指南假定您部署了 Red Hat Quay，并已准备好开始使用它。

第 1 章 RED HAT QUAY 中的用户和机构

在开始创建软件仓库以在 Red Hat Quay 中保存容器镜像之前，您应该考虑如何组织这些存储库。Red Hat Quay 实例中的每个软件仓库都必须与一个机构或用户关联。

1.1. RED HAT QUAY TENANCY 模型

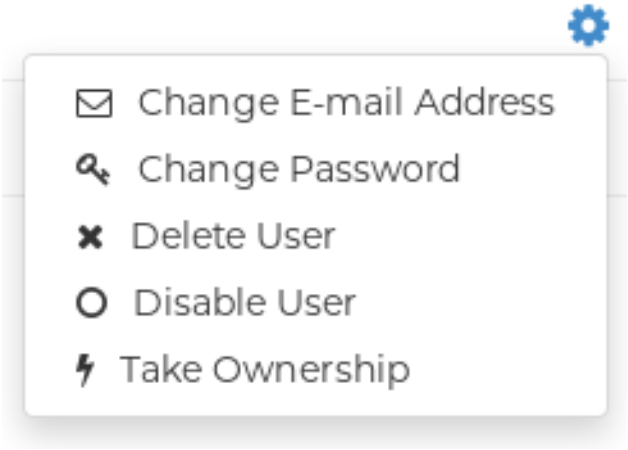


- **组织** 在通用命名空间内提供共享存储库的方式，该命名空间不属于一个用户，而是属于共享设置中的许多用户（如公司）。
- **团队** 为机构提供了一种将权限（包括全局和特定仓库以及特定存储库）设置或用户组的方法
- **用户可以通过** Quay Web UI 或客户端（如 **podman login**）登录到 registry。每个用户自动获得用户命名空间，如 **quay-server.example.com/user/<username>**
- **超级用户** 通过在用户界面中的超级用户管理员面板通过 Super User API 调用来增强访问权限和特权，以及对普通用户可见或无法访问的 Super User API 调用
- **机器人帐户** 提供对非human 用户的存储库的自动访问，如管道工具，并且与 OpenShift 服务帐户类似。通过像任何其他用户或团队一样添加该帐户，可以为存储库中的机器人帐户授予权限。

1.2. 创建用户帐户

为您的 Red Hat Quay 实例创建新用户：

1. 以超级用户（默认情况下为quay）登录到 Red Hat Quay。
2. 从主页右上角选择您的帐户名称，然后选择 Super User Admin Panel.
3. 从左列中选择用户图标。
4. 选择创建用户按钮。
5. 输入新用户的用户名和电子邮件地址，然后选择创建用户按钮。
6. 返回到 Users 页面，选择新 Username 右侧的 Options 图标。此时会出现一个下拉菜单，如下图所示：



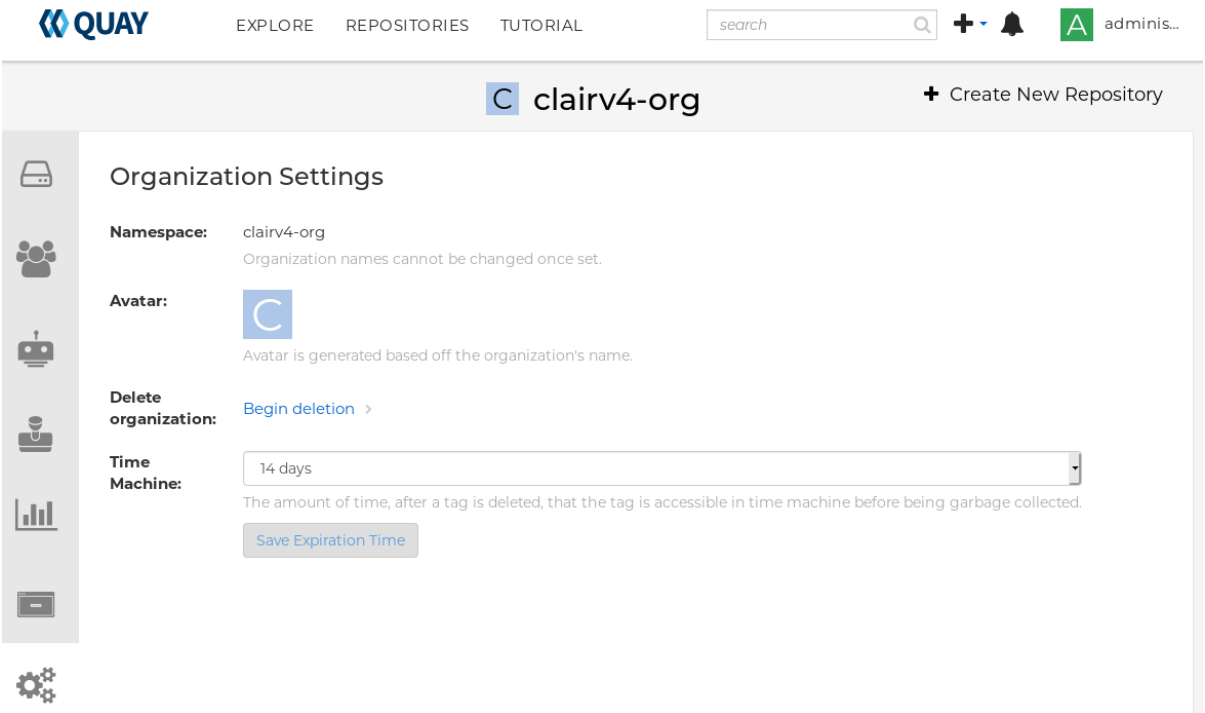
7. 从菜单中选择 Change Password。
8. 添加新密码并确认密码，然后选择"更改密码"按钮。

新用户现在可以使用该用户名和密码通过 web ui 或一些容器客户端登录。

1.3. 创建机构帐户

任何用户都可以创建自己的组织，以共享容器镜像的存储库。要创建新机构，请执行以下操作：

1. 以任何用户身份登录时，从主页右上角选择加号(+)，然后选择 New Organization。
2. 键入机构的名称。名称必须是字母数字，所有小写，以及 2 到 255 个字符的长度
3. 选择 Create Organization。新组织会出现，可以从左列中的图标开始添加存储库、团队、机器人帐户和其他功能。下图显示了使用选择设置选项卡的新组织的页面示例。



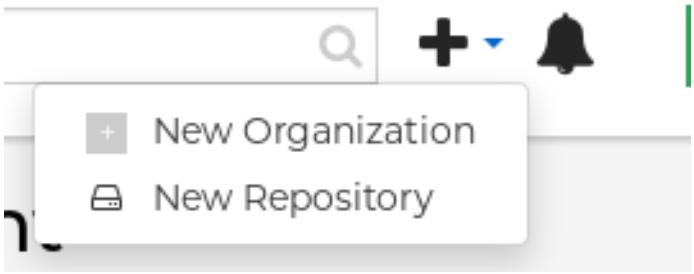
第 2 章 创建软件仓库

存储库提供存储相关容器镜像集的中央位置。在 Red Hat Quay 中创建存储库的方法有两种：通过 push（从 **docker** 或 **podman**）和 Red Hat Quay UI。它们基本上是相同的，无论您使用的是 Quay.io，还是您自己的 Red Hat Quay 实例。

2.1. 通过 UI 创建镜像存储库

在 Red Hat Quay UI 中创建用户在用户帐户下存储库：通过 Web UI 登录用户帐户。点击主页（或者与用户关联的其他页面）右上角的 + 图标，然后选择 New Repository，如下图所示：

+



1. 在出现的 Create New Repository 页面上
   - 在您的用户名中添加新存储库名称
   - 单击 Repository Description 并键入存储库的描述
   - 在 Repository Visibility 中，选择您是否要是公共仓库还是私有
   - 单击创建存储库按钮。

创建新的存储库，从空开始。您可以使用 docker pull 命令从这个仓库中拉取镜像（减去镜像名称）。

要在 Red Hat Quay UI 中创建在机构下的软件仓库：

1. 以具有组织的 Admin 或 Write 权限的用户身份登录。
2. 在 Repositories 视图中，从 Users 和 Organizations 下的右列中选择机构名称。这个机构页面会出现，类似于图 2.x 中显示的页面：
3. 单击页面右上角的 +Create New Repository。
4. 在出现的 Create New Repository 页面中：
   - 将新存储库名称添加到机构名称
   - 单击 Repository Description 并键入存储库的描述
   - 在 Repository Visibility 中，选择您是否要是公共仓库还是私有
   - 单击创建存储库按钮。

创建新的存储库，从空开始。您可以使用 docker pull 命令从这个仓库中拉取镜像（减去镜像名称）。

2.2. 通过 DOCKER 或 PODMAN 创建镜像存储库

假设您有正确的凭证，将镜像推送到 Red Hat Quay 实例中不存在的软件仓库中，会在将镜像推送到该存储库时创建该存储库。**docker** 或 **podman** 命令适用于这些示例。

1. 标记镜像：在本地系统中提供了 **docker** 或 **podman** 的镜像，使用新存储库名称和镜像名称标记该镜像。以下是将镜像推送到 Quay.io 或您自己的 Red Hat Quay 设置的示例（如 reg.example.com）。例如，将 namespace 替换为您的 Red Hat Quay 用户名或机构，将 repo_name 替换为您要创建的存储库的名称：

```
# sudo podman tag myubi-minimal quay.io/namespace/repo_name
# sudo podman tag myubi-standard reg.example.com/namespace/repo_name
```

2. 推送到适当的 registry。例如：

```
# sudo podman push quay.io/namespace/repo_name
# sudo podman push reg.example.com/namespace/repo_name
```

注意

要创建应用程序存储库，请按照您执行创建容器镜像存储库的步骤操作。

第 3 章 管理对软件仓库的访问

作为 Red Hat Quay 用户，您可以创建自己的软件仓库，并使其可以被 Red Hat Quay 实例上的其他用户访问。另外，您可以创建机构来允许基于团队访问存储库。在用户和组织存储库中，您可以通过创建与机器人帐户关联的凭据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）可以轻松访问您的仓库，而无需客户端具有 Red Hat Quay 用户帐户。

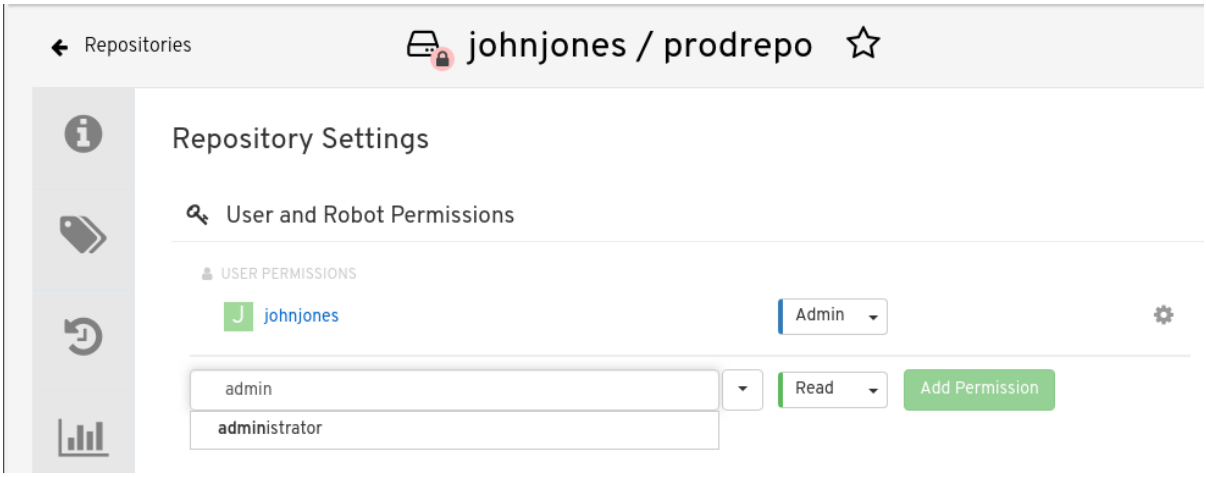
3.1. 允许访问用户仓库

在用户命名空间中创建存储库时，您可以将对此存储库的访问权限添加到用户帐户或通过机器人帐户添加访问权限。

3.1.1. 允许用户访问用户存储库

要访问与用户帐户关联的仓库，请执行以下操作：

1. 登录到您的 Red Hat Quay 用户帐户。
2. 选择要共享访问权限的用户命名空间下的仓库。
3. 从左列中选择 Settings 图标。
4. 键入您要授予您存储库访问权限的用户名称。用户名应显示为您类型，如下图所示：



5. 在权限框中选择以下其中之一：
   - read - 允许用户查看存储库并从中拉取存储库。
   - write - 允许用户查看存储库，以及从镜像拉取(pull)到存储库的镜像。
   - admin - 允许存储库的所有管理设置，以及所有 Read 和 Write 权限。
6. 选择 Add Permission 按钮。用户现在有分配的权限。

要删除存储库的用户权限，请选择用户条目右侧的 Options 图标，然后选择 Delete Permission。

3.2. 允许访问用户存储库的机器机器

机器人帐户用于设置对 Red Hat Quay registry 中存储库的自动访问。它们与 OpenShift 服务帐户类似。设置机器人帐户时，您：

- 生成与机器人帐户关联的凭证

- 识别机器可以将镜像推送到或拉取镜像的存储库和镜像
- 复制并粘贴生成的凭证，以用于不同的容器客户端（如 Docker、podman、Kubernetes、Metesos 等）访问每个定义的存储库

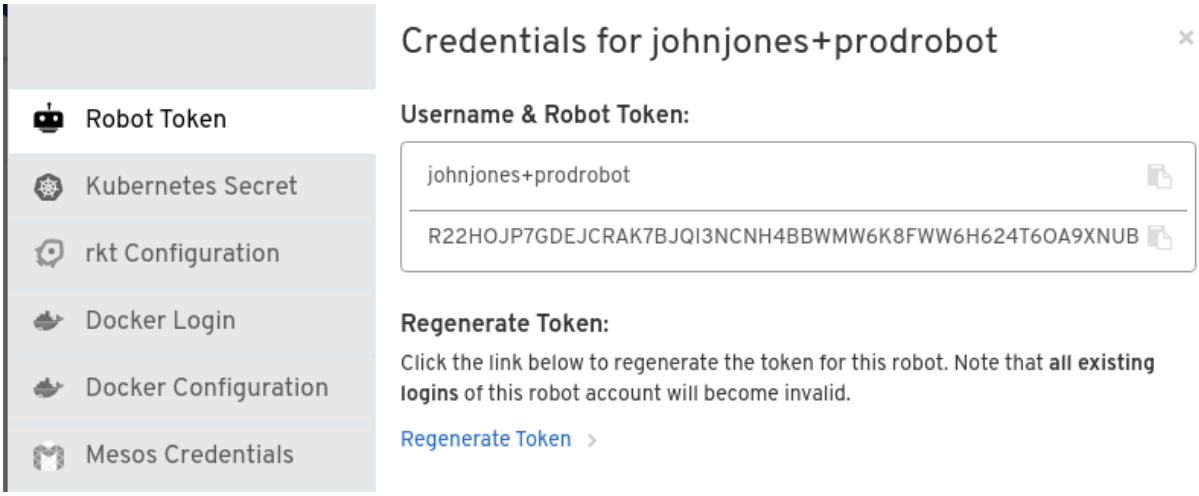
请记住，每个机器人帐户都限制为一个用户命名空间或机构。例如，机器人可以提供对用户 jsmith 访问的所有存储库的访问权限，但不能提供给不在用户存储库列表中的任何存储库。

以下流程通过设置机器人帐户来允许访问您的存储库。

1. 选择 Robot 图标：在 Repositories 视图中，从左列中选择 Robot 图标。
2. 创建 Robot 帐户：选择 Create Robot Account 按钮。
3. set Robot name：输入名称和描述，然后选择 Create robot account 按钮。机器人名称成为您的用户名的组合，以及您设置的机器人名称（如 jsmith+myrobot）
4. 向机器人帐户添加权限：在机器人帐户的 Add permissions 屏幕中，定义您要机器人可访问的存储库，如下所示：
   - 在每个存储库旁边放置一个检查标记，机器人可以访问该人可以访问
   - 对于每个存储库，选择以下之一并点 Add permissions:
     - none - Robot 没有对存储库的权限
     - 读取 - Robot 可以从存储库查看和拉取
     - write - Robot 可从中读取（拉取）并写入(push)到存储库
     - admin - 完全从存储库拉取(pull)和推送到存储库的功能，以及执行与存储库关联的管理任务
   - 选择添加权限按钮来应用设置
5. 获取凭证以通过机器人访问存储库： Back on the Robot Accounts 页面，选择 Robot account name 来查看该机器的凭据信息。
6. 获取令牌：选择 Robot Token，如下图所示，查看为机器人生成的令牌。如果要重置令牌，请选择 Regenerate Token。

注意

务必要清楚，重新生成令牌使这个人的前一个令牌都无效。



7. get credentials：在对生成的令牌满意后，使用以下命令获取生成的凭证：
   - Kubernetes Secret：选择它以 Kubernetes pull secret yaml 文件的形式下载凭证。
   - rkt Configuration：选择这个来为 rkt 容器运行时下载凭证，格式为 json 文件。
   - docker Login：选择它复制包含凭据的完整 **docker login** 命令行。
   - Docker 配置：选择此文件以下载要用作 Docker config.json 文件的文件，以永久存储您的客户端系统中的凭据。
   - Mesos Credentials：选择它下载 tarball，提供可在 Mesos 配置文件的 uris 字段中标识的凭据。

3.3. 允许访问机构仓库

创建机构后，您可以将一组存储库直接关联到该组织。要为该机构中的存储库添加访问权限，您可以添加团队（具有相同权限的用户集）和单独的用户。基本上，组织具有与用户相同的创建存储库和机器人帐户的功能，但组织旨在通过一组用户（团队或单独）设置共享存储库。

有关机构的其他知识：

- 您不能在另一个机构中有一个机构。为了整理组织，请使用团队。
- 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。
- 可以将团队作为使用存储库和关联镜像的成员设置，也可以作为管理员设置，以管理该机构具有特殊特权

3.3.1. 将团队添加到机构

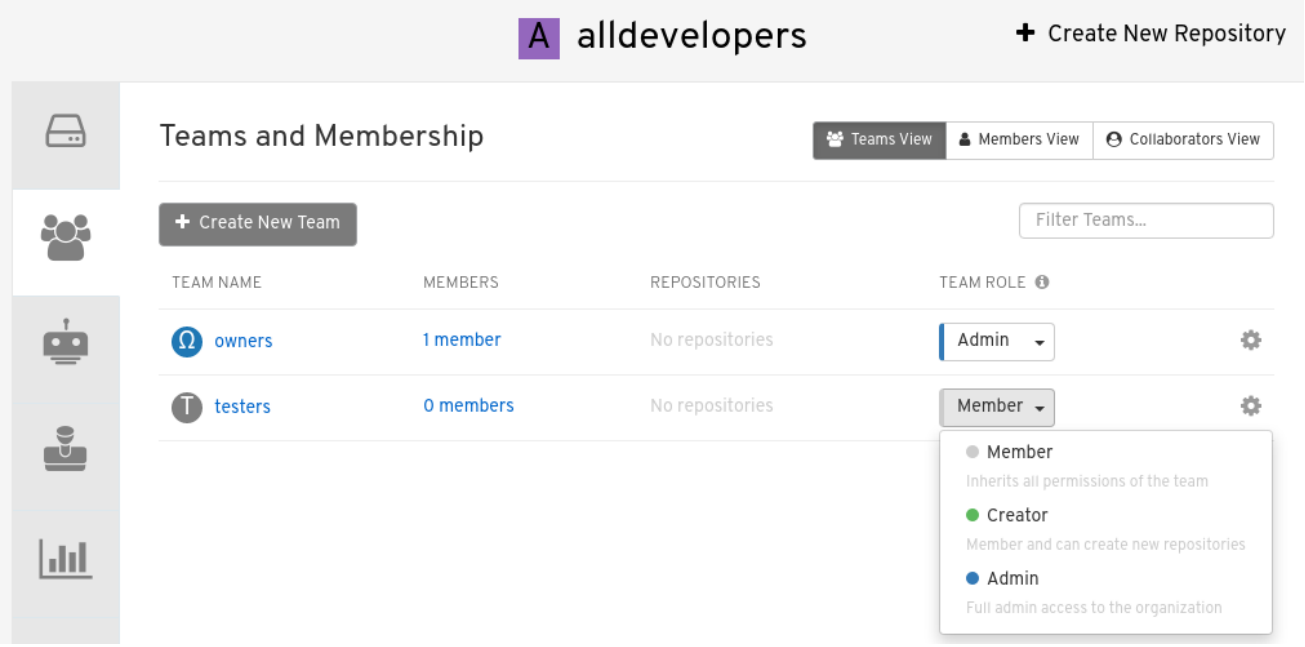
当您为您的机构创建团队时，可以选择团队名称，选择哪些软件仓库可供团队使用，并决定团队的访问权限级别。

1. 在 Organization 视图中，从左列中选择 Teams 和 Membership 图标。您将看到所有者团队存在，其具有创建该组织的用户的 Admin 权限。
2. 选择 Create New Team。系统会提示您输入要与机构关联的新团队名称。输入团队名称，该名称必须以小写字母开头，且团队名称的其余部分是小写字母和数字的任意组合（不允许大写字母或特殊字符）。
3. 选择 Create team 按钮。此时会出现 Add permissions 窗口，显示机构中的存储库列表。

4. 检查每个您希望团队能够访问的存储库。然后为每个方法选择一个权限：
   - 读取 - 团队成员可以查看和拉取镜像
   - write - 团队成员可以查看、拉取和推送镜像
   - admin - 团队成员具有完全的读/写权限，以及执行与存储库相关的管理任务
5. 选择 Add permissions 为团队保存存储库权限。

3.3.2. 设置团队角色

在添加了团队后，您可以在机构内设置该团队的角色。从组织内的 Teams 和 Membership 屏幕中，选择 TEAM ROLE 下拉菜单，如下图所示：



对于所选团队，选择以下角色之一：

- 成员 - 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
- admin - 完全管理对组织的访问权限，包括创建团队、添加成员和设置权限。

3.3.3. 将用户添加到团队

作为拥有组织的 Admin 权限的用户，您可以向团队添加用户和机器人。添加用户时，它会向该用户发送电子邮件。用户会一直处于等待状态，直到该用户接受邀请。

要将用户或机器添加到团队，请从机构屏幕开始并执行以下操作：

1. 选择您要添加用户或机器人的团队。
2. 在 Team Members 框中，键入以下之一：
   - 来自 Red Hat Quay registry 账户的用户名
   - registry 中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 orgname+robotname 的形式

3. 如果是机器人帐户，它会立即添加到团队中。对于用户帐户，需要加入的邀请被发送给用户。在用户接受该邀请前，用户仍然处于 INVITED TO JOIN 状态。

接下来，用户接受加入团队的电子邮件邀请。用户下一次登录 Red Hat Quay 实例时，用户将从 INVITED TO JOIN 列表中移到该机构的 MEMBERS 列表中。

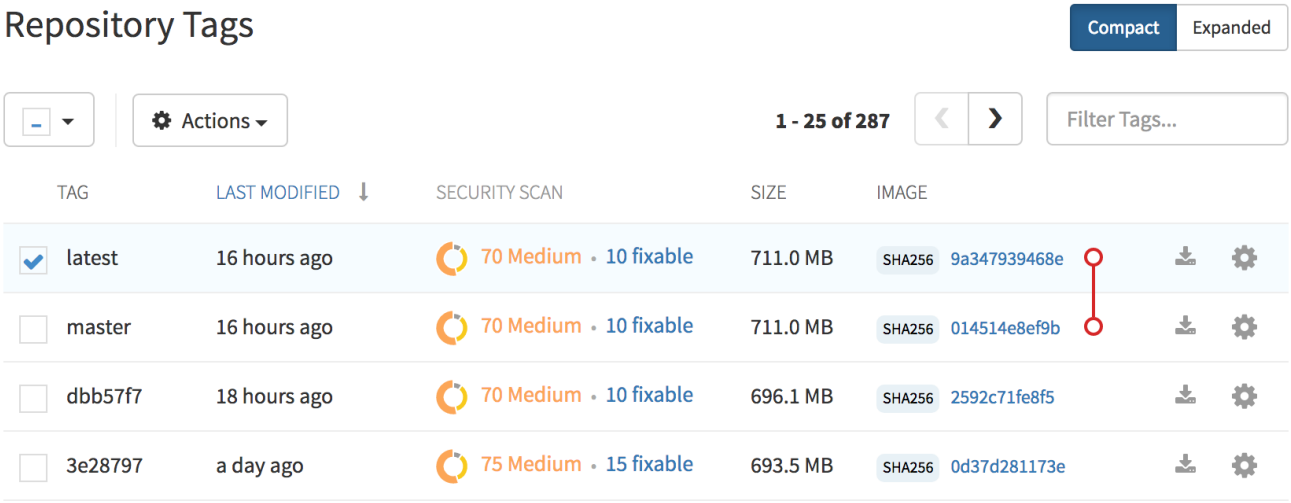
第 4 章 使用标签

标签提供了一种方式来识别镜像的版本，以及以不同方式提供命名同一镜像的方法。除了镜像的版本外，镜像标签还可识别其用途（如 devel、test 或 prod）或事实（最新）。

在镜像存储库 **的标签** 选项卡中，您可以查看、修改、添加、移动、删除和查看标签历史记录。您还可以使用不同的命令获取特定镜像（基于其名称和标签）下载（拉取）特定镜像的命令行。

4.1. 查看和修改标签

存储库的标签可以在存储库页面的 tags 面板中查看和修改，方法是单击 **Tags** 选项卡。



4.1.1. 添加新标签到带标记的镜像

单击标签旁边的 gear 图标并选择 **Add New Tag**，即可将新标签添加到标签。Red Hat Quay 将确认向镜像添加新标签。

4.1.2. 移动标签

将标签移到不同的镜像可以通过执行与添加新标签相同的操作，但提供现有的标签名称来实现。Red Hat Quay 将确认您希望标签移动，而不是添加。

4.1.3. 删除标签

特定标签及其所有镜像可以通过单击标签的 gear 图标并选择 **Delete Tag** 来删除。这将删除该标签以及其唯一的任何镜像。除非没有标签直接通过父子关系或间接引用镜像，否则不会删除镜像。

4.1.4. 查看标签历史记录并返回到时间

4.1.4.1. 查看标签历史记录

要查看标签的镜像历史记录，请单击 **Actions** 菜单下的 **View Tags History** 菜单项。显示页面将显示过去和指向该镜像时标记所指向的每个镜像。

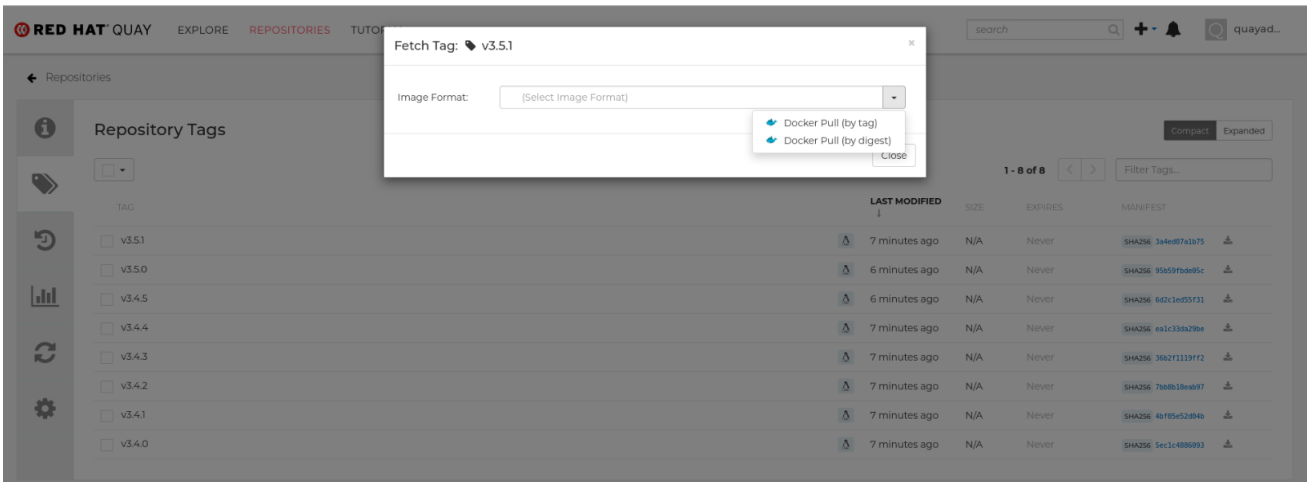
4.1.4.2. 返回时间

要将标签恢复到以前的镜像，请找到覆盖所需镜像的历史记录行，并点击 Restore 链接。

4.1.5. 通过标签或摘要获取镜像

在 **Tags** 选项卡中，您可以查看从准备好使用这些镜像的客户端中拉取镜像的不同方法。

1. 选择特定的存储库/镜像
2. 在左侧列中选择标签
3. 为特定镜像/标签组合选择 Fetch Tag 图标
4. 当显示 Fetch Tag 弹出时，选择 Image format 复选框，以查看显示可用于拉取镜像的不同方法的下拉菜单。选择提供将特定容器镜像拉取到本地系统的完整命令行：



您可以通过标签名称或使用 **docker** 命令摘要名称来拉取镜像的常规名称。选择您想要的拉取类型，然后选择 **Copy Command**。完整的命令行复制到您的剪贴板中。这两个命令显示标签和摘要的 **docker pull**:

```
docker pull quay.io/cnegus/whatever:latest
docker pull
quay.io/cnegus/whatever@sha256:e02231a6aa8ba7f5da3859a359f99d77e371cb47e643ce78e101958
782581fb9
```

将命令粘贴到系统中可用的 **docker** 命令和服务可用的命令行 shell 中，然后按 Enter 键。此时，容器镜像已准备好在您的本地系统上运行。

在 RHEL 和 Fedora 系统中，您可以替换 **docker** 的 **podman** 来拉取和运行所选镜像。

4.2. 标签过期

可以使用名为 **标签过期 的功能，设置为从所选日期和时间的 Red Hat Quay 存储库过期** 镜像。有关标签过期的信息包括：

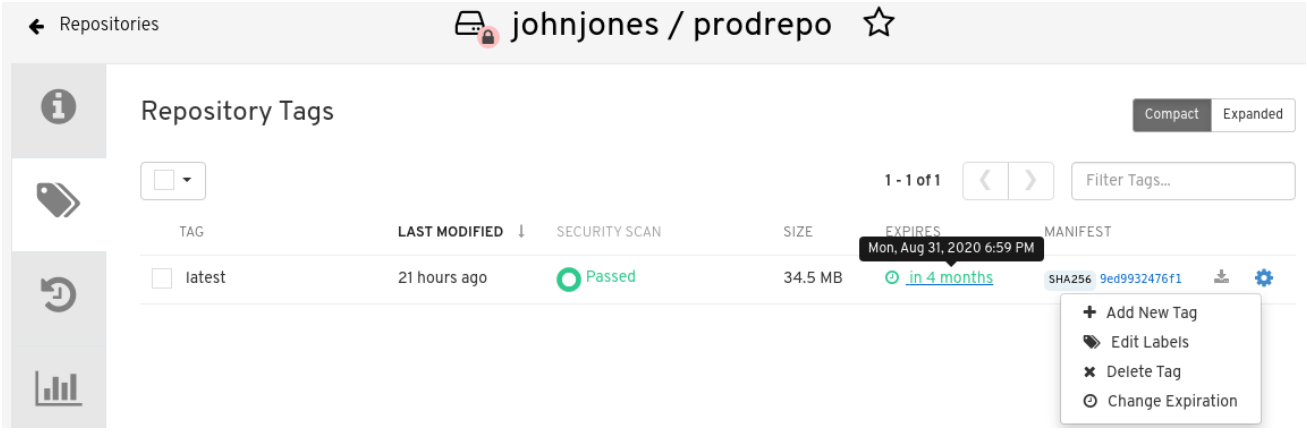
- 当标签过期时，标签将从存储库中删除。如果是特定镜像的上标签，则镜像将被设为已删除。
- 过期以每个标签为基础设置，而不是针对整个仓库设置。
- 当标签过期或删除时，不会立即从 registry 中删除。Time Machine（在用户设置中）的值定义了删除的标签实际被删除和收集的垃圾回收的时间。默认情况下，这个值为 14 天。直到此时间为止，标签可以被回复到过期或删除的镜像。

- Red Hat Quay 超级用户没有与从用户存储库中删除过期镜像相关的特殊权限。超级用户没有中央机制来收集信息并针对用户存储库执行操作。各个存储库的所有者是管理过期和最终删除其镜像的存储库。

可使用不同方法设置标签过期：

- 在创建镜像时，通过在 Dockerfile 中设置 **quay.expires-after=** LABEL。这将设置从构建镜像时过期的时间。
- 从存储库标签选择 EXPIRES 栏中的过期日期，然后选择要过期的特定日期和时间。

下图显示了在标签过期时更改标签过期的 Options 条目和标签过期的 EXPIRES 字段。将鼠标悬停在 EXPIRES 字段上，以查看当前设定的过期日期和时间。



4.2.1. 从 Dockerfile 设置标签过期

通过 Dockerfile LABEL 命令添加标签，如 **quay.expires-after=20h** 会导致标签在指定的时间后自动过期。时间值可能像 **1h**、**2d**、**3w** （小时）、3w（小时）、天和周（从镜像构建之时）。

4.2.2. 从仓库设置标签过期

Repository Tag 页面中有一个标题为 **EXPIRES** 的 UI 列，表示标签何时到期。用户可以单击其过期的时间，或者单击右侧的 Settings 按钮（gear 图标），然后选择 **Change Expiration**。

选择提示时的日期和时间，然后选择" **更改过期** "。当达到过期时间时，标签将设置为从存储库中删除。

4.3. 安全扫描

点击标签页旁边的漏洞或可修复计数，您可以跳到该标签的安全扫描信息。您可以发现您的镜像容易受到影响，以及您可能可用的补救选项。

请记住，镜像扫描只列出 Clair 镜像扫描程序发现的漏洞。每个用户都对未发现的漏洞的作用完全取决于该用户。Red Hat Quay 超级用户不操作这些发现的漏洞。

第 5 章 查看和导出日志

为 Red Hat Quay 中的所有仓库和命名空间（用户和机构）收集活动日志。访问日志文件的方法有多种，包括：

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

要访问日志，必须具有所选存储库或命名空间的 Admin 权限。

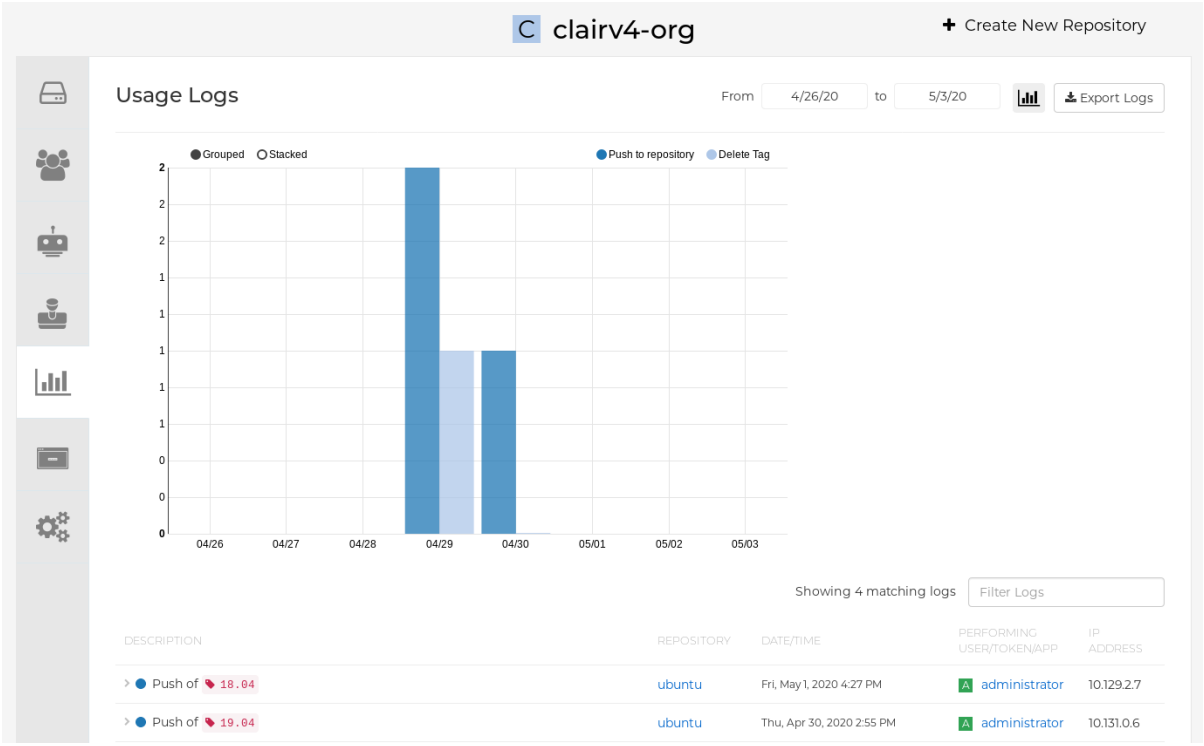
注意

通过 API 一次提供最多 100 个日志记录结果。要收集更多结果，您必须使用本章中介绍的日志导出器功能。

5.1. 查看日志

要从 Web UI 查看存储库或命名空间的日志条目，请执行以下操作：

1. 选择具有 Admin 权限的存储库或命名空间（机构或用户）。
2. 从左列中选择 Usage Logs 图标。此时会出现一个 Usage Logs 屏幕，如下例所示：



3. 在 Usage Logs 页面中，您可以：
   - 通过将日期添加到 From 和 to box，设置用于查看日志条目的日期范围。默认情况下会显示日志条目的最新一周。
   - 在 Filter Logs 框中键入字符串，以显示给定字符串的容器的日志条目。
   - 将箭头切换到任何日志条目左侧的箭头，以查看与该日志条目关联的一个或多个文本。

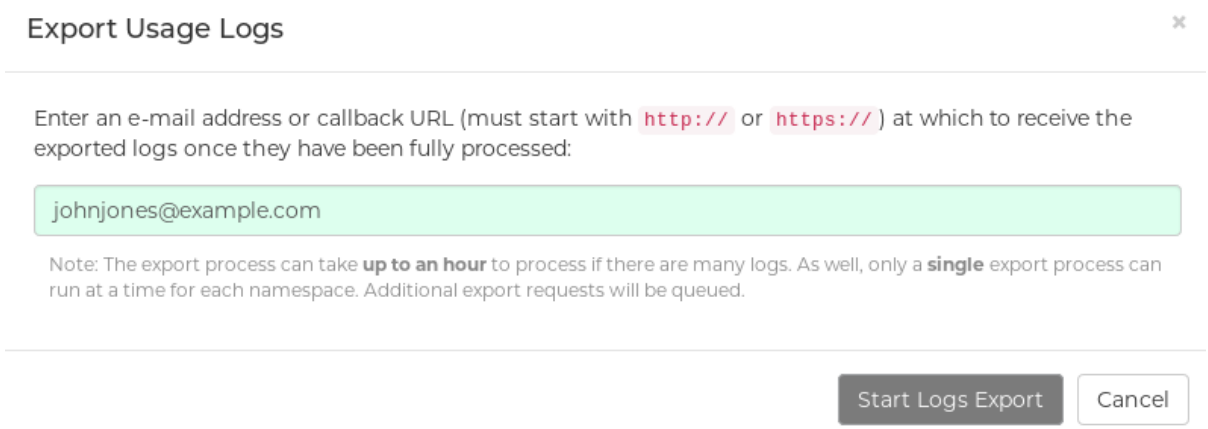
5.2. 导出存储库日志

为了获取大量日志文件并在 Red Hat Quay 数据库之外保存它们，您可以使用 Export Logs 功能。以下是您应该有关使用导出日志的一些问题：

- 您可以为要从仓库收集的日志选择一系列日期。
- 您可以请求通过电子邮件附件发送到您的日志或定向到回调 URL。
- 您需要具有对存储库或命名空间的 Admin 权限才能导出日志
- 一次最多可以导出 30 天日志数据
- 导出日志仅收集之前生成的日志数据。它不会流传输日志数据。
- 必须为此功能配置您的 Red Hat Quay 实例（本地存储将无法工作）。
- 收集日志并可用后，如果您想要保存数据，应立即复制这些数据。默认情况下，数据会在一小时内过期。

使用导出日志功能：

1. 选择一个具有 Admin 权限的存储库。
2. 从左列中选择 Usage Logs 图标。此时会出现 Usage Logs 屏幕。
3. 选择您要收集的日志条目的 From 和 date 范围。
4. 选择"导出日志"按钮。此时会出现一个出口使用日志弹出，如下所示



5. 输入您要接收导出的日志的电子邮件地址或回调 URL。对于回调 URL，您可以使用 URL 作为位置，如 webhook.site。
6. 选择 Start Logs Export。这会导致 Red Hat Quay 开始收集所选日志条目。根据收集的日志数据量，这可能需要一分钟到一小时才能完成。
7. 日志导出完成后，您将：
   - 收到一封电子邮件，提醒您请求的导出日志条目的可用性。
   - 从 webhook URL 查看日志导出请求成功状态。系统将为您选择下载日志的链接。

请记住，URL 指向 Red Hat Quay 外部存储中的位置，并将其设置为一小时内过期。因此，如果您打算保留它们，请确保在过期时间之前复制导出的日志。

第 6 章 使用构建 WORKER 自动构建 DOCKERFILE

Red Hat Quay 支持在 OpenShift 或 Kubernetes 上使用一组 worker 节点构建 Dockerfile。构建触发器（如 GitHub Webhook）可以配置为在提交新代码时自动构建新版存储库。本文档将指导您使用 Red Hat Quay 安装启用构建，并设置一个或多个 OpenShift/K8s 集群来接受来自 Red Hat Quay 的构建。使用 Red Hat Quay 3.4 时，底层构建管理器已作为 Red Hat Quay 2 从 Python 2 迁移到 Python 3 的一部分进行了彻底重写。因此，构建器节点现在作为 Kubernetes 作业与构建节点动态创建，这些节点在 Red Hat Quay 3.3 及更早版本中持续运行。这大大简化了 Red Hat Quay 管理构建的方式，并提供在每天处理数千个容器镜像构建时所用的相同机制 quay.io。当前在 Red Hat Quay 3.3 下运行静态（"企业"构建程序）的客户将需要迁移到基于 Kubernetes 的构建机制。

6.1. 架构概述

Red Hat Quay Build 系统专为可扩展性而设计（因为它用来托管 quay.io 的所有构建）。Red Hat Quay 的 Build Manager 组件提供了一个编配层，用于跟踪构建请求并确保构建执行程序（OpenShift/K8s 集群）将执行每个请求。每个构建都由一个 Kubernetes 作业处理，它会启动一个小型虚拟机来完全隔离并包含镜像构建过程。这样可确保容器构建不会影响相互影响或底层构建系统。可以配置多个可执行文件，以确保即使在基础架构出现故障时也会执行构建。Red Hat Quay 会自动将构建发送到不同的 Executor，如果它检测到有困难。

注意

Red Hat Quay 的上游版本提供了如何配置基于 AWS/EC2 的 Executor 的说明。Red Hat Quay 客户不支持此配置。

6.1.1. 构建管理器

构建管理器负责调度构建的生命周期。需要更新构建队列、构建阶段和运行作业状态的操作由构建管理器处理。

6.1.2. 构建 worker 的 control plane

构建作业在单独的 worker 节点上运行，并调度到单独的 control planes（执行器）。目前，Red Hat Quay 支持在 AWS 和 Kubernetes 上运行作业。构建通过 quay.io/quay/quay-builder 执行。在 AWS 上，构建调度到 EC2 实例。在 k8s 上，构建被调度为作业资源。

6.1.3. 编配器

编配器用于存储当前运行的构建作业的状态，并为构建管理器发布事件以消耗。例如，到期事件。目前，支持的编配器后端是 Redis。

6.2. OPENSHIFT 要求

Red Hat Quay 构建支持 Kubernetes 和 OpenShift 4.5 或更高版本。需要裸机（非虚拟化）worker 节点，因为构建 pod 需要能够运行 kvm 虚拟化。每个构建都在临时虚拟机中执行，以确保在构建运行时完整的隔离和安全性。另外，您的 OpenShift 集群应该允许与 Red Hat Quay 构建关联的 ServiceAccount 使用所需的 SecurityContextConstraint 运行，以支持特权容器。

6.3. 编配要求

Red Hat Quay 构建需要访问 Redis 实例来跟踪构建状态信息。可以使用已部署到 Red Hat Quay 安装中的同一 Redis 实例。所有构建队列都在 Red Hat Quay 数据库中管理，因此不需要高可用性 Redis 实例。

6.4. 使用 OPENSHIFT 设置 RED HAT QUAY BUILDER

6.4.1. OpenShift TLS 组件

Red Hat Quay 3.6 Operator 引入了 **tls** 组件，可让您控制 TLS 配置。

注意

当 Operator 管理 TLS 组件时，Red Hat Quay 3.6 不支持构建器。

如果将 **tls** 设置为**非受管状**态，您可以提供自己的 **ssl.cert** 和 **ssl.key** 文件。在本实例中，如果您希望集群支持构建器，您必须将 Quay 路由和构建器路由名称添加到证书中的 SAN 列表中，或者使用通配符。要添加构建器路由，请使用以下格式：

```
[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]
```

6.4.2. 准备 OpenShift 进行 Red Hat Quay 构建

在 OpenShift 集群中需要几个操作，然后才能接受来自 Red Hat Quay 的构建。

1. 创建一个将运行构建的项目（如 'builder'）

```
$ oc new-project builder
```

2. 在此项目中创建一个 **ServiceAccount**，它将用于运行构建。确保有足够的权限来创建 **作业和 Pod**。复制 **ServiceAccount**的令牌，以便稍后使用。

```
$ oc create sa -n builder quay-builder
$ oc policy add-role-to-user -n builder edit system:serviceaccount:builder:quay-builder
$ oc sa get-token -n builder quay-builder
```

3. 识别 OpenShift 集群 API 服务器的 URL。这可以从 OpenShift 控制台找到。

4. 识别调度构建作业时使用的 worker **节点标签**。因为构建 pod 需要在裸机 worker 节点上运行，因此通常这些 pod 使用特定的标签标识。检查您的集群管理员，以确定应使用哪个节点标签。

5. 如果集群使用自签名证书，获取 kube apiserver 的 CA 以添加到 Red Hat Quay 的额外证书。

   a. 获取包含 CA 的 secret 的名称：

```
$ oc get sa openshift-apiserver-sa --namespace=openshift-apiserver -o json | jq
'.secrets[] | select(.name | contains("openshift-apiserver-sa-token"))'.name
```

   b. 从 Openshift 控制台中的 secret 获取 **ca.crt** 的键值。该值应该以"-----BEGIN CERTIFICATE-----"开始。

   c. 使用 ConfigTool，在 Red Hat Quay 中导入 CA。确保此文件的名称与 **K8S_API_TLS_CA** 匹配。

6. 为 **ServiceAccount** 创建所需的安全上下文/角色绑定：

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
```

```
  metadata:
    name: quay-builder
  priority: null
  readOnlyRootFilesystem: false
  requiredDropCapabilities: null
  runAsUser:
    type: RunAsAny
  seLinuxContext:
    type: RunAsAny
  seccompProfiles:
  - '*'
  supplementalGroups:
    type: RunAsAny
  volumes:
  - '*'
  allowHostDirVolumePlugin: true
  allowHostIPC: true
  allowHostNetwork: true
  allowHostPID: true
  allowHostPorts: true
  allowPrivilegeEscalation: true
  allowPrivilegedContainer: true
  allowedCapabilities:
  - '*'
  allowedUnsafeSysctls:
  - '*'
  defaultAddCapabilities: null
  fsGroup:
    type: RunAsAny
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    name: quay-builder-scc
    namespace: builder
  rules:
  - apiGroups:
    - security.openshift.io
    resourceNames:
    - quay-builder
    resources:
    - securitycontextconstraints
    verbs:
    - use
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: quay-builder-scc
    namespace: builder
  subjects:
  - kind: ServiceAccount
    name: quay-builder
  roleRef:
```

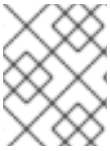
```
apiGroup: rbac.authorization.k8s.io
kind: Role
name: quay-builder-scc
```

6.4.3. 启用构建器，并将构建配置添加到 Red Hat Quay Configuration Bundle

1. 确保已在 Red Hat Quay 配置中启用了 Builds。

```
FEATURE_BUILD_SUPPORT: True
```

1. 在 Red Hat Quay 配置捆绑包中添加以下内容，使用特定于您的安装的值替换每个值。



注意

目前，只能通过 Red Hat Quay Config 工具启用构建功能。在 config.yaml 文件中必须手动完成 Build Manager 和 Executors 的实际配置。

```
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  ORCHESTRATOR:
    REDIS_HOST: quay-redis-host
    REDIS_PASSWORD: quay-redis-password
    REDIS_SSL: true
    REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
  - EXECUTOR: kubernetes
    BUILDER_NAMESPACE: builder
    K8S_API_SERVER: api.openshift.somehost.org:6443
    K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt
    VOLUME_SIZE: 8G
    KUBERNETES_DISTRIBUTION: openshift
    CONTAINER_MEMORY_LIMITS: 5120Mi
    CONTAINER_CPU_LIMITS: 1000m
    CONTAINER_MEMORY_REQUEST: 3968Mi
    CONTAINER_CPU_REQUEST: 500m
    NODE_SELECTOR_LABEL_KEY: beta.kubernetes.io/instance-type
    NODE_SELECTOR_LABEL_VALUE: n1-standard-4
    CONTAINER_RUNTIME: podman
    SERVICE_ACCOUNT_NAME: *****
    SERVICE_ACCOUNT_TOKEN: *****
    QUAY_USERNAME: quay-username
    QUAY_PASSWORD: quay-password
    WORKER_IMAGE: <registry>/quay-quay-builder
    WORKER_TAG: some_tag
    BUILDER_VM_CONTAINER_IMAGE: <registry>/quay-quay-builder-qemu-rhcos:v3.4.0
    SETUP_TIME: 180
    MINIMUM_RETRY_THRESHOLD: 0
    SSH_AUTHORIZED_KEYS:
    - ssh-rsa 12345 someuser@email.com
    - ssh-rsa 67890 someuser2@email.com
```

以下介绍了每个配置字段。

ALLOWED_WORKER_COUNT

定义每个 Red Hat Quay Pod 实例化多少个 Build Worker。这通常是 '1'。

ORCHESTRATOR_PREFIX

定义添加到所有 Redis 键的唯一前缀（可用于将 Orchestrator 值与其他 Redis 密钥隔离）。

REDIS_HOST

Redis 服务的主机名。

REDIS_PASSWORD

用于向 Redis 服务进行身份验证的密码。

REDIS_SSL

定义您的 Redis 连接是否使用 SSL。

REDIS_SKIP_KEYSPACE_EVENT_SETUP

默认情况下，Red Hat Quay 不会在运行时设置关键事件所需的密钥空间事件。为此，请将 REDIS_SKIP_KEYSPACE_EVENT_SETUP 设置为 **false**。

EXECUTOR

启动此类型的 Executor 的定义。有效值为 'kubernetes' 和 'ec2'

BUILDER_NAMESPACE

Red Hat Quay 构建所需的 Kubernetes 命名空间

K8S_API_SERVER

OpenShift 集群的 API 服务器的主机名，进行构建需要

K8S_API_TLS_CA

构建集群的 CA 证书的 **Quay** 容器中的 filepath，以便在发出 API 调用时信任 Quay 应用。

KUBERNETES_DISTRIBUTION

指明正在使用的 Kubernetes 类型。有效值为 'openshift' 和 'k8s'。

CONTAINER_*

定义每个构建 Pod 的资源请求和限值。

NODE_SELECTOR_*

定义应调度构建 Pod 的节点选择器标签名称/值对。

CONTAINER_RUNTIME

指定构建器是否应该运行 **docker** 还是 **podman**。使用红帽 **quay-builder** 镜像的客户应将其设置为 **podman**。

SERVICE_ACCOUNT_NAME/SERVICE_ACCOUNT_TOKEN

定义构建 Pod 将要使用的服务帐户名称/令牌。

QUAY_USERNAME/QUAY_PASSWORD

定义拉取在 WORKER_IMAGE 字段中指定的 Red Hat Quay 构建 worker 镜像所需的 registry 凭证。客户应该提供针对 registry.redhat.io 的"创建注册表服务帐户"一节中的红帽服务帐户凭证，网址为 https://access.redhat.com/RegistryAuthentication

WORKER_IMAGE

Red Hat Quay 构建器镜像的镜像引用。registry.redhat.io/quay/quay-builder

WORKER_TAG

所需构建器镜像的标签。最新版本为 v3.4.0。

BUILDER_VM_CONTAINER_IMAGE

对包含每个 Red Hat Quay 构建(**registry.redhat.io/quay-builder-qemu-rhcos:v3.4.0**)所需的内部虚拟机的完整引用。

SETUP_TIME

如果构建还没有在 Build Manager 中注册，则指定超时的秒数（默认为 500 秒）。超时的构建尝试重启三次。如果构建在三个尝试失败后没有注册，它会被视为失败。

MINIMUM_RETRY_THRESHOLD

此设置与多个可执行文件一起使用；它指示在选择不同的执行器前尝试启动构建的次数。设置为 0 意味着构建作业需要没有多少限制。这个值应该被有意保留小（三个或更少），确保在基础架构出现故障时迅速发生故障转移。例如，Kubernetes 设置为第一个 executor 和 EC2 作为第二个执行者。如果我们希望最后一次尝试在 EC2 上始终执行作业，而不是 Kubernetes，我们将 Kubernetes 执行程序的 **MINIMUM_RETRY_THRESHOLD** 设置为 1 和 EC2 的 **MINIMUM_RETRY_THRESHOLD** 为 0（如果不设置 0）。在这种情况下，kubernetes' **MINIMUM_RETRY_THRESHOLD** > retries_remaining(1)将评估为 False，因此回退到配置的第二个 executor

SSH_AUTHORIZED_KEYS

ignition 配置中 bootstrap 的 ssh 密钥列表。这允许使用其他密钥 ssh 到 EC2 实例或 QEMU 虚拟机

6.5. OPENSHIFT 路由限制

注意

本节只适用于您在带有受管 **路由** 组件的 OpenShift 上使用 Quay Operator。

由于 OpenShift **路由** 只能向单个端口提供流量，因此设置构建需要额外的步骤。确保将 **kubectl** 或 **oc** CLI 工具配置为处理安装 Quay Operator 的集群，并且您的 **QuayRegistry** 存在（不一定与构建程序运行的裸机集群相同）。

- 按照以下步骤，确保 OpenShift 集群上启用了 HTTP/2 入口。
- Quay Operator 将创建一个 **Route**，用于将 gRPC 流量定向到现有 Quay pod 内运行的构建管理器服务器。如果要使用自定义主机名（如 **builder.registry.example.com**的子域），请确保创建一个带有 DNS 供应商的 CNAME 记录，该记录指向所创建的 **Route** 的 **status.ingress[0].host**。

```
$ kubectl get -n <namespace> route <quayregistry-name>-quay-builder -o jsonpath=
{.status.ingress[0].host}
```

- 使用 OpenShift UI 或 CLI，将 **QuayRegistry** 的 **spec.configBundle Secret** 引用的 Secret 使用构建集群 CA 证书（将密钥 **extra_ca_cert_build_cluster.cert**）更新 **config.yaml** 条目，并使用上述构建器配置中引用的值（取决于构建 executor）和 **BUILDMANHOSTNAME** 字段：

```
BUILDMAN_HOSTNAME: <build-manager-hostname>
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  ORCHESTRATOR:
    REDIS_HOST: quay-redis-host
    REDIS_PASSWORD: quay-redis-password
    REDIS_SSL: true
    REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
  - EXECUTOR: kubernetes
    BUILDER_NAMESPACE: builder
    ...
```

额外配置字段解释如下：

BUILDMAN_HOSTNAME

外部可访问的服务器主机名，用于向构建管理器进行通信。默认值与 **SERVER_HOSTNAME** 相同。对于 OpenShift **Route**，如果使用自定义主机名，可以是 **status.ingress[0].host** 或 CNAME 条目。**BUILDMAN_HOSTNAME 需要包含** 端口号，例如，对于 Openshift Route 的 **somehost:443**，因为用于与构建管理器通信的 gRPC 客户端在忽略时不会推断任何端口。

6.6. 构建故障排除

由构建管理器启动的构建器实例是临时的。这意味着，他们可以在超时/失败或 control plane(EC2/K8s) 收集的垃圾回收时被 Red Hat Quay} 关闭。这意味着，若要获取构建器日志，构建 **运行时** 需要执行此操作。

6.6.1. DEBUG 配置标志

可以设置 DEBUG 标志，以防止在完成/失败后清理构建器实例。为此，在所需的 executor 配置中将 DEBUG 设置为 true。例如：

```
EXECUTORS:
  - EXECUTOR: ec2
    DEBUG: true
    ...
  - EXECUTOR: kubernetes
    DEBUG: true
    ...
```

当设置为 true 时，DEBUG 将阻止构建节点在 quay-builder 服务完成或失败后关闭，并阻止构建管理器清理实例（终止 EC2 实例或删除 k8s 作业）。这将允许调试构建器节点问题，**不应** 在生产环境中设置。生命周期服务仍然存在。例如，实例仍然会在约 2 小时后关闭（EC2 实例将终止，k8s 作业将完成），设置 DEBUG 也会影响 ALLOWED_WORKER_COUNT，因为未终止的实例/作业仍将计入运行 worker 的总数。这意味着，如果 ALLOWED_WORKER_COUNT 能够调度新构建，则需要手动删除现有的构建器 worker。

使用以下步骤：

1. 客户机虚拟机将其 SSH 端口(22)转发到其主机的端口 2222。端口将构建器 Pod 的端口 2222 转发到 localhost 上的端口。例如

```
$ kubectl port-forward <builder pod> 9999:2222
```

2. 使用从 SSH_AUTHORIZED_KEYS 中的密钥集，通过 SSH_AUTHORIZED_KEYS 连接到容器中运行的虚拟机：

```
$ ssh -i /path/to/ssh/key/set/in/ssh_authorized_keys -p 9999 core@localhost
```

3. 获取 quay-builder 服务日志：

```
$ systemctl status quay-builder
$ journalctl -f -u quay-builder
```

- 也可以使用单个 SSH 命令执行第 2-3 步：

```
$ ssh -i /path/to/ssh/key/set/in/ssh_authorized_keys -p 9999 core@localhost 'systemctl
status quay-builder'
$ ssh -i /path/to/ssh/key/set/in/ssh_authorized_keys -p 9999 core@localhost 'journalctl -f
-u quay-builder'
```

6.7. 设置 GITHUB 构建（可选）

如果您的组织计划通过推送到 GitHub（或 GitHub Enterprise）进行构建，继续在 *GitHub 中创建 OAuth 应用程序*。

第 7 章 构建 DOCKERFILE

Red Hat Quay 支持在我们的构建 fleet 上构建 Dockerfile，并将生成的镜像推送到存储库。

7.1. 查看和管理构建

单击存储库视图中的 Builds 选项卡，可以查看和管理 **存储库** 构建。

7.2. 手动启动构建

要手动启动存储库构建，请单击任何存储库页面上标题右侧的 **+** 图标，然后选择 **New Dockerfile Build**。上传的 **Dockerfile .tar.gz** 或 HTTP URL 可用于构建。

注意

在手动启动构建时，您将无法指定 Docker 构建上下文。

7.3. 构建触发器

存储库构建也可由事件自动触发，如推送到 SCM（GitHub、BitBucket 或 GitLab）或 调用 Webhook。

7.3.1. 创建新构建触发器

要设置构建触发器，点 Builds 视图页面上的 **Create Build Trigger** 按钮，并按照对话框的说明进行操作。您需要向您的存储库授予 Red Hat Quay 访问权限，以设置触发器，并且您的帐户 *需要 SCM 存储库的 admin 访问权限*。

7.3.2. 手动触发构建触发器

要手动触发构建触发器，请单击构建触发器旁边的图标，然后选择 **Run Now**。

7.3.3. 构建上下文

使用 Docker 构建镜像时，指定目录使其成为构建上下文。对于手动构建和构建触发器，这包括 true，因为 Red Hat Quay 执行的构建与您自己的机器上运行 **docker 构建** 没有不同。

如果未指定，Red Hat Quay 构建上下文始终来自构建设置，并将回退到构建源的根目录。*触发构建时，Red Hat Quay 构建 worker 会将 git 存储库克隆到 worker 机器，并在执行构建前输入构建上下文。*

对于基于 tar 归档的构建，构建工作程序提取存档并输入构建上下文。例如：

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

想象上面的例子是 GitHub 存储库的目录结构，名为"example"。如果在构建触发器设置中或手动启动构建时没有指定子目录，则构建将在示例目录中操作。

*如果将 **subdir** 指定为构建触发器设置中的子目录，则只有其中的 Dockerfile 才可以出现在构建中。这意味着您无法在 Dockerfile 中使用 **ADD** 命令来添加 **文件**，因为它不在构建上下文之外。*

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

第 8 章 设置自定义 GIT TRIGGER

Custom Git Trigger 是任何 git 服务器的通用方法，充当构建触发器。它完全依赖于 SSH 密钥和 Webhook 端点；其他一切都留给用户实施。

8.1. 创建触发器

创建自定义 Git Trigger 与创建带有几个细微差别的任何其他触发器类似：

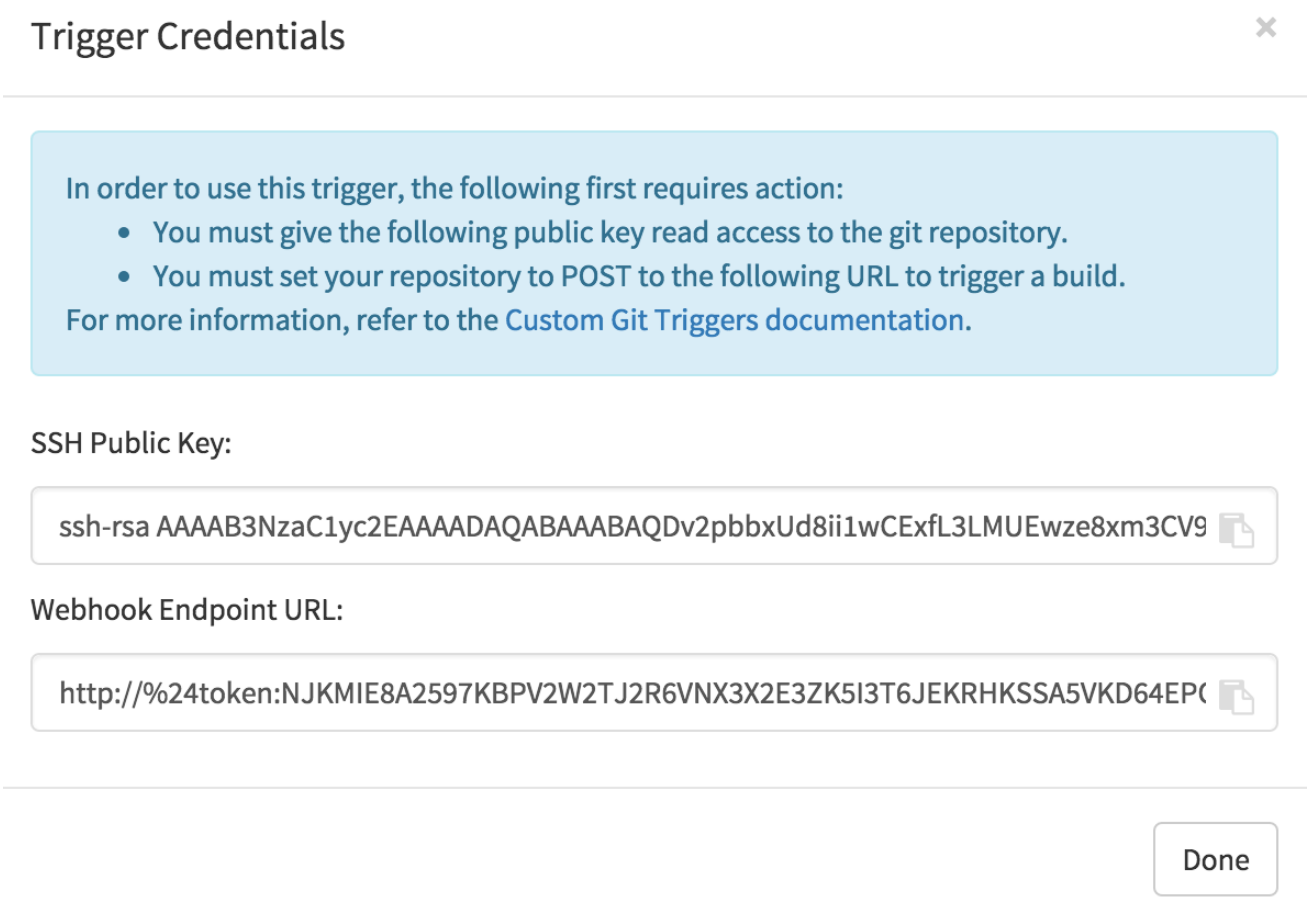
- Red Hat Quay 无法自动检测与触发器搭配使用的适当机器人帐户。这必须在创建过程中手动完成。
- 创建触发器后必须执行额外的步骤才能使用触发器。这些步骤的详情如下。

8.2. 触发器(TRIGGER-CREATION)设置

创建触发器后，在使用触发器 **前需要 2 个额外的步骤**：

- 提供对创建触发器时生成的 SSH 公钥 的读取访问权限。
- 设置 POST 到 Red Hat Quay 端点的 webhook，以触发构建。

key 和 URL 可随时通过从位于触发器列表中的 gear 选择 **View Credentials** 即可。



8.2.1. SSH 公钥访问

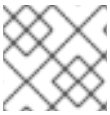
根据 Git 服务器设置，可以通过各种方式安装 Red Hat Quay 为自定义 git 触发器生成的 SSH 公钥。例如，Git 文档 描述了一个小型服务器设置，只需将密钥添加到 **$HOME/.ssh/authorize_keys**，即可提供

对构建器的访问来克隆存储库。对于未正式支持的任何 git 存储库管理软件，通常有一个位置输入通常标记为 **Deploy Keys** 的键。

8.2.2. Webhook

要自动触发构建，必须使用以下格式将 JSON 有效负载 POST 到 webhook URL ：

```
{
  "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
    }
  }
}
```



注意

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

再一次，这可以通过不同的方法完成，具体取决于服务器设置，但对于大多数情况，可以通过 post-receive git hook 进行。

第 9 章 跳过源控制触发的构建

要指定 Red Hat Quay 构建系统应忽略提交，请在提交消息中的任何位置添加文本 **[skip build]** 或 **[build skip]**。

第 10 章 设置 GITHUB 构建触发器标签

Red Hat Quay 支持使用 GitHub 或 GitHub Enterprise 作为构建镜像的触发器。如果您尚未这样做，请继续并在 Red Hat Quay 中启用构建支持。

10.1. 了解构建触发器的标签命名

在 Red Hat Quay 3.3 之前，从构建触发器创建的镜像已限制。由构建触发器构建的镜像的名称为：

- 调用触发器的分支或标签
- 带有使用默认分支的镜像的 **latest** 标签

从 Red Hat Quay 3.3 及更高版本开始，在设置镜像标签方面具有更大的灵活性。第一种操作是输入自定义标签，使任何字符串指定为每个构建的镜像标签。但是，作为替代方案，您可以使用以下标签模板使用每个提交中的信息标记镜像：

- **${commit_info.short_sha}**:提交的简短 SHA
- **${commit_info.date}** ：提交的时间戳
- **${commit_info.author}** ：提交中的作者
- **${commit_info.committer}**: 提交的提交者
- **${parsed_ref.branch}**: 分支名称

以下流程描述了如何为构建触发器设置标记。

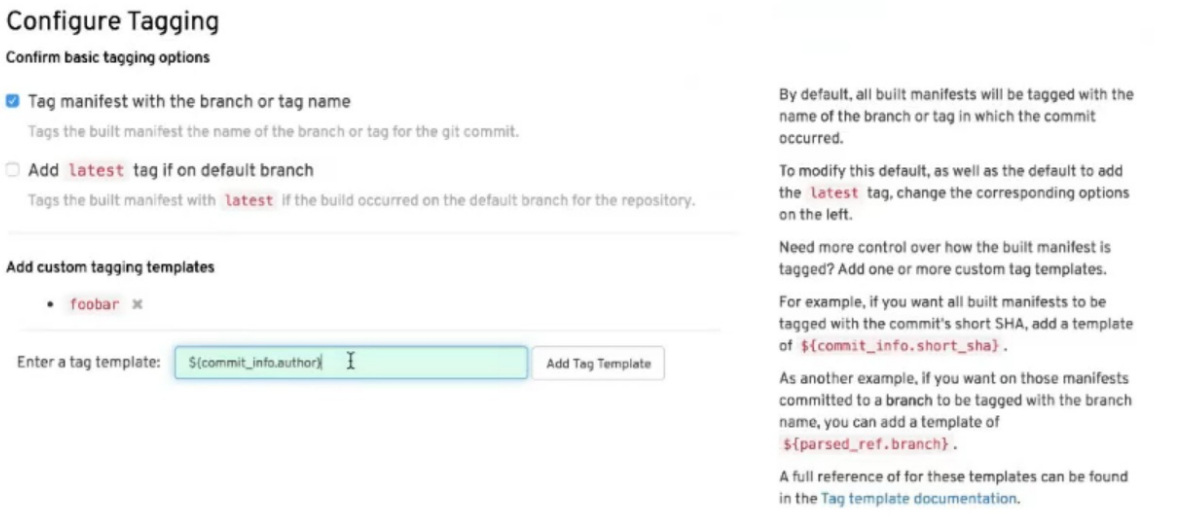
10.2. 为构建触发器设置标签名称

按照以下步骤为构建触发器配置自定义标签：

1. 在存储库视图中，从左侧导航中选择 Builds 图标。
2. 选择 Create Build Trigger 菜单，然后选择您想要的存储库推送类型（GitHub、Bitbucket、GitLab 或 Custom Git 存储库 push）。本例中选择了 GitHub Repository Push，如下图所示。

3. 当显示 Setup Build Trigger 页面时，请选择您要设置的仓库和命名空间。

4. 在 Configure Trigger 下，为所有分支和标签选择 Trigger，或者仅在分支和标签匹配正则表达式中选择 Trigger。然后选择 Continue。此时会出现 Configure Tagging 部分，如下图所示：



5. 向下滚动 以配置标记，并从以下选项中选择：

   - **使用分支或标签名称标记清单** ：选中此框，以使用作为镜像上使用的标签所发生的分支或标签的名称。默认启用。
   - **如果默认分支为**：选中，则使用镜像的最新标签（如果位于存储库的默认分支上），则添加 **latest** 标签。默认启用。
   - **添加自定义标记模板** ：在 Enter a tag template 框中输入自定义标签或模板。在这里可以输入多个标签模板，如本节前面所述。它们包括使用短 SHA、时间戳、作者名称、提交者和分支名称的方法，作为标签。

6. 选择 Continue。系统会提示您输入 Docker 构建的目录构建上下文。构建上下文目录标识包含 Dockerfile 的目录的位置，以及触发构建时所需的其他文件。如果 Dockerfile 位于 git 存储库的根目录中，则输入 "/"。

7. 选择 Continue。此时会提示您输入一个可选的 Robot 帐户。如果要在构建过程中拉取私有基础镜像，请执行以下操作。机器人帐户需要访问构建。

8. 选择 Continue 以完成构建触发器的设置。

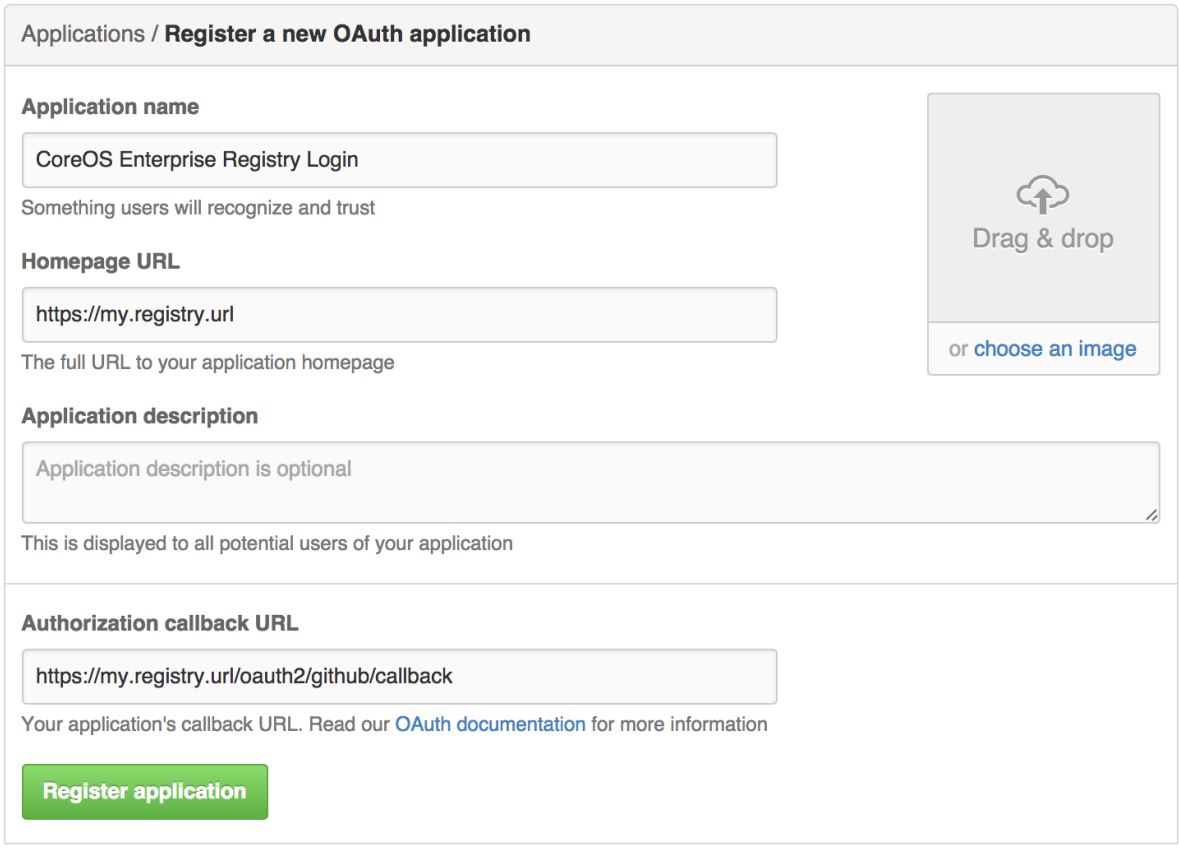
如果您要返回到存储库的 Repository Builds 页面，您设置的构建触发器将在 Build Triggers 标题下列出。

第 11 章 在 GITHUB 中创建 OAUTH 应用程序

您可以通过将 registry 注册为 GitHub OAuth 应用来授权 registry 访问 GitHub 帐户及其存储库。

11.1. 创建新的 GITHUB 应用程序

1. *登录 GitHub（企业）*
2. *访问您机构设置下的应用程序页面。*
3. *点 Register New Application。此时会显示* ***Register a new OAuth 应用程序配置*** *页面：*



4. *设置主页 URL：输入 Quay Enterprise URL 作为* ***主页 URL***

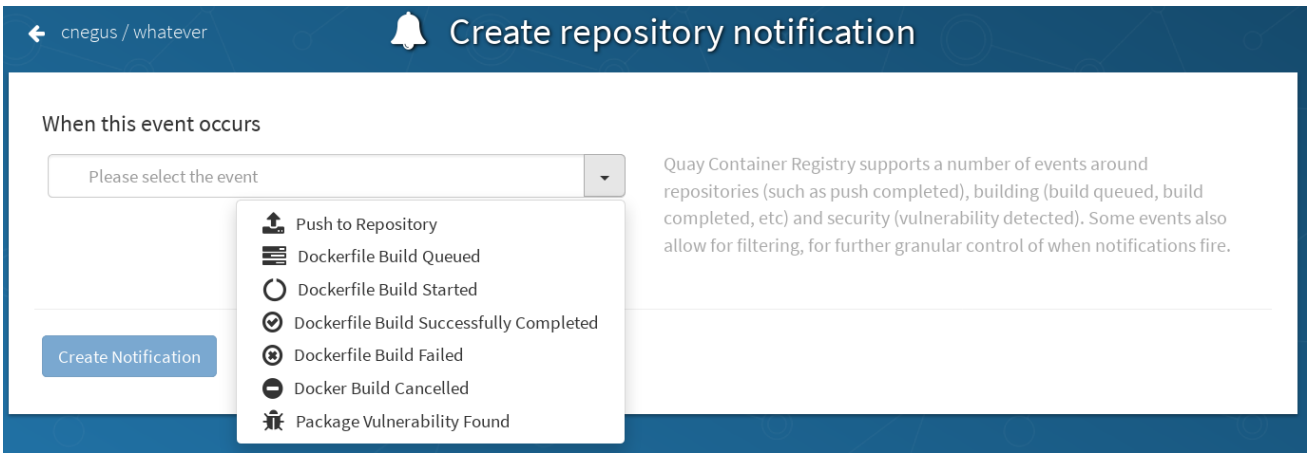
注意

如果使用公共 GitHub，则输入的主页 URL 必须可以被您的用户访问。它也可以是内部 URL。

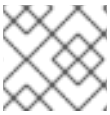
5. *设置授权回调 URL：输入 https://{$RED_HAT_QUAY_URL}/oauth2/github/callback 作为 Authorization 回调 URL。*
6. *点 Register application 按钮保存您的设置。此时会显示新的应用程序概述：*
7. *记录为新应用显示的客户端 ID 和客户端 Secret。*

第 12 章 存储库通知

Quay 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。要添加通知，请在查看存储库时单击 **Settings** 选项卡，然后选择 **Create Notification**。在 **when this event occurs** 字段中，选择要接收通知的项目：



选择某个事件后，通过添加您如何通知该事件来进一步配置它。



注意

添加通知需要 存储库管理权限。

以下是存储库事件的示例。

12.1. 仓库事件

12.1.1. repository Push

成功将一个或多个镜像推送(push)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"
  ]
}
```

12.1.2. Dockerfile 构建队列

以下是 Dockerfile 构建的示例响应已放入构建系统中。响应可能会根据使用可选属性的不同。

```
{
  "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"
}
```

12.1.3. Dockerfile 构建已启动

以下是由构建系统启动的 Dockerfile 构建的示例。根据某些属性是可选的，响应可能会有所不同。

```
{
  "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"
}
```

12.1.4. Dockerfile 构建成功完成

以下是已由构建系统成功完成的 Dockerfile 构建示例。

注意

构建的镜像的 Repository Push 事件将会 **同时进行** 此事件。

```
{
  "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"
  ]
}
```

12.1.5. Dockerfile 构建失败

Dockerfile 构建失败

```
{
  "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"
}
```

12.1.6. Dockerfile 构建取消

已取消 Dockerfile 构建

```
{
  "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"
}
```

12.1.7. 安全漏洞被检测到

程序库中检测到了一个漏洞

```
{
  "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
  }
}
```

12.2. 通知操作

12.2.1. Quay 通知

将向 Quay.io 通知区域添加通知。单击任何 Quay.io 页面右上角的 bell 图标，即可找到通知区域。

Quay.io 通知可以设置为将所有用户、团队 或组织作为整体发送到 用户、团队 或 组织。

12.2.2. 电子邮件

电子邮件地址将发送到指定地址，描述发生的事件。

注意

必须基于每个 软件仓库验证所有 电子邮件地址

12.2.3. Webhook POST

将向指定的 URL 发出 HTTP POST 调用，其中包含事件的数据（请参阅上述各个事件的数据格式）。

当 URL 是 HTTPS 时，调用将从 Quay.io 设置 SSL 客户端证书。此证书的验证将证明该调用源自 Quay.io。带有 2xx 范围内的状态代码的响应被视为成功。带有任何其他状态代码的响应将被视为失败，并导致重试 Webhook 通知。

12.2.4. Flowdock 通知

向 Flowdock 发送消息。

12.2.5. HipChat 通知

向 HipChat 发送消息。

12.2.6. Slack 通知

向 Slack 发送消息。

第 13 章 OCI 支持和 RED HAT QUAY

Red Hat Quay 等容器注册表最初设计为支持 Docker 镜像格式的容器镜像。为了促进在 Docker 外使用额外的运行时，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 registry 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，它的支持不仅限于独立应用程序，而是整个 Kubernetes 平台。这些范围包括 Open Policy Agent(OPA)策略，用于安全并监管到 Helm chart 和 Operator，以帮助应用程序部署。

Red Hat Quay 是一个私有容器 registry，它不仅存储容器镜像，而且支持整个工具生态系统，以帮助管理容器。在 3.6 中对基于 OCI 的工件的支持已从只有 Helm 扩展到默认包含 cosign 和 ztsd 压缩方案。因此，F **EATURE_HELM_OCI_SUPPORT** 已被弃用。

当使用 OpenShift Operator 部署 Red Hat Quay 3.6 时，在 **FEATURE_GENERAL_OCI_SUPPORT** 配置下默认启用对 Helm 和 OCI 工件的支持。如果您需要显式启用该功能，例如之前禁用了该功能，或者已经从未启用的版本升级时，请参考"启用 OCI 和 Helm 支持 "部分。

13.1. HELM 和 OCI 的先决条件

- **可信证书：** Helm 客户端和 Quay 间的通信可以通过 HTTPS 和 Helm 3.5 进行沟通，因此仅支持通过 HTTPS 与可信证书通信的 registry。另外，操作系统必须信任 registry 公开的证书。在以后的 Helm 版本中的支持将允许与远程 registry 进行安全通信。因此，请确保将您的操作系统配置为信任 Quay 使用的证书，例如：

```
$ sudo cp rootCA.pem   /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

- **实验性功能：** 许多用于与 Helm 和 OCI registry 交互的命令使用 **helm chart** 子命令。编写本文时，Helm 中的 OCI 支持仍标记为"实验性"功能，必须明确启用。这可以通过设置环境变量 **HELM_EXPERIMENTAL_OCI=1** 来完成。

- **安装 Helm 客户端：** 从 https://github.com/helm/helm/releases 下载所需版本，例如 https://get.helm.sh/helm-v3.5.3-linux-amd64.tar.gz。解包它并将 helm 二进制文件移到所需的目的地：

```
$ tar -zxvf helm-v3.5.3-linux-amd64.tar.gz
$ mv linux-amd64/helm /usr/local/bin/helm
```

- **在 Quay 中创建机构：** 使用 Quay registry UI 创建一个用于存储 Helm chart 的新组织。例如，创建名为 **helm** 的组织。

13.2. 使用 QUAY 的 HELM CHART

作为 Cloud Native Computing Foundation(CNCF)的研究项目，Helm 已成为 Kubernetes 的事实软件包管理器，因为它简化了应用被打包和部署的方式。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。可以为存储库中的常规分发和使用提供 chart。Helm 仓库是提供 index.yaml 元数据文件的 HTTP 服务器，以及一组可选的打包 chart。从 Helm 版本 3 开始，支持以替代传统仓库的方式在 OCI registry 中分发 chart。要演示如何将 Quay 用作 Helm chart 的 registry，会使用 Helm 仓库中的现有 chart 来展示与 OCI registry for Chart 开发者和用户的交互。

在以下示例中，从红帽实践(CoP)软件仓库中下载了一个 etherpad chart 示例，并使用以下步骤将其推送到本地 Red Hat Quay 存储库：

- *添加适当的软件仓库*
- *使用最新的元数据更新存储库*
- *下载并解压缩 chart，以创建名为 **etherpad** 的本地目录*

例如：

```
$ helm repo add redhat-cop https://redhat-cop.github.io/helm-charts
$ helm repo update
$ helm pull redhat-cop/etherpad --version=0.0.4 --untar
```

*标记 chart 需要使用 **helm chart save** 命令 - 这与使用 **podman tag** 进行标记镜像。*

```
$ helm chart save ./etherpad example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  6850d9b21dd4b87cf20ad49f2e2c7def9655c52ea573e1ddb9d1464eeb6a46a6
size:    3.5 KiB
name:    etherpad
version: 0.0.4
0.0.4: saved
```

*使用 **helm chart list** 命令查看 chart 的本地实例：*

```
helm chart list

REF                                                     NAME     VERSION DIGEST SIZE   CREATED
example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4 etherpad 0.0.4
ce0233f 3.5 KiB 23 seconds
```

*在推送 chart 前，使用 **helm registry login** 命令登录到存储库：*

```
$ helm registry login example-registry-quay-quay-enterprise.apps.user1.example.com
Username: quayadmin
Password:
Login succeeded
```

*使用 **helm chart push** 命令将 chart 推送到本地 Quay 存储库：*

```
$ helm chart push example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

The push refers to repository [example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad]
ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  ce0233fd014992b8e27cc648cdabbebd4dd6850aca8fb8e50f7eef6f2f49833d
size:    3.5 KiB
name:    etherpad
version: 0.0.4
0.0.4: pushed to remote (1 layer, 3.5 KiB total)
```

要测试推送是否正常工作，请删除本地副本，然后从存储库中拉取 chart：

```
$ helm chart rm example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
$ rm -rf etherpad
$ helm chart pull example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

0.0.4: Pulling from example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad
ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  6850d9b21dd4b87cf20ad49f2e2c7def9655c52ea573e1ddb9d1464eeb6a46a6
size:    3.5 KiB
name:    etherpad
version: 0.0.4
Status: Downloaded newer chart for example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4
```

使用 **helm chart 导出** 命令提取 chart 文件：

```
$ helm chart export example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  ce0233fd014992b8e27cc648cdabbebd4dd6850aca8fb8e50f7eef6f2f49833d
size:    3.5 KiB
name:    etherpad
version: 0.0.4
Exported chart to etherpad/
```

13.3. OCI 和 HELM 配置

现在，在 **FEATURE_GENERAL_OCI_SUPPORT** 属性中支持 Helm 的支持。如果需要显式启用该功能，例如之前禁用了该功能，或者已经从未默认启用的版本升级，则需要在 Quay 配置中添加两个属性来启用 OCI 工件的使用：

```
FEATURE_GENERAL_OCI_SUPPORT: true
FEATURE_HELM_OCI_SUPPORT: true
```

表 13.1. OCI 和 Helm 配置

字段	类型	描述
FEATURE_GENERAL_OCI_SUPPORT	布尔值	启用对 OCI 工件的支持 **Default:** True
FEATURE_HELM_OCI_SUPPORT	布尔值	启用对 Helm 工件的支持 **Default:** True

重要

从 Red Hat Quay 3.6 开始，**FEATURE_HELM_OCI_SUPPORT** 已被弃用，并将在以后的 Red Hat Quay 版本中删除。在 Red Hat Quay 3.6 中，默认支持 Helm 工件，并包括在 **FEATURE_GENERAL_OCI_SUPPORT** 属性中。用户不再需要更新其 config.yaml 文件来启用支持。

13.4. COSIGN OCI 支持 RED HAT QUAY

Cosign 是一个工具，可用于签名和验证容器镜像。它使用 ECDSA-P256 签名算法和红帽的简单签名有效负载格式来创建存储在 PKIX 文件中的公钥。私钥作为加密的 PEM 文件存储。

Cosign 目前支持以下内容：

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

13.5. 使用带有 QUAY 的 COSIGN

如果您有 Go 1.16+，您可以使用以下命令直接安装 cosign：

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

接下来，生成密钥对：

```
$ cosign generate-key-pair
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

使用以下命令为密钥对签名：

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

有些用户可能会遇到以下错误：

```
error: 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[]
```

由于 cosign 依赖于 ~/.docker/config.json 进行授权，因此您可能需要执行以下命令：

-

```
$ podman login --authfile ~/.docker/config.json quay-server.example.com
Username:
Password:
Login Succeeded!
```

您可以使用以下命令查看更新的授权配置：

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

13.6. 向 QUAY 添加其他 OCI 介质类型

Helm、cosign 和 ztsd 压缩方案工件默认内置在 Red Hat Quay 3.6 中。对于默认情况下不支持的任何其他 OCI 介质类型，您可以使用以下格式将它们添加到 Quay config.yaml 中的 **ALLOWED_OCI_ARTIFACT_TYPES** 配置中：

```
ALLOWED_OCI_ARTIFACT_TYPES:
  <oci config type 1>:
  - <oci layer type 1>
  - <oci layer type 2>

  <oci config type 2>:
  - <oci layer type 3>
  - <oci layer type 4>
...
```

例如，您可以通过在 config.yaml 中添加以下内容来添加 Singularity(SIF) 支持：

```
...
ALLOWED_OCI_ARTIFACT_TYPES:
  application/vnd.oci.image.config.v1+json
  - application/vnd.dev.cosign.simplesigning.v1+json
  application/vnd.cncf.helm.config.v1+json
  - application/tar+gzip
  application/vnd.sylabs.sif.config.v1+json
  - application/vnd.sylabs.sif.layer.v1+tar
...
```

注意

当添加默认配置的 OCI 介质类型时，用户需要手动添加对 cosign 和 Helm 的支持。在默认情况下，支持 ztsd 压缩方案，因此用户不需要将 OCI 介质类型添加到其 config.yaml 中以启用支持。

13.7. 在 QUAY 中禁用 OCI 工件

如果要禁用 OCI 工件支持，您可以在 config.yaml 中将 **FEATURE_GENERAL_OCI_SUPPORT** 设置为 **False** ：

```
...
FEATURE_GENERAL_OCI_SUPPORT = False
...
```

第 14 章 使用 RED HAT QUAY API

Red Hat Quay 提供了一个完整的 OAuth 2、RESTful API，它：

- 可以从 URL https://<yourquayhost>/api/v1的端点获取每个 Red Hat Quay 实例的端点
- 通过启用 Swagger UI，通过浏览器连接到端点，以获取、删除、发布和放置 Red Hat Quay 设置
- 可以被使用 API 调用和使用 OAuth 令牌的应用程序访问
- 以 JSON 格式发送和接收数据

下面的文本描述了如何访问 Red Hat Quay API，并使用它来查看和修改 Red Hat Quay 集群中的设置。下一节列出了和描述 API 端点。

14.1. 从 QUAY.IO 访问 QUAY API

如果您还没有运行自己的 Red Hat Quay 集群，您可以从 Web 浏览器浏览 Quay.io 提供的 Red Hat Quay API：

> https://docs.quay.io/api/swagger/

显示 Quay.io API 端点的 API Explorer。您将无法看到在 Quay.io 上未启用的 Red Hat Quay 功能的超级用户 API 端点或端点（如存储库镜像）。

从 API Explorer，您可以获得并有时会更改以下信息：

- 账单、订阅和计划
- 存储库构建和构建触发器
- 错误消息和全局信息
- 仓库镜像、清单、权限、通知、漏洞和镜像签名
- 使用日志
- 机构、成员和 OAuth 应用程序
- 用户和机器人帐户
- and more...

选择打开端点来查看端点的每一个部分的 Model Schema。打开端点，输入所有必要的参数（如存储库名称或镜像），然后选择 **尝试退出！** 按钮来查询或更改与 Quay.io 端点关联的设置。

14.2. 创建 OAUTH 访问令牌

要创建 OAuth 访问令牌，以便您可以访问机构的 API：

1. 登录 Red Hat Quay 并选择您的机构（或创建一个新的机构）。
2. 点击左侧导航栏中的 Applications 图标。
3. 选择 Create New Application 并在出现提示时为新应用程序提供名称。

4. 选择新应用。
5. 从左侧导航中选择 Generate Token。
6. 选择设置令牌范围的复选框，然后选择 Generate Access Token。
7. 查看允许并选择授权应用程序的权限来批准它。
8. 复制新生成的令牌，以用于访问 API。

14.3. 从网页浏览器访问 QUAY API

通过启用 Swagger，您可以通过网页浏览器访问您自己的 Red Hat Quay 实例的 API。这个 URL 通过 Swagger UI 和这个 URL 公开 Red Hat Quay API explorer：

```
https://<yourquayhost>/api/v1/discovery.
```

这样，访问 API 的方法不包括在 Red Hat Quay 安装中可用的超级用户端点。以下是通过运行 swagger-ui 容器镜像访问本地系统上运行的 Red Hat Quay API 接口的示例：

```
# export SERVER_HOSTNAME=<yourhostname>
# sudo podman run -p 8888:8080 -e API_URL=https://$SERVER_HOSTNAME:8443/api/v1/discovery
docker.io/swaggerapi/swagger-ui
```

在运行 swagger-ui 容器时，打开 Web 浏览器到 localhost 端口 8888，以通过 swagger-ui 容器查看 API 端点。

为避免在日志中出现错误，如 "API 调用，如果使用浏览器调用 X-Requested-With 标头来调用，"如果从浏览器中调用，请将下面这一行添加到集群中的所有节点上的 **config.yaml** 中，再重启 Red Hat Quay：

```
BROWSER_API_CALLS_XHR_ONLY: false
```

14.4. 从命令行访问 RED HAT QUAY API

您可以通过 Red Hat Quay 集群的 API，使用 **curl** 命令 GET、PUT、POST 或 DELETE 设置。将 **<token>** 替换为您之前创建的 OAuth 访问令牌，以便在以下示例中获取或更改设置。

14.4.1. 获取超级用户信息

```
$ curl -X GET -H "Authorization: Bearer <token_here>" \
    "https://<yourquayhost>/api/v1/superuser/users/"
```

例如：

```
$ curl -X GET -H "Authorization: Bearer mFCdgS7SAIoMcnTsHCGx23vcNsTgziAa4CmmHIsg"
http://quay-server:8080/api/v1/superuser/users/ | jq

{
  "users": [
    {
      "kind": "user",
      "name": "quayadmin",
      "username": "quayadmin",
```

```
    "email": "quayadmin@example.com",
    "verified": true,
    "avatar": {
      "name": "quayadmin",
      "hash": "357a20e8c56e69d6f9734d23ef9517e8",
      "color": "#5254a3",
      "kind": "user"
    },
    "super_user": true,
    "enabled": true
  }
 ]
}
```

14.4.2. 使用 API 创建超级用户

- 配置超级用户名称，如 Deploy Quay book 所述：
  - 使用配置编辑器 UI 或者
  - 直接使用配置 API 来验证（并下载）更新的配置捆绑包，直接编辑 **config.yaml** 文件
- 为超级用户名称创建用户帐户：
  - 获取授权令牌（如上所述），并使用 **curl** 来创建用户：

    ```
    $ curl -H "Content-Type: application/json"  -H "Authorization: Bearer
    Fava2kV9C92p1eXnMawBZx9vTqVnksvwNm0ckFKZ" -X POST --data '{
     "username": "quaysuper",
     "email": "quaysuper@example.com"
    }'  http://quay-server:8080/api/v1/superuser/users/ | jq
    ```

  - 返回的内容包括新用户帐户的生成密码：

    ```
    {
      "username": "quaysuper",
      "email": "quaysuper@example.com",
      "password": "EH67NB3Y6PTBED8H0HC6UVHGGGA3ODSE",
      "encrypted_password":
    "fn37AZAUQH0PTsU+vlO9lS0QxPW9A/boXL4ovZjIFtlUPrBz9i4j9UDOqMjuxQ/0HTfy38go
    KEpG8zYXVeQh3lOFzuOjSvKic2Vq7xdtQsU="
    }
    ```

现在，当您请求用户列表时，它将以超级用户身份显示 **quaysuper** ：

```
$ curl -X GET -H "Authorization: Bearer mFCdgS7SAIoMcnTsHCGx23vcNsTgziAa4CmmHIsg"
http://quay-server:8080/api/v1/superuser/users/ | jq

{
  "users": [
  {
    "kind": "user",
    "name": "quayadmin",
    "username": "quayadmin",
    "email": "quayadmin@example.com",
```

```
      "verified": true,
      "avatar": {
        "name": "quayadmin",
        "hash": "357a20e8c56e69d6f9734d23ef9517e8",
        "color": "#5254a3",
        "kind": "user"
      },
      "super_user": true,
      "enabled": true
    },
    {
      "kind": "user",
      "name": "quaysuper",
      "username": "quaysuper",
      "email": "quaysuper@example.com",
      "verified": true,
      "avatar": {
        "name": "quaysuper",
        "hash": "c0e0f155afcef68e58a42243b153df08",
        "color": "#969696",
        "kind": "user"
      },
      "super_user": true,
      "enabled": true
    }
  ]
}
```


14.4.3. 列出用量日志

一个内部 API **/api/v1/superuser/logs** 可用于列出当前系统的用量日志。这将分页结果，因此在以下示例中创建了超过 20 个仓库，以介绍如何使用多个调用来访问整个结果集。

14.4.3.1. 分页示例

首次调用

```
$ curl -X GET -k -H "Authorization: Bearer qz9NZ2Np1f55CSZ3RVOvxjeUdkzYuCp0pKggABCD"
https://example-registry-quay-quay-enterprise.apps.example.com/api/v1/superuser/logs | jq
```

初始输出


```
{
  "start_time": "Sun, 12 Dec 2021 11:41:55 -0000",
  "end_time": "Tue, 14 Dec 2021 11:41:55 -0000",
  "logs": [
    {
      "kind": "create_repo",
      "metadata": {
        "repo": "t21",
        "namespace": "namespace1"
      },
      "ip": "10.131.0.13",
      "datetime": "Mon, 13 Dec 2021 11:41:16 -0000",
```

```
      "performer": {
        "kind": "user",
        "name": "user1",
        "is_robot": false,
        "avatar": {
          "name": "user1",
          "hash": "5d40b245471708144de9760f2f18113d75aa2488ec82e12435b9de34a6565f73",
          "color": "#ad494a",
          "kind": "user"
        }
      },
      "namespace": {
        "kind": "org",
        "name": "namespace1",
        "avatar": {
          "name": "namespace1",
          "hash": "6cf18b5c19217bfc6df0e7d788746ff7e8201a68cba333fca0437e42379b984f",
          "color": "#e377c2",
          "kind": "org"
        }
      }
    },
    {
      "kind": "create_repo",
      "metadata": {
        "repo": "t20",
        "namespace": "namespace1"
      },
      "ip": "10.131.0.13",
      "datetime": "Mon, 13 Dec 2021 11:41:05 -0000",
      "performer": {
        "kind": "user",
        "name": "user1",
        "is_robot": false,
        "avatar": {
          "name": "user1",
          "hash": "5d40b245471708144de9760f2f18113d75aa2488ec82e12435b9de34a6565f73",
          "color": "#ad494a",
          "kind": "user"
        }
      },
      "namespace": {
        "kind": "org",
        "name": "namespace1",
        "avatar": {
          "name": "namespace1",
          "hash": "6cf18b5c19217bfc6df0e7d788746ff7e8201a68cba333fca0437e42379b984f",
          "color": "#e377c2",
          "kind": "org"
        }
      }
    },
...

  {
    "kind": "create_repo",
```

```
      "metadata": {
        "repo": "t2",
        "namespace": "namespace1"
      },
      "ip": "10.131.0.13",
      "datetime": "Mon, 13 Dec 2021 11:25:17 -0000",
      "performer": {
        "kind": "user",
        "name": "user1",
        "is_robot": false,
        "avatar": {
          "name": "user1",
          "hash": "5d40b245471708144de9760f2f18113d75aa2488ec82e12435b9de34a6565f73",
          "color": "#ad494a",
          "kind": "user"
        }
      },
      "namespace": {
        "kind": "org",
        "name": "namespace1",
        "avatar": {
          "name": "namespace1",
          "hash": "6cf18b5c19217bfc6df0e7d788746ff7e8201a68cba333fca0437e42379b984f",
          "color": "#e377c2",
          "kind": "org"
        }
      }
    }
  ],
  "next_page":
"gAAAAABhtzGDsH38x7pjWhD8MJq1_2FAgqUw2X9S2LoCLNPH65QJqB4XAU2qAxYb6QqtlcWj9eI6
DUiMN_q3e3I0agCvB2VPQ8rY75WeaiUzM3rQlMc4i6ElR78t8oUxVfNp1RMPIRQYYZyXP9h6E8LZZhq
TMs0S-SedaQJ3kVFtkxZqJwHVjgt23Ts2DonVoYwtKgI3bCC5"
}
```


使用 next_page 进行第二个调用

```
$ curl -X GET -k -H "Authorization: Bearer qz9NZ2Np1f55CSZ3RVOvxjeUdkzYuCp0pKggABCD"
https://example-registry-quay-quay-enterprise.apps.example.com/api/v1/superuser/logs?
next_page=gAAAAABhtzGDsH38x7pjWhD8MJq1_2FAgqUw2X9S2LoCLNPH65QJqB4XAU2qAxYb6Q
qtlcWj9eI6DUiMN_q3e3I0agCvB2VPQ8rY75WeaiUzM3rQlMc4i6ElR78t8oUxVfNp1RMPIRQYYZyXP9h
6E8LZZhqTMs0S-SedaQJ3kVFtkxZqJwHVjgt23Ts2DonVoYwtKgI3bCC5 | jq
```

第二个调用的输出


```
{
  "start_time": "Sun, 12 Dec 2021 11:42:46 -0000",
  "end_time": "Tue, 14 Dec 2021 11:42:46 -0000",
  "logs": [
    {
      "kind": "create_repo",
      "metadata": {
        "repo": "t1",
        "namespace": "namespace1"
      },
```

```
    "ip": "10.131.0.13",
    "datetime": "Mon, 13 Dec 2021 11:25:07 -0000",
    "performer": {
      "kind": "user",
      "name": "user1",
      "is_robot": false,
      "avatar": {
        "name": "user1",
        "hash": "5d40b245471708144de9760f2f18113d75aa2488ec82e12435b9de34a6565f73",
        "color": "#ad494a",
        "kind": "user"
      }
    },
    "namespace": {
      "kind": "org",
      "name": "namespace1",
      "avatar": {
        "name": "namespace1",
        "hash": "6cf18b5c19217bfc6df0e7d788746ff7e8201a68cba333fca0437e42379b984f",
        "color": "#e377c2",
        "kind": "org"
      }
    }
  },
  ...
 ]
}
```

14.4.4. 目录同步

要在机构 **testadminorg** 中为团队 **newteam** 启用目录同步，其中 LDAP 中对应的组名称是 **ldapgroup** ：

```
$ curl -X POST -H "Authorization: Bearer 9rJYBR3v3pXcj5XqIA2XX6Thkwk4gld4TCYLLWDF" \
    -H "Content-type: application/json" \
    -d '{"group_dn": "cn=ldapgroup,ou=Users"}' \
    http://quay1-server:8080/api/v1/organization/testadminorg/team/newteam/syncing
```

为同一团队禁用同步：

```
$ curl -X DELETE -H "Authorization: Bearer 9rJYBR3v3pXcj5XqIA2XX6Thkwk4gld4TCYLLWDF" \
    http://quay1-server:8080/api/v1/organization/testadminorg/team/newteam/syncing
```

14.4.5. 通过 API 创建存储库构建

要从指定的输入构建存储库，并使用自定义标签标记构建，用户可以使用 requestRepoBuild 端点。它采用以下数据：

```
{
"docker_tags": [
  "string"
],
"pull_robot": "string",
```

```
"subdirectory": "string",
"archive_url": "string"
}
```

archive_url 参数应指向包含 Dockerfile 和其他构建所需文件的 **tar** 或 **zip** 存档。**file_id** 参数是我们较旧的构建系统外。它无法再使用。如果 Dockerfile 位于子目录中，也需要指定它。

该存档应公开访问。OAuth 应用应具有 "Administer Organization" 范围，因为只有机构管理员可以访问机器人的帐户令牌。否则，某人可以通过简单地授予构建对机器人（没有访问权限本身）来获取机器人权限，并使用它来获取镜像内容。如果出现错误，请检查 json 块返回的，并确保归档位置、拉取机器机器和其他参数正确传递。点单个构建页面右上角的" 下载日志 " 检查日志以了解更详细的消息。

14.4.6. 创建机构机器人

```
$ curl -X PUT https://quay.io/api/v1/organization/{orgname}/robots/{robot shortname} \
  -H 'Authorization: Bearer <token>"
```

14.4.7. 触发构建

```
$ curl -X POST https://quay.io/api/v1/repository/YOURORGNAME/YOURREPONAME/build/ \
  -H 'Authorization: Bearer <token>'
```

带有请求的 Python

```
import requests
r = requests.post('https://quay.io/api/v1/repository/example/example/image', headers={'content-type':
'application/json', 'Authorization': 'Bearer <redacted>'}, data={[<request-body-contents>})
print(r.text)
```

14.4.8. 创建私有存储库

```
$ curl -X POST https://quay.io/api/v1/repository \
   -H 'Authorization: Bearer {token}' \
   -H 'Content-Type: application/json' \
   -d '{"namespace":"yournamespace", "repository":"yourreponame",
   "description":"descriptionofyourrepo", "visibility": "private"}' | jq
```