Red Hat Quay 3.11

使用 Red Hat Quay

使用 Red Hat Quay

Last Updated: 2024-06-06

Red Hat Quay 3.11 使用 Red Hat Quay

使用 Red Hat Quay

法律通告

摘要

了解如何使用 Red Hat Quay

目录

前言

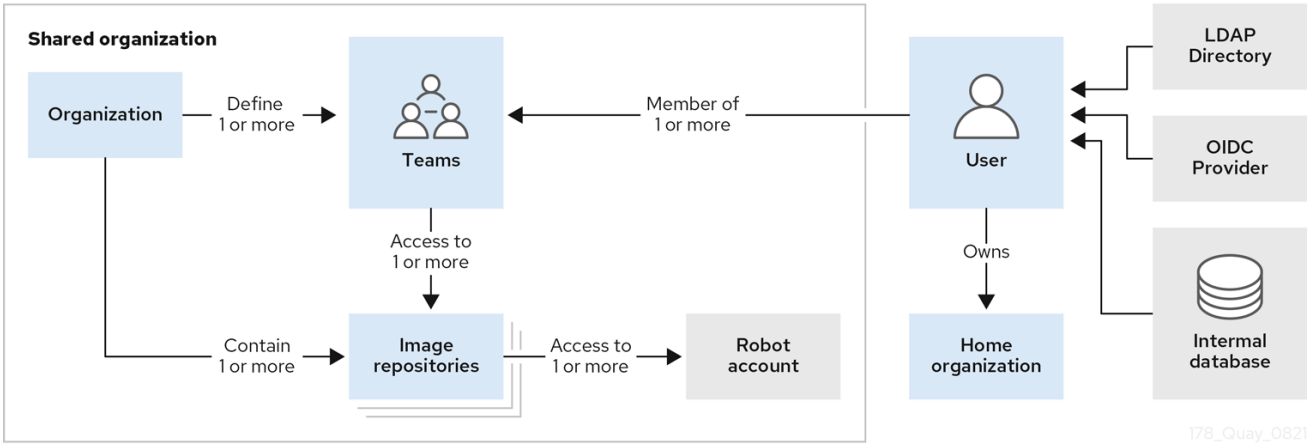
Red Hat Quay 容器镜像 registry 可让您将容器镜像存储在中央位置。作为 Red Hat Quay registry 的常规用户，您可以创建存储库来组织镜像，并选择性地添加对您控制的存储库的读取(pull)和写入(push)访问权限。具有管理特权的用户可以执行更广泛的任务，例如添加用户和控制默认设置。

本指南假定您已部署 Red Hat Quay，并准备好开始设置并使用它。

第 1 章 用户和机构

在创建存储库以在 Red Hat Quay 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 Red Hat Quay，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。

1.1. 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&gt**; 或 **quay.io/<username>**。
- **超级用户** 通过用户界面中的 **Super User Admin Panel** 增强了访问权限和特权。超级用户 API 调用也可用，这对普通用户不可见或访问。
- **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。

1.2. 创建用户帐户

Red Hat Quay 的用户帐户代表个人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 Red Hat Quay 中的容器镜像管理的基础。

使用以下步骤为您的 Red Hat Quay 存储库创建新用户。

先决条件

- 您已在 **config.yaml** 文件中配置了超级用户。如需更多信息，请参阅配置 Red Hat Quay 超级用户。

流程

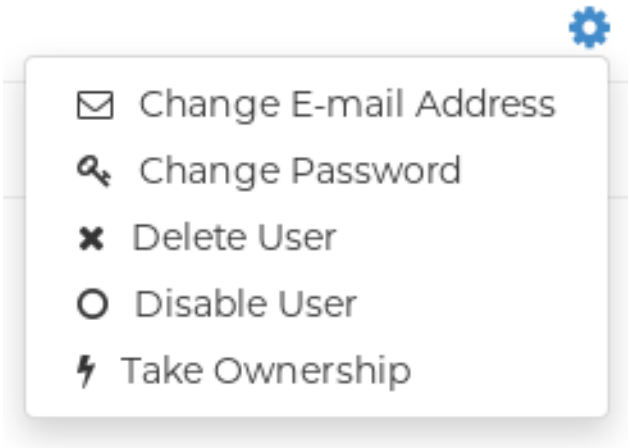
1. 以超级用户身份登录到您的 Red Hat Quay 存储库。

2. 在导航窗格中，选择您的帐户名称，然后单击 **Super User Admin Panel**。
3. 单击列中的 **Users** 图标。
4. 点 **创建用户** 按钮。
5. 输入新用户的用户名和电子邮件地址，然后单击 **创建用户** 按钮。
6. 您会被重定向到 **Users** 页面，其中现在还有另一个 Red Hat Quay 用户。

注意

您可能需要刷新 **Users** 页面来显示其他用户。

7. 在 **Users** 页面中，单击与新用户关联的 **Options** cogwheel。此时会出现一个下拉菜单，如下图所示：



8. 单击 **Change Password**。
9. 添加新密码，然后单击 **Change User Password**。
新用户现在可以使用该用户名和密码通过 Web UI 或其首选的容器客户端（如 Docker 或 Podman）登录。

1.3. 从命令行删除 RED HAT QUAY 用户

在 Red Hat Quay UI 的 **Superuser Admin** 面板中访问 **Users** 选项卡时，您可能会遇到没有列出用户的情况。相反，会显示一条消息，表示 Red Hat Quay 被配置为使用外部身份验证，用户只能在该系统中创建。

这个错误由以下两个原因之一：

- 加载用户时，Web UI 会超时。发生这种情况时，用户无法对其执行任何操作。
- 在 LDAP 身份验证中。当更改 userID 时，但关联的电子邮件不是。目前，Red Hat Quay 不允许创建具有旧电子邮件地址的新用户。

在出现这个问题时，使用以下步骤从 Red Hat Quay 中删除用户。

流程

- 输入以下 **curl** 命令从命令行删除用户：

```
$ curl -X DELETE -H "Authorization: Bearer <insert token here>"
https://<quay_hostname>/api/v1/superuser/users/<name_of_user>
```

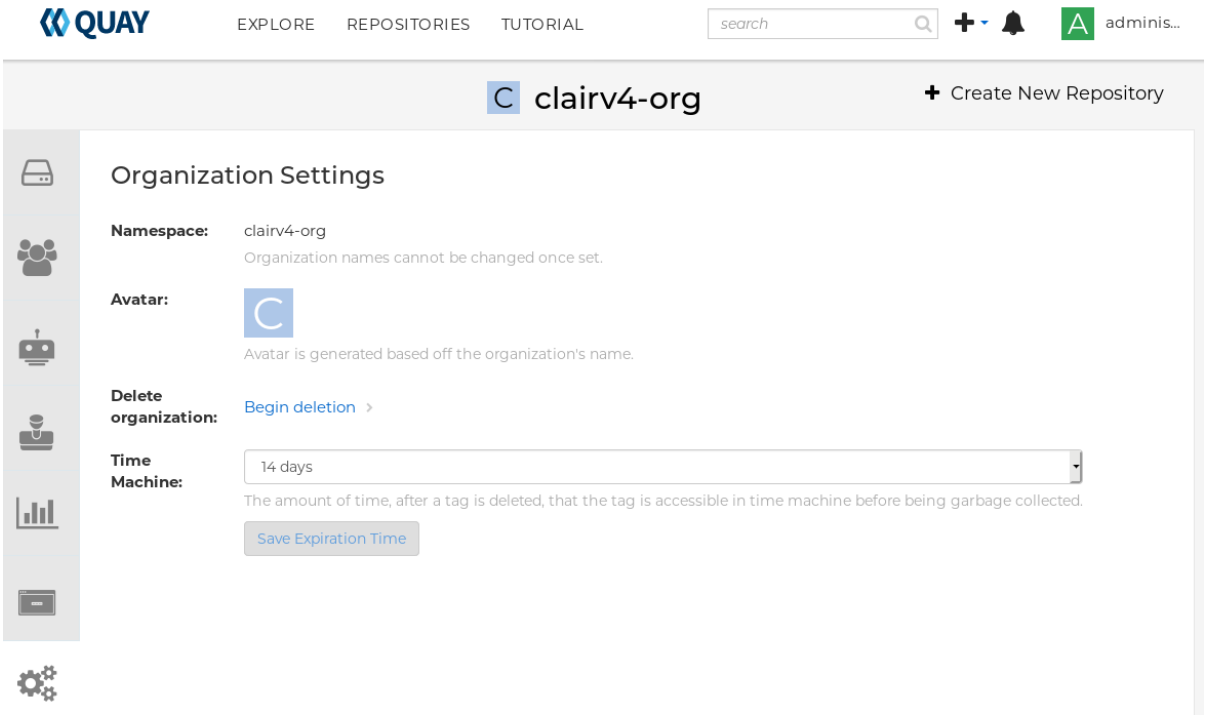
注意

删除用户后，此用户在其专用帐户中具有的任何存储库都不可用。

1.4. 创建机构帐户

任何用户都可以创建自己的组织来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 个字符之间的长度
3. 选择 Create Organization。这时将显示新组织，可供您开始从左列中的图标添加存储库、团队、机器人帐户和其他功能。下图显示了选择设置选项卡的新组织页面示例。



第 2 章 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。

Red Hat Quay 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。

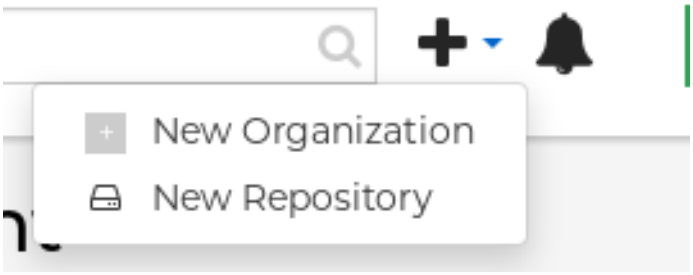
在 Red Hat Quay 中创建存储库的方法有两种：使用相关 **docker** 或 **podman** 命令推送镜像，或者使用 Red Hat Quay UI。

2.1. 使用 UI 创建镜像存储库

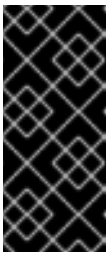
使用以下步骤使用 Red Hat Quay UI 创建存储库。

流程

1. 通过 Web UI 登录您的用户帐户。
2. 在 Red Hat Quay 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。

2.2. 使用 CLI 创建镜像存储库

使用正确的凭证，您可以使用 Red Hat Quay 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。

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：

```
$ sudo 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：

```
$ sudo podman tag docker.io/library/busybox quay-
server.example.com/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。

```
$ sudo podman push --tls-verify=false quay-server.example.com/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
```

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 Red Hat Quay 用户帐户。

3.1. 允许访问用户软件仓库

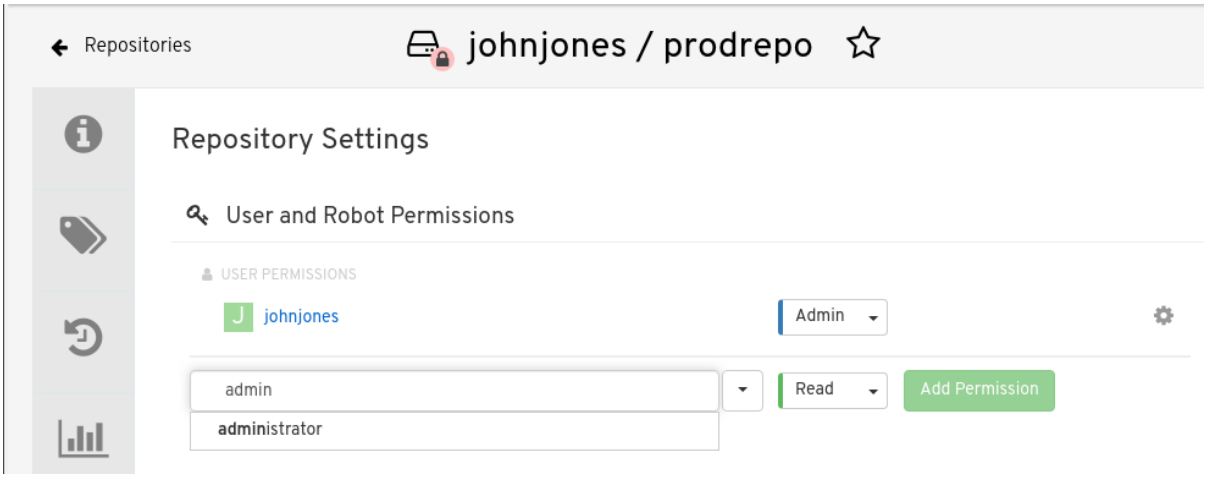
在用户命名空间中创建存储库时，您可以将对该存储库的访问权限添加到用户帐户或通过 Robot Accounts 添加。

3.1.1. 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 Red Hat Quay。
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。

3.1.2. 允许机器人访问用户存储库

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 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。

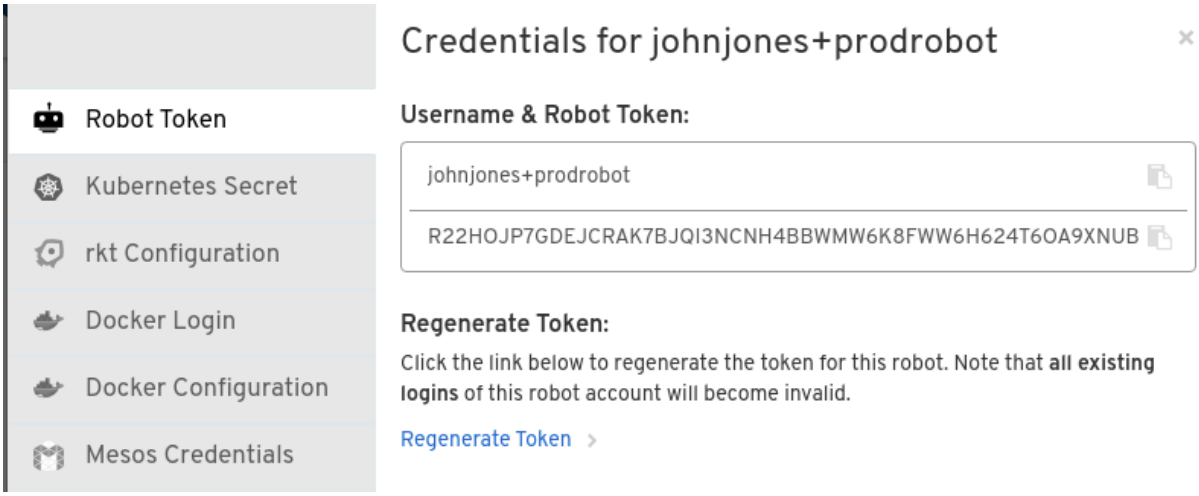
使用以下步骤设置允许访问您的存储库的 Robot 帐户。

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：

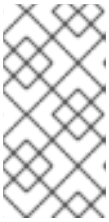
- **Kubernetes Secret**: 选择此项以 Kubernetes pull secret yaml 文件的形式下载凭证。
- **rkt Configuration** ：选择此项以 **.json** 文件的形式下载 rkt 容器运行时的凭证。
- **docker Login** ：选择此项以复制包含凭据的完整 **docker login** 命令行。
- **docker Configuration** ：选择此项以下载用作 Docker **config.json** 文件的文件，以将凭证永久存储在客户端系统上。
- **Mesos Credentials**: 选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

3.2. 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 Red Hat Quay 中，用户组可以是 *团队*，也可以是具有相同权限的用户集合，或是 *个人用户*。

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。

3.2.1. 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. 点 Create Organization 以完成创建。

3.2.1.1. 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"
```

3.2.2. 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。

3.2.3. 设置团队角色

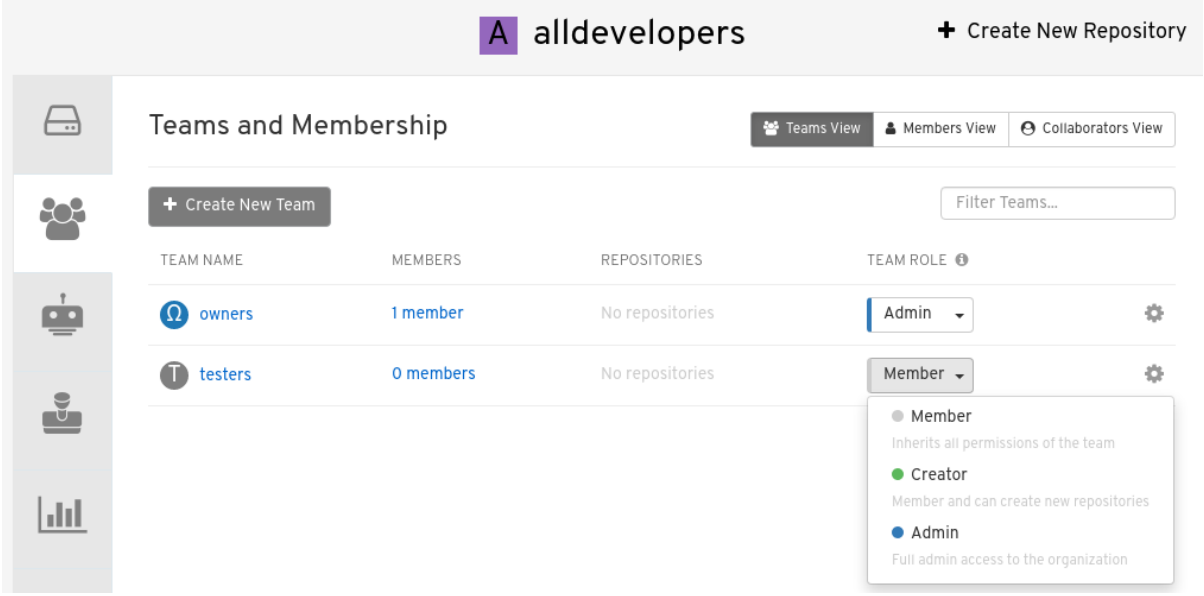
添加团队后，您可以在机构中设置该团队的角色。

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。

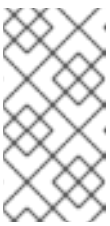
3.2.4. 将用户添加到团队

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，Red Hat Quay 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

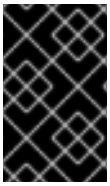
机器人帐户会立即添加到团队。对于用户帐户，加入的邀请将发送给用户。在用户接受该邀请前，用户仍然处于 **INVITED TO JOIN** 状态。用户接受电子邮件邀请加入团队后，他们会从 **INVITED TO JOIN** 列表移到机构的 **MEMBERS** 列表。

其他资源

创建 OAuth 访问令牌

3.3. 禁用机器人帐户

Red Hat Quay 管理员可以通过禁止用户创建新的机器人帐户来管理机器人帐户。



重要

机器人帐户对于存储库镜像是必需的。将 **ROBOTS_DISALLOW** 配置字段设置为 **true** 可中断镜像配置。用户镜像仓库不应在其 **config.yaml** 文件中将 **ROBOTS_DISALLOW** 设置为 **true**。这是一个已知问题，并将在以后的 Red Hat Quay 版本中解决。

使用以下步骤禁用机器人帐户创建。

先决条件

- 您已创建了多个机器人帐户。

流程

1. 更新 **config.yaml** 字段以添加 **ROBOTS_DISALLOW** 变量，例如：

```
ROBOTS_DISALLOW: true
```

2. 重启 Red Hat Quay 部署。

验证：创建新的机器人帐户

1. 导航到您的 Red Hat Quay 存储库。
2. 点存储库的名称。
3. 在导航窗格中，单击 **Robot Accounts**。
4. 单击 **Create Robot Account**。
5. 输入机器人帐户的名称，例如 < **organization-name/username>+<robot-name** >。
6. 单击 **Create robot account** 以确认创建。显示以下消息： **Cannot create robot account.机器人帐户被禁用。请联系您的管理员。**

验证：登录到机器人帐户

1. 在命令行界面(CLI)中，输入以下命令尝试以机器人帐户身份登录：

```
$ podman login -u="<organization-name/username>+<robot-name>" -
p="KETJ6VN0WT8YLLNXUJJ4454ZI6TZJ98NV41OE02PC2IQXVXRFQ1EJ36V12345678"
<quay-server.example.com>
```

返回以下出错信息：

```
Error: logging into "<quay-server.example.com>": invalid username/password
```

2. 您可以传递 **log-level=debug** 标志，以确认机器人帐户已被取消激活：

```
$ podman login -u="<organization-name/username>+<robot-name>" -
p="KETJ6VN0WT8YLLNXUJJ4454ZI6TZJ98NV41OE02PC2IQXVXRFQ1EJ36V12345678" -
-log-level=debug <quay-server.example.com>
```

```
...
DEBU[0000] error logging into "quay-server.example.com": unable to retrieve auth token:
invalid username/password: unauthorized: Robot accounts have been disabled. Please
contact your administrator.
```

第 4 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：

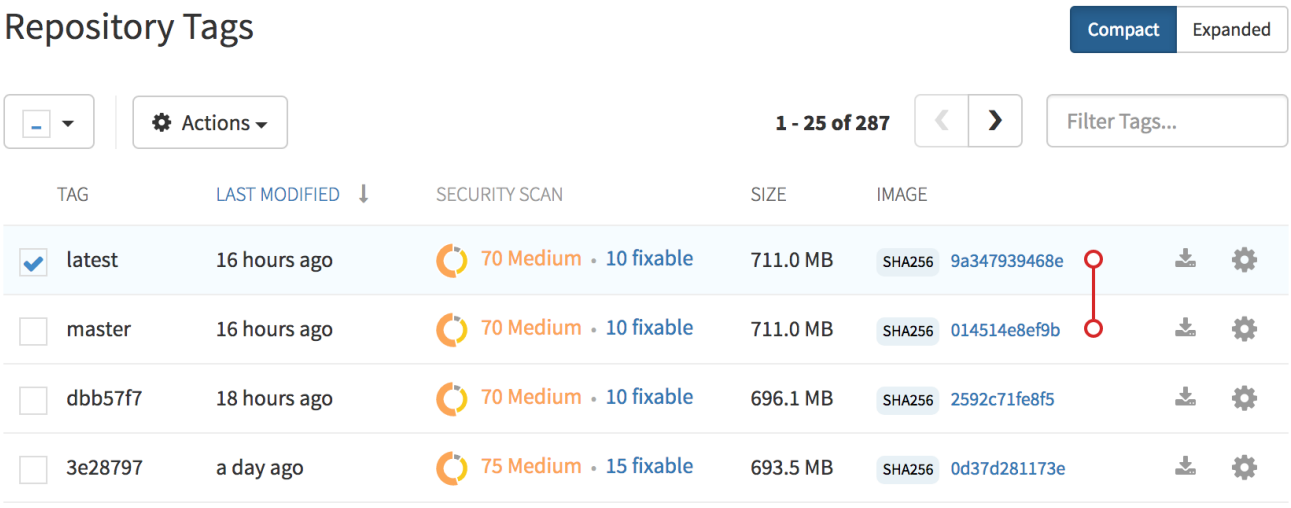
- **版本和发布** ：通过镜像标签，您可以表示应用程序或软件的不同版本或版本。例如，您可能有一个标记为 *v1.0* 的镜像，以代表更新版本的初始发行版本和 *v1.1*。这有助于维护镜像版本的清晰记录。
- **回滚和测试** ：如果您遇到新镜像版本的问题，您可以通过指定标签来轻松恢复到以前的版本。这在调试和测试阶段特别有用。
- **开发环境** ：在使用不同环境时镜像标签很有用。您可以使用 *dev* 标签进行开发版本，*qa* 用于质量保证测试，以及生产环境的 *prod*，每个标签都有自己的功能和配置。
- **持续集成/持续部署(CI/CD)** ：CI/CD 管道通常使用镜像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。

4.1. 查看和修改标签

要查看 Red Hat Quay 上的镜像标签，请导航到存储库并单击 **Tags** 选项卡。例如：

查看和修改存储库中的标签



4.1.1. 向镜像添加新镜像标签

您可以向 Red Hat Quay 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。

4.1.2. 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。Red Hat Quay 确认您希望标签移动，而不是添加标签。

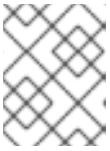
4.1.3. 删除镜像标签

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"。

4.1.3.1. 查看标签历史记录

Red Hat Quay 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。

4.1.3.2. 恢复标签更改

Red Hat Quay 提供了一个全面 *的时间机器* 功能，它允许旧的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。

4.1.4. 通过标签或摘要获取镜像

Red Hat Quay 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-server.example.com/quayadmin/busybox:test2
```

4.2. 标签过期

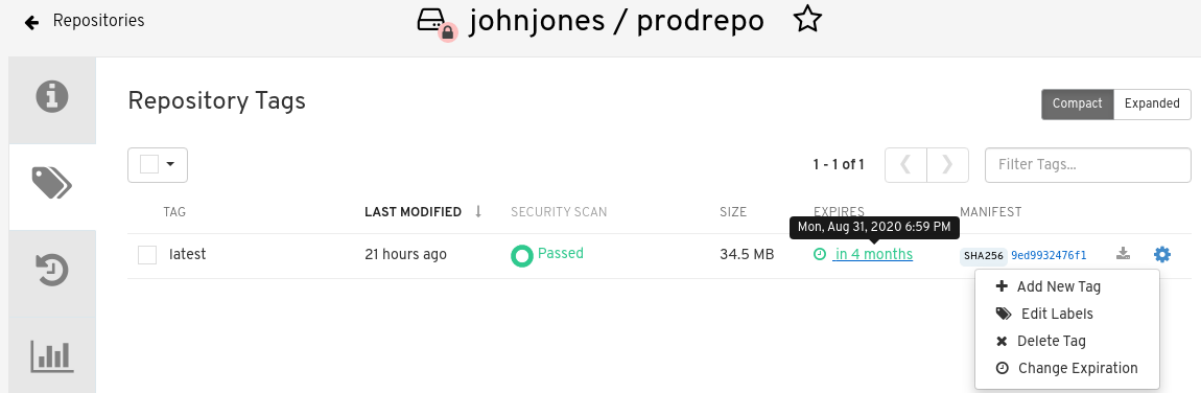
可以使用 *标签到期功能，将镜像设置为在所选日期和时间时从 Red Hat Quay 存储库过期*。此功能包括以下特征：

- 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。

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特权。超级用户没有中央机制来收集信息并操作用户存储库。每个存储库的所有者是管理过期和删除其镜像的所有者。

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 Red Hat Quay UI 上的过期日期。例如：



4.2.1. 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。

4.2.2. 从存储库设置标签过期

标签过期可以在 Red Hat Quay UI 中设置。

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。

4.3. 查看 CLAIR 安全扫描

默认情况下，Red Hat Quay 不启用 Clair 安全扫描程序。要启用 Clair，请参阅 Red Hat Quay 上的 Clair。

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。Red Hat Quay 超级用户不作于发现的漏洞。

第 5 章 查看和导出日志

为 Red Hat Quay 中的所有存储库和命名空间收集活动日志。

查看 Red Hat Quay 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。

5.1. 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。

5.2. 导出存储库日志

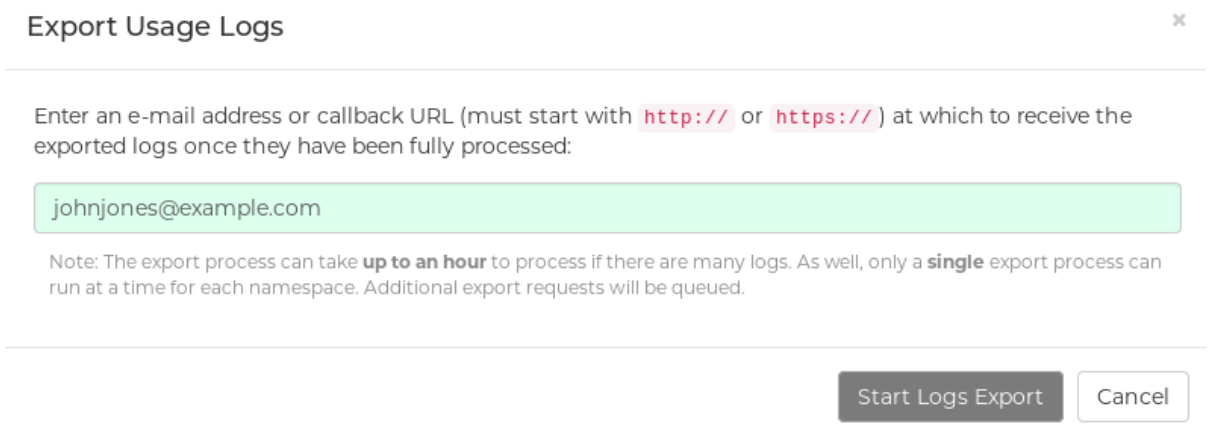
您可以使用 **Export Logs** 功能获取大量日志文件，并将它们保存在 Red Hat Quay 数据库之外。这个功能有以下优点和限制：

- 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。
- 必须为您的 Red Hat Quay 实例配置此功能的外部存储。本地存储无法导出日志。
- 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。

注意

URL 指向 Red Hat Quay 外部存储中的位置，并设置为在一小时内过期。如果要保留日志，请确保在过期时间之前复制导出的日志。

第 6 章 自动使用构建 WORKER 构建 DOCKERFILE

Red Hat Quay 支持使用 OpenShift Container Platform 或 Kubernetes 上的一组 worker 节点来构建 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本的存储库。

本文档演示了如何使用 Red Hat Quay 安装启用构建，并设置多个 OpenShift Container Platform 或 Kubernetes 集群以接受 Red Hat Quay 的构建。

6.1. 使用 OPENSHIFT CONTAINER PLATFORM 设置 RED HAT QUAY BUILDER

在 OpenShift Container Platform 中使用前，您必须预先配置 Red Hat Quay Builders。

6.1.1. 配置 OpenShift Container Platform TLS 组件

tls 组件允许您控制 TLS 配置。

注意

当 TLS 组件由 Red Hat Quay Operator 管理时，Red Hat Quay 不支持 Builder。

如果将 **tls** 设置为 **unmanaged**，则提供自己的 **ssl.cert** 和 **ssl.key** 文件。在本实例中，如果希望集群支持 Builder，您必须将 **Quay** 路由和 Builder 路由名称添加到证书中的 SAN 列表中；也可以使用通配符。

要添加 builder 路由，请使用以下格式：

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

6.1.2. 为 Red Hat Quay Builder 准备 OpenShift Container Platform

使用以下步骤为 OpenShift Container Platform 准备 Red Hat Quay Builders。

先决条件

- 您已配置了 OpenShift Container Platform TLS 组件。

流程

1. 输入以下命令来创建运行 Builds 的项目，如 **builder** ：

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

2. 输入以下命令在 **构建器** 命名空间中创建一个新的 **ServiceAccount** ：

```
$ oc create sa -n builder quay-builder
```

3. 输入以下命令在 **builder** 命名空间中授予用户 **edit** 角色：

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

4. 输入以下命令来检索与 **builder** 命名空间中 **quay-builder** 服务帐户关联的令牌。此令牌用于身份验证并与 OpenShift Container Platform 集群的 API 服务器交互。

```
$ oc sa get-token -n builder quay-builder
```

5. 识别 OpenShift Container Platform 集群的 API 服务器的 URL。这可以在 OpenShift Container Platform Web 控制台中找到。

6. 识别调度构建作业时要使用的 worker 节点标签。因为构建 pod 需要在裸机 worker 节点上运行，所以通常它们使用特定的标签标识。
使用集群管理员进行检查，以确定应使用哪些节点标签。

7. 可选。如果集群使用自签名证书，则必须获取 Kube API 服务器的证书颁发机构(CA)以添加到 Red Hat Quay 的额外证书。

   a. 输入以下命令获取包含 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 Container Platform Web 控制台中的 secret 获取 **ca.crt** 键值。该值以 **"-----BEGIN CERTIFICATE-----"'** 开头。

   c. 将 CA 导入到 Red Hat Quay。确保此文件的名称与 **K8S_API_TLS_CA** 匹配。

8. 为 **ServiceAccount** 创建以下 **SecurityContextConstraints** 资源：

```
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
```

6.1.3. 配置 Red Hat Quay Builders

使用以下步骤启用 Red Hat Quay Builders。

流程

1. 确保您的 Red Hat Quay **config.yaml** 文件启用了 Builds，例如：

```
FEATURE_BUILD_SUPPORT: True
```

2. 在 Red Hat Quay **config.yaml** 文件中添加以下信息，将每个值替换为与您的特定安装相关的信息：

```
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
```

有关每个配置字段的更多信息，请参阅

6.2. OPENSHIFT CONTAINER PLATFORM *路由* 限制

当您将 Red Hat Quay Operator on OpenShift Container Platform 与受管 **路由** 组件搭配使用时，会有以下限制：

- 目前，OpenShift *Container Platform 路由* 只能为单个端口提供流量。设置 Red Hat Quay 构建需要额外的步骤。
- 确保您的 **kubectl** 或 **oc** CLI 工具被配置为与安装 Red Hat Quay Operator 的集群一起工作，且您的 **QuayRegistry** 存在；QuayRegistry 不必位于运行 *Builders* 的同一裸机集群中。
- 按照以下步骤，确保 OpenShift 集群上启用了 HTTP/2 ingress。
- Red Hat Quay Operator 创建一个 **Route** 资源，将 gRPC 流量定向到现有 **Quay** pod 或 pod 内运行的 Build manager 服务器。如果要使用自定义主机名或 < **builder-registry.example.com** > 等子域，请确保使用 DNS 供应商创建一个 CNAME 记录，指向创建 **Route** 资源的 **status.ingress[0].host**。例如：

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

- 使用 OpenShift Container Platform UI 或 CLI，使用 Build 集群 CA 证书更新 **QuayRegistry** 的 **spec.configBundle Secret** 引用的 Secret。将密钥命名为 **extra_ca_cert_build_cluster.cert**。使用配置 Red Hat Quay Builders 时创建的 Builder 配置中引用的正确值更新 **config.yaml** 文件条目，并添加 **BUILDMAN_HOSTNAME** CONFIGURATION FIELD：

```
BUILDMAN_HOSTNAME: <build-manager-hostname> 1
BUILD_MANAGER:
```

```
- ephemeral
- ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  JOB_REGISTRATION_TIMEOUT: 600
  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
    ...
```

1 构建作业用来回 Build 管理器的外部访问服务器主机名。默认为与 **SERVER_HOSTNAME** 相同。对于 OpenShift **Route**，如果使用自定义主机名，则为 **status.ingress[0].host** 或 CNAME 条目。**BUILDMAN_HOSTNAME** 必须包含端口号，例如，OpenShift Container Platform Route 的 **somehost:443**，因为用于与构建管理器通信的 gRPC 客户端如果省略，则不会推断任何端口。

6.3. 构建故障排除

由 Build manager 启动的 *Builder* 实例是临时的。这意味着，Red Hat Quay 会在超时或失败时关闭，或者 control plane (EC2/K8s)收集的垃圾回收。要获取构建日志，您必须在构建运行时这样做。

6.3.1. DEBUG config 标志

DEBUG 标志可以设置为 **true**，以防止 Builder 实例在完成或失败后被清理。例如：

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

当设置为 **true** 时，debug 功能可防止 Build 节点在 **quay-builder** 服务完成后或失败后关闭。它还可防止 Build manager 通过终止 EC2 实例或删除 Kubernetes 作业来清理实例。这允许调试 Builder 节点问题。

不应在生产环境中设置调试。生命周期服务仍然存在；例如，实例在大约 2 小时后仍然关闭。发生这种情况时，EC2 实例将被终止，Kubernetes 作业已完成。

启用调试也会影响 **ALLOWED_WORKER_COUNT**，因为未终止的实例和作业仍计入正在运行的 worker 的总数。因此，如果达到 **ALLOWED_WORKER_COUNT**，则必须手动删除现有的 Builder worker，才能调度新的构建。

设置 DEBUG 也会影响 **ALLOWED_WORKER_COUNT**，因为未确定的实例/作业仍会计入正在运行的 worker 总数。这意味着，如果达到 **ALLOWED_WORKER_COUNT**，则需要手动删除现有的构建程序，才能调度新的构建。

6.3.2. OpenShift Container Platform 和 Kubernetes 构建故障排除

使用以下步骤对 OpenShift Container Platform Kubernetes 构建进行故障排除。

流程

1. 输入以下命令在本地机器和使用 OpenShift Container Platform 集群或 Kubernetes 集群运行的 pod 之间创建一个端口转发隧道：

```
$ oc port-forward <builder_pod> 9999:2222
```

2. 使用指定的 SSH 密钥和端口建立到远程主机的 SSH 连接，例如：

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

3. 输入以下命令来获取 **quay-builder** 服务日志：

```
$ systemctl status quay-builder
```

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

6.4. 设置 GITHUB 构建

如果您的组织计划通过推送到 Github 或 Github Enterprise 来执行构建，*请在 GitHub 中继续创建 OAuth 应用程序*。

第 7 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。

Red Hat Quay 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。

7.1. 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，因为由 Red Hat Quay 创建的 Build 与在本地机器上运行 **docker build** 或 **podman build** 不同。

Red Hat Quay Build 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，Red Hat Quay Build worker 会将 Git 存储库克隆到 worker 机器，然后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 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。

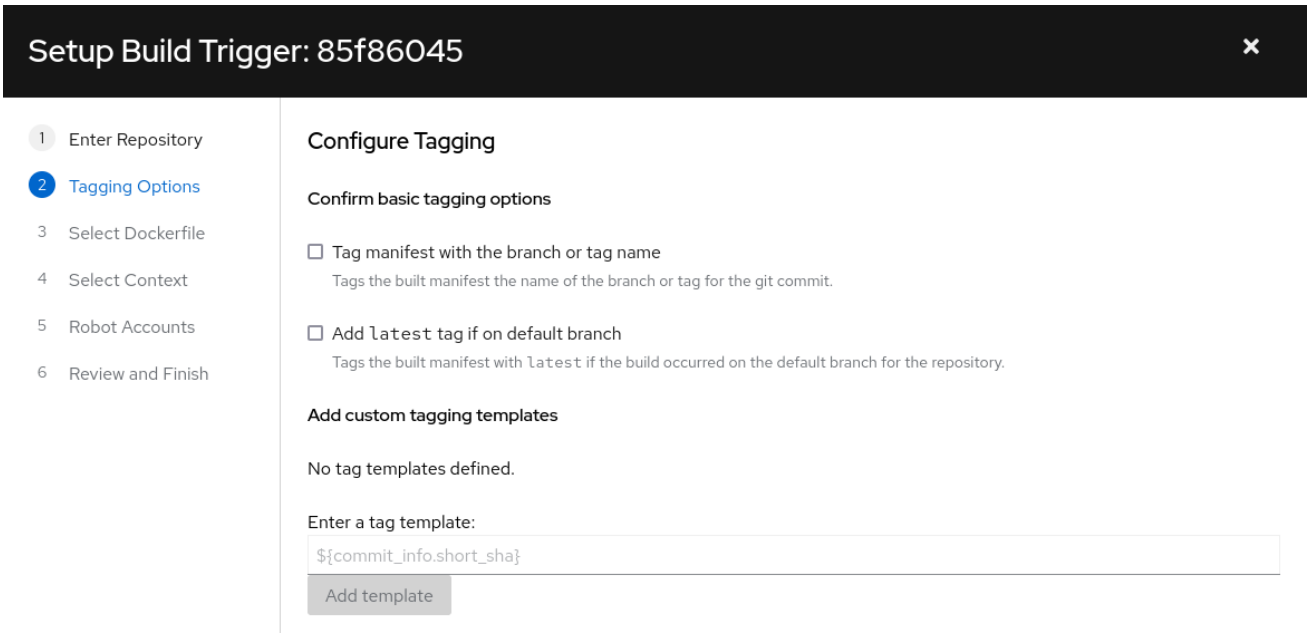
如果在 Build 触发器设置中指定子目录，如 **subdir**，则只有其中的 Dockerfile 可用于构建。这意味着您无法使用 Dockerfile 中的 **ADD** 命令来添加文件，因为它位于构建上下文之外。

与 Docker Hub 不同，Dockerfile 是 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。

7.2. 构建触发器的标签命名

自定义标签可用于 Red Hat Quay。

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

7.3. 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，请在提交消息的任何位置添加文本 [skip build] 或 [build skip]。

7.4. 查看和管理构建

存储库构建可以在 Red Hat Quay UI 上查看和管理。

流程

1. 使用 UI 导航到 Red Hat Quay 存储库。
2. 在导航窗格中，选择 Builds。

7.5. 创建新构建

Red Hat Quay 可以创建新构建，只要在其 config.yaml 文件中将 FEATURE_BUILD_SUPPORT 设置为 true。

先决条件

- 您已导航到存储库的 Builds 页面。
- 在 config.yaml 文件中，FEATURE_BUILD_SUPPORT 被设置为 true。

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**。

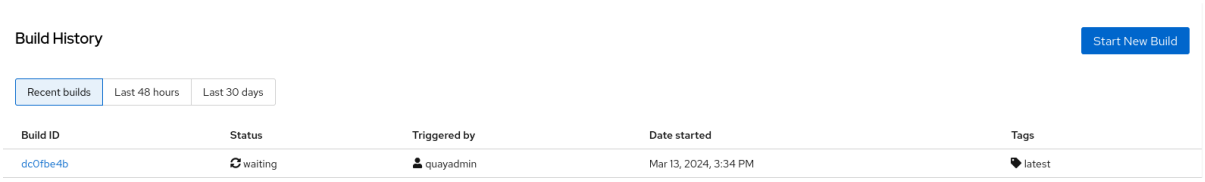
注意

- 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**。



7.6. 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。

7.6.1. 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。

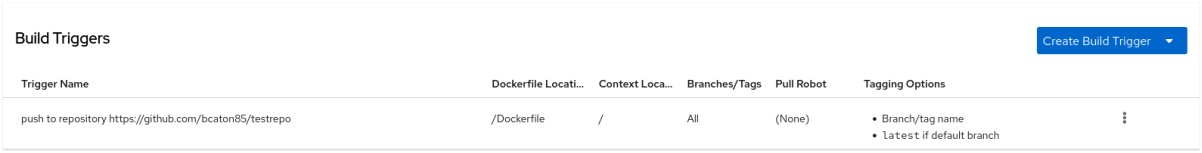
8. 点击 **Next。**检查任何验证警告。如有必要，请在单击 **Finish 前修复**问题。

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：



7.6.2. 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**。

7.7. 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。

7.7.1. 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：

- Red Hat Quay 无法自动检测要与触发器搭配使用的正确 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。

7.7.2. 自定义触发器创建设置

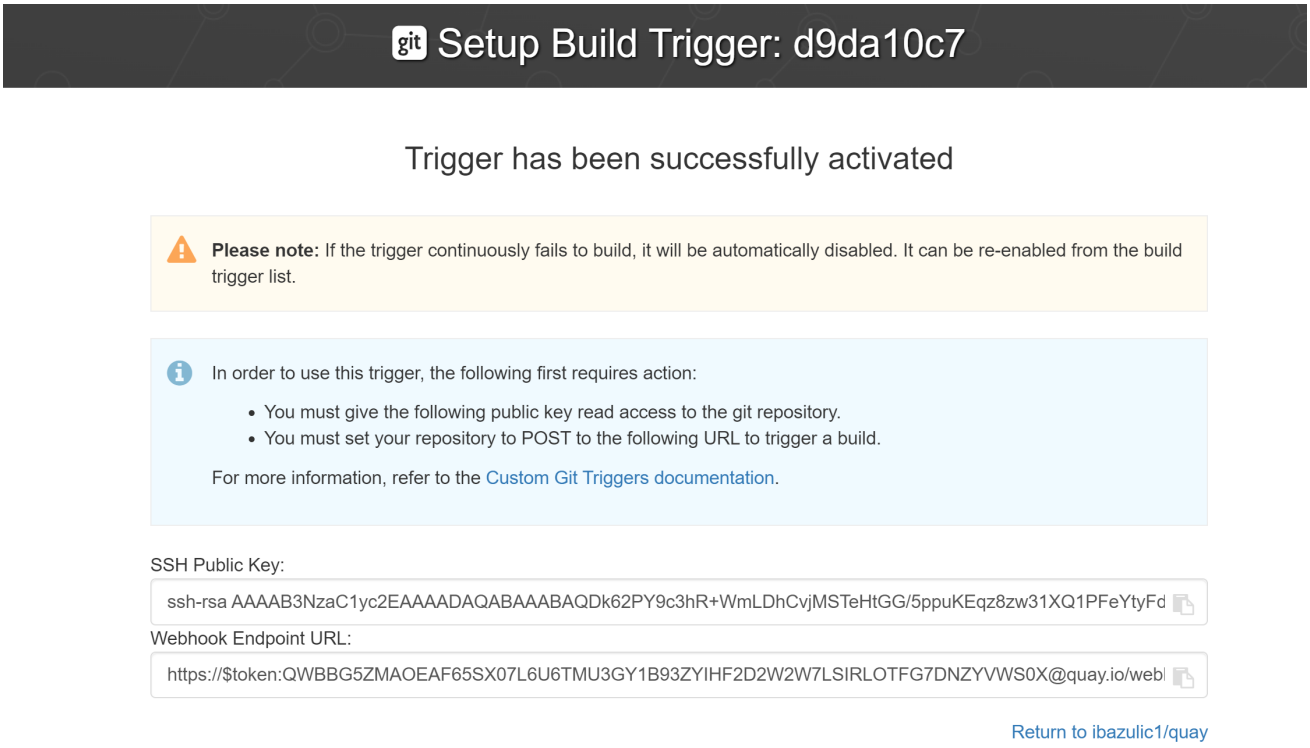
在创建自定义 Git 触发器时，需要额外的步骤：

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. 您必须设置一个将 POST 到 Red Hat Quay 端点的 webhook 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



7.7.2.1. SSH 公钥访问

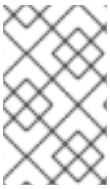
根据 Git 服务器配置，可以通过多种方式安装 Red Hat Quay 为自定义 Git 触发器生成的 SSH 公钥。

例如，Git 文档描述了一个小型服务器设置，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。

7.7.2.2. 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 8 章 在 GITHUB 中创建 OAUTH 应用程序

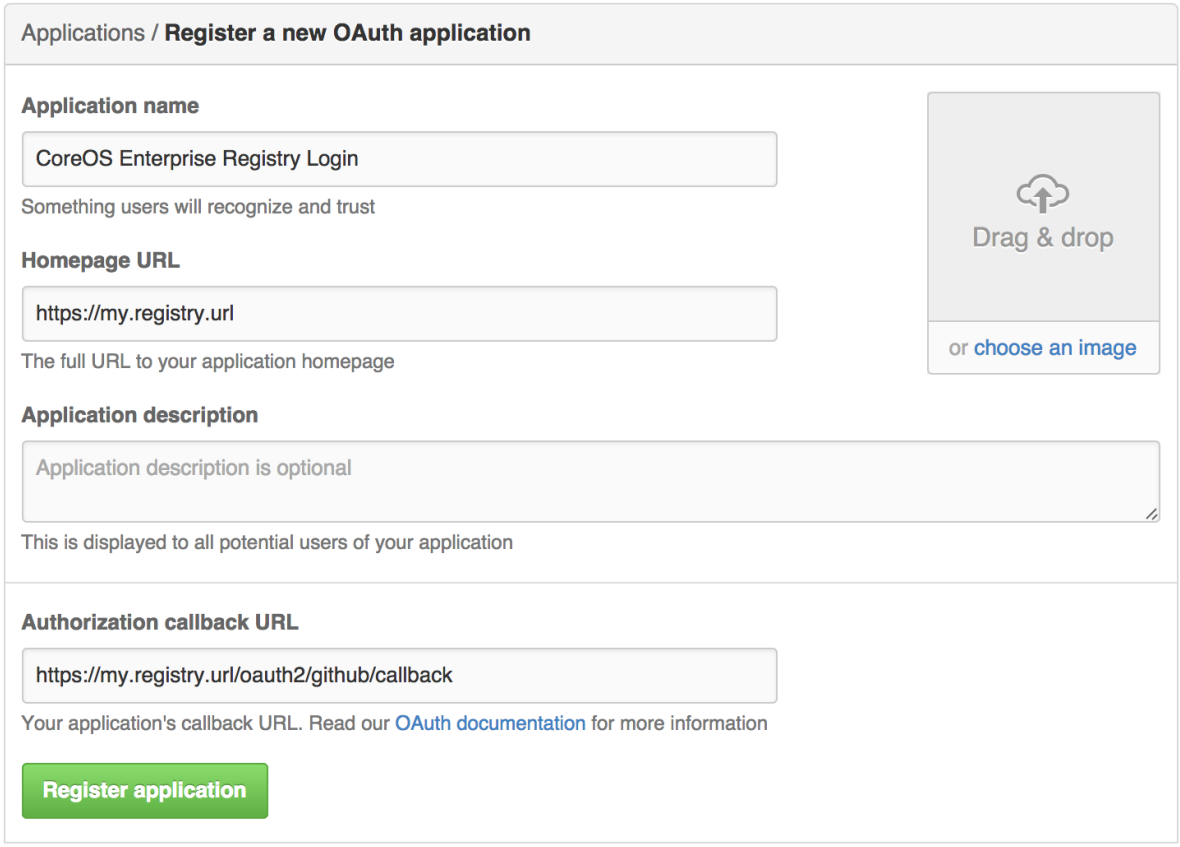
您可以通过将 Red Hat Quay OAuth 应用程序注册为 GitHub 帐户及其存储库来授权您的 Red Hat Quay registry 访问 GitHub 帐户及其存储库。

8.1. 创建新 GITHUB 应用程序

使用以下步骤在 Github 中创建 OAuth 应用程序。

流程

1. 登录 Github Enterprise。
2. 在导航窗格中，选择您的 username → Your organizations。
3. 在导航窗格中，选择 Applications。
4. 点 Register New Application。此时会显示 Register a new OAuth application configuration 屏幕，例如：



5. **在 Application name 文本框中输入应用程序的名称。**

6. **在 Homepage URL 文本框中，输入您的 Red Hat Quay URL。**

注意

如果您使用公共 GitHub，则输入的 Homepage URL 必须可以被您的用户访问。它仍然可以是内部 URL。

7. **在 Authorization 回调 URL 中，输入 https://<RED_HAT_QUAY_URL>/oauth2/github/callback。**

8. **点 Register application 保存您的设置。**

9. **当显示新应用程序的概述时，记录客户端 ID 和为新应用显示的客户端 Secret。**

第 9 章 仓库通知

Red Hat Quay 支持在存储库中为在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

导航到 Red Hat Quay 上的存储库。

1. 在导航窗格中，单击 Settings。
2. 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。
3. 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**

4. 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。

5. 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

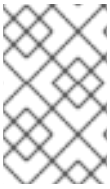
响应可能会因使用可选属性的不同而有所不同。

```
{
  "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 Red Hat Quay 导航窗格中的 bell 图标找到。

Red Hat Quay 通知可以设置为作为一个整体发送给 用户、团队 或组织。

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 Red Hat Quay 设置了 SSL 客户端证书。此证书的验证证明了调用源自 Red Hat Quay。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。

Red Hat Quay 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。Red Hat Quay 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 Red Hat Quay 迭代不同。因为 Red Hat Quay 现在可以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外的介质类型，所以现在它比标准容器镜像格式更广泛，而且是标准的容器镜像格式，也是新的或不合比例的类型。

除了其对 novel 介质类型的扩展支持外，Red Hat Quay 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 Red Hat Quay 的承诺为 Docker 用户提供无缝体验。此外，Red Hat Quay 继续扩展对 Docker V1 拉取的支持，并满足可能仍然依赖这个早期版本的 Docker 镜像的用户。

对 OCI 工件的支持会被默认启用。在以前的版本中，在 FEATURE_GENERAL_OCI_SUPPORT 配置字段下启用了 OCI 介质类型。

注意

因为现在默认启用所有 OCI 介质类型，所以不再需要使用 FEATURE_GENERAL_OCI_SUPPORT、ALLOWED_OCI_ARTIFACT_TYPES 和 IGNORE_UNKNOWN_MEDIATYPES。

另外，FEATURE_HELM_OCI_SUPPORT 配置字段已被弃用。此配置字段不再被支持，并将在以后的 Red Hat Quay 版本中删除。

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。Red Hat Quay 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。

10.1.3. 启用您的系统信任 Red Hat Quay 使用的 SSL/TLS 证书

通过 HTTPS 促进 Helm 客户端和 Red Hat Quay 之间的通信。从 Helm 3.5 开始，支持仅适用于通过 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。您必须确保您的操作系统已配置为信任 Red Hat Quay 使用的证书。使用以下步骤使您的系统信任自定义证书。

流程

1. 输入以下命令将 rootCA.pem 文件复制到 /etc/pki/ca-trust/source/anchors/ 文件夹：

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

2. 输入以下命令更新 CA 信任存储：

```
$ sudo update-ca-trust extract
```

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 Red Hat Quay。

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 Red Hat Quay：

```
$ helm registry login quay370.apps.quayperf370.perfscale.devcluster.openshift.com
```

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://quay370.apps.quayperf370.perfscale.devcluster.openshift.com
```

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://quay370.apps.quayperf370.perfscale.devcluster.openshift.com/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。*
- *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-server.example.com/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-server.example.com
```

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-server.example.com/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-server.example.com
```

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"
  }
 }
```

10.5. 使用其他工件类型

默认情况下，启用其他工件类型供 Red Hat Quay 使用。

使用以下步骤添加额外的 OCI 介质类型。

先决条件

- 您已在 config.yaml 文件中将 FEATURE_GENERAL_OCI_SUPPORT 设置为 true。

流程

1. 在 config.yaml 文件中，添加 ALLOWED_OCI_ARTIFACT_TYPES 配置字段。例如：

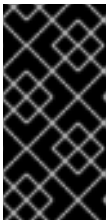
```
FEATURE_GENERAL_OCI_SUPPORT: true
ALLOWED_OCI_ARTIFACT_TYPES:
  <oci config type 1>:
  - <oci layer type 1>
  - <oci layer type 2>

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

2. 通过将以下内容添加到 config.yaml 文件中，添加对所需工件类型的支持，例如 Singularity Image Format (SIF)：

```
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
```



重要

当添加默认情况下不配置的工件类型时，Red Hat Quay 管理员还需要手动添加对 Cosign 和 Helm 的支持。

现在，用户可以为其 Red Hat Quay registry 标记 SIF 镜像。

10.6. 在 RED HAT QUAY 中禁用 OCI 工件

使用以下步骤禁用对 OCI 工件的支持。

流程

- 通过在 config.yaml 文件中将 FEATURE_GENERAL_OCI_SUPPORT 设置为 false 来禁用 OCI 工件支持。例如：

```
FEATURE_GENERAL_OCI_SUPPORT = false
```

第 11 章 RED HAT QUAY 配额管理和强制概述

使用 Red Hat Quay，用户可以通过建立配置的存储配额限制来报告存储消耗并包含 registry 增长。内部 Red Hat Quay 用户现在带有以下功能来管理其环境的容量限制：

- 配额报告：通过此功能，超级用户可以跟踪其所有组织的存储消耗。此外，用户还可以跟踪其所分配组织的存储消耗。
- 配额管理：通过此功能，超级用户可以定义 Red Hat Quay 用户的软和硬性检查。软检查告知用户机构的存储消耗是否达到其配置的阈值。硬检查可防止用户在存储消耗达到配置的限制时推送到 registry。

这些功能一起允许 Red Hat Quay registry 的服务所有者定义服务级别协议并支持健康的资源预算。

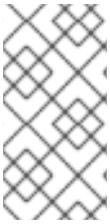
11.1. 配额管理架构

启用配额管理功能后，单独的 blob 大小在存储库和命名空间级别总和。例如，如果同一存储库中的两个标签引用同一 blob，则该 Blob 的大小仅计算为仓库总计一次。另外，清单列表总数计算为存储库总计。

重要

因为清单列表总数被计算为仓库总计，所以从以前的 Red Hat Quay 版本升级时消耗的总配额可能会在 Red Hat Quay 3.9 中被报告不同。在某些情况下，新总数可能会超过存储库的之前设置的限制。Red Hat Quay 管理员可能需要调整存储库所分配的配额，以考虑这些更改。

配额管理功能的工作原理是，使用回填 worker 计算现有存储库和命名空间的大小，然后从总镜像添加或减去，这些镜像被推送或垃圾收集到每个镜像。另外，在收集清单时，从总的减法中减去。



注意

因为减法发生在清单垃圾回收时的总数，所以大小计算会有一个延迟，直到它能够收集垃圾回收为止。有关垃圾回收的更多信息，请参阅 Red Hat Quay 垃圾回收。

以下数据库表包含机构中 Red Hat Quay 仓库的配额存储库大小、配额命名空间大小和配额 registry

大小（以字节为单位）：

- QuotaRepositorySize
- QuotaNameSpaceSize
- QuotaRegistrySize

组织大小由回填 worker 计算，以确保不会重复。初始化镜像推送时，会验证用户的机构存储来检查它是否超出配置的配额限制。如果镜像推送超过定义的配额限制，则会出现软或硬检查：

- 对于软检查，用户会收到通知。
- 对于硬检查，推送将停止。

如果存储消耗在配置的配额限制内，则允许推送进行。

镜像清单删除遵循类似的流程，其中关联的镜像标签和清单之间的链接会被删除。另外，在镜像清单被删除后，会在 QuotaRepositorySize、QuotaNameSpaceSize 和 QuotaRegistrySize 表中重新计算和更新存储库大小。

11.2. 配额管理限制

配额管理有助于组织维护资源消耗。配额管理的一个限制是计算推送上的资源消耗，导致计算成为推送的关键路径的一部分。如果没有这种情况，使用数据可能会偏移。

最大存储配额大小取决于所选数据库：

表 11.1. worker 数量环境变量

变量	Description
Postgres	8388608 TB

变量	Description
MySQL	8388608 TB
SQL Server	16777216 TB

11.3. 配额管理配置字段

表 11.2. 配额管理配置

字段	类型	描述
FEATURE_QUOTA_MANAGEMENT	布尔值	为配额管理功能启用配置、缓存和验证。 **Default:** `False`
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	字符串	为所有机构启用系统默认配额拒绝字节允许。 默认情况下，不会设置限制。
QUOTA_BACKFILL	布尔值	启用配额回填工作程序来计算预先存在的 Blob 的大小。 **默认:True**
QUOTA_TOTAL_DELAY_SECONDS	字符串	启动配额回填的时间延迟。滚动部署可能会导致总部署不正确。此字段 **必须设置为** 超过滚动部署完成所需的时间。 **默认 ：1800**
PERMANENTLY_DELETE_TAGS	布尔值	启用与从时间窗中删除标签相关的功能。 **默认 ：False**
RESET_CHILD_MANIFEST_EXPIRATION	布尔值	重置以子清单为目标的临时标签的过期。将这个功能设置为 **True** 时，子清单会立即收集垃圾回收。 **默认 ：False**

11.3.1. 配额管理配置示例

以下 YAML 是启用配额管理时推荐的配置。

配额管理 YAML 配置

```
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_GARBAGE_COLLECTION: true
PERMANENTLY_DELETE_TAGS: true
QUOTA_TOTAL_DELAY_SECONDS: 1800
RESET_CHILD_MANIFEST_EXPIRATION: true
```

11.4. 使用 RED HAT QUAY API 建立配额

首次创建机构时，它不会应用配额。使用 /api/v1/organization/{organization}/quota 端点：

示例命令

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota  | jq
```

输出示例

```
[]
```

11.4.1. 设置配额

要为机构设置配额，请将数据 POST 到 /api/v1/organization/{orgname}/quota 端点：.Sample

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
'{"limit_bytes": 10485760}'  https://example-registry-quay-quay-
enterprise.apps.docs.quayteam.org/api/v1/organization/testorg/quota | jq
```

输出示例

```
"Created"
```

11.4.2. 查看配额

要查看应用的配额，请来自 /api/v1/organization/{orgname}/quota 端点的 GET 数据：

示例命令

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota  | jq
```

输出示例

```
[
  {
    "id": 1,
    "limit_bytes": 10485760,
    "default_config": false,
    "limits": [],
    "default_config_exists": false
  }
]
```

11.4.3. 修改配额

要修改现有的配额（在这个实例中从 10 MB 改为 100 MB），使用到 /api/v1/organization/{orgname}/quota/{quota_id} 端点的 PUT 数据：

示例命令

```
$ curl -k -X PUT -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
'{"limit_bytes": 104857600}'  https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1 | jq
```

输出示例

```
{
  "id": 1,
  "limit_bytes": 104857600,
  "default_config": false,
  "limits": [],
  "default_config_exists": false
}
```

11.4.4. 推送镜像

要查看消耗的存储，请将各种镜像推送到机构。

11.4.4.1. 推送 iwl:18.04

从命令行将 ubuntu:18.04 推送到机构：

示例命令

```
$ podman pull ubuntu:18.04

$ podman tag docker.io/library/ubuntu:18.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04

$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

11.4.4.2. 使用 API 查看配额使用量

要查看消耗的存储，针对 /api/v1/repository 端点使用 GET 数据：

示例命令

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/repository?
last_modified=true&namespace=testorg&popularity=true&public=true'  | jq
```

输出示例

```
{
  "repositories": [
    {
      "namespace": "testorg",
      "name": "ubuntu",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 27959066,
        "configured_quota": 104857600
      },
      "last_modified": 1651225630,
      "popularity": 0,
      "is_starred": false
    }
  ]
}
```

11.4.4.3. 推送另一个镜像

1. 拉取、标签和推送第二个镜像，如 nginx ：

示例命令

```
$ podman pull nginx

$ podman tag docker.io/library/nginx example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/testorg/nginx

$ podman push --tls-verify=false example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

2. 要查看机构中存储库的配额报告，请使用 /api/v1/repository 端点：

示例命令

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' 'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/repository?last_modified=true&namespace=testorg&popularity=true&public=true'
```

输出示例

```
{
  "repositories": [
    {
      "namespace": "testorg",
      "name": "ubuntu",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 27959066,
        "configured_quota": 104857600
```

```
    },
    "last_modified": 1651225630,
    "popularity": 0,
    "is_starred": false
   },
   {
    "namespace": "testorg",
    "name": "nginx",
    "description": null,
    "is_public": false,
    "kind": "image",
    "state": "NORMAL",
    "quota_report": {
     "quota_bytes": 59231659,
     "configured_quota": 104857600
    },
    "last_modified": 1651229507,
    "popularity": 0,
    "is_starred": false
   }
  ]
}
```

3. 要查看机构详情中的配额信息，请使用 /api/v1/organization/{orgname} 端点：

示例命令

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
'https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg' | jq
```

输出示例

```
{
  "name": "testorg",
  ...
  "quotas": [
    {
      "id": 1,
      "limit_bytes": 104857600,
```

```
    "limits": []
   }
  ],
  "quota_report": {
   "quota_bytes": 87190725,
   "configured_quota": 104857600
  }
 }
```

11.4.5. 使用配额限制拒绝推送

如果镜像推送超过定义的配额限制，则会出现软或硬检查：

- 对于软检查，或警告，用户会收到通知。
- 对于硬检查 或拒绝，推送将被终止。

11.4.5.1. 设置 reject 和 warning 限制

要设置 reject 和 warning 限制，POST 数据到 /api/v1/organization/{orgname}/quota/{quota_id}/limit 端点：

reject limit 命令示例

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
'{"type":"Reject","threshold_percent":80}'  https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1/limit
```

警告限制命令示例

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
'{"type":"Warning","threshold_percent":50}'  https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1/limit
```

11.4.5.2. 查看拒绝和警告限制

要查看 reject 和 warning 限制，请使用 /api/v1/organization/{orgname}/quota 端点：

查看配额限制

```
$  curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq
```

配额限制的输出示例

```
[
  {
    "id": 1,
    "limit_bytes": 104857600,
    "default_config": false,
    "limits": [
      {
        "id": 2,
        "type": "Warning",
        "limit_percent": 50
      },
      {
        "id": 1,
        "type": "Reject",
        "limit_percent": 80
      }
    ],
    "default_config_exists": false
  }
]
```

11.4.5.3. 超过 reject 限制时推送镜像

在本例中，reject 限制(80%)已设置为低于当前存储库大小(~83%)，因此下一个推送应自动被拒绝。

从命令行将示例镜像推送到机构：

镜像推送示例

```
$ podman pull ubuntu:20.04

$ podman tag docker.io/library/ubuntu:20.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04

$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

当超过配额时的输出示例

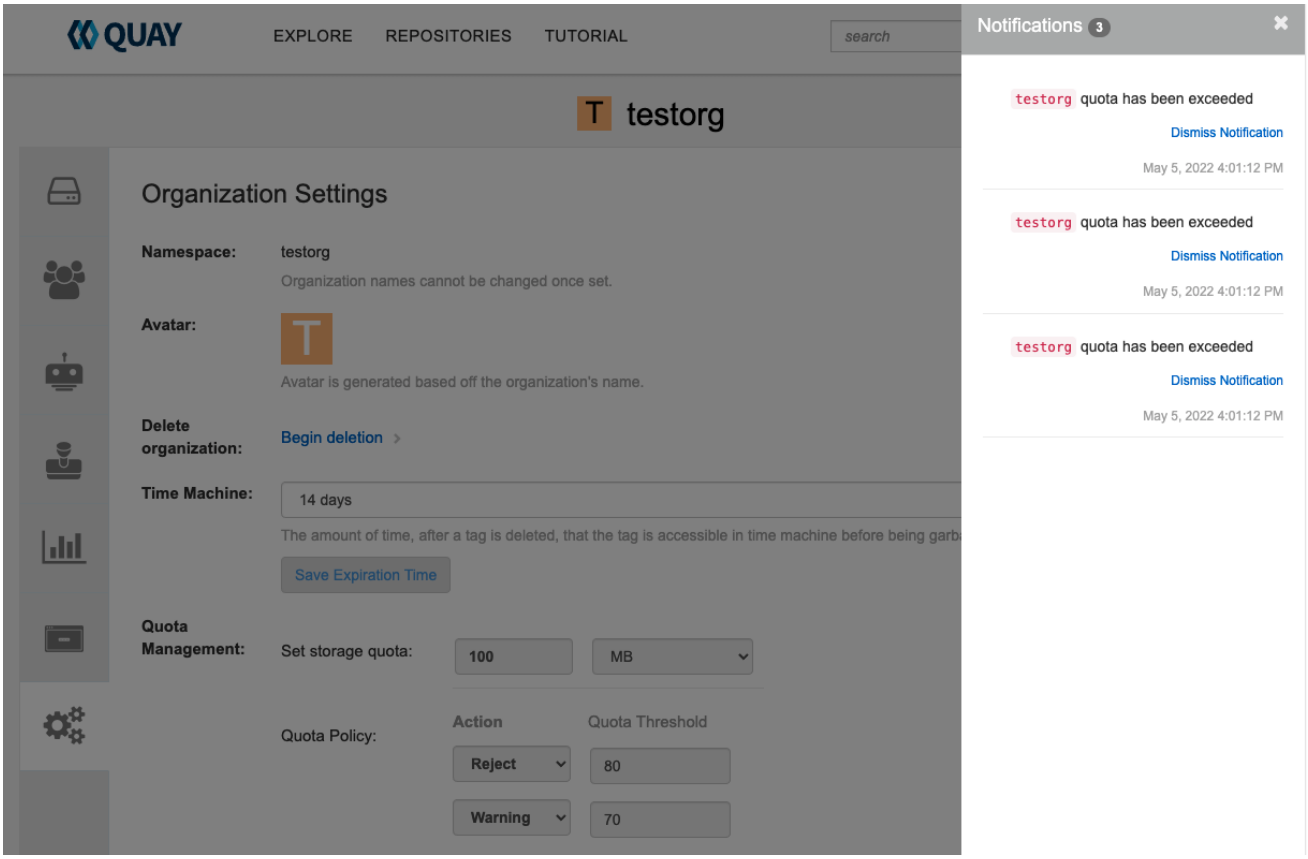
```
Getting image source signatures
Copying blob d4dfaa212623 [--------------------------------------] 8.0b / 3.5KiB
Copying blob cba97cc5811c [--------------------------------------] 8.0b / 15.0KiB
Copying blob 0c78fac124da [--------------------------------------] 8.0b / 71.8MiB
WARN[0002] failed, retrying in 1s ... (1/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [--------------------------------------] 8.0b / 3.5KiB
Copying blob cba97cc5811c [--------------------------------------] 8.0b / 15.0KiB
Copying blob 0c78fac124da [--------------------------------------] 8.0b / 71.8MiB
WARN[0005] failed, retrying in 1s ... (2/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [--------------------------------------] 8.0b / 3.5KiB
Copying blob cba97cc5811c [--------------------------------------] 8.0b / 15.0KiB
Copying blob 0c78fac124da [--------------------------------------] 8.0b / 71.8MiB
WARN[0009] failed, retrying in 1s ... (3/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [--------------------------------------] 8.0b / 3.5KiB
Copying blob cba97cc5811c [--------------------------------------] 8.0b / 15.0KiB
Copying blob 0c78fac124da [--------------------------------------] 8.0b / 71.8MiB
```

> ***Error: Error writing blob: Error initiating layer upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace***

11.4.5.4. 超过限制的通知

超过限制时，会出现通知：

配额通知



第 12 章 RED HAT QUAY 作为上游 REGISTRY 的代理缓存

随着容器开发的流行，客户越来越依赖于上游 registry （如 Docker 或 Google Cloud Platform）中的容器镜像来获取服务。现在，registry 对用户可从这些 registry 中拉取的次数有速率限制和节流。

通过此功能，Red Hat Quay 将作为代理缓存来绕过上游 registry 中的 pull-rate 限制。添加缓存功能也会加快拉取性能，因为镜像是从缓存而不是上游依赖项中拉取的。只有在上游镜像摘要与缓存的镜像不同时，缓存的镜像才会更新，从而减少速率限制和潜在的节流。

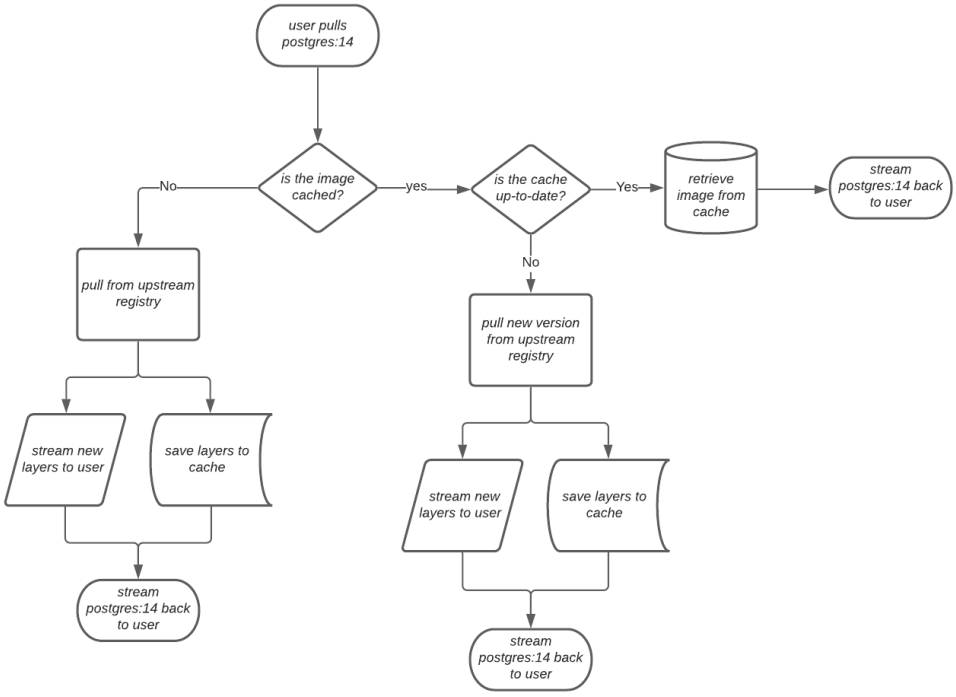
在 Red Hat Quay 缓存代理中，提供以下功能：

- 特定的机构可以定义为上游 registry 的缓存。
- 配置充当特定上游 registry 的缓存的 Quay 组织。此存储库可以使用 Quay UI 定义，并提供以下配置：
  - 私有存储库或增加速率限制的上游 registry 凭证。
  - 过期计时器以避免超过缓存机构大小。
- 通过配置应用程序对/关闭全局配置。
- 整个上游 registry 或只缓存单个命名空间，例如，所有 docker.io 或只包括 docker.io/library。
- 记录所有缓存拉取。
- Clair 的缓存镜像扫描性。

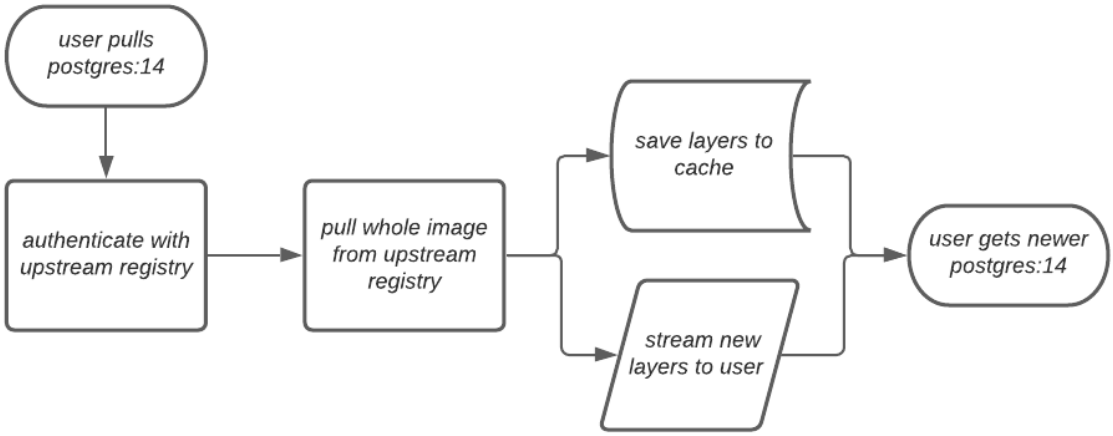
12.1. 代理缓存架构

下图显示了代理缓存功能的预期设计流和架构。

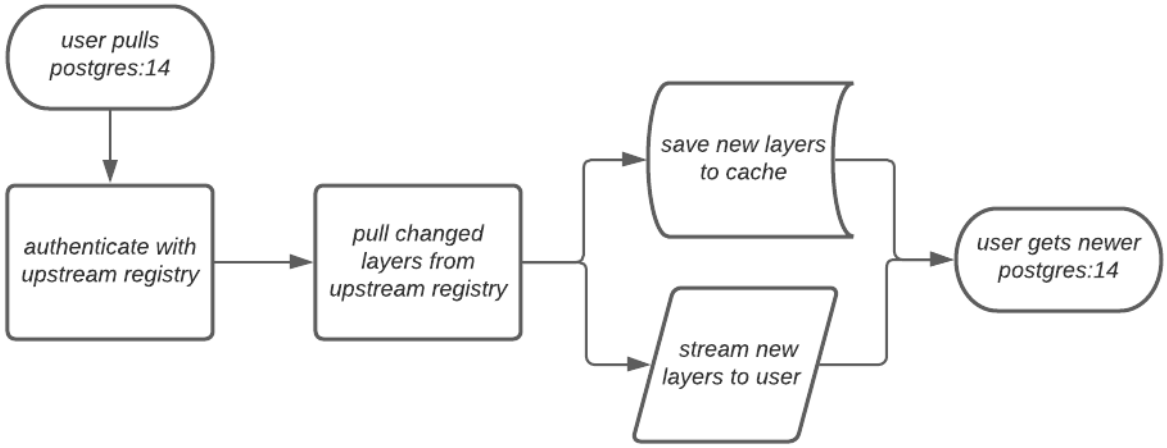
Overview



当用户从 Red Hat Quay 上的上游存储库拉取镜像（例如 postgres:14 ）时，存储库会检查镜像是否存在。如果镜像不存在，则启动一个新的拉取(pull)。拉取后，镜像层会保存到并行用户的缓存和服务器。下图描述了这种情况的架构概述：

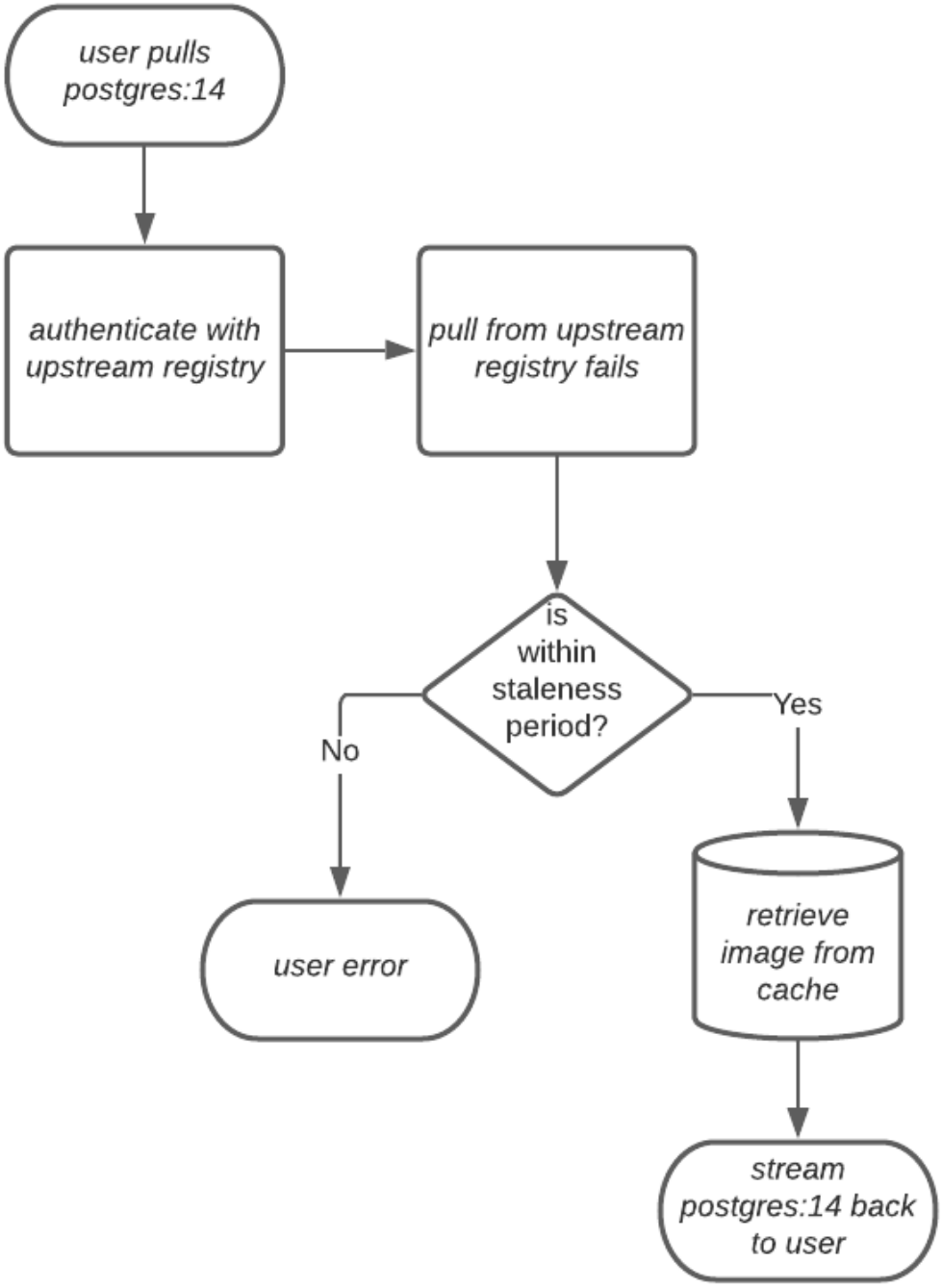


如果缓存中存在镜像，用户可以依赖 Quay 的缓存来与上游源保持最新状态，以便自动拉取缓存中较新的镜像。当上游 registry 中覆盖原始镜像的标签时会出现这种情况。下图描述了当上游镜像和缓存镜像版本不同时发生的情况概述：



如果上游镜像和缓存的版本相同，则不会拉取层，并将缓存的镜像传送到用户。

在某些情况下，用户会在上游 registry 停机时启动拉取。如果在配置的过时的周期内发生这种情况，则会传输存储在缓存中的镜像。如果在配置的过时的周期后拉取发生，则会向用户传播错误。下图描述了在配置过时的时间后进行拉取时的架构概述：



Quay 管理员可以利用组织的可配置大小限制来限制缓存大小，以便后端存储消耗保持可预测状态。这可以通过根据使用镜像的频率从缓存中丢弃镜像来实现。下图描述了这种情况的架构概述：

12.2. 代理缓存限制

Red Hat Quay 的代理缓存有以下限制：

-

您的代理缓存的大小限制必须大于或等于您要缓存的镜像。例如，如果您的代理缓存组织的最大大小为 500 MB，并且用户希望拉取的镜像为 700 MB，则镜像将被缓存，并将溢出超过配置的限制。

- 缓存的镜像必须具有与 Quay 存储库中的镜像相同的属性。

12.3. 使用 RED HAT QUAY 代理远程 REGISTRY

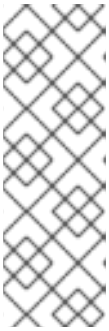
以下流程描述了如何使用 Red Hat Quay 代理远程 registry。此流程设置为代理 quay.io，允许用户使用 podman 从 quay.io 上的任意命名空间拉取任何公共镜像。

前提条件

- config.yaml 中的 FEATURE_PROXY_CACHE 设置为 true。
- 分配 Member 团队角色。有关团队角色的更多信息，请参阅 Red Hat Quay 中的用户和机构。

流程

1. 在 UI 上的 Quay 机构中，例如 cache-quayio，单击左侧窗格中的 Organization Settings。
2. 可选：点 Add Storage Quota 为您的机构配置配额管理。有关配额管理的更多信息，请参阅配额管理。



注意

在某些情况下，在拉取过程中达到配额限制时，使用 Podman 拉取镜像可能会返回以下错误： Error parsing image configuration: Error fetch blob: invalid status code from registry 403 (Forbidden)。错误 403 isaccurate 发生，因为 Podman 在命名空间上隐藏了正确的 API 错误： 配额已超过命名空间。此已知问题将在以后的 Podman 更新中解决。

3. 在 Remote Registry 中，输入要缓存的远程 registry 的名称，如 quay.io，然后单击 Save。

注意

通过将命名空间添加到远程 Registry 中，如 quay.io/<namespace >，您机构中的用户只能从该命名空间代理。

4. 可选：添加远程 Registry Username 和 Remote Registry 密码。

注意

如果没有设置 Remote Registry Username 和 Remote Registry Password，则无法在删除代理缓存和创建新 registry 的情况下添加一个。

5. 可选：在 Expiration 字段中设置一个时间。

注意

- 代理机构中缓存的镜像的默认标签 Expiration 字段被设置为 86400 秒。在代理机构中，每次拉取标签时，标签过期都会刷新到 UI 的 Expiration 字段中设置的值。此功能与 Quay 的默认 单个标签过期 功能不同。在代理机构中，可以覆盖单个标签功能。发生这种情况时，根据代理机构的 Expiration 字段重置各个标签的过期时间。
- 过期的镜像将在分配的时间后消失，但仍然存储在 Quay 中。镜像被完全删除或收集的时间取决于机构的 Time Machine 设置。垃圾回收的默认时间为 14 天，除非另有指定。

6. 点击 Save。

7. 在 CLI 上，从 registry 中拉取公共镜像，如 quay.io，充当代理缓存：

```
$ podman pull <registry_url>/<organization_name>/<quayio_namespace>/<image_name>
```

重要

如果将您的机构设置为从远程 registry 中的单个命名空间拉取，必须在 URL 中省略远程 registry 命名空间。例如，podman pull <registry_url>/<organization_name>/<image_name>。

12.3.1. 在代理机构中利用存储配额限制

在 Red Hat Quay 3.8 中，代理缓存功能已被改进，它有一个自动运行标记的镜像的功能。只有代理命名空间配置了配额限制时，才会使用镜像标签的自动运行。目前，如果镜像大小大于组织的配额，则从上传时跳过镜像，直到管理员创建必要的空间为止。现在，当推送镜像超过分配的空间时，自动运行增强会标记最近用于删除的标签。因此，会存储新的镜像标签，而最小使用的镜像标签被标记为删除。

重要

- 作为 auto-pruning 功能的一部分，标记为删除的标签最终由垃圾收集器(gc) worker 进程收集垃圾回收。因此，在此期间不会完全强制实施配额大小限制。
- 目前，命名空间配额大小计算不会考虑清单子的大小。这是一个已知问题，并将在以后的 Red Hat Quay 版本中解决。

12.3.1.1. 测试代理机构中的存储配额限制功能

使用以下步骤测试启用了代理缓存和存储配额限制的机构自动运行功能。

先决条件

- 您的机构被配置为充当代理机构。以下示例来自 quay.io 的代理。
- 在 config.yaml 文件中，FEATURE_PROXY_CACHE 被设置为 true。
- 在 config.yaml 文件中，FEATURE_QUOTA_MANAGEMENT 设置为 true。
- 您的组织配置了配额限制，例如 150 MB。

流程

1. 从代理机构拉取镜像到存储库，例如：

```
$ podman pull quay-server.example.com/proxytest/projectquay/quay:3.7.9
```

2. 根据存储库中的空间，您可能需要从代理机构拉取其他镜像，例如：

```
$ podman pull quay-server.example.com/proxytest/projectquay/quay:3.6.2
```

3. 在 Red Hat Quay registry UI 中，点存储库的名称。

- 点导航窗格中的标签，并确保 quay:3.7.9 和 quay:3.6.2 已标记。

4. 拉取导致您的存储库超过分配配额的最后镜像，例如：

```
$ podman pull quay-server.example.com/proxytest/projectquay/quay:3.5.1
```

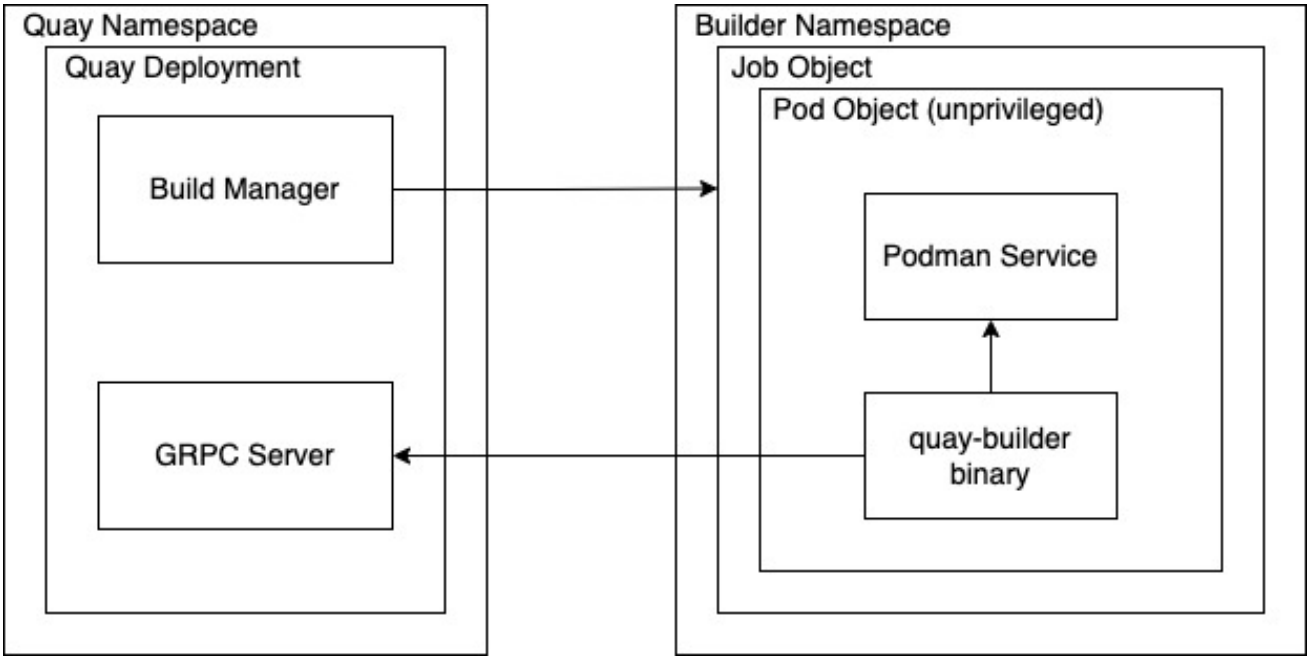
5. 刷新 Red Hat Quay registry 的 Tags 页面。您推送的第一个镜像（例如 quay:3.7.9 ）应该已自动修剪。Tags 页面现在应当显示 quay:3.6.2 和 quay:3.5.1。

第 13 章 RED HAT QUAY 构建增强

Red Hat Quay 构建可以在虚拟平台上运行。对运行之前构建配置的后向兼容性也可用。

13.1. RED HAT QUAY 增强的构建架构

下图显示了增强功能的预期设计流和架构：



在这个版本中，构建管理器首先会创建作业对象。然后，作业对象 使用 quay-builder-image 创建 pod。quay-builder-image 将包含 quay-builder 二进制文件 和 Podman 服务。创建的 pod 作为非特权运行。然后，quay-builder 二进制文件 会在通信状态和从 Build Manager 检索构建信息时构建镜像。

13.2. RED HAT QUAY 构建限制

在非特权上下文中在 Red Hat Quay 中运行构建可能会导致一些在以前的构建策略下工作的命令失败。尝试更改构建策略可能会导致构建出现性能问题和可靠性。

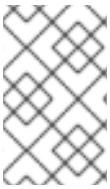
直接在容器中运行构建与使用虚拟机的隔离没有相同的隔离。更改构建环境也可能会导致以前工作的构建失败。

13.3. 使用 OPENSHIFT CONTAINER PLATFORM 创建 RED HAT QUAY 构建器环境

本节中的步骤解释了如何使用 OpenShift Container Platform 创建 Red Hat Quay 虚拟构建器环境。

13.3.1. OpenShift Container Platform TLS 组件

tls 组件允许您控制 TLS 配置。



注意

当 TLS 组件由 Operator 管理时，Red Hat Quay 3.11 不支持构建程序。

如果将 tls 设置为 unmanaged，则提供自己的 ssl.cert 和 ssl.key 文件。在本实例中，如果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，或使用通配符。

要添加 builder 路由，请使用以下格式：

```
[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443
```

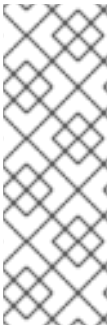
13.3.2. 使用 OpenShift Container Platform for Red Hat Quay builders

构建器需要 SSL/TLS 证书。有关 SSL/TLS 证书的更多信息，请参阅 向 Red Hat Quay 容器添加 TLS 证书。

如果使用 Amazon Web Service (AWS) S3 存储，您必须在运行构建器前修改 AWS 控制台中的存储桶。如需所需参数，请参阅以下部分"修改 AWS S3 存储桶"。

13.3.2.1. 为虚拟构建器准备 OpenShift Container Platform

使用以下步骤为 Red Hat Quay 虚拟构建器准备 OpenShift Container Platform。



注意

- 此流程假设您已置备集群并运行 Quay Operator。
- 此流程是在 OpenShift Container Platform 上设置虚拟命名空间。

流程

1. 使用集群管理员帐户登录到 Red Hat Quay 集群。
2. 运行以下命令，创建一个运行您的虚拟构建器（如 virtual-builders ）的新项目：

```
$ oc new-project virtual-builders
```

3. 输入以下命令在项目中创建一个 ServiceAccount，它将用于运行构建：

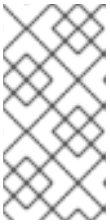
```
$ oc create sa -n virtual-builders quay-builder
```

4. 为创建的服务帐户提供编辑权限，以便它可以运行构建：

```
$ oc adm policy -n virtual-builders add-role-to-user edit
system:serviceaccount:virtual-builders:quay-builder
```

5. 输入以下命令授予 Quay builder anyuid scc 权限：

```
$ oc adm policy -n virtual-builders add-scc-to-user anyuid -z quay-builder
```



注意

此操作需要集群管理员特权。这是必要的，因为构建器必须以 Podman 用户身份运行，才能使非特权或无根构建正常工作。

6. 获取 Quay builder 服务帐户的令牌。

a. *如果使用 OpenShift Container Platform 4.10 或更早的版本，请输入以下命令：*

```
oc sa get-token -n virtual-builders quay-builder
```

b. *如果使用 OpenShift Container Platform 4.11 或更高版本，请输入以下命令：*

```
$ oc create token quay-builder -n virtual-builders
```

注意

当令牌过期时，您需要请求新令牌。另外，您还可以添加自定义过期。例如，指定 --duration 20160m 以保留令牌两周。

输出示例

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ItTHZ0dGZMYjhlWnYxZTQzN2dJVEJxcDJscldSdEUtYWsifQ...
```

7. *输入以下命令确定构建程序路由：*

```
$ oc get route -n quay-enterprise
```

输出示例

```
NAME                          HOST/PORT                                                  PATH   SERVICES                      PORT   TERMINATION     WILDCARD
...
example-registry-quay-builder   example-registry-quay-builder-quay-enterprise.apps.docs.quayteam.org   example-registry-quay-app   grpc   edge/Redirect   None
...
```

8. 输入以下命令使用 .crt 扩展生成自签名 SSL/TIS 证书：

```
$ oc extract cm/kube-root-ca.crt -n openshift-apiserver
```

输出示例

```
ca.crt
```

9. 输入以下命令将 ca.crt 文件重命名为 extra_ca_cert_build_cluster.crt ：

```
$ mv ca.crt extra_ca_cert_build_cluster.crt
```

10. 在控制台中找到配置捆绑包的 secret，然后选择 Actions → Edit Secret 并添加适当的构建程序配置：

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- <superusername>
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: <sample_build_route> 1
BUILD_MANAGER:
 - ephemeral
 - ALLOWED_WORKER_COUNT: 1
   ORCHESTRATOR_PREFIX: buildman/production/
   JOB_REGISTRATION_TIMEOUT: 3600 2
   ORCHESTRATOR:
     REDIS_HOST: <sample_redis_hostname> 3
     REDIS_PASSWORD: ""
     REDIS_SSL: false
     REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
   EXECUTORS:
     - EXECUTOR: kubernetesPodman
       NAME: openshift
       BUILDER_NAMESPACE: <sample_builder_namespace> 4
       SETUP_TIME: 180
       MINIMUM_RETRY_THRESHOLD: 0
       BUILDER_CONTAINER_IMAGE: <sample_builder_container_image> 5
       # Kubernetes resource options
```

```
K8S_API_SERVER: <sample_k8s_api_server> 6
K8S_API_TLS_CA: <sample_crt_file> 7
VOLUME_SIZE: 8G
KUBERNETES_DISTRIBUTION: openshift
CONTAINER_MEMORY_LIMITS: 300m 8
CONTAINER_CPU_LIMITS: 1G 9
CONTAINER_MEMORY_REQUEST: 300m 10
CONTAINER_CPU_REQUEST: 1G 11
NODE_SELECTOR_LABEL_KEY: ""
NODE_SELECTOR_LABEL_VALUE: ""
SERVICE_ACCOUNT_NAME: <sample_service_account_name>
SERVICE_ACCOUNT_TOKEN: <sample_account_token> 12
```

1

构建路由通过运行 oc get route -n 及 OpenShift Operator 的命名空间的名称来获取。路由末尾必须提供一个端口，它应使用以下格式：[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443。

2

如果设置了 JOB_REGISTRATION_TIMEOUT 参数，您可能会收到以下错误：failed to register job to build manager: rpc error: code = Unauthenticated desc = Invalid build token: Signature has expired.建议将此参数设置为至少 240。

3

如果您的 Redis 主机有密码或 SSL/TLS 证书，您必须相应地更新。

4

设置，以匹配虚拟构建器命名空间的名称，如 virtual-builders。

5

对于早期访问，BUILDER_CONTAINER_IMAGE 目前是 quay.io/projectquay/quay-builder:3.7.0-rc.2。请注意，这可能会在早期访问窗口中更改。如果出现这种情况，则会警告客户。

6

K8S_API_SERVER 通过运行 oc cluster-info 获取。

7

您必须手动创建并添加自定义 CA 证书，例如 K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt。

8

如果未指定，则默认为 5120Mi。

9

对于虚拟构建，您必须确保集群中有足够的资源。如果未指定，则默认为 1000m。

10

如果未指定，则默认为 3968Mi。

11

如果未指定，则默认为 500m。

12

在运行 oc create sa 时获得。

配置示例

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org:443
BUILD_MANAGER:
 - ephemeral
 - ALLOWED_WORKER_COUNT: 1
   ORCHESTRATOR_PREFIX: buildman/production/
   JOB_REGISTRATION_TIMEOUT: 3600
   ORCHESTRATOR:
     REDIS_HOST: example-registry-quay-redis
     REDIS_PASSWORD: ""
     REDIS_SSL: false
     REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
   EXECUTORS:
     - EXECUTOR: kubernetesPodman
       NAME: openshift
       BUILDER_NAMESPACE: virtual-builders
       SETUP_TIME: 180
       MINIMUM_RETRY_THRESHOLD: 0
       BUILDER_CONTAINER_IMAGE: quay.io/projectquay/quay-builder:3.7.0-rc.2
```

```
        # Kubernetes resource options
        K8S_API_SERVER: api.docs.quayteam.org:6443
        K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt
        VOLUME_SIZE: 8G
        KUBERNETES_DISTRIBUTION: openshift
        CONTAINER_MEMORY_LIMITS: 1G
        CONTAINER_CPU_LIMITS: 1080m
        CONTAINER_MEMORY_REQUEST: 1G
        CONTAINER_CPU_REQUEST: 580m
        NODE_SELECTOR_LABEL_KEY: ""
        NODE_SELECTOR_LABEL_VALUE: ""
        SERVICE_ACCOUNT_NAME: quay-builder
        SERVICE_ACCOUNT_TOKEN:
"eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ItTHZ0dGZMYjhIWnYxZTQzN2dJVEJ
xcDJscldSdEUtYWsifQ"
```

13.3.2.2. 手动添加 SSL/TLS 证书

由于配置工具的已知问题，您必须手动将自定义 SSL/TLS 证书添加到正确运行。使用以下步骤手动添加自定义 SSL/TLS 证书。

有关创建 SSL/TLS 证书的更多信息，请参阅在 Red Hat Quay 容器中添加 TLS 证书。

13.3.2.2.1. 创建和签名证书

使用以下步骤创建并签署 SSL/TLS 证书。

流程

- 创建证书颁发机构并签署证书。如需更多信息，请参阅创建证书颁发机构并签署证书。

openssl.cnf

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

```
subjectAltName = @alt_names
[alt_names]
DNS.1 = example-registry-quay-quay-enterprise.apps.docs.quayteam.org (1)
DNS.2 = example-registry-quay-builder-quay-enterprise.apps.docs.quayteam.org (2)
```

(1) 必须包含 Red Hat Quay registry URL 的 alt_name。

(2) BUILDMAN_HOSTNAME的 alt_name

示例命令

```
$ openssl genrsa -out rootCA.key 2048
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
$ openssl genrsa -out ssl.key 2048
$ openssl req -new -key ssl.key -out ssl.csr
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -
out ssl.cert -days 356 -extensions v3_req -extfile openssl.cnf
```

13.3.2.2.2. 将 TLS 设置为非受管

使用以下步骤将 king:tls 设置为 unmanaged。

流程

1. 在 Red Hat Quay Registry YAML 中，将 kind: tls 设置为 managed: false ：

```
- kind: tls
  managed: false
```

2. 在 Events 页面中，更改会被阻断，直到您设置了适当的 config.yaml 文件。例如：

```
- lastTransitionTime: '2022-03-28T12:56:49Z'
  lastUpdateTime: '2022-03-28T12:56:49Z'
  message: >-
    required component `tls` marked as unmanaged, but `configBundleSecret`
    is missing necessary fields
  reason: ConfigInvalid
  status: 'True'
```

13.3.2.2.3. 创建临时 secret

使用以下步骤为 CA 证书创建临时 secret。

流程

1. 在默认命名空间中创建 CA 证书的 secret：

```
$ oc create secret generic -n quay-enterprise temp-crt --from-file
extra_ca_cert_build_cluster.crt
```

2. 在 default 命名空间中为 ssl.key 和 ssl.cert 文件创建一个 secret：

```
$ oc create secret generic -n quay-enterprise quay-config-ssl --from-file ssl.cert --from-file
ssl.key
```

13.3.2.2.4. 将 secret 数据复制到配置 YAML 中

使用以下步骤将 secret 数据复制到 config.yaml 文件中。

流程

1. 在控制台 UI 中找到 Workloads → Secrets 的新 secret。

2. 对于每个 secret，找到 YAML 视图：

```
kind: Secret
apiVersion: v1
metadata:
  name: temp-crt
  namespace: quay-enterprise
  uid: a4818adb-8e21-443a-a8db-f334ace9f6d0
  resourceVersion: '9087855'
  creationTimestamp: '2022-03-28T13:05:30Z'
```

```
...
data:
  extra_ca_cert_build_cluster.crt: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0l....
type: Opaque
```

```
kind: Secret
apiVersion: v1
metadata:
  name: quay-config-ssl
  namespace: quay-enterprise
  uid: 4f5ae352-17d8-4e2d-89a2-143a3280783c
  resourceVersion: '9090567'
  creationTimestamp: '2022-03-28T13:10:34Z'
...
data:
  ssl.cert: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-

LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

3. 在 UI 中查找 Red Hat Quay registry 配置捆绑包的 secret，或通过运行以下命令来使用命令行：

```
$ oc get quayregistries.quay.redhat.com -o jsonpath="
{.items[0].spec.configBundleSecret}{'\n'}"  -n quay-enterprise
```

4. 在 OpenShift Container Platform 控制台中，选择配置捆绑包 secret 的 YAML 选项卡，并从您创建的两个 secret 中添加数据：

```
kind: Secret
apiVersion: v1
metadata:
  name: init-config-bundle-secret
  namespace: quay-enterprise
  uid: 4724aca5-bff0-406a-9162-ccb1972a27c1
  resourceVersion: '4383160'
  creationTimestamp: '2022-03-22T12:35:59Z'
...
data:
  config.yaml: >-
    RkVBVFVSRV9VU0VSX0lOSVRJQUxJWkU6lHRydWUKQlJ...
  extra_ca_cert_build_cluster.crt: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0ldw..
..
  ssl.cert: >-
```

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-

LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

5. 点击 **Save**。

6. 输入以下命令查看您的 pod 是否重启：

```
$ oc get pods -n quay-enterprise
```

输出示例

```
NAME                                             READY   STATUS              RESTARTS   AGE
...
example-registry-quay-app-6786987b99-vgg2v       0/1     ContainerCreating   0          2s
example-registry-quay-app-7975d4889f-q7tvl       1/1     Running             0          5d21h
example-registry-quay-app-7975d4889f-zn8bb       1/1     Running             0          5d21h
example-registry-quay-app-upgrade-lswsn          0/1     Completed           0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv   0/1     ContainerCreating   0          2s
example-registry-quay-config-editor-c6c4d9ccd-2mwg2    1/1     Running             0          5d21h
example-registry-quay-database-66969cd859-n2ssm  1/1     Running             0          6d1h
example-registry-quay-mirror-764d7b68d9-jmlkk    1/1     Terminating         0          5d21h
example-registry-quay-mirror-764d7b68d9-jqzwg    1/1     Terminating         0          5d21h
example-registry-quay-redis-7cc5f6c977-956g8     1/1     Running             0          5d21h
```

7. 重新配置 Red Hat Quay registry 后，输入以下命令检查 Red Hat Quay 应用程序 pod 是否正在运行：

```
$ oc get pods -n quay-enterprise
```

输出示例

```
example-registry-quay-app-6786987b99-sz6kb            1/1     Running            0
7m45s
example-registry-quay-app-6786987b99-vgg2v            1/1     Running            0
9m1s
example-registry-quay-app-upgrade-lswsn               0/1     Completed          0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv  1/1     Running            0
9m1s
example-registry-quay-database-66969cd859-n2ssm       1/1     Running            0
6d1h
example-registry-quay-mirror-758fc68ff7-5wxlp         1/1     Running            0
8m29s
example-registry-quay-mirror-758fc68ff7-lbl82         1/1     Running            0          8m29s
example-registry-quay-redis-7cc5f6c977-956g8          1/1     Running            0
5d21h
```

8. 在您的浏览器中，访问 registry 端点并验证证书是否已适当更新。例如：

```
Common Name (CN) example-registry-quay-quay-enterprise.apps.docs.quayteam.org
Organisation (O) DOCS
Organisational Unit (OU) QUAY
```

13.3.2.3. 使用 UI 创建构建触发器

使用以下步骤使用 UI 创建构建触发器。

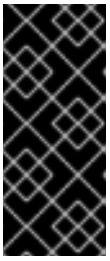
流程

1. 登录到您的 Red Hat Quay 存储库。

2. 单击 Create New Repository，再创建一个新 registry，如 testrepo。

3. 在 Repositories 页面上，单击导航窗格上的 Builds 选项卡。或者，直接使用对应的 URL：

```
https://example-registry-quay-quay-
enterprise.apps.docs.quayteam.org/repository/quayadmin/testrepo?tab=builds
```



重要

在某些情况下，构建器可能会遇到解析主机名的问题。这个问题可能与作业对象上的 dnsPolicy 设置为 default 相关。目前，这个问题还没有临时解决方案。它将在以后的 Red Hat Quay 版本中解决。

4. 点 Create Build Trigger → Custom Git Repository Push。

5. 输入用于克隆 Git 存储库的 HTTPS 或 SSH 样式 URL，然后单击 Continue。例如：

```
https://github.com/gabriel-rh/actions_test.git
```

6. 使用分支或标签名称检查 Tag 清单，然后单击 Continue。

7. 在调用触发器时输入要构建的 Dockerfile 的位置，如 /Dockerfile，然后单击 Continue。

8. 输入 Docker 构建的上下文的位置，如 /，然后单击 Continue。

9. 如果保证，请创建一个 Robot 帐户。否则，点 Continue。

10. 点 Continue 来验证参数。

11. 在 Builds 页面中，点 Trigger Name 的 Options 图标，然后点 Run Trigger Now。

12. 从 Git 存储库输入提交 SHA，然后单击 Start Build。

13. 您可以通过单击 Build History 页面中的提交或运行 oc get pods -n virtual-builders 来检查构建的状态。例如：

```
$ oc get pods -n virtual-builders
```

输出示例

```
NAME                                     READY   STATUS    RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Running   0          7s
```

```
$ oc get pods -n virtual-builders
```

输出示例

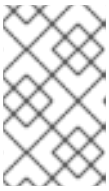
```
NAME                                     READY   STATUS        RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Terminating   0          9s
```

```
$ oc get pods -n virtual-builders
```

输出示例

```
No resources found in virtual-builders namespace.
```

14. 构建完成后，您可以检查导航窗格上 **Tags** 下的标签状态。



注意

通过早期访问，目前构建的完整构建日志和时间戳当前不可用。

13.3.2.4. 修改 AWS S3 存储桶

如果使用 **AWS S3** 存储，则必须在运行构建器前在 **AWS** 控制台中更改存储桶。

流程

1. *在 s3.console.aws.com 上登录到 AWS 控制台。*

2. *在搜索栏中，搜索 S3，然后单击 S3。*

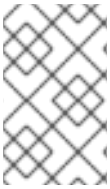
3. *点存储桶的名称，如 myawsbucket。*

4. *单击权限选项卡。*

5. *在 Cross-origin 资源共享(CORS) 下，包含以下参数：*

```
[
    {
        "AllowedHeaders": [
            "Authorization"
        ],
        "AllowedMethods": [
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    },
    {
        "AllowedHeaders": [
            "Content-Type",
            "x-amz-acl",
            "origin"
        ],
        "AllowedMethods": [
            "PUT"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    }
]
```

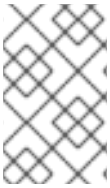
13.3.2.5. 修改 Google Cloud Platform 对象存储桶



注意

目前，IBM Power 和 IBM Z 不支持修改 Google Cloud Platform 对象存储桶。

使用以下步骤为虚拟构建器配置跨原始资源共享(CORS)。



注意

如果没有 CORS 配置，上传构建 Dockerfile 会失败。

流程

1. 使用以下引用来创建 JSON 文件，以满足您的特定 CORS 需求。例如：

```
$ cat gcp_cors.json
```

输出示例

```
[
  {
    "origin": ["*"],
    "method": ["GET"],
    "responseHeader": ["Authorization"],
    "maxAgeSeconds": 3600
  },
  {
    "origin": ["*"],
    "method": ["PUT"],
    "responseHeader": [
          "Content-Type",
          "x-goog-acl",
          "origin"],
    "maxAgeSeconds": 3600
  }
]
```

2. 输入以下命令更新 GCP 存储桶：

```
$ gcloud storage buckets update gs://<bucket_name> --cors-file=./gcp_cors.json
```

输出示例

```
Updating
  Completed 1
```

3. 您可以运行以下命令来显示 GCP 存储桶的更新 CORS 配置：

```
$ gcloud storage buckets describe gs://<bucket_name>  --format="default(cors)"
```

输出示例

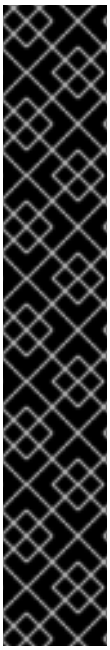
```
cors:
- maxAgeSeconds: 3600
  method:
  - GET
  origin:
  - '*'
  responseHeader:
  - Authorization
- maxAgeSeconds: 3600
  method:
  - PUT
  origin:
  - '*'
  responseHeader:
  - Content-Type
  - x-goog-acl
  - origin
```

第 14 章 使用 V2 UI

使用以下步骤配置和使用 Red Hat Quay v2 UI。

14.1. V2 用户界面配置

启用 FEATURE_UI_V2 后，您可以在用户界面的当前版本和用户界面的新版本间切换。



重要

- 这个 UI 目前处于 beta 阶段，可能会有变化。在其当前状态中，用户只能创建、查看和删除机构、存储库和镜像标签。
- 使用旧 UI 时，超时会话将要求用户在弹出窗口中再次输入密码。使用新的 UI 时，用户将返回到主页面，并需要输入其用户名和密码凭证。这是一个已知问题，并将在以后的新 UI 版本中解决。
- 在传统 UI 和新 UI 之间如何报告镜像清单大小时，有一个差异。在传统 UI 中，镜像清单以兆字节为单位报告。v2 UI 使用标准定义兆字节(MB)来报告镜像清单大小。

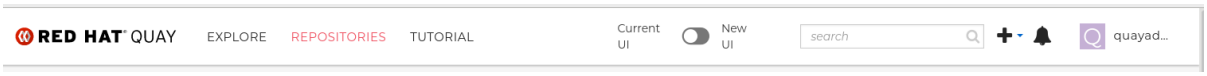
流程

1. 在部署的 config.yaml 文件中，添加 FEATURE_UI_V2 参数并将其设置为 true，例如：

```
---
FEATURE_TEAM_SYNCING: false
FEATURE_UI_V2: true
FEATURE_USER_CREATION: true
---
```

2. 登录到您的 Red Hat Quay 部署。

3. 在部署的导航窗格中，您可以选择在 Current UI 和 New UI 之间切换。点击切换按钮将其设置为新 UI，然后点 Use Beta Environment，例如：



14.1.1. 使用 v2 UI 创建新机构

先决条件

- 您已将部署切换为使用 v2 UI。

使用以下步骤使用 v2 UI 创建机构。

流程

1. 单击导航窗格中的 Organization。
2. 单击 Create Organization。
3. 输入 组织名称，例如 testorg。
4. 点 Create。

现在，您的示例组织应在 Organizations 页面下填充。

14.1.2. 使用 v2 UI 删除机构

使用以下步骤使用 v2 UI 删除机构。

流程

1. 在 Organizations 页面上，选择您要删除的机构的名称，如 testorg。
2. 点 More Actions 下拉菜单。

3. 点 Delete。

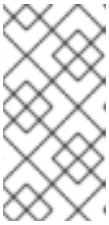
注意

在 Delete 页面中，有一个 搜索 输入框。有了此框，用户可以搜索特定的组织，以确保它们被正确地安排删除。例如，如果用户正在删除 10 个机构，并希望确保删除了一个特定的机构，他们可以使用 搜索 输入框确认机构标记为删除。

4. 通过在框中输入 confirm 确认您要永久删除组织。

5. 点 Delete。

删除后，您将返回到 Organizations 页面。



注意

您可以通过选择多个机构，然后点 More Actions → Delete 一次删除多个机构。

14.1.3. 使用 v2 UI 创建新存储库

使用以下步骤使用 v2 UI 创建存储库。

流程

1. 单击导航窗格上的 Repositories。

2. 单击 Create Repository。

3. 选择一个命名空间，如 quayadmin，然后输入 Repository name，如 testrepo。

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。

4. 点 Create。

现在，您的示例存储库应在 Repositories 页面下填充。

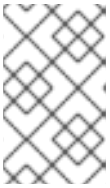
14.1.4. 使用 v2 UI 删除存储库

先决条件

- 您已创建了一个存储库。

流程

1. 在 v2 UI 的 Repositories 页面中，单击您要删除的镜像的名称，如 quay/admin/busybox。
2. 点 More Actions 下拉菜单。
3. 点 Delete。



注意

如果需要，您可以点击 Make Public 或 Make Private。

4. 在框中键入 confirm，然后单击 Delete。
5. 删除后，您将返回到 Repositories 页面。

14.1.5. 将镜像推送到 v2 UI

使用以下步骤将镜像推送到 v2 UI。

流程

1. 从外部 registry 拉取示例镜像：

```
$ podman pull busybox
```

2. 标记镜像：

```
$ podman tag docker.io/library/busybox quay-
server.example.com/quayadmin/busybox:test
```

3. 将镜像推送到 registry：

```
$ podman push quay-server.example.com/quayadmin/busybox:test
```

4. 导航到 v2 UI 上的 Repositories 页面，并确保您的镜像已被正确推送。

5. 您可以选择镜像标签来检查安全详情，然后导航到 Security Report 页面。

14.1.6. 使用 v2 UI 删除镜像

使用以下步骤通过 v2 UI 删除镜像。

先决条件

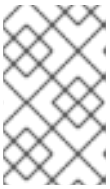
- 您已将镜像推送到 registry。

流程

1. 在 v2 UI 的 Repositories 页面中，单击您要删除的镜像的名称，如 quay/admin/busybox。

2. 点 More Actions 下拉菜单。

3. 点 Delete。



注意

如果需要，您可以点击 Make Public 或 Make Private。

4. 在框中键入 confirm，然后单击 Delete。

5. 删除后，您将返回到 Repositories 页面。

14.1.7. 使用 Red Hat Quay v2 UI 创建新团队

使用以下步骤使用 Red Hat Quay v2 UI 创建新团队。

先决条件

- 您已创建了一个带有存储库的组织。

流程

1. 在 Red Hat Quay v2 UI 上，点机构的名称。

2. 在您的机构页面中，点 Teams 和 membership。

3. 点 Create new team 复选框。

4. 在 Create team 弹出窗口中，为您的新团队提供一个名称。

5. 可选。为您的新团队提供描述。

6. 单击 **Proceed**。此时会出现一个新的弹出窗口。

7. 可选。将此团队添加到存储库，并将权限设置为 **Read**、**Write**、**Admin** 或 **None** 之一。

8. 可选。添加团队成员或机器人帐户。要添加团队成员，请输入其 **Red Hat Quay** 帐户的名称。

9. 检查并填写信息，然后点 **Review and Finish**。新团队会出现在 **Teams** 和 **membership** 页面 下。在这里，您可以点 **kebab** 菜单并选择以下选项之一：

   - 管理团队成员.在此页面上，您可以查看所有成员、团队成员、机器人帐户或被邀请的用户。您还可以通过单击 **Add new member** 来添加新团队成员。
   - 设置存储库权限。在本页中，您可以将存储库权限设置为 **Read**、**Write**、**Admin** 或 **None** 之一。
   - 删除。此弹出窗口允许您通过单击 **Delete** 来删除团队。

10. 可选。您可以点击以下选项之一来显示有关团队、成员和协作者的更多信息：

    - 团队视图.此菜单显示所有团队名称、成员数量、存储库数量以及每个团队的角色。
    - 成员视图.此菜单显示团队成员的所有用户名，以及他们所属的团队，以及用户的存储库权限。
    - collaborators View.此菜单显示存储库协作器。collaborator 是不属于机构中的任何团队的用户，但他们对属于该组织的一个或多个存储库具有直接权限。

14.1.8. 使用 v2 UI 创建机器人帐户

使用以下步骤，使用 v2 UI 创建机器人帐户。

流程

1. 在 v2 UI 上，单击 **Organizations**。

2. 单击您要为其创建机器人帐户的组织名称，如 **test-org**。

3. 点 **Robot accounts** 选项卡 → **Create robot account**。

4. 在 **Provide a name for your robot account** 框中，输入名称，如 **robot1**。

5. 可选。如果需要，可以使用以下选项：

   a. 将机器人添加到团队。

   b. 将机器人添加到存储库。

   c. 调整机器人的权限。

6. 在 **Review and finish** 页面中，查看您提供的信息，然后点 **Review and finish**。此时会出现以下警报： **Successfully created robot account with robot name: <organization_name> + <robot_name>.**

   或者，如果您试图创建与另一个机器人帐户相同的机器人帐户，您可能会收到以下错误消息： **Error create robot account**。

7. 可选。您可以单击 **Expand** 或 **Collapse** 以显示有关机器人帐户的描述性信息。

8. 可选。您可以点 **kebab** 菜单 → **Set repository** 权限来更改机器人帐户的权限。显示以下消息： **Successfully updated repository** 权限。

9. 可选。要删除您的机器人帐户，请选中机器人帐户的复选框，然后单击回收站图标。此时会出现弹出窗口。在文本框中键入 **confirm**，然后单击 **Delete**。或者，您可以点 **kebab** 菜单 →

Delete。显示以下消息： Successfully deleted robot account.

14.1.8.1. 使用 Red Hat Quay v2 UI 批量管理机器人帐户存储库访问权限

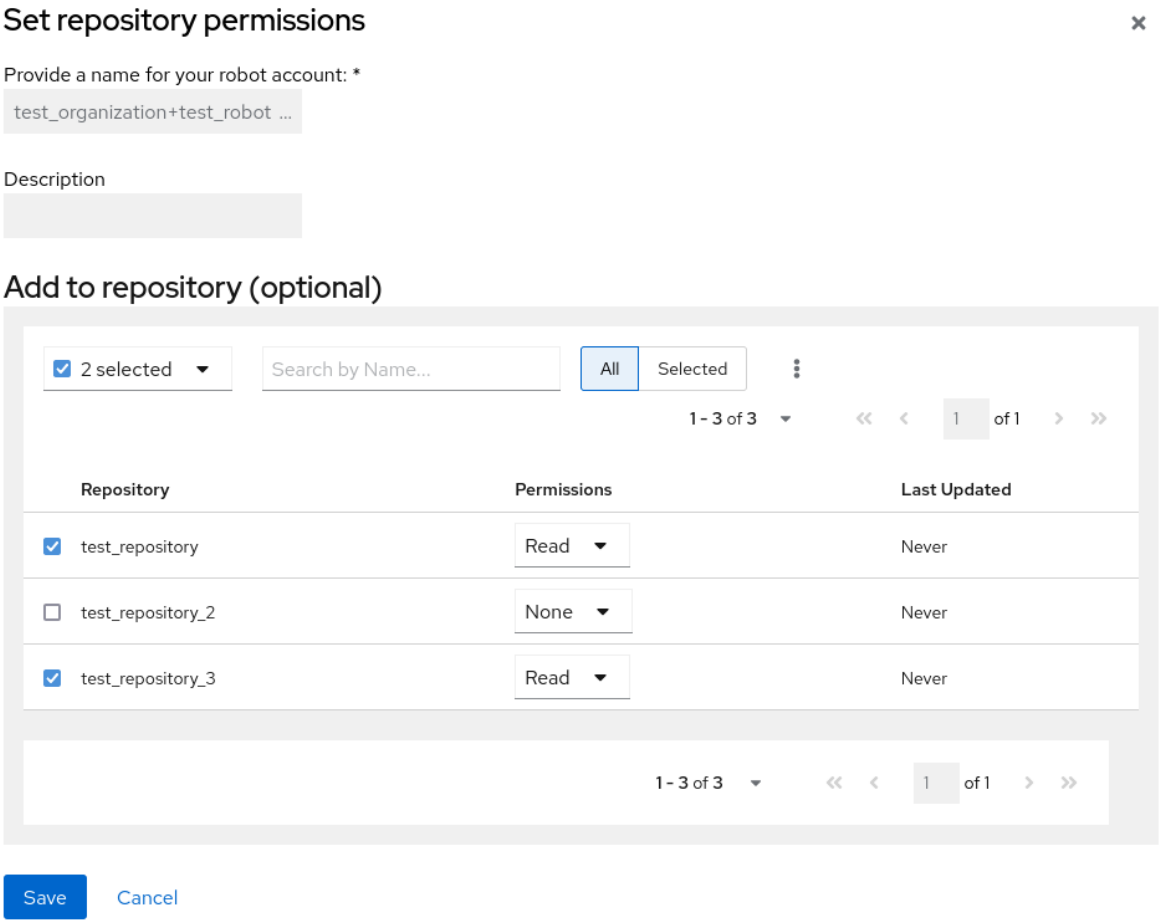
使用以下步骤，使用 Red Hat Quay v2 UI 在批量、机器人帐户存储库访问中。

先决条件

- 您已创建了机器人帐户。
- 您已在单个机构中创建多个软件仓库。

流程

1. 在 Red Hat Quay v2 UI 登录页面中，单击导航窗格中的 Organizations。
2. 在 Organizations 页面上，选择具有多个存储库的组织的名称。单个机构下的存储库数量可在 Repo Count 列下找到。
3. 在您的组织页面中，单击 Robot 帐户。
4. 对于将添加到多个存储库的机器人帐户，点 kebab 图标 → Set repository 权限。
5. 在 Set repository permissions 页面上，选中机器人帐户要添加到的存储库的框。例如：



6. *设置机器人帐户的权限，如 None、Read、Write、Admin。*

7. *单击保存。显示 Success alert: Successfully updated repository 权限 的警报会出现在 Set repository permissions 页面中，确认更改。*

8. *返回到 Organizations → Robot accounts 页面。现在，您的机器人帐户的 Repositories 列显示机器人帐户已添加到的存储库数量。*

14.1.9. 使用 Red Hat Quay v2 UI 创建默认权限

默认权限定义在创建存储库时应自动授予的权限，除了存储库的默认创建者之外。根据创建存储库的用户分配权限。

使用以下步骤使用 Red Hat Quay v2 UI 创建默认权限。

流程

1. 点机构的名称。

2. 单击 **Default permissions**。

3. 单击 创建默认权限。此时会出现切换 **drawer**。

4. 选择 **Anyone** 或 **Specific** 用户，在创建存储库时创建默认权限。

   a. 如果选择 **Anyone**，则必须提供以下信息：

   - 应用到。搜索、邀请或添加用户/机器/团队。
   - 权限.将权限设置为 **Read,Write,** 或 **Admin** 之一。

   b. 如果选择特定用户，则必须提供以下信息：

   - 存储库创建者.提供用户或机器人帐户。
   - 应用到。提供用户名、机器人帐户或团队名称。
   - 权限.将权限设置为 **Read,Write,** 或 **Admin** 之一。

5. 点 **Create default permissions**。此时会出现确认框，返回以下警报： **Successfully created default permissions for creator**。

14.1.10. v2 UI 的组织设置

使用以下步骤使用 **v2 UI** 更改您的机构设置。

流程

1. 在 v2 UI 上，单击 Organizations。
2. 单击您要为其创建机器人帐户的组织名称，如 test-org。
3. 点 Settings 选项卡。
4. 可选。输入与机构关联的电子邮件地址。
5. 可选。将 Time Machine 功能的分配时间设置为以下之一：
   - 1 周
   - 1 个月
   - 1 年
   - Never
6. 点击 Save。

14.1.11. 使用 v2 UI 查看镜像标签信息

使用以下步骤通过 v2 UI 查看镜像标签信息。

流程

1. 在 v2 UI 上，单击 Repositories。
2. 点存储库的名称，例如 quayadmin/busybox。

3. 单击标签的名称，例如 test。您会进入标签的 Details 页面。该页面显示以下信息：

   - Name
   - 软件仓库
   - 摘要
   - 安全漏洞
   - 创建
   - modified
   - Size
   - 标签
   - 如何获取镜像标签

4. 可选。点 Security Report 查看标签的漏洞。您可以扩展公告列以打开 CVE 数据。

5. 可选。点 Packages 查看标签的软件包。

6. 单击存储库的名称，如 busybox，以返回到 Tags 页面。

7. 可选。将鼠标悬停在 Pull 图标上，以显示获取标签的方法。

8.

选中标签框或多个标签，单击 **Actions** 下拉菜单，然后单击 **Delete** 以删除该标签。在弹出框中单击 **Delete** 来确认删除。

14.1.12. 使用 v2 UI 调整存储库设置

使用以下步骤使用 **v2 UI** 调整存储库的各种设置。

流程

1. 在 **v2 UI** 上，单击 **Repositories**。
2. 点存储库的名称，例如 **quayadmin/busybox**。
3. 点 **Settings** 选项卡。
4. 可选。单击 **User and robot permissions**。您可以通过单击 **权限** 下的下拉菜单选项来调整用户或机器人帐户的设置。您可以将设置更改为 **Read**、**Write** 或 **Admin**。
5. 可选。单击 **Events 和 notification**。您可以通过单击 **Create Notification** 来创建事件和通知。可用的事件选项如下：
   - 推送到存储库
   - 发现软件包漏洞
   - 镜像构建失败
   - 镜像构建已排队
   - 镜像构建已启动

- *镜像构建成功*
- *镜像构建已取消*

然后，发出通知。可用的选项如下：

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

选择事件选项和通知方法后，包括一个 Room ID #, a Room Notification Token，然后点 Submit。

6. *可选。单击 Repository visibility。您可以通过单击 Make Public 使存储库为私有或公共存储库。*

7. *可选。单击 Delete repository。您可以通过单击 Delete Repository 来删除存储库。*

14.2. 查看 RED HAT QUAY 标签历史记录

使用以下步骤查看 Red Hat Quay v2 UI 上的标签历史记录。

流程

1. *在 Red Hat Quay v2 UI 仪表板中，单击导航窗格中的 Repositories。*

2. 单击具有镜像标签的存储库的名称。

3. 单击 Tag History。在这个页面中，您可以执行以下操作：

- 根据标签名称搜索
- 选择日期范围
- 查看标签更改
- 查看标签修改日期及其更改的时间

14.3. 在 RED HAT QUAY V2 UI 中添加和管理标签

Red Hat Quay 管理员可以按照以下流程为标签添加和管理标签标签。

流程

1. 在 Red Hat Quay v2 UI 仪表板中，单击导航窗格中的 Repositories。

2. 单击具有镜像标签的存储库的名称。

3. 单击镜像的 kebab 菜单，再选择 Edit labels。

4. 在 Edit labels 窗口中，点 Add new label。

5. 使用 key=value 格式输入镜像标签的标签，如 com.example.release-date=2023-11-14。

注意

当无法使用 key=value 格式时返回以下错误： Invalid label format，必须为由 = 分隔的键值。

6. 单击框的空格来添加标签。

7. 可选。添加第二个标签。

8. 点 Save labels 将标签保存到镜像标签。返回以下通知： Created labels successfully。

9. 可选。点标签上的同一镜像标签菜单 kebab → Edit labels → X 将其删除；或者，您可以编辑文本。点 Save labels。标签现已被删除或编辑。

14.4. 在 RED HAT QUAY V2 UI 中设置标签过期

Red Hat Quay 管理员可以为存储库中的特定标签设置过期日期。这有助于自动清理旧的或未使用的标签，有助于减少存储空间。

流程

1. 在 Red Hat Quay v2 UI 仪表板中，单击导航窗格中的 Repositories。

2. 单击具有镜像标签的存储库的名称。

3. 单击镜像的 kebab 菜单，然后选择 Change expiration。

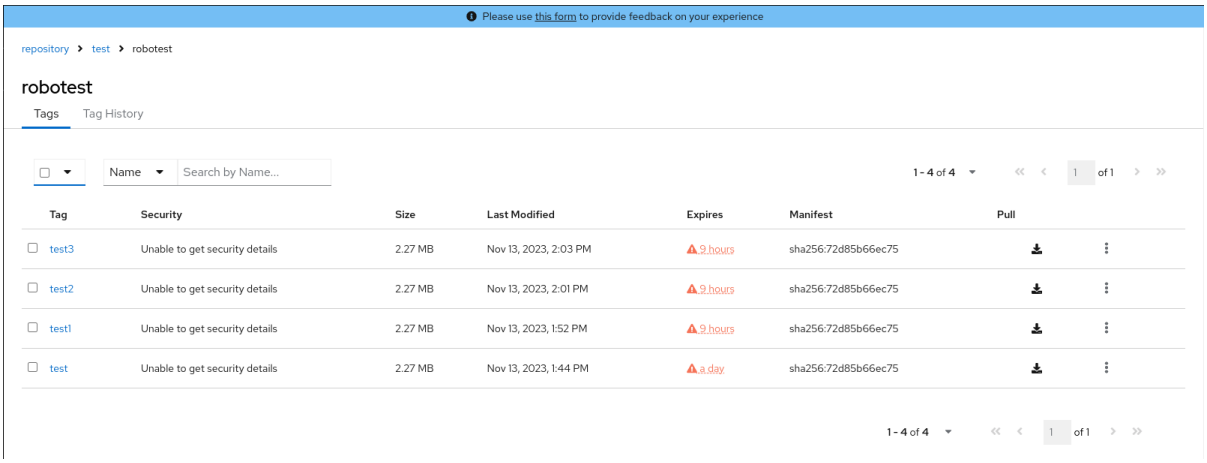
4. 可选。或者，您可以通过点击多个标签框来批量添加过期日期，然后选择 Actions → Set expiration。

5. 在 Change Tags Expiration 窗口中，设置一个过期日期，指定星期几、月份、月份和年的日期。例如，2023 年 11 月 15 日星期三。或者，您也可以单击日历按钮并手动选择日期。

6. 设置时间，例如 2:30 PM。

7. 点 Change Expiration 确认日期和时间。返回以下通知：成功将 标签测试的过期时间设置为 2023 年 11 月 15 日，2:26 PM。

8. 在 Red Hat Quay v2 UI Tags 页面中，您可以看到标签被设置为过期的时间。例如：



14.5. 在 RED HAT QUAY V2 UI 上选择颜色首选项

在使用 v2 UI 时，用户可以在 light 和 dark 模式间切换。此功能还包括自动模式选择，它根据用户的浏览器首选项在 light 或 dark 模式之间进行选择。

使用以下步骤在自动、light 和 dark 模式间切换。

流程

1. 登录到您的 Red Hat Quay 存储库。

2. 在导航窗格中，点您的用户名，例如 quayadmin。

3. 在 Appearance 下，选择 Light theme、Dark 主题 和 Device-based theme。基于主题的设备会根据浏览器的颜色首选项设置模式。

14.6. 查看 RED HAT QUAY V2 UI 中的使用日志

Red Hat Quay 日志可以提供有关 Red Hat Quay registry 使用的方式的有价值的信息。可以通过 v2 UI 上的机构、存储库或命名空间来查看日志。

流程

1. 登录到您的 Red Hat Quay registry。
2. 导航到您作为管理员的 Organization、repository 或 namespace。
3. 点 Logs。

4. 可选。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。
5. 可选。点 Export 来导出日志。您必须输入电子邮件地址或以 http:// 或 https:// 开头的有效回调 URL。根据存在多少个日志，此过程可能需要一小时。

14.7. 启用旧的 UI

1. 在导航窗格中，您可以选择在 Current UI 和 New UI 之间切换。点击切换按钮，来将它设为 Current UI。


RED HAT QUAY
EXPLORE
REPOSITORIES
TUTORIAL
Current UI
New UI
search
quayad...

第 15 章 使用 RED HAT QUAY API

Red Hat Quay 提供了完整的 OAuth 2、RESTful API：

- 可从 URL https://<yourquayhost>/api/v1中每个 Red Hat Quay 实例的端点提供
- 允许您通过浏览器连接到端点，通过启用 Swagger UI 获取、删除、发布和放置 Red Hat Quay 设置
- 可以被发出 API 调用和使用 OAuth 令牌的应用程序访问
- 以 JSON 的形式发送和接收数据

以下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。下一节会列出并描述了 API 端点。

15.1. 从 QUAY.IO 访问 QUAY API

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

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

出现的 API Explorer 显示 Quay.io API 端点。您将看到 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）。

在 API Explorer 中，您可以获取并有时更改信息：

- 账单、订阅和计划
- 存储库构建和构建触发器

- 错误消息和全局消息
- 存储库镜像、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
- 更多信息

选择打开一个端点，以查看端点的每个部分的 Model Schema。打开端点，输入任何所需参数（如存储库名称或镜像），然后选择 Try it out! 按钮来查询或更改与 Quay.io 端点关联的设置。

15.2. 创建 OAUTH 访问令牌

OAuth 访问令牌是允许您以安全的方式访问受保护的资源的凭证。使用 Red Hat Quay，您必须先创建 OAuth 访问令牌，然后才能访问机构的 API 端点。

使用以下步骤创建 OAuth 访问令牌。

先决条件

- 您已以管理员身份登录到 Red Hat Quay。

流程

1. 在主页上，选择一个 Organization。
2. 在导航窗格中，选择 Applications。

3. 点 Create New Application 并提供新应用程序名称，然后按 Enter。

4. 在 OAuth Applications 页面中，选择应用程序的名称。

5. 可选。输入以下信息：

   a. 应用程序名称

   b. 主页 URL

   c. 描述

   d. avatar E-mail

   e. 重定向/Callback URL 前缀

6. 在导航窗格中，选择 Generate Token。

7. 选中以下选项框：

   a. 管理机构

   b. 管理软件仓库

   c. 创建软件仓库

   d. 查看所有可见的存储库

e. 对任何可访问的软件仓库的读/写

f. 超级用户访问

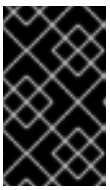
g. 管理用户

h. 读取用户信息

8. 点 Generate Access Token。您将被重定向到一个新页面。

9. 查看您允许的权限，然后点 Authorize Application。点 Authorize Application 来确认您的决定。

10. 您将被重定向到 Access Token 页面。复制并保存访问令牌。



重要

这是复制和保存访问令牌的唯一机会。在离开此页后无法保留它。

15.3. 从 WEB 浏览器访问 QUAY API

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

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

这种访问 API 的方式不包括在 Red Hat Quay 安装中提供的超级用户端点。以下是通过运行 swagger-ui 容器镜像访问本地系统中运行的 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
```

运行 swagger-ui 容器后，打开 Web 浏览器来 localhost 端口 8888，以通过 swagger-ui 容器查看

API 端点。

为了避免日志中出现错误，如 "API 调用必须通过 X-Requested-With 标头调用（如果从浏览器调用），请在集群中的所有节点上添加以下行到集群中所有节点上的 config.yaml，并重启 Red Hat Quay：

```
BROWSER_API_CALLS_XHR_ONLY: false
```

15.4. 从命令行访问 RED HAT QUAY API

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

15.4.1. 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 mFCdgS7SAIoMcnTsHCGx23vcNsTgziAa4CmmHlsg"
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
    }
  ]
}
```

15.4.2. 使用 API 创建超级用户

- 配置超级用户名称，如 Deploy Quay 书所述：
  - 使用配置编辑器 UI 或
  - 直接编辑 config.yaml 文件，使用 选项使用配置 API 验证（和下载）更新的配置捆绑包
- 为超级用户名称创建用户帐户：
  - 获取以上所述的授权令牌，并使用 curl 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
```

  - 返回的内容包括为新用户帐户生成的密码：

```
{
  "username": "quaysuper",
  "email": "quaysuper@example.com",
  "password": "EH67NB3Y6PTBED8H0HC6UVHGGGA3ODSE",
  "encrypted_password":
"fn37AZAUQH0PTsU+vlO9lS0QxPW9A/boXL4ovZjlFtlUPrBz9i4j9UDOqMjuxQ/0HTf
y38goKEpG8zYXVeQh3lOFzuOjSvKic2Vq7xdtQsU="
}
```

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 mFCdgS7SAIoMcnTsHCGx23vcNsTgziAa4CmmHlsg"
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
  }
 ]
}
```


15.4.3. 列出用量日志

可以使用 intrnal API /api/v1/superuser/logs 列出当前系统的使用情况日志。结果被分页，因此在以下示例中创建了超过 20 个仓库，以显示如何使用多个调用来访问整个结果集。

15.4.3.1. 分页示例

第一次调用

```
$ 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_2FAgqUw2X9S2LoCLNPH65QJqB4XAU2qAxYb6QqtlcWj9el6DUiMN_q3e3l0agCvB2VPQ8rY75WeaiUzM3rQlMc4i6ElR78t8oUxVfNp1RMPlRQYYZyXP9h6E8LZZhqTMs0S-SedaQJ3kVFtkxZqJwHVjgt23Ts2DonVoYwtKgl3bCC5"
}
```


使用 next_page 进行第二次调用

```
$ curl -X GET -k -H "Authorization: Bearer
qz9NZ2Np1f55CSZ3RVOvxjeUdkzYuCp0pKggABCD" https://example-registry-quay-quay-
```

```
enterprise.apps.example.com/api/v1/superuser/logs?
next_page=gAAAAABhtzGDsH38x7pjWhD8MJq1_2FAgqUw2X9S2LoCLNPH65QJqB4XAU2qA
xYb6QqtlcWj9el6DUiMN_q3e3l0agCvB2VPQ8rY75WeaiUzM3rQlMc4i6ElR78t8oUxVfNp1RMPlR
QYYZyXP9h6E8LZZhqTMs0S-SedaQJ3kVFtkxZqJwHVjgt23Ts2DonVoYwtKgl3bCC5 | 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"
        }
      }
    },
    ...
  ]
}
```

15.4.4. 目录同步

要在组织 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
```

禁用同一团队的同步：

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

15.4.5. 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 和其他所需文件的 tar 或 zip 存档。file_id 参数是我们较旧的构建系统的一部分。它无法再使用。如果 Dockerfile 位于子目录中，则需要同时指定它。

归档应该可以被公开访问。OAuth 应用应具有 "Administer Organization" 范围，因为只有机构管理员有权访问机器人的帐户令牌。否则，通过只向机器人授予构建访问权限（无需访问自身），并使用它来获取镜像内容，从而获得机器人权限。如果出现错误，请检查返回的 json 块，并确保正确传递存档位置、拉取机器人和其他参数。单击单个构建页面右上角的"下载日志"，以检查日志以了解更多信息。

15.4.6. 创建机构机器人

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

15.4.7. 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)
```

15.4.8. 创建私有软件仓库

```
$ 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
```

15.4.9. 创建已镜像的存储库

最小配置

```
curl -X POST
  -H "Authorization: Bearer ${bearer_token}"
  -H "Content-Type: application/json"
  --data '{"external_reference": "quay.io/minio/mc", "external_registry_username": "",
"sync_interval": 600, "sync_start_date": "2021-08-06T11:11:39Z", "root_rule": {"rule_kind":
"tag_glob_csv", "rule_value": [ "latest" ]}, "robot_username": "orga+robot"}'
https://${quay_registry}/api/v1/repository/${orga}/${repo}/mirror | jq
```

扩展配置

```
$ curl -X POST
  -H "Authorization: Bearer ${bearer_token}"
  -H "Content-Type: application/json"
  --data '{"is_enabled": true, "external_reference": "quay.io/minio/mc",
"external_registry_username": "username", "external_registry_password": "password",
"external_registry_config": {"unsigned_images":true, "verify_tls": false, "proxy":
{"http_proxy": "http://proxy.tld", "https_proxy": "https://proxy.tld", "no_proxy": "domain"}},
```

```
"sync_interval": 600, "sync_start_date": "2021-08-06T11:11:39Z", "root_rule": {"rule_kind":
"tag_glob_csv", "rule_value": [ "*" ]}, "robot_username": "orga+robot"}'
https://${quay_registry}/api/v1/repository/${orga}/${repo}/mirror | jq
```