Red Hat Quay 3.11

Red Hat Quay の使用

Red Hat Quay の使用

Last Updated: 2024-04-05

Red Hat Quay 3.11 Red Hat Quay の使用

Red Hat Quay の使用

法律上の通知

概要

Red Hat Quay の使用法を学ぶ

目次

はじめに

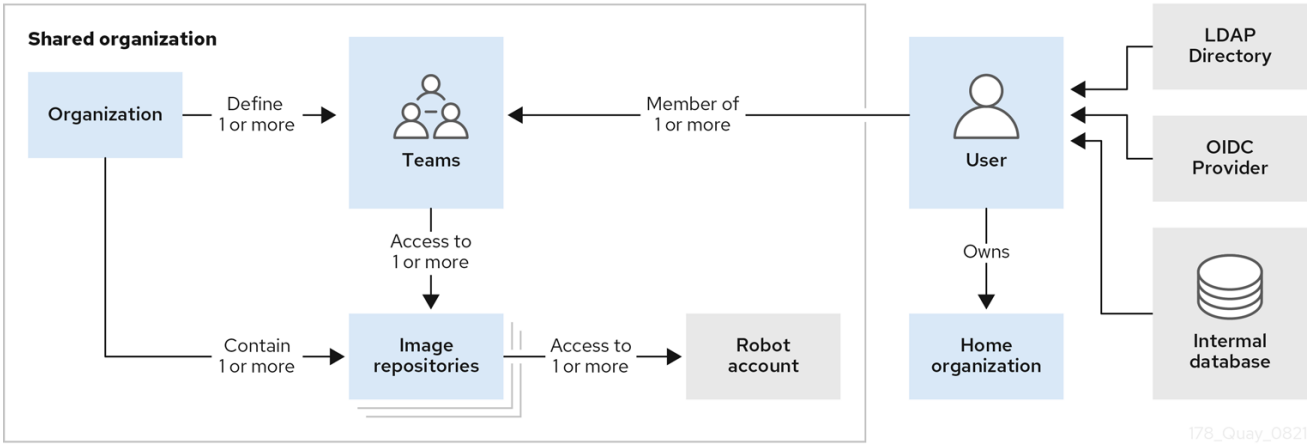
Red Hat Quay コンテナーイメージレジストリーでは、コンテナーイメージを中央の場所に保存できます。Red Hat Quay レジストリーの通常のユーザーとして、イメージを整理するためのリポジトリーを作成し、自分が管理するリポジトリーへの読み取り (プル) と書き込み (プッシュ) のアクセスを選択的に追加することができます。管理者権限を持つユーザーは、ユーザーの追加やデフォルト設定の制御など、より広範なタスクを実行することができます。

このガイドでは、Red Hat Quay がデプロイされ、設定と使用を開始する準備ができていることを前提としています。

第1章 ユーザーと組織

Red Hat Quay でコンテナーイメージを追加するリポジトリーを作成する前に、そのリポジトリーをどのように構成するかを検討する必要があります。Red Hat Quay では、各リポジトリーを **組織** または **ユーザー** と関係付ける必要があります。この関係により、リポジトリーの所有権とアクセス制御が定義されます。

1.1. テナンシーモデル



- **組織** は、単一のユーザーに属さない共通の名前空間のもとでリポジトリーを共有する方法を提供します。このようなリポジトリーは、会社などの共有設定内の複数のユーザーに属します。
- **チーム** は、組織から権限を委任する方法を提供します。権限は、グローバルレベル (たとえば、すべてのリポジトリー全体) で設定することも、特定のリポジトリーに対して設定することもできます。特定のユーザーのセットまたはグループに対して設定することもできます。
- **ユーザー** は、Web UI を介してレジストリーにログインすることも、Podman や Docker などのクライアントを使用して、それぞれのログインコマンド (**$ podman login** など) を使用してレジストリーにログインすることもできます。各ユーザーは、ユーザー名前空間 **<quay-server.example.com>/<user>/<username>**、**quay.io/<username>** など) を自動的に取得します。
- **スーパーユーザー** は、ユーザーインターフェイスの **Super User Admin Panel** を通じて、強化されたアクセス権と特権を持ちます。スーパーユーザー API 呼び出しも利用できます。通常のユーザーは、これを表示することもアクセスすることもできません。
- **ロボットアカウント** は、パイプラインツールなどの人間以外のユーザーにリポジトリーへの自動アクセスを提供します。ロボットアカウントは OpenShift Container Platform の **サービスアカウント** に似ています。リポジトリー内のロボットアカウントに権限を付与するには、そのアカウントを他のユーザーやチームと同様に追加します。

1.2. ユーザーアカウントの作成

Red Hat Quay のユーザーアカウントは、プラットフォームの機能への認証アクセス権を持つ個人を表します。このアカウントを通じて、リポジトリーの作成と管理、コンテナーイメージのアップロードと取得、およびこれらのリソースへのアクセス権の制御を行うことができます。このアカウントは、Red Hat Quay 内でのコンテナーイメージの管理を組織化および監視するうえで極めて重要です。

以下の手順を使用して、Red Hat Quay リポジトリーの新しいユーザーを作成します。

前提条件

- **config.yaml** ファイルでスーパーユーザーを設定している。詳細は、Red Hat Quay スーパーユーザーの設定 を参照してください。

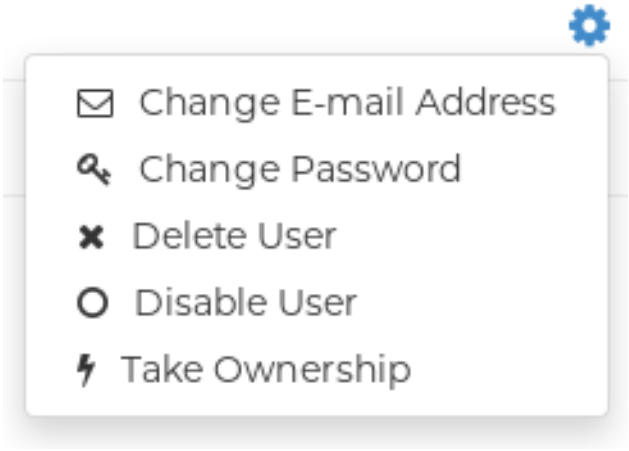
手順

1. Red Hat Quay リポジトリーにスーパーユーザーとしてログインします。
2. ナビゲーションウィンドウでアカウント名を選択し、**Super User Admin Panel** をクリックします。
3. 列の **Users** アイコンをクリックします。
4. **Create User** ボタンをクリックします。
5. 新しいユーザーのユーザー名とメールアドレスを入力し、**Create User** ボタンをクリックします。
6. **Users** ページにリダイレクトされ、別の Red Hat Quay ユーザーが表示されます。

注記

追加したユーザーを表示するには、**Users** ページを更新する必要がある場合があります。

7. **Users** ページで、新しいユーザーに関連する **Options** の歯車をクリックします。下図のようなドロップダウンメニューが表示されます。



8. **Change Password** をクリックします。
9. 新しいパスワードを追加し、**Change User Password** をクリックします。
新しいユーザーが、そのユーザー名とパスワードを使用して、Web UI または Docker や Podman など、希望するコンテナークライアントを通じてログインできるようになります。

1.3. コマンドラインからの RED HAT QUAY ユーザーの削除

Red Hat Quay UI の **Superuser Admin** パネルの **Users** タブにアクセスすると、ユーザーが一覧表示されていない状況が発生する可能性があります。代わりに、Red Hat Quay が外部認証を使用するように設定されていて、ユーザーはそのシステムでのみ作成できることを示すメッセージが表示されます。

このエラーは、次の 2 つの理由のいずれかで発生します。

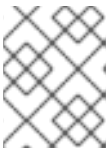
- ユーザーを読み込むときに Web UI がタイムアウトします。この問題が発生すると、ユーザーはアクセスして操作を実行できなくなります。
- LDAP 認証: ユーザー ID が変更されたが、関連付けられたメールアドレスが変更されていない場合。現在、Red Hat Quay では、古いメールアドレスを使用して新しいユーザーを作成することはできません。

この問題が発生した場合は、次の手順を使用して Red Hat Quay からユーザーを削除します。

手順

- 以下の **curl** コマンドを入力して、コマンドラインからユーザーを削除します。

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



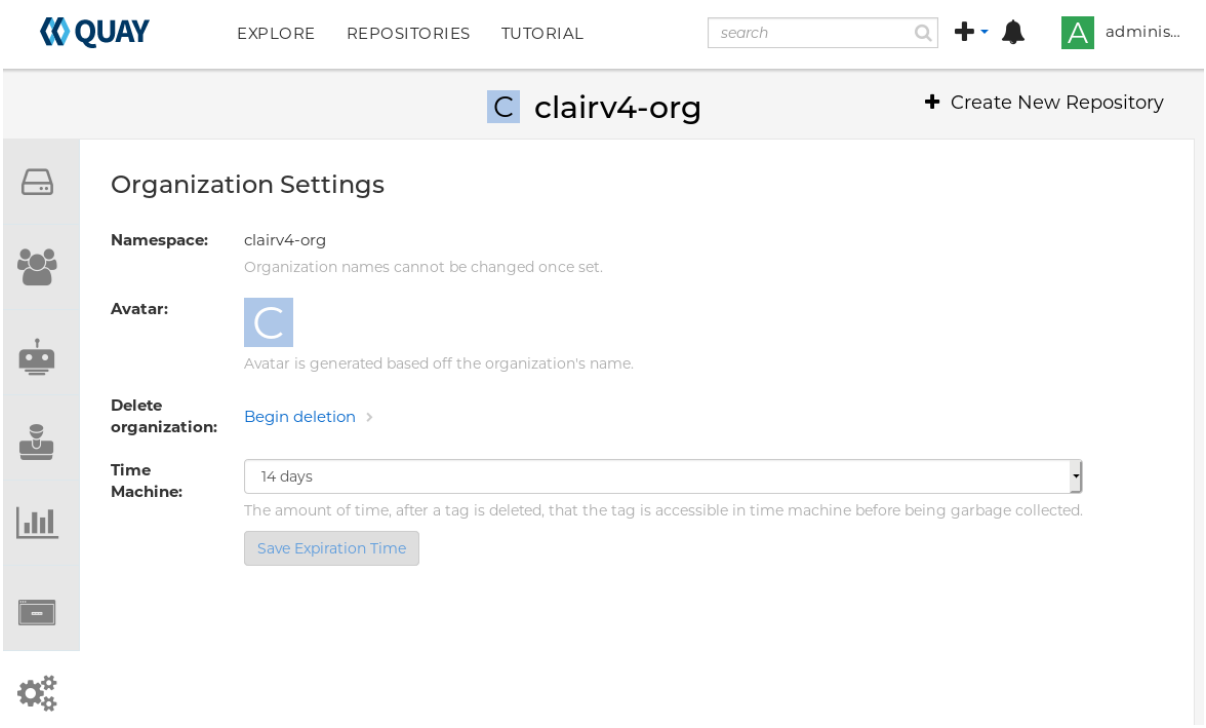
注記

ユーザーを削除すると、このユーザーが自分のプライベートアカウントにあったリポジトリーはすべて使用できなくなります。

1.4. 組織アカウントの作成

ユーザーは誰でも自分の組織を作り、コンテナーイメージのリポジトリーを共有することができます。新しい組織を作るには、以下の手順に従います。

1. 任意のユーザーでログインした状態で、ホームページの右上隅にある正符号 (+) を選択し、New Organization を選択します。
2. 組織の名前を入力します。名前は英数字で、すべて小文字で、2 ~ 255 文字の間でなければなりません。
3. Create Organization を選択します。新しい組織が表示され、リポジトリー、チーム、ロボットアカウント、その他の機能を左カラムのアイコンから追加できます。次の図は、設定タブを選択した場合の新組織のページの例です。


QUAY
EXPLORE
REPOSITORIES
TUTORIAL
search
adminis...
C clairv4-org
+ Create New Repository
Organization Settings
Namespace:
clairv4-org
Organization names cannot be changed once set.
Avatar:
C
Avatar is generated based off the organization's name.
Delete organization:
Begin deletion
Time Machine:
14 days
The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected.
Save Expiration Time

第2章 リポジトリーの作成

リポジトリーは、関連するコンテナーイメージのセットを一元的に保存するための場所を提供します。これらのイメージを使用して、アプリケーションとその依存関係を標準化された形式で構築できます。

リポジトリーは名前空間を使用して整理します。名前空間には、それぞれ複数のリポジトリーを含めることができます。たとえば、個人プロジェクト用の名前空間、会社用の名前空間、または組織内の特定のチーム用の名前空間を設定できます。

Red Hat Quay では、ユーザーがリポジトリーへのアクセスを制御できます。リポジトリーをパブリックにすると、誰でもリポジトリーからイメージをプルまたはダウンロードできるようになります。リポジトリーをプライベートにすると、許可されたユーザーまたはチームのみにアクセスが制限されます。

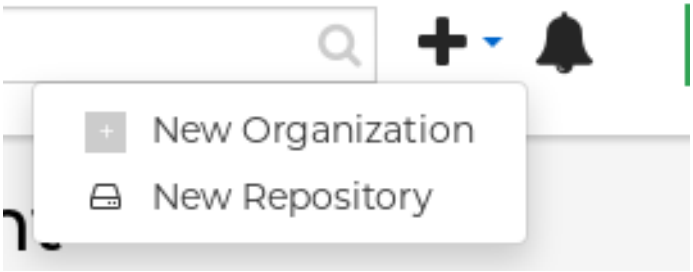
Red Hat Quay でリポジトリーを作成するには、関連する **docker** または **podman** コマンドを使用してイメージをプッシュする方法と、Red Hat Quay UI を使用する方法の 2 つがあります。

2.1. UI を使用したイメージリポジトリーの作成

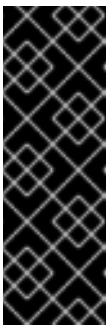
Red Hat Quay UI を使用してリポジトリーを作成するには、次の手順を実行します。

手順

1. Web UI からユーザーアカウントにログインします。
2. Red Hat Quay のランディングページで、**Create New Repository** をクリックします。または、**+** アイコン → **New Repository** をクリックすることもできます。以下に例を示します。



3. **Create New Repository** ページで、以下を行います。
   a. 使用するユーザー名または組織に **Repository Name** を追加します。



重要

リポジトリー名には次の単語を使用しないでください。* **build** * **trigger** * **tag**

これらの単語をリポジトリー名に使用すると、ユーザーがリポジトリーにアクセスできなくなり、リポジトリーを完全に削除できなくなります。このようなリポジトリーを削除しようとすると、**Failed to delete repository <repository_name>, HTTP404 - Not Found.** というエラーが返されます。

   b. オプション: **Click to set repository description** をクリックして、リポジトリーの説明を追加します。
   c. ニーズに合わせて、**Public** または **Private** をクリックします。
   d. オプション: 必要なリポジトリーの初期化を選択します。

4. **Create Private Repository**をクリックして、新しい空のリポジトリーを作成します。

2.2. CLI を使用したイメージリポジトリーの作成

適切な認証情報がある場合は、Docker または Podman を使用して、Red Hat Quay インスタンスにまだ存在しないリポジトリーにイメージを **プッシュ** できます。イメージのプッシュとは、コンテナーイメージをローカルシステムまたは開発環境から Quay.io などのコンテナーレジストリーにアップロードするプロセスを指します。イメージを Quay.io にプッシュすると、リポジトリーが作成されます。

イメージをプッシュしてイメージリポジトリーを作成するには、次の手順を実行します。

前提条件

- **podman** CLI をダウンロードしてインストールしている。
- Quay.io にログインしている。
- イメージ (busybox など) をプルしている。

手順

1. サンプルレジストリーからサンプルページを取得します。以下に例を示します。

```
$ sudo podman pull busybox
```

出力例

```
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. ローカルシステム上のイメージに、新しいリポジトリーとイメージ名をタグ付けします。以下に例を示します。

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

3. イメージをレジストリーにプッシュします。この手順の後に、ブラウザーを使用して、リポジトリーでタグ付けされたイメージを確認できます。

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

出力例

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

第3章 リポジトリーへのアクセス管理

Red Hat Quay ユーザーは、独自のリポジトリーを作成し、インスタンスに含まれる他のユーザーにそのリポジトリーへのアクセスを許可できます。または、特定の組織を作成して、定義されたチームに基づいてリポジトリーへのアクセスを許可することもできます。

ユーザーリポジトリーと組織リポジトリーのどちらでも、ロボットアカウントに関連する認証情報を作成すると、そのリポジトリーへのアクセスを許可できます。ロボットアカウントを使用すると、Red Hat Quay ユーザーアカウントを持っていないさまざまなコンテナークライアント (Docker や Podman など) がリポジトリーに簡単にアクセスできるようになります。

3.1. ユーザーリポジトリーへのアクセスの許可

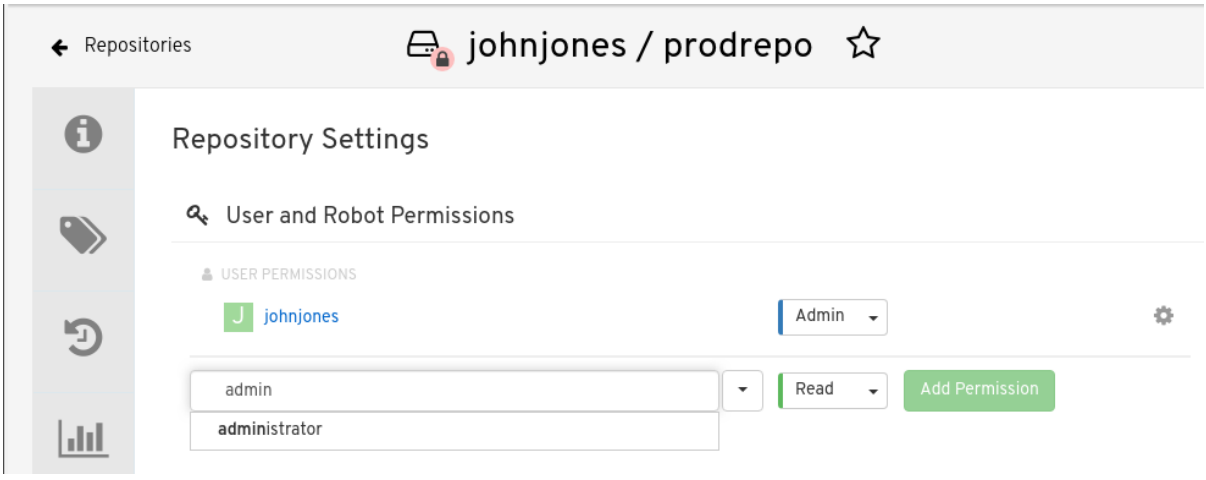
ユーザー名前空間にリポジトリーを作成すると、そのリポジトリーへのアクセスを、ユーザーアカウントに、またはロボットアカウントを通じて追加できます。

3.1.1. ユーザーリポジトリーへのユーザーアクセスの許可

ユーザーアカウントに関連付けられたリポジトリーへのアクセスを許可するには、次の手順を実行します。

手順

1. ユーザーアカウントを使用して Red Hat Quay にログインします。
2. 複数のユーザー間で共有されるユーザー名前空間配下のリポジトリーを選択します。
3. ナビゲーションペインで **Settings** を選択します。
4. 自分のリポジトリーへのアクセスを許可するユーザーの名前を入力します。入力すると、名前が表示されます。以下に例を示します。



5. パーミッションボックスで、以下のいずれかを選択します。
   - **Read**。リポジトリーの表示とリポジトリーからのプルをユーザーに許可します。
   - **Write**。リポジトリーの表示、リポジトリーからのプル、リポジトリーへのイメージのプッシュをユーザーに許可します。
   - **Admin**。リポジトリーに対するすべての管理設定と、すべての **Read** および **Write** 権限をユーザーに提供します。

6. **Add Permission** ボタンを選択します。これで、ユーザーに権限が割り当てられました。

7. オプション: リポジトリーに対するユーザー権限を削除または変更するには、**Options** アイコンを選択し、**Delete Permission** を選択します。

3.1.2. ユーザーリポジトリーへのロボットアクセスの許可

ロボットアカウントは、Red Hat Quay レジストリー内のリポジトリーへの自動アクセスを設定するのに使用します。ロボットアカウントは OpenShift Container Platform のサービスアカウントに似ています。

ロボットアカウントを設定すると、以下が実行されます。

- ロボットアカウントに関連付けられた認証情報が生成されます。
- ロボットアカウントがイメージをプッシュおよびプルできるリポジトリーとイメージが特定されます。
- 生成された認証情報をコピー/ペーストして、Docker、Podman、Kubernetes、Mesos などのさまざまなコンテナークライアントで使用し、定義された各リポジトリーにアクセスできます。

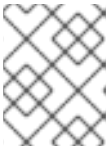
各ロボットアカウントは、1 つのユーザー名前空間または組織に制限されます。たとえば、ロボットアカウントは、ユーザー **jsmith** にすべてのリポジトリーへのアクセスを提供できます。しかし、ユーザーのリポジトリーリストにないリポジトリーへのアクセスは提供できません。

次の手順を使用して、リポジトリーへのアクセスを許可できるロボットアカウントを設定します。

手順

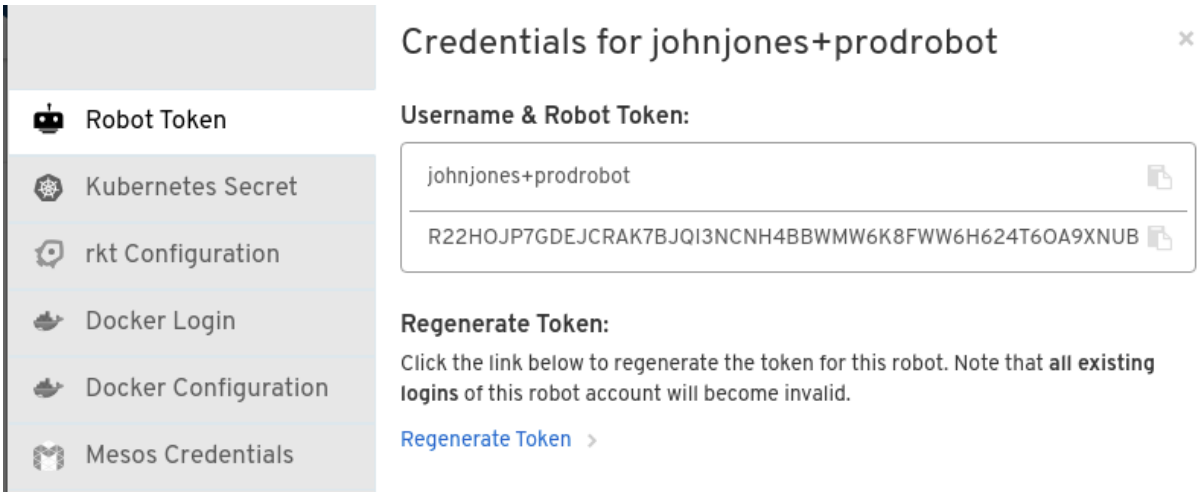
1. **Repositories** ランディングページで、ユーザーの名前をクリックします。
2. ナビゲーションペインで **Robot Accounts** をクリックします。
3. **Create Robot Account** をクリックします。
4. ロボットアカウントの名前を入力します。
5. オプション: ロボットアカウントの説明を入力します。
6. **Create Robot Account** をクリックします。ロボットアカウントの名前は、ユーザー名とロボットの名前を組み合わせたものになります (例: **jsmith+robot)**。
7. ロボットアカウントを関連付けるリポジトリーを選択します。
8. ロボットアカウントの権限を次のいずれかに設定します。
   - **None**。ロボットアカウントにリポジトリーに対する権限は付与されません。
   - **Read**。ロボットアカウントがリポジトリーの表示とリポジトリーからのプルを行えるようになります。
   - **Write**。ロボットアカウントがリポジトリーからの読み取り (プル) とリポジトリーへの書き込み (プッシュ) を行えるようになります。
   - **Admin**。プルおよびプッシュを行うためのリポジトリーへのフルアクセスに加えて、リポジトリーに関連する管理作業を行う権限を付与します。

9. **Add permissions** ボタンをクリックして設定を適用します。
10. **Robot Accounts** ページで、ロボットアカウントを選択して、そのロボットの認証情報を表示します。
11. **Robot Account** オプションで、**Copy to Clipboard** をクリックして、ロボット用に生成されたトークンをコピーします。新しいトークンを生成するには、**Regenerate Token** をクリックします。



注記

トークンを再生成すると、このロボットの以前のトークンがすべて無効になります。



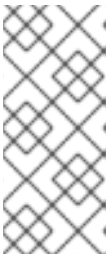
12. 生成された認証情報を次の方法で取得します。
    - **Kubernetes Secret**: Kubernetes プルシークレット yaml ファイルの形式で認証情報をダウンロードするには、これを選択します。
    - **rkt Configuration**: rkt コンテナーランタイムの認証情報を **.json** ファイルの形式でダウンロードするには、これを選択します。
    - **Docker Login**: 認証情報を含む完全な **docker login** コマンドラインをコピーするには、これを選択します。
    - **Docker Configuration**: Docker **config.json** ファイルとして使用するファイルがダウンロードし、クライアントシステムに認証情報を永続的に保存するには、これを選択します。
    - **Mesos Credentials**: Mesos 設定ファイルの URI フィールドで識別できる認証情報を提供する tarball をダウンロードするには、これを選択します。

3.2. 組織リポジトリー

組織を作成すると、リポジトリーのセットを直接その組織に関連付けることができます。組織リポジトリーは、組織がユーザーのグループを通じて共有リポジトリーをセットアップすることを目的としているという点で、基本的なリポジトリーとは異なります。Red Hat Quay では、ユーザーのグループは、**チーム**、同じ権限を持つユーザーのセット、または **個々のユーザー** のいずれかです。

組織に関するその他の有用な情報を以下に示します。

- 組織を別の組織内に組み込むことはできません。組織を細分化するには、チームを使用します。
- 組織にユーザーを直接含めることはできません。まずチームを追加してから、各チームに 1 人以上のユーザーを追加する必要があります。



注記

個々のユーザーは、組織内の特定のリポジトリーに追加できます。そのため、これらのユーザーは、**Repository Settings** ページのどのチームのメンバーでもありません。**Teams and Memberships**ページの **Collaborators View** に、特に組織に所属する必要がなく、その組織内の特定のリポジトリーに直接アクセスできるユーザーが表示されます。

- チームは、リポジトリーおよび関連イメージを使用する単なるメンバーとして、または組織を管理するための特別な権限を持つ管理者として、組織内に設定できます。

3.2.1. 組織の作成

組織を作成するには、次の手順を実行します。

手順

1. **Repositories** ランディングページで、**Create New Organization** をクリックします。
2. **Organization Name** に、2 文字以上 225 文字未満の名前を入力します。
3. **Organization Email** に、アカウントのメールアドレスとは異なるメールアドレスを入力します。
4. **Create Organization** をクリックして作成を完了します。

3.2.1.1. API を使用した別の組織の作成

API を使用して別の組織を作成できます。これを行うには、UI を使用して最初の組織を作成しておく必要があります。OAuth アクセストークンも生成しておく必要があります。

Red Hat Quay API エンドポイントを使用して別の組織を作成するには、次の手順を実行します。

前提条件

- UI を使用して少なくとも 1 つの組織をすでに作成している。
- OAuth アクセストークンを生成している。詳細は、「OAuth アクセストークンの作成」を参照してください。

手順

1. 次のコマンドを入力して、**data.json** という名前のファイルを作成します。

```
$ touch data.json
```

2. 次の内容をファイルに追加します。これが新しい組織の名前になります。

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

3. 次のコマンドを入力して、OAuth アクセストークンと Red Hat Quay レジストリーエンドポイントを渡し、API エンドポイントを使用して新しい組織を作成します。

```
$ 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/
```

出力例

```
"Created"
```

3.2.2. 組織へのチームの追加

組織のためにチームを作成する際に、チーム名を選択し、チームが利用できるリポジトリーを選択し、チームのアクセスレベルを決定できます。

組織のチームを作成するには、次の手順を実行します。

前提条件

- 組織を作成している。

手順

1. **Repositories** ランディングページで、チームを追加する組織を選択します。
2. ナビゲーションペインで、**Teams and Membership** を選択します。デフォルトでは、組織を作成したユーザーの **Admin** 権限を持つ **owners** チームが存在します。
3. **Create New Team** をクリックします。
4. 新しいチームの名前を入力します。チーム名の先頭は小文字である必要があります。また、使用できるのは小文字と数字のみです。大文字や特殊文字は使用できません。
5. **Create team** をクリックします。
6. チームの名前をクリックすると、**Team** ページにリダイレクトされます。ここで、チームの説明を追加したり、登録ユーザー、ロボット、メールアドレスなどのチームメンバーを追加したりできます。詳細は、「チームへのユーザーの追加」を参照してください。
7. **No repositories** テキストをクリックすると、使用可能なリポジトリーのリストが表示されます。チームにアクセスを許可する各リポジトリーのボックスを選択します。
8. チームに付与する適切な権限を選択します。
   - **None**。チームメンバーにリポジトリーに対する権限は付与されません。
   - **Read**。チームメンバーがリポジトリーの表示とリポジトリーからのプルを行えるようになります。
   - **Write**。チームメンバーがリポジトリーの読み取り (プル) とリポジトリーへの書き込み (プッシュ) を行えるようになります。

- **Admin**。プルおよびプッシュを行うためのリポジトリーへのフルアクセスに加えて、リポジトリーに関連する管理作業を行う権限を付与します。

9. **Add permissions** を選択して、チームのリポジトリー権限を保存します。

3.2.3. チームロールの設定

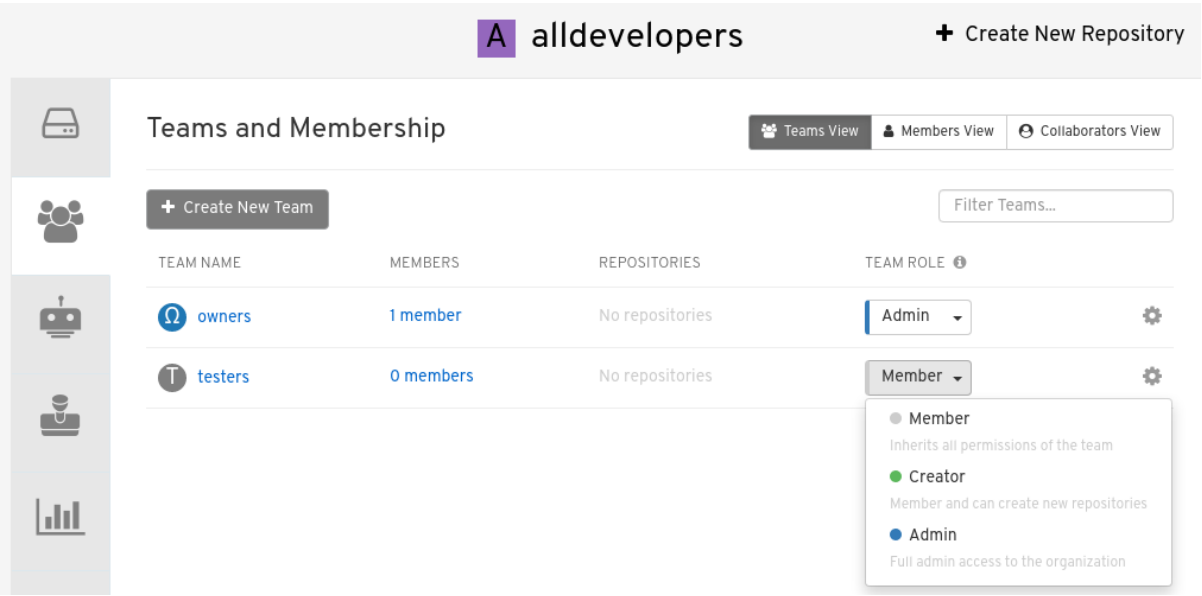
チームを追加したら、そのチームの組織内でのロールを設定できます。

前提条件

- チームを作成している。

手順

1. **Repository** ランディングページで、組織の名前をクリックします。
2. ナビゲーションウィンドウで、**Teams and Membership** をクリックします。
3. 次の図に示すように、**TEAM ROLE** ドロップダウンメニューを選択します。



4. 選択したチームについて、以下のロールのいずれかを選択します。

- **Member**。チームに設定されているすべての権限を継承します。
- **Creator**。メンバーのすべての権限に加えて、新しいリポジトリーを作成する権限を付与します。
- **Admin**。チームの作成、メンバーの追加、権限の設定など、組織への完全な管理アクセス権を付与します。

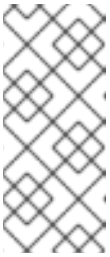
3.2.4. チームへのユーザーの追加

組織に対する管理者権限を持っていれば、ユーザーとロボットアカウントをチームに追加できます。ユーザーを追加すると、Red Hat Quay はそのユーザーにメールを送信します。そのユーザーが招待を受け入れるまで、ユーザーは保留状態のままになります。

ユーザーまたはロボットアカウントをチームに追加するには、次の手順を実行します。

手順

1. **Repository** ランディングページで、組織の名前をクリックします。
2. ナビゲーションウィンドウで、**Teams and Membership** をクリックします。
3. ユーザーまたはロボットアカウントを追加するチームを選択します。
4. **Team Members** ボックスに、次のいずれかの情報を入力します。
   - レジストリー上のアカウントからのユーザー名。
   - レジストリー上のユーザーアカウントのメールアドレス。
   - ロボットアカウントの名前。名前は、<組織名>+<ロボット名> の形式である必要があります。



注記

ロボットアカウントはすぐにチームに追加されます。ユーザーアカウントの場合、参加の招待がユーザーにメールで送信されます。ユーザーがその招待を受け入れるまで、ユーザーは **INVITED TO JOIN** 状態のままになります。ユーザーがチームへの参加招待メールを受け入れると、**INVITED TO JOIN** リストから組織の **MEMBERS** リストに移動します。

関連情報

OAuth アクセストークンの作成

3.3. ロボットアカウントの無効化

Red Hat Quay 管理者は、ユーザーに新しいロボットアカウントの作成を禁止することでロボットアカウントを管理できます。

重要

リポジトリーのミラーリングにはロボットアカウントが必要です。**ROBOTS_DISALLOW** 設定フィールドを **true** に設定すると、ミラーリング設定が破棄されます。リポジトリーをミラーリングするユーザーは、**config.yaml** ファイルで **ROBOTS_DISALLOW** を **true** に設定しないでください。これは既知の問題であり、Red Hat Quay の今後のリリースで修正される予定です。

ロボットアカウントの作成を無効にするには、次の手順を実行します。

前提条件

- 複数のロボットアカウントを作成している。

手順

1. **config.yaml** フィールドを更新して **ROBOTS_DISALLOW** 変数を追加します。次に例を示します。

```
ROBOTS_DISALLOW: true
```

2. Red Hat Quay デプロイメントを再起動します。

検証: 新しいロボットアカウントの作成

1. Red Hat Quay リポジトリーに移動します。
2. リポジトリーの名前をクリックします。
3. ナビゲーションウィンドウで、**Robot Accounts** をクリックします。
4. **Create Robot Account** をクリックします。
5. ロボットアカウントの名前を入力します (例: **<organization-name/username>+<robot-name>)**。
6. **Create robot account** をクリックして作成を確定します。次のメッセージが表示されます。**Cannot create robot account.Robot accounts have been disabled.Please contact your administrator.**

検証: ロボットアカウントへのログイン

1. コマンドラインインターフェイス (CLI) で、次のコマンドを入力してロボットアカウントの 1 つとしてログインを試みます。

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

次のエラーメッセージが返されます。

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

2. **log-level=debug** フラグを渡すと、ロボットアカウントが無効化されたことを確認できます。

```
$ 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 タグの使用

イメージタグ は、コンテナーイメージの特定のバージョンまたはバリアントに割り当てられたラベルまたは識別子を指します。コンテナーイメージは通常、イメージのさまざまな部分を表す複数のレイヤーで構成されます。イメージタグは、異なるバージョンのイメージを区別したり、イメージに関する追加情報を提供したりするために使用します。

イメージタグには次の利点があります。

- **バージョン管理とリリース**: イメージタグを使用すると、アプリケーションまたはソフトウェアのさまざまなバージョンまたはリリースを示すことができます。たとえば、初期リリースを表すために **v1.0** とイメージにタグ付けしたり、更新されたバージョンを表すために **v1.1** とタグ付けしたりできます。これは、イメージのバージョンの明確な履歴を維持するのに役立ちます。

- **ロールバックとテスト**: 新しいイメージのバージョンで問題が発生した場合は、タグを指定することで以前のバージョンに簡単に戻すことができます。これは、デバッグ段階とテスト段階で特に役立ちます。

- **開発環境**: イメージタグは、さまざまな環境で作業する場合に役立ちます。たとえば、開発バージョンには **dev** タグ、品質保証テストには **qa**、本番環境には **prod** タグを使用して、それぞれ機能と設定を異なるものにすることができます。

- **継続的インテグレーション/継続的デプロイメント (CI/CD)**: CI/CD パイプラインでは、イメージタグを使用してデプロイメントプロセスを自動化することがよくあります。新しいコードの変更により、特定のタグを持つ新しいイメージの作成をトリガーすることで、シームレスな更新が可能になります。

- **機能ブランチ**: 複数の開発者が異なる機能やバグ修正に取り組んでいる場合は、変更ごとに個別のイメージタグを作成できます。これは、個々の機能を分離してテストするのに役立ちます。

- **カスタマイズ**: イメージタグを使用すると、各バリアントを追跡しながら、さまざまな設定、依存関係、または最適化を使用してイメージをカスタマイズできます。

- **セキュリティーとパッチ適用**: セキュリティーの脆弱性が発見された場合は、更新されたタグを使用して、パッチが適用されたバージョンのイメージを作成し、セキュアな最新バージョンをシステムで確実に使用できます。

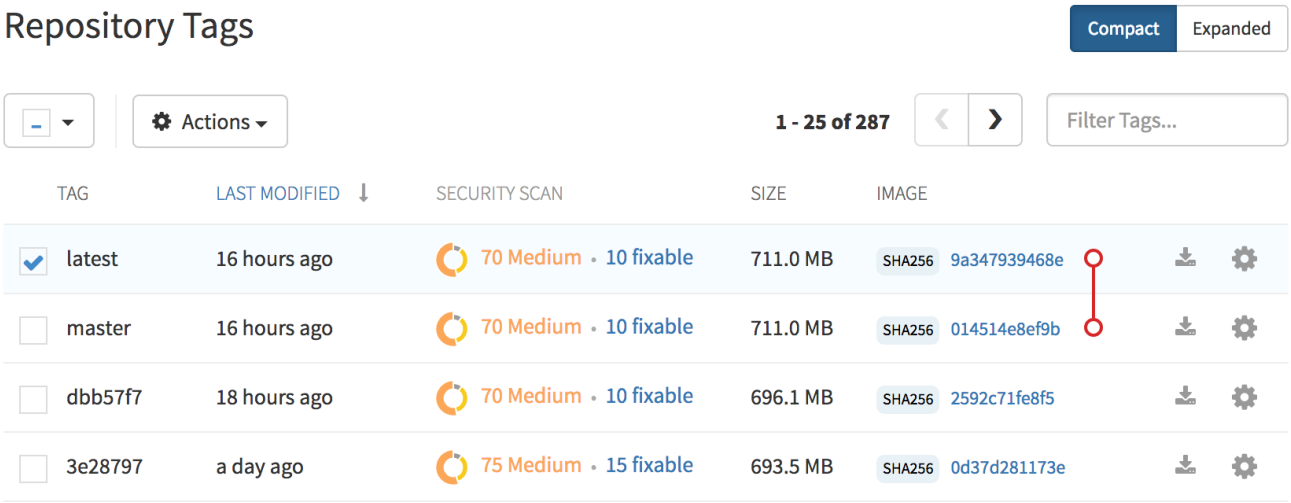
- **Dockerfile の変更**: Dockerfile またはビルドプロセスを変更する場合は、イメージタグを使用して、以前の Dockerfile と更新された Dockerfile からビルドしたイメージを区別できます。

全体として、イメージタグはコンテナーイメージを管理および整理するための構造化された方法を提供し、開発、デプロイ、および保守の効率的なワークフローを可能にします。

4.1. タグの表示と変更

Red Hat Quay でイメージタグを表示するには、リポジトリーに移動して、**Tags** タブをクリックします。以下に例を示します。

リポジトリーからのタグの表示および変更



4.1.1. イメージへの新しいイメージタグの追加

Red Hat Quay のイメージに新しいタグを追加できます。

手順

1. タグの横にある **Settings** または **歯車** アイコンをクリックし、**Add New Tag** をクリックします。
2. タグの名前を入力し、**Create Tag** をクリックします。
   新しいタグが **Repository Tags** ページにリストされます。

4.1.2. イメージタグの移動

必要に応じて、タグを別のイメージに移動できます。

手順

- タグの横にある **Settings** または **歯車** アイコンをクリックし、**Add New Tag** をクリックして既存のタグ名を入力します。タグを追加するのではなく移動することを確認するメッセージが表示されます。

4.1.3. イメージタグの削除

イメージタグを削除すると、その特定のバージョンのイメージがレジストリーから実質的に削除されます。

イメージタグを削除するには、次の手順を実行します。

手順

1. リポジトリーの **Tags** ページに移動します。
2. **Delete Tag** をクリックします。これにより、タグとそのタグに固有のイメージが削除されます。

注記

イメージタグの削除は、**タイムマシン** 機能に割り当てられている時間に基づいて元に戻すことができます。詳細は、「タグの変更を元に戻す」を参照してください。

4.1.3.1. タグの履歴の表示

Red Hat Quay では、イメージとそれぞれのイメージタグの包括的な履歴を確認できます。

手順

- リポジトリーの **Tag History** ページに移動して、イメージタグの履歴を表示します。

4.1.3.2. タグの変更を元に戻す

Red Hat Quay は、古いイメージタグを一定期間リポジトリーに保持できる包括的な **タイムマシン** 機能を備えており、タグに加えられた変更を元に戻すことができます。この機能を使用すると、タグの削除などのタグの変更を元に戻すことができます。

手順

1. リポジトリーの **Tag History** ページに移動します。
2. タイムライン内でイメージタグが変更または削除された時点を見つけます。次に、**Revert** の下のオプションをクリックしてタグをイメージに復元するか、**Permanently Delete** の下のオプションをクリックしてイメージタグを完全に削除します。

4.1.4. タグやダイジェストによるイメージの取得

Red Hat Quay では、Docker クライアントと Podman クライアントを使用して、複数の方法でイメージをプルできます。

手順

1. リポジトリーの **Tags** ページに移動します。
2. **Manifest** で、**Fetch Tag** アイコンをクリックします。
3. ポップアップボックスが表示され、次のオプションが表示されます。
   - Podman Pull (by tag)
   - Docker Pull (by tag)
   - Podman Pull (by digest)
   - Docker Pull (by digest)
     4 つのオプションのいずれかを選択すると、イメージのプルに使用できる各クライアント用のコマンドが返されます。
4. **Copy Command** をクリックしてコマンドをコピーします。このコマンドはコマンドラインインターフェイス (CLI) で使用できます。以下に例を示します。

```
$ podman pull quay-server.example.com/quayadmin/busybox:test2
```

4.2. タグの有効期限

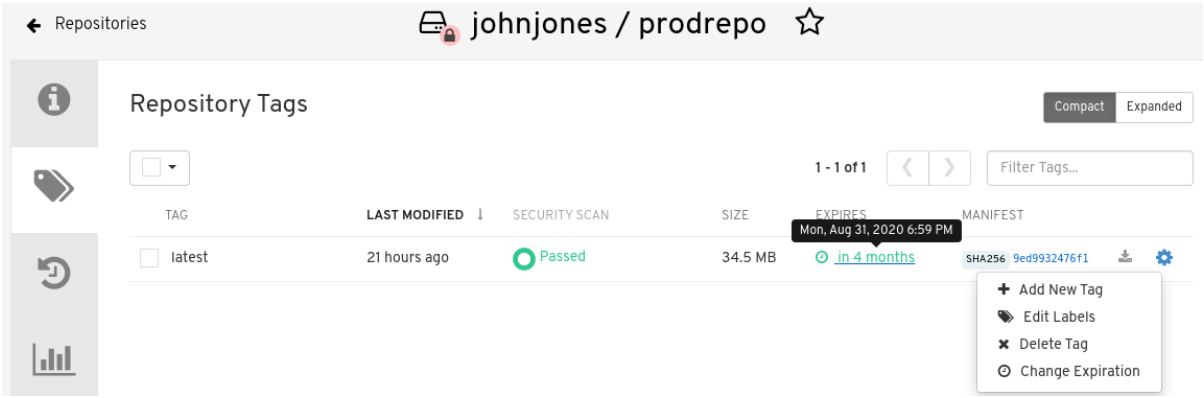
タグの有効期限 機能を使用すると、選択した日時に Red Hat Quay リポジトリーのイメージが期限切れになるように設定できます。この機能には次の特徴があります。

- イメージタグの有効期限が切れると、そのタグがリポジトリーから削除されます。特定のイメージに対する最後のタグであれば、そのイメージも削除するように設定されます。
- 有効期限はタグごとに設定されます。リポジトリー全体に対して設定されるものではありません。
- タグは期限切れになったり、削除されたりしても、レジストリーからすぐには削除されません。これは、**タイムマシン** 機能で設計された割り当て時間によって決まります。この時間により、タグを完全に削除するタイミング、またはガベージコレクションを行うタイミングが定義されます。デフォルトでは、この値は **14 日** に設定されていますが、管理者は複数のオプションから 1 つ選択することで、この時間を調整できます。ガベージコレクションが発生するまでは、タグの変更を元に戻すことができます。

Red Hat Quay のスーパーユーザーには、ユーザーリポジトリーからの期限切れイメージの削除に関する特別な権限はありません。スーパーユーザーがユーザーリポジトリーの情報を収集し、操作するための一元的なメカニズムはありません。有効期限やイメージ削除の管理は、各リポジトリーの所有者に委ねられています。

タグの有効期限は、次の 2 つの方法のいずれかで設定できます。

- イメージの作成時に Dockerfile で **quay.expires-after=** LABEL を設定する方法。これは、イメージをビルドした時点からの有効期間を設定するものです。
- Red Hat Quay UI で有効期限を選択する方法。以下に例を示します。



4.2.1. Dockerfile からのタグの有効期限の設定

docker label コマンドを使用してラベル (例: **quay.expires-after=20h**) を追加すると、指定した時間の経過後に、タグが自動的に期限切れになります。以下に示す時間、日、または週の値を指定できます。

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

有効期限は、イメージがレジストリーにプッシュされた時点から始まります。

4.2.2. リポジトリーからのタグの有効期限の設定

タグの有効期限は Red Hat Quay UI で設定できます。

手順

1. リポジトリーに移動し、ナビゲーションペインで **Tags** をクリックします。
2. イメージタグの **Settings** または **gear** アイコンをクリックし、**Change Expiration** を選択します。
3. プロンプトが表示されたら日付と時刻を選択し、**Change Expiration** を選択します。有効期限に達するとタグがリポジトリーから削除されるように設定されます。

4.3. CLAIR セキュリティースキャンの表示

Clair セキュリティースキャナーは、デフォルトでは Red Hat Quay に対して有効になっていません。Clair を有効にするには、Red Hat Quay の Clair を参照してください。

手順

1. リポジトリーに移動し、ナビゲーションペインで **Tags** をクリックします。このページに、セキュリティースキャンの結果が表示されます。
2. マルチアーキテクチャーイメージに関する詳細情報を表示するには、**See Child Manifests** をクリックして、展開されたビューでマニフェストのリストを確認します。
3. **See Child Manifests** の下の関連リンクをクリックします。たとえば、**1 Unknown** をクリックすると、**Security Scanner** ページにリダイレクトされます。
4. **Security Scanner** ページには、イメージがどの CVE の影響を受けるか、利用可能な修復オプションなど、タグに関する情報が表示されます。

注記

イメージスキャンでは、Clair セキュリティースキャナーによって検出された脆弱性がリストされるだけです。発見された脆弱性への対処は、当該ユーザーに委ねられています。Red Hat Quay のスーパーユーザーが発見された脆弱性に対処することはありません。

第5章 ログの表示とエクスポート

アクティビティーログは、Red Hat Quay のすべてのリポジトリーおよび名前空間について収集されます。

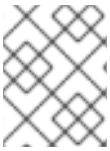
Red Hat Quay の使用状況ログを表示すると、運用とセキュリティーの両方の目的で貴重な洞察と利点を得ることができます。使用状況ログから次の情報が明らかになる可能性があります。

- **リソースプランニング**: 使用状況ログは、イメージのプル、プッシュの数、レジストリーへの全体的なトラフィックに関するデータを提供します。
- **ユーザーアクティビティー**: ログを使用すると、ユーザーアクティビティーを追跡し、どのユーザーがレジストリー内のイメージにアクセスして操作しているかを把握できます。これは、監査、ユーザー行動の理解、アクセス制御の管理に役立ちます。
- **使用パターン**: 使用パターンを調査することで、よく使用されるイメージ、頻繁に使用されるバージョン、ほとんどアクセスされないイメージについて詳細な情報を得ることができます。この情報は、イメージのメンテナンスとクリーンアップの作業に優先順位を付けるのに役立ちます。
- **セキュリティー監査**: 使用状況ログにより、誰がいつイメージにアクセスしたかを追跡できます。これは、セキュリティー監査、コンプライアンス、および不正または不審なアクティビティーの調査にとって非常に重要です。
- **イメージのライフサイクル管理**: ログにより、どのイメージがプル、プッシュ、削除されているかが明らかになります。この情報は、古いイメージを廃止し、許可されたイメージだけを確実に使用させるなど、イメージのライフサイクル管理に不可欠です。
- **コンプライアンスと規制要件**: 多くの業界には、機密リソースへのアクセスの追跡と監査を義務付けるコンプライアンス要件があります。使用状況ログは、そのような規制への準拠を証明するのに役立ちます。
- **異常な動作の特定**: 使用状況ログ内の正常でないパターンや異常なパターンは、潜在的なセキュリティー違反または悪意のあるアクティビティーを示している可能性があります。このログを監視すると、セキュリティーインシデントをより効果的に検出して対応することができます。
- **傾向分析**: 使用状況ログは、レジストリーがどのように使用されているかに関する経時的な傾向と詳細情報を提供します。これは、リソースの割り当て、アクセス制御、イメージ管理戦略について、情報に基づいた意思決定を行うのに役立ちます。

ログファイルにアクセスする方法は、以下のように複数あります。

- Web UI によりログを閲覧する
- 外部に保存できるようにログをエクスポートする
- API を使用してログエントリーへのアクセスする

ログにアクセスするには、選択したリポジトリーまたは名前空間の管理者権限が必要です。



注記

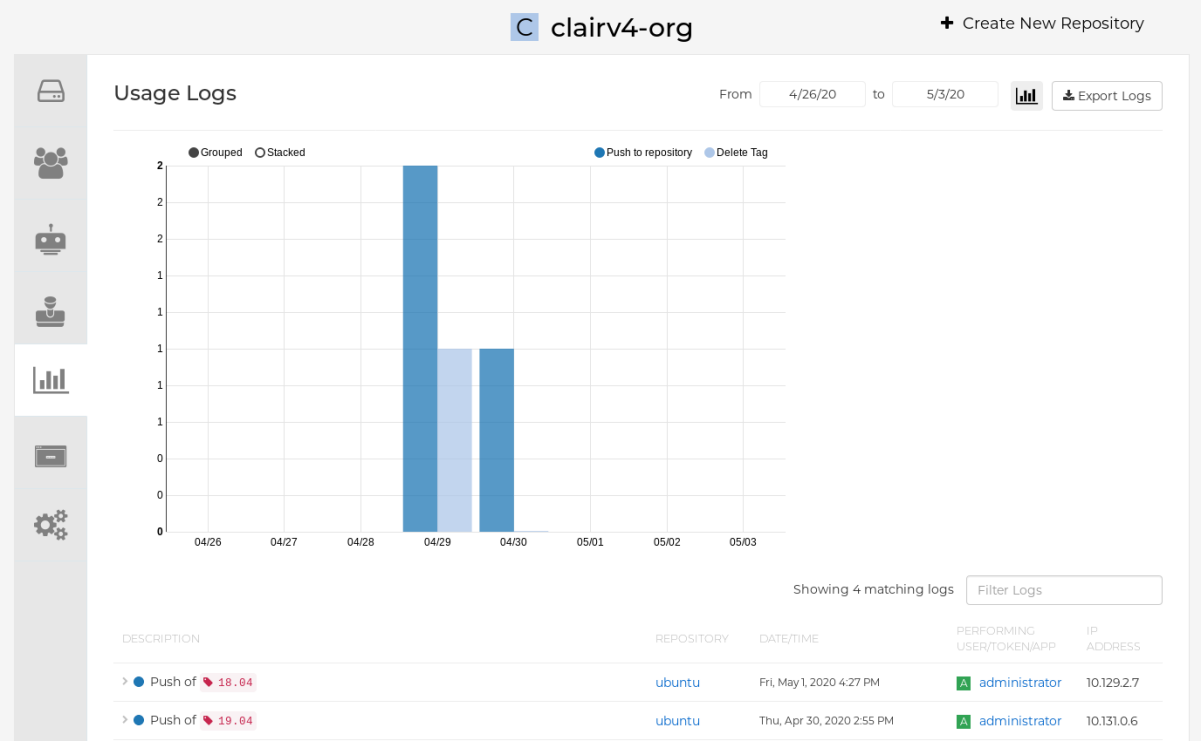
API を介して一度に利用できるログ結果は最大 100 件です。それ以上の結果を集めるには、この章で紹介するログエクスポーター機能を使う必要があります。

5.1. UI を使用したログの表示

Web UI を使用してリポジトリーまたは名前空間のログエントリーを表示するには、次の手順を実行します。

手順

1. 自分が管理者であるリポジトリーまたは名前空間に移動します。
2. ナビゲーションペインで、**Usage Logs** を選択します。



3. オプション: Usage Logs ページで以下を行います。
   a. **From** ボックスと **to** ボックスに日付を追加して、ログエントリーを表示する日付範囲を設定します。デフォルトでは、UI には最新週のログエントリーが表示されます。
   b. **Filter Logs** ボックスに文字列を入力すると、指定したキーワードのログエントリーが表示されます。たとえば、**delete** と入力してログをフィルタリングし、削除されたタグを表示できます。
   c. **Description** で、ログエントリーの矢印を切り替えると、特定のログエントリーに関連するテキストが表示されます。

5.2. リポジトリーログのエクスポート

Export Logs 機能を使用すると、より多くのログファイルを取得し、Red Hat Quay データベースの外部に保存できます。この機能には次の利点と制約があります。

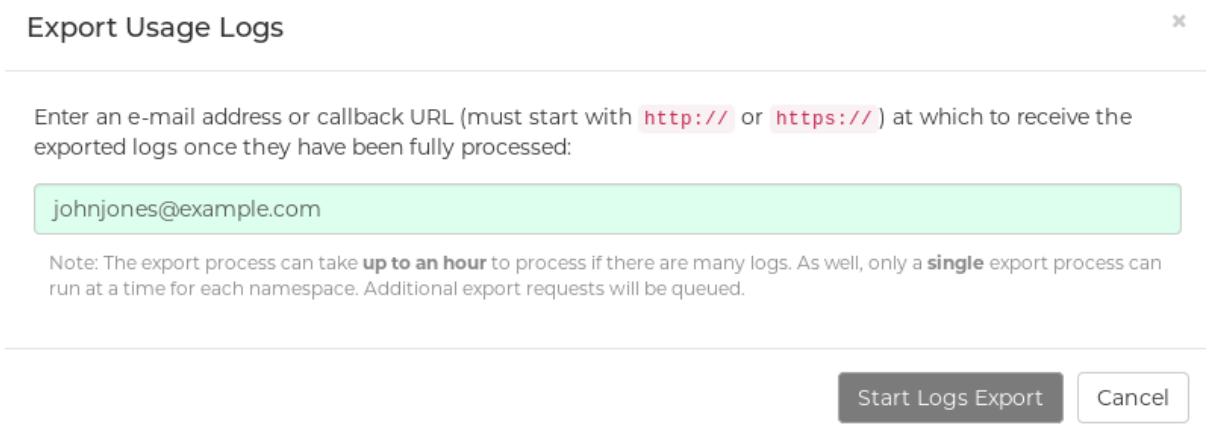
- リポジトリーから収集するログの日付の範囲を選択できます。
- ログをメールの添付ファイルで送信するか、コールバック URL に移動することを要求できます。
- ログをエクスポートするには、リポジトリーまたは名前空間の管理者である必要があります。
- すべてのユーザーのログが 30 日分保持されます。

- Export Logs 機能は、過去に生成されたログデータのみを収集します。ログ取得中のデータのストリーミングは行いません。
- この機能を使用するには、Red Hat Quay インスタンスを外部ストレージ用に設定する必要があります。ローカルストレージはログのエクスポートには使用できません。
- ログが収集されて利用可能になったら、そのデータを保存する場合は、すぐにコピーする必要があります。デフォルトでは、データは 1 時間後に期限切れになります。

ログをエクスポートするには、次の手順を実行します。

手順

1. 管理者権限のあるリポジトリーを選択します。
2. ナビゲーションペインで、**Usage Logs** を選択します。
3. オプション: 特定の日付を指定する場合は、**From** ボックスと **to** ボックスに範囲を入力します。
4. **Export Logs** ボタンをクリックします。以下のような Export Usage Logs のポップアップが表示されます。



5. エクスポートされたログを受信するメールアドレスまたはコールバック URL を入力します。コールバック URL には、指定ドメインへの URL (例: <webhook.site>) を使用できます。
6. **Start Logs Export** を選択して、選択したログエントリーを収集するプロセスを開始します。収集されるログデータの量に応じて、これが完了するまでに数分から数時間かかる場合があります。
7. ログのエクスポートが完了すると、次の 2 つのイベントのいずれかが発生します。
   - 要求したエクスポート済みログエントリーが利用可能であることを通知するメールが受信されます。
   - webhook URL からのログエクスポート要求の成功ステータスが返されます。さらに、エクスポートされたデータへのリンクを使用して、ログをダウンロードできるようになります。

注記

URL は Red Hat Quay 外部ストレージ内の場所を指しており、1時間以内に期限切れになるように設定されています。ログを保持する場合は、エクスポートされたログを有効期限が切れる前に必ずコピーしてください。

第6章 ビルドワーカーを使用した DOCKERFILE の自動ビルド

Red Hat Quay は、OpenShift または Kubernetes 上のワーカーノードのセットを使用した Dockerfile のビルドをサポートしています。GitHub webhook などのビルドトリガーを設定することで、新しいコードがコミットされたときに自動的に新しいバージョンのリポジトリーを構築できます。

このドキュメントでは、Red Hat Quay インストールでビルドを有効にし、Red Hat Quay からのビルドを受け入れるように OpenShift Container Platform または Kubernetes クラスターをもう 1 つセットアップする方法を説明します。

6.1. OPENSHIFT CONTAINER PLATFORM を使用した RED HAT QUAY ビルダーのセットアップ

Red Hat Quay ビルダーを OpenShift Container Platform で使用する前に、事前設定する必要があります。

6.1.1. OpenShift Container Platform TLS コンポーネントの設定

tls コンポーネントを使用すると、TLS 設定を制御できます。

注記

TLS コンポーネントが Red Hat Quay Operator によって管理されている場合、Red Hat Quay は Builders をサポートしません。

tls を **unmanaged** に設定する場合は、独自の **ssl.cert** ファイルと **ssl.key** ファイルを提供します。このとき、クラスターで Builder をサポートする場合は、**Quay** ルート名と Builder ルート名の両方を証明書の SAN リストに追加する必要があります。あるいは、ワイルドカードを使用することもできます。

ビルダールートを追加するには、次の形式を使用します。

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

6.1.2. Red Hat Quay ビルダー用の OpenShift Container Platform の準備

次の手順を使用して、OpenShift Container Platform 用に Red Hat Quay ビルダーを準備します。

前提条件

- OpenShift Container Platform TLS コンポーネントが設定されている。

手順

1. 次のコマンドを入力して、ビルドを実行するプロジェクト (**builder** など) を作成します。

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

2. 次のコマンドを入力して、**builder** namespace に新しい **ServiceAccount** を作成します。

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

3. 次のコマンドを入力して、**builder** namespace 内の **edit** ロールをユーザーに付与します。

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

4. 次のコマンドを入力して、**builder** namespace の **quay-builder** サービスアカウントに関連付けられたトークンを取得します。このトークンは、OpenShift Container Platform クラスターの API サーバーの認証と対話に使用されます。

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

5. OpenShift Container Platform クラスターの API サーバーの URL を特定します。これは、OpenShift Container Platform Web コンソールで確認できます。

6. ビルド **jobs** をスケジュールするときに使用するワーカーノードラベルを特定します。ビルド pods はベアメタルのワーカーノードで実行する必要があるため、通常、これらは特定のラベルで識別されます。
どのノードラベルを使用すべきかは、クラスター管理者に確認してください。

7. オプション: クラスターが自己署名証明書を使用している場合は、Kube API Server の認証局 (CA) を取得して Red Hat Quay の追加証明書に追加する必要があります。

   a. 次のコマンドを入力して、CA を含むシークレットの名前を取得します。

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

   b. OpenShift Container Platform Web コンソールで、シークレットから **ca.crt** キーの値を取得します。値は **"-----BEGIN CERTIFICATE-----"`** で始まります。

   c. CA を Red Hat Quay にインポートします。このファイルの名前が **K8S_API_TLS_CA** と一致することを確認します。

8. **ServiceAccount** に対して次の **SecurityContextConstraints** リソースを作成します。

```
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. Red Hat Quay ビルダーの設定

Red Hat Quay ビルダーを有効にするには、次の手順を実行します。

手順

1. Red Hat Quay の **config.yaml** ファイルでビルドが有効になっていることを確認します。次に例を示します。

```
FEATURE_BUILD_SUPPORT: True
```

2. 以下の情報を Red Hat Quay の **config.yaml** ファイルに追加し、各値をお客様のインストールに関連する情報に置き換えます。

```
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
```

各設定フィールドの詳細は、を参照してください。

6.2. OPENSHIFT CONTAINER PLATFORM のルート の制限事項

OpenShift Container Platform 上で Red Hat Quay Operator をマネージド **route** コンポーネントとともに使用する場合、次の制限が適用されます。

- 現在、OpenShift Container Platform の **ルート** は、単一ポートへのトラフィックのみを処理できます。Red Hat Quay ビルドをセットアップするには追加の手順が必要です。
- Red Hat Quay Operator がインストールされているクラスターで **kubectl** または **oc** CLI ツールが動作するように設定されていること、および **QuayRegistry** が存在することを確認します。**QuayRegistry** は、**Builders** が実行されるのと同じベアメタルクラスター上にある必要はありません。
- こちらの手順 に従って、OpenShift クラスター上で HTTP/2 ingress が有効になっていることを確認します。
- Red Hat Quay Operator は、既存の **Quay** Pod 内で実行されているビルドマネージャーサーバーに gRPC トラフィックを送信する **Route** リソースを作成します。カスタムホスト名、または **<builder-registry.example.com>** などのサブドメインを使用する場合は、作成した **Route** リソースの **status.ingress[0].host** を参照する DNS プロバイダーで CNAME レコードを作成してください。以下に例を示します。

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

- OpenShift Container Platform UI または CLI を使用して、**QuayRegistry** の **spec.configBundleSecret** によって参照される **Secret** をビルドクラスター CA 証明書で更新します。キーに **extra_ca_cert_build_cluster.cert** という名前を付けます。Red Hat Quay ビルダーの設定時に作成した Builder 設定で参照されている正しい値で **config.yaml** ファイルのエントリーを更新し、**BUILDMAN_HOSTNAME** CONFIGURATION FIELD を追加します。

```
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. ビルドジョブがビルドマネージャーとの通信に使用する、外部からアクセス可能なサーバーのホスト名です。デフォルトは **SERVER_HOSTNAME** と同じです。OpenShift **Route** の場合は **status.ingress[0].host**、カスタムホスト名を使用している場合は CNAME エントリーのいずれかになります。**BUILDMAN_HOSTNAME** にはポート番号を含める必要があります (たとえば、OpenShift Container Platform Route の場合は **somehost:443**)。これを省略すると、ビルドマネージャーとの通信に使用される gRPC クライアントがポートを推測しません。

6.3. ビルドのトラブルシューティング

ビルドマネージャーが起動した **ビルダー** インスタンスは、一時的なものです。これは、タイムアウトまたは失敗時に Red Hat Quay によってシャットダウンされるか、コントロールプレーン (EC2/K8s) によってガベージコレクションされることを意味します。ビルドログを取得するには、ビルドの実行中に取得する必要があります。

6.3.1. DEBUG 設定フラグ

DEBUG フラグを **true** に設定すると、完了または失敗後にビルダーのインスタンスがクリーンアップされるのを防ぐことができます。以下に例を示します。

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

true に設定すると、デバッグ機能により、**quay-builder** サービスの完了または失敗後にビルドノードがシャットダウンされなくなります。また、ビルドマネージャーが EC2 インスタンスを終了したり、Kubernetes ジョブを削除したりしてインスタンスをクリーンアップすることもなくなります。これにより、ビルダーノードの問題をデバッグできるようになります。

デバッグは実稼働サイクルでは設定しないでください。設定しても、ライフタイムサービスが残ります。そのため、たとえばインスタンスが約 2 時間後にシャットダウンします。これが発生すると、EC2 インスタンスが終了し、Kubernetes ジョブが完了します。

デバッグを有効にすると、終了していないインスタンスとジョブも実行中のワーカーの総数にカウントされるため、**ALLOWED_WORKER_COUNT** にも影響します。そのため、**ALLOWED_WORKER_COUNT** に達した場合、新しいビルドをスケジュールできるようにするには、既存のビルダーワーカーを手動で削除する必要があります。

DEBUG を設定すると、終了していないインスタンス/ジョブも実行中のワーカーの総数にカウントされるため、**ALLOWED_WORKER_COUNT** にも影響します。このため、新しいビルドをスケジュールして、**ALLOWED_WORKER_COUNT** に達した場合は、既存のビルダーワーカーを手動で削除する必要があります。

6.3.2. OpenShift Container Platform および Kubernetes ビルドのトラブルシューティング

OpenShift Container Platform Kubernetes ビルドのトラブルシューティングを行うには、次の手順を実行します。

手順

1. 次のコマンドを入力して、ローカルマシンと OpenShift Container Platform クラスターまたは Kubernetes クラスターのいずれかで実行されている Pod の間にポート転送トンネルを作成します。

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

2. 指定した SSH 鍵とポートを使用して、リモートホストへの SSH 接続を確立します。次に例を示します。

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

3. 次のコマンドを入力して、**quay-builder** サービスログを取得します。

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

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

6.4. GITHUB ビルドの設定

Github または Github Enterprise へのプッシュでビルドを行う場合は、**GitHub での OAuth アプリケーションの作成** に進みます。

第7章 コンテナーイメージのビルド

コンテナーイメージをビルドするには、コンテナー化されたアプリケーションのブループリントを作成する必要があります。ブループリントは、アプリケーションのインストール方法と設定方法を定義する他のパブリックリポジトリーのベースイメージに依存します。

Red Hat Quay は、Docker および Podman コンテナーイメージをビルドする機能をサポートしています。この機能は、コンテナーとコンテナーオーケストレーションを利用する開発者や組織に役立ちます。

7.1. ビルドコンテキスト

Docker または Podman でイメージをビルドする際には、**ビルドコンテキスト** となるディレクトリーを指定します。これは手動ビルドとビルドトリガーの両方に当てはまります。Red Hat Quay によって作成されるビルドは、ローカルマシン上で **docker build** または **podman build** を実行することと変わらないためです。

Red Hat Quay のビルドコンテキストは、常にビルドセットアップから指定された **サブディレクトリー** であり、ディレクトリーが指定されていない場合はビルドソースのルートにフォールバックします。

ビルドがトリガーされると、Red Hat Quay のビルドワーカーは Git リポジトリーをワーカーマシンにクローンし、ビルドを行う前にビルドコンテキストに入ります。

.tar アーカイブをベースにしたビルドでは、ビルドワーカーがアーカイブを抽出し、ビルドコンテキストに入ります。以下に例を示します。

展開されたビルドアーカイブ

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

上記の **展開されたビルドアーカイブ** は、**example** という Github リポジトリーのディレクトリー構造を持っていると考えてみてください。ビルドトリガーの設定でサブディレクトリーが指定されていない場合、またはビルドを手動で開始する場合、ビルドは example ディレクトリーで行われます。

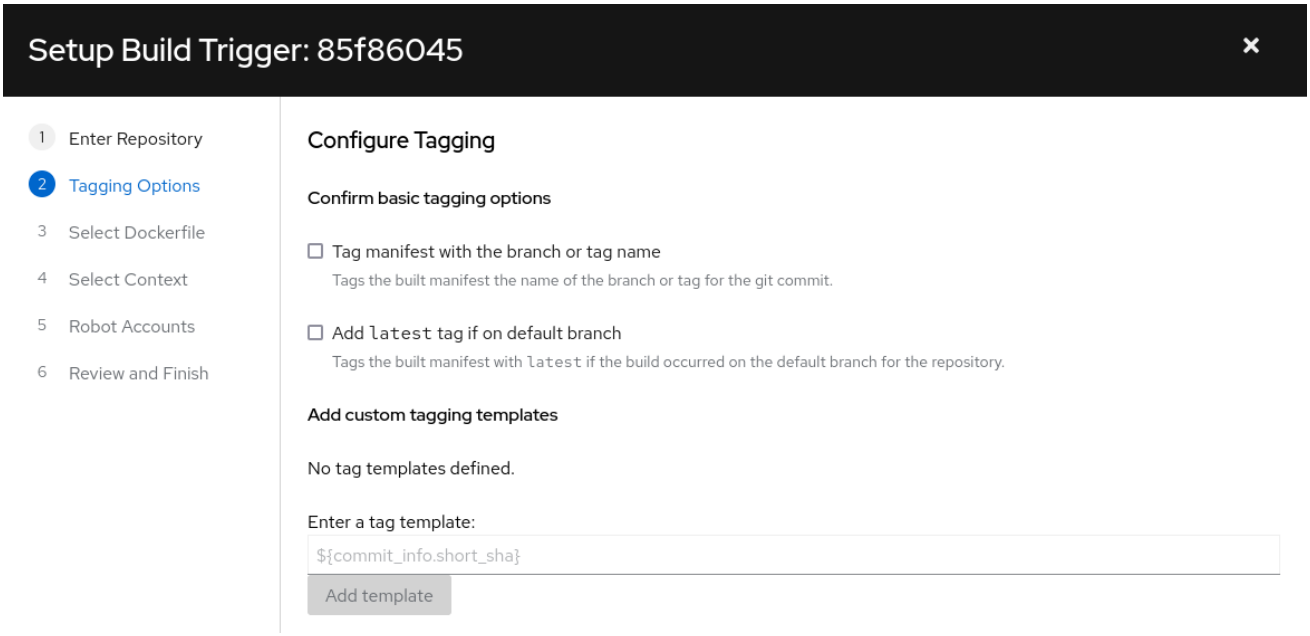
ビルドトリガーの設定でサブディレクトリー (**subdir** など) を指定した場合は、その中の Dockerfile のみがビルドの対象になります。つまり、Dockerfile の **ADD** コマンドを使用して **file** を追加することは、ビルドコンテキストの外にあるためできません。

Docker Hub とは異なり、Dockerfile は Red Hat Quay のビルドコンテキストの一部です。そのため、Dockerfile を **.dockerignore** ファイル内に含めることはできません。

7.2. ビルドトリガーのタグ命名

Red Hat Quay ではカスタムタグを使用できます。

1 つの方法として、ビルドした各イメージにタグとして割り当てる文字列を含める方法があります。または、ビルドトリガーの **Configure Tagging** セクションで次のタグテンプレートを使用して、各コミットからの情報でイメージにタグ付けすることもできます。



- **${commit}**: 発行されたコミットの完全な SHA
- **${parsed_ref.branch}**: ブランチ情報 (利用可能な場合)
- **${parsed_ref.tag}**: タグ情報 (利用可能な場合)
- **${parsed_ref.remote}**: リモート名
- **${commit_info.date}**: コミットが発行された日付
- **${commit_info.author.username}**: コミットの作成者のユーザー名
- **${commit_info.short_sha}**: コミット SHA の最初の 7 文字
- **${committer.properties.username}**: コミッターのユーザー名

以上がすべてではありませんが、これらはタグ付けに最も役立つタグテンプレートです。完全なタグテンプレートスキーマは、こちらのページ を参照してください。

詳細は、Set up custom tag templates in build triggers for Red Hat Quay and Quay.io を参照してください。

7.3. ソースコントロールをトリガーとしたビルドのスキップ

Red Hat Quay ビルドシステムがコミットを無視するように指定するには、コミットメッセージの任意の場所に **[skip build]** または **[build skip]** というテキストを追加します。

7.4. ビルドの表示および管理

リポジトリービルドは Red Hat Quay UI で表示および管理できます。

手順

1. UI を使用して Red Hat Quay リポジトリーに移動します。
2. ナビゲーションペインで、**Builds** を選択します。

7.5. 新しいビルドの作成

Red Hat Quay は、**config.yaml** ファイルで **FEATURE_BUILD_SUPPORT** が **true** に設定されている限り、新しいビルドを作成できます。

前提条件

- リポジトリーの **Builds** ページに移動している。
- **config.yaml** ファイルで **FEATURE_BUILD_SUPPORT** が **true** に設定されている。

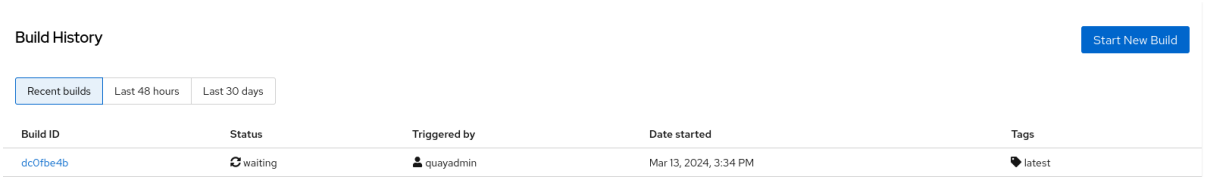
手順

1. **Builds** ページで、**Start New Build** をクリックします。
2. プロンプトが表示されたら、**Upload Dockerfile** をクリックして、ルートディレクトリーに Dockerfile または Dockerfile を含むアーカイブをアップロードします。
3. **Start Build** をクリックします。

注記

- 現在、ユーザーは手動でビルドを開始するときに Docker ビルドコンテキストを指定できません。
- 現在、BitBucket は Red Hat Quay v2 UI ではサポートされていません。

4. ビルドにリダイレクトされます。ビルドはリアルタイムで確認できます。Dockerfile ビルドが完了してプッシュされるまで待ちます。
5. オプション: **Download Logs** をクリックしてログをダウンロードしたり、**Copy Logs** をクリックしてログをコピーしたりできます。
6. 戻るボタンをクリックして **Repository Builds** ページに戻り、ビルド履歴を表示できます。



7.6. ビルドトリガー

ビルドトリガーは、ソースコントロールのプッシュ、webhook 呼び出しの作成 など、トリガー条件が満たされるとビルドを呼び出します。

7.6.1. ビルドトリガーの作成

次の手順に従って、カスタム Git リポジトリーを使用してビルドトリガーを作成します。

注記

以下の手順は、**config.yaml** ファイルに Github 認証情報が含まれていないことを前提としています。

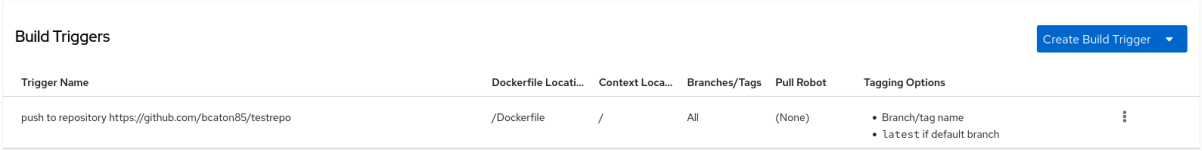
前提条件

- リポジトリーの **Builds** ページに移動している。

手順

1. **Builds** ページで、**Create Build Trigger**をクリックします。
2. Github、BitBucket、Gitlab などの目的のプラットフォームを選択するか、カスタム Git リポジトリーを使用します。この例では、Github のカスタム Git リポジトリーを使用しています。
3. カスタム Git リポジトリー名を入力します (例: **git@github.com:<username>/<repo>.git**)。**Next** をクリックします。
4. プロンプトが表示されたら、次のオプションのいずれかまたは両方を選択して、タグ付けオプションを設定します。
   - **Tag manifest with the branch or tag name**。このオプションを選択すると、ビルドされたマニフェストにブランチの名前または git コミットのタグがタグ付けされます。
   - **Add latest tag if on default branch**。このオプションを選択すると、リポジトリーのデフォルトブランチでビルドが行われた場合、ビルドされたマニフェストに latest がタグ付けされます。
     オプションで、カスタムのタグ付けテンプレートを追加できます。ここに入力できるタグテンプレートは複数あります。短い SHA ID、タイムスタンプ、作成者名、コミッター、コミットからのブランチ名をタグとして使用することもできます。詳細は、「ビルドトリガーのタグ命名」を参照してください。

     タグ付けを設定したら、**Next** をクリックします。
5. プロンプトが表示されたら、トリガーの呼び出し時にビルドする Dockerfile の場所を選択します。Dockerfile が git リポジトリーのルートにあり、Dockerfile という名前が付けられている場合は、Dockerfile パスとして **/Dockerfile** を入力します。**Next** をクリックします。
6. プロンプトが表示されたら、Docker ビルドのコンテキストを選択します。Dockerfile が Git リポジトリーのルートにある場合は、ビルドコンテキストディレクトリーとして / を入力します。**Next** をクリックします。
7. オプション: 任意のロボットアカウントを選択します。これにより、ビルドプロセス中にプライベートのベースイメージをプルできます。プライベートベースイメージが使用されていないことを把握している場合は、この手順を省略できます。
8. **Next** をクリックします。検証の警告がないか確認します。必要に応じて、**Finish** をクリックする前に問題を修正します。
9. トリガーが正常にアクティベートされたという警告が表示されます。このトリガーを使用するには、以下のアクションが必要になることに注意してください。
   - 以下の公開鍵に git リポジトリーへの読み取りアクセス権を与える必要があります。
   - ビルドをトリガーするには、リポジトリーを **POST** に設定する必要があります。
     SSH 公開鍵を保存し、**Return to <organization_name>/<repository_name>**をクリックします。リポジトリーの **Builds** ページにリダイレクトされます。
10. **Builds** ページに、ビルドトリガーが表示されます。以下に例を示します。



7.6.2. ビルドの手動トリガー

次の手順を使用して、ビルドを手動でトリガーできます。

手順

1. **Builds** ページで、**Start new build** をクリックします。
2. プロンプトが表示されたら、**Invoke Build Trigger** を選択します。
3. **Run Trigger Now** をクリックして、プロセスを手動で開始します。
   ビルドが開始したら、**Repository Builds** ページでビルド ID を確認できます。

7.7. カスタム GIT トリガーの設定

カスタム Git トリガー は、Git サーバーをビルドトリガーとして機能させるための一般的な方法です。カスタム Git トリガーは SSH 鍵と webhook エンドポイントのみに依存します。それ以外の実装はユーザーに委ねられています。

7.7.1. トリガーの作成

カスタム Git トリガーの作成は、他のトリガーの作成と似ていますが、次の点が違います。

- Red Hat Quay は、トリガーで使用する適切なロボットアカウントを自動的に検出することはできません。これは、作成時に手動で行う必要があります。
- トリガーの作成後に追加の手順を実施する必要があります。この手順については、次のセクションで詳しく説明します。

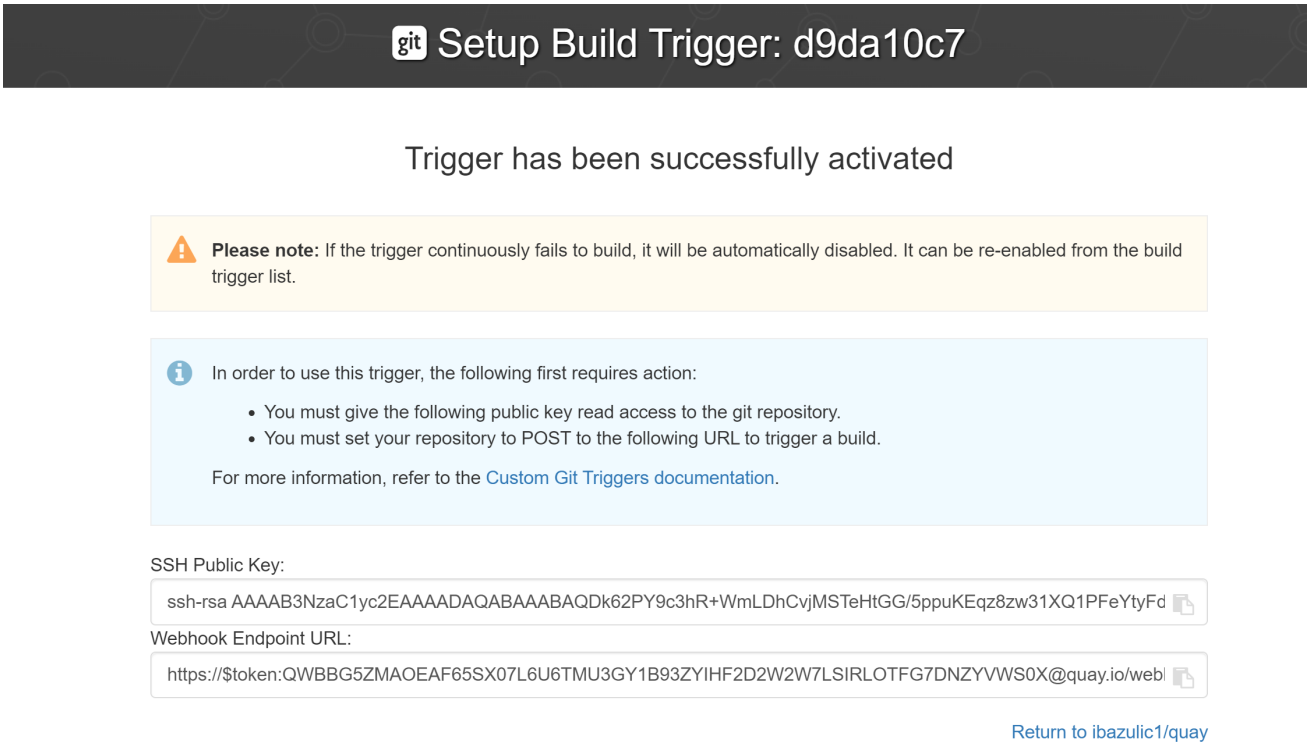
7.7.2. カスタムトリガー作成の設定

カスタム Git トリガーを作成する場合は、次の 2 つの追加手順が必要です。

1. トリガーの作成時に生成された SSH 公開鍵への読み取りアクセスを付与する必要があります。
2. ビルドをトリガーする Red Hat Quay のエンドポイントに POST する webhook をセットアップする必要があります。

鍵と URL は、**Settings** または **歯車** アイコンから **View Credentials** を選択することで利用できます。

リポジトリーからのタグの表示および変更



7.7.2.1. SSH 公開鍵へのアクセス

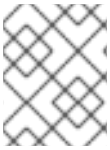
Git サーバーの設定に応じて、Red Hat Quay がカスタム Git トリガー用に生成する SSH 公開鍵をインストールする方法はさまざまです。

たとえば、Git のドキュメント では、小規模なサーバーのセットアップを説明しています。この場合は、鍵を **$HOME/.ssh/authorize_keys** に追加すると、ビルダーがリポジトリーをクローンするためのアクセス権が付与されます。公式にサポートされていない git リポジトリー管理ソフトウェアの場合は、通常、**Deploy Keys** というラベルが付いた、鍵を入力する場所があります。

7.7.2.2. Webhook

ビルドを自動的にトリガーするには、次の形式を使用して **.json** ペイロードを webhook URL に **POST** する必要があります。

これはサーバーの設定に応じてさまざまな方法で行うことができますが、ほとんどの場合は **post-receive** Git フック で行うことができます。



注記

このリクエストが有効であるためには、**application/json** を含む **Content-Type** ヘッダーが必要です。

webhook の例

```
{
  "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 GITHUB での OAUTH アプリケーションの作成

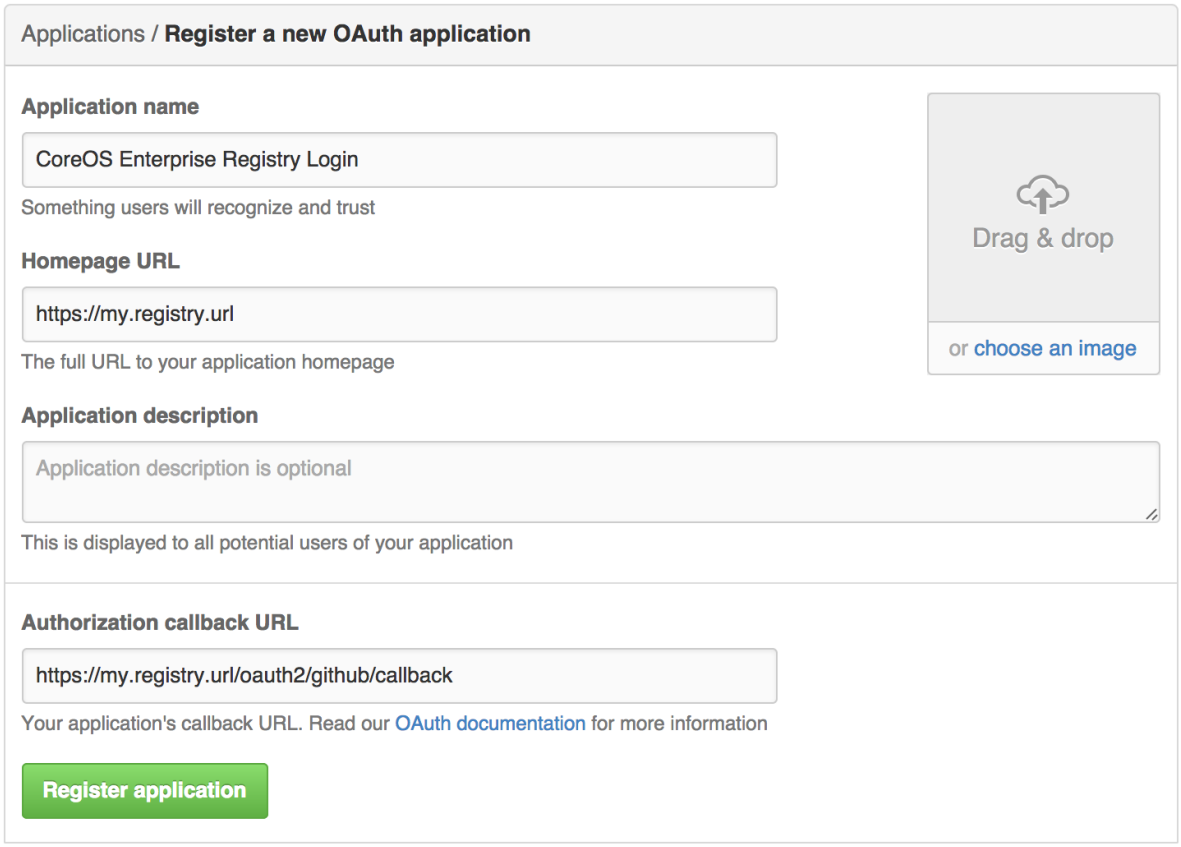
Red Hat Quay レジストリーを GitHub OAuth アプリケーションとして登録することで、GitHub アカウントとそのリポジトリーへのアクセスを許可できます。

8.1. GITHUB アプリケーションの新規作成

Github で OAuth アプリケーションを作成するには、次の手順を実行します。

手順

1. Github Enterprise にログインします。
2. ナビゲーションペインで、ユーザー名 → **Your organizations** を選択します。
3. ナビゲーションペインで、**Applications** を選択します。
4. Register New Application をクリックします。**Register a new OAuth application** 設定画面が表示されます。次に例を示します。



5. **Application name** テキストボックスにアプリケーションの名前を入力します。
6. **Homepage URL** テキストボックスに、Red Hat Quay URL を入力します。

注記

公開されている GitHub を使用する場合、入力するホームページの URL は、ユーザーがアクセスできるものである必要があります。その URL が内部用のままである可能性があります。

7. **Authorization callback URL** に、https://<RED_HAT_QUAY_URL>/oauth2/github/callback と入力します。

8. **Register application** をクリックして設定を保存します。

9. 新しいアプリケーションの概要が表示されたら、新しいアプリケーションに対して表示されたクライアント ID とクライアントシークレットを記録します。

第9章 リポジトリー通知

Red Hat Quay では、リポジトリーのライフサイクルで発生するさまざまなイベントの **通知** をリポジトリーに追加できます。

9.1. 通知の作成

通知を追加するには、次の手順を実行します。

前提条件

- リポジトリーを作成している。
- リポジトリーの管理者権限がある。

手順

Red Hat Quay のリポジトリーに移動します。

1. ナビゲーションペインで、**Settings** をクリックします。
2. **Events and Notifications** カテゴリーで、**Create Notification** をクリックして、リポジトリーイベントの新しい通知を追加します。**Create repository notification** ページにリダイレクトされます。
3. **Create repository notification** ページで、ドロップダウンメニューを選択してイベントのリストを表示します。次のタイプのイベントの通知を選択できます。
   - リポジトリーへのプッシュ
   - Dockerfile ビルドのキューへの追加
   - Dockerfile ビルドの開始
   - Dockerfile ビルドの正常な完了
   - Docker ビルドのキャンセル
   - パッケージの脆弱性の検出
4. イベントの種類を選択したら、通知方法を選択します。次の方法を使用できます。
   - Quay 通知
   - メール
   - Webhook POST
   - Flowdock チーム通知
   - HipChat ルーム通知
   - Slack ルーム通知
     選択した方法に応じて、追加情報を入力する必要があります。たとえば、**E-mail** を選択した場合は、メールアドレスと通知タイトル (省略可能) を入力する必要があります。
5. イベントと通知方法を選択したら、**Create Notification** をクリックします。

9.2. リポジトリーイベントの説明

次のセクションでは、リポジトリーイベントについて詳しく説明します。

9.2.1. リポジトリープッシュ

1 つまたは複数のイメージのリポジトリーへのプッシュが成功しました。

```
{
  "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 ビルドのキューへの追加

次の例は、ビルドシステムのキューに追加された Dockerfile ビルドからの応答です。

注記

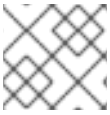
応答は、オプションの属性の使用によって異なる場合があります。

```
{
  "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 ビルドの開始

次の例は、ビルドシステムのキューに追加された Dockerfile ビルドからの応答です。



注記

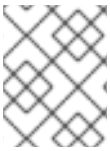
応答は、オプションの属性の使用によって異なる場合があります。

```
{
  "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 ビルドの正常な完了

次の例は、ビルドシステムによって正常に完了した Dockerfile ビルドからの応答です。



注記

このイベントは、ビルドされたイメージの **リポジトリープッシュ** イベントと同時に発生します。

```
{
  "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@sha256:2a7af5265344cc3704d5d47c4604b1efcbd227a7a6a6ff73d6e4e08a27fd7d99",

"quay.io/dgangaia/test@sha256:569e7db1a867069835e8e97d50c96eccafde65f08ea3e0d5debaf16e2545d9d1"
  ]
}
```

9.2.5. Dockerfile ビルドの失敗

次の例は、失敗した Dockerfile ビルドからの応答です。

```
{
  "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 ビルドのキャンセル

次の例は、キャンセルされた Dockerfile ビルドからの応答です。


```
{
  "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. 脆弱性の検出

次の例は、リポジトリーで脆弱性を検出した Dockerfile ビルドからの応答です。


```
{
  "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. 通知アクション

9.3.1. 通知の追加

通知は **Events and Notifications** ページの **Repository Settings** セクションに追加されます。通知は **Notifications** ウィンドウにも追加されます。このウィンドウは Red Hat Quay のナビゲーションペインにある **ベル** アイコンをクリックすると表示されます。

Red Hat Quay の通知は、**ユーザー**、**チーム**、または**組織** 全体に送信するように設定できます。

9.3.2. メール通知

指定のイベントを説明するメールが、指定のアドレスに送信されます。メールアドレスは **リポジトリーごとに** 認証する必要があります。

9.3.3. Webhook POST 通知

HTTP **POST** 呼び出しは、イベントのデータを使用して、指定の URL に対して行われます。イベントデータの詳細は、「リポジトリーイベントの説明」を参照してください。

URL が HTTPS の場合、呼び出しには Red Hat Quay の SSL クライアント証明書が設定されます。この証明書を検証することで、Red Hat Quay からの呼び出しが証明されます。ステータスコードが **2xx** の範囲の応答は成功とみなされます。それ以外のステータスコードを持つ応答は失敗とみなされ、webhook 通知が再試行されます。

9.3.4. Flowdock 通知

Flowdock にメッセージを投稿します。

9.3.5. Hipchat 通知

HipChat にメッセージを投稿します。

9.3.6. Slack 通知

Slack にメッセージを投稿します。

第10章 OPEN CONTAINER INITIATIVE のサポート

コンテナーレジストリーは、当初 Docker イメージ形式のコンテナーイメージをサポートするように設計されていました。Docker 以外で追加のランタイムの使用をプロモートするために、コンテナーランタイムとイメージ形式に関連する標準化を提供するために Open Container Initiative (OCI) が作成されました。ほとんどのコンテナーレジストリーは、Docker イメージマニフェスト V2、Schema 2 形式をベースとして OCI 標準化をサポートします。

コンテナーイメージのほかにも、個別のアプリケーションだけでなく、Kubernetes プラットフォームを全体としてサポートする各種のアーティファクトが新たに出現しました。これらは、アプリケーションのデプロイメントを支援するセキュリティーおよびガバナンスの Open Policy Agent (OPA) ポリシーから Helm チャートおよび Operator に及びます。

Red Hat Quay は、コンテナーイメージを格納するだけでなく、コンテナーの管理を支援するツールのエコシステム全体もサポートするプライベートコンテナーレジストリーです。Red Hat Quay は、OCI 1.0 **Image and Distribution specifications**と可能な限り互換性を保つよう努めており、**Helm チャート** (OCI をサポートする Helm のバージョンを使用してプッシュされている場合に限る) などの一般的なメディアタイプや、コンテナーイメージのマニフェストまたはレイヤーコンポーネント内部のさまざまな任意のメディアタイプをサポートしています。このような新しいメディアタイプのサポートが、以前の Red Hat Quay のバージョンと異なる点です。以前のバージョンでは、受け入れるメディアタイプについてより厳しい制限がレジストリーに設けられていました。Red Hat Quay は、以前はサポート範囲外だったメディアタイプも含め、より幅広いメディアタイプに対応できるようになりました。そのため、標準のコンテナーイメージ形式だけでなく、新しいタイプや従来とは異なるタイプにも対応できるようになり、より多用途になりました。

新しいメディアタイプへのサポート拡張に加えて、Red Hat Quay は、V2_2 および V2_1 形式を含む Docker イメージとの互換性を確保しています。このような Docker V2_2 および V2_1 イメージとの互換性は、Docker ユーザーにシームレスなエクスペリエンスを提供するという Red Hat Quay の取り組みを表すものです。さらに、Red Hat Quay は、引き続き Docker V1 プルのサポートを拡張し、このような以前のバージョンの Docker イメージを現在も利用している可能性のあるユーザーに対応します。

OCI アーティファクトのサポートはデフォルトで有効になっています。以前は、OCI メディアタイプは **FEATURE_GENERAL_OCI_SUPPORT** 設定フィールドで有効にされていました。

注記

すべての OCI メディアタイプがデフォルトで有効になったため、**FEATURE_GENERAL_OCI_SUPPORT**、**ALLOWED_OCI_ARTIFACT_TYPES**、および **IGNORE_UNKNOWN_MEDIATYPES** を使用する必要がなくなりました。

さらに、**FEATURE_HELM_OCI_SUPPORT** 設定フィールドが非推奨になりました。この設定フィールドはサポートされなくなり、Red Hat Quay の将来のバージョンでは削除される予定です。

10.1. HELM および OCI の前提条件

Helm は、アプリケーションのパッケージ化とデプロイの方法を簡素化します。Helm は、アプリケーションを表す Kubernetes リソースが含まれる **チャート** というパッケージ形式を使用します。Red Hat Quay は、OCI でサポートされているバージョンの Helm チャートをサポートします。

次の手順に従って、Helm およびその他の OCI メディアタイプを使用するようにシステムを事前設定します。

10.1.1. Helm のインストール

Helm クライアントをインストールするには、次の手順を実行します。

手順

1. Helm リリース ページから Helm の最新バージョンをダウンロードします。
2. 次のコマンドを入力して Helm バイナリーを解凍します。

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

3. Helm バイナリーを目的の場所に移動します。

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

Helm のインストールの詳細は、Helm のインストール ドキュメントを参照してください。

10.1.2. Helm 3.8 へのアップグレード

OCI レジストリーチャートをサポートするには、Helm が少なくとも 3.8 にアップグレードされている必要があります。すでに Helm をダウンロードしていて、Helm 3.8 にアップグレードする必要がある場合は、Helm アップグレード のドキュメントを参照してください。

10.1.3. Red Hat Quay で使用される SSL/TLS 証明書をシステムが信頼できるようにする

Helm クライアントと Red Hat Quay の間の通信は、HTTPS 経由で促進されます。Helm 3.5 では、信頼できる証明書を使用して HTTPS 経由で通信するレジストリーのみがサポートされています。さらに、オペレーティングシステムはレジストリーで公開される証明書を信頼する必要があります。Red Hat Quay で使用される証明書を信頼するようにオペレーティングシステムが設定されていることを確認する必要があります。システムがカスタム証明書を信頼できるようにするには、次の手順を実行します。

手順

1. 次のコマンドを入力して、**rootCA.pem** ファイルを **/etc/pki/ca-trust/source/anchors/** フォルダーにコピーします。

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

2. 次のコマンドを入力して、CA トラストストアを更新します。

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

10.2. HELM チャートの使用

以下の例を使用して、Red Hat Community of Practice (CoP) リポジトリーから etherpad チャートをダウンロードしてプッシュします。

前提条件

- Red Hat Quay にログインしている。

手順

1. 次のコマンドを入力して、チャートリポジトリーを追加します。

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

2. 次のコマンドを入力して、チャートリポジトリーから、ローカルで使用可能なチャートの情報を更新します。

```
$ helm repo update
```

3. 次のコマンドを入力して、リポジトリーからチャートを取得します。

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

4. 次のコマンドを入力して、チャートをチャートアーカイブにパッケージ化します。

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

出力例

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

5. **helm registry login** を使用して Red Hat Quay にログインします。

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

6. **helm push** コマンドを使用して、チャートをリポジトリーにプッシュします。

```
$ helm push etherpad-0.0.4.tgz
oci://quay370.apps.quayperf370.perfscale.devcluster.openshift.com
```

出力例:

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

7. ローカルコピーを削除してから、リポジトリーからチャートをプルして、プッシュが機能したことを確認します。

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

```
$ helm pull oci://quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad --
version 0.0.4
```

出力例:

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

10.3. COSIGN OCI のサポート

Cosign は、コンテナーイメージの署名および検証に使用できるツールです。**ECDSA-P256** 署名アルゴリズムおよび Red Hat の Simple Signing ペイロード形式を使用して、PKIX ファイルに保存される公開鍵を作成します。秘密鍵は暗号化された PEM ファイルとして保存されます。

Cosign は現在、以下をサポートしています。

- ハードウェアおよび KMS の署名
- 自身の PKI を使用
- OIDC PKI
- 組み込みのバイナリー透過性およびタイムスタンプサービス

Cosign を直接インストールするには、次の手順を実行します。

前提条件

- Go バージョン 1.16 以降がインストールされている。
- **config.yaml** ファイルで **FEATURE_GENERAL_OCI_SUPPORT** を **true** に設定している。

手順

1. 次の **go** コマンドを入力して、Cosign を直接インストールします。

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

出力例

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

2. 次のコマンドを入力して、Cosign 用のキーと値のペアを生成します。

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

出力例

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

3. 次のコマンドを入力して、キーと値のペアに署名します。

```
$ cosign sign -key cosign.key quay-server.example.com/user1/busybox:test
```

出力例

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

認証に **~./docker/config.json** に依存していることが原因で起こる **error: signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[]** エラーが発生した場合は、次のコマンドを実行する必要がある場合があります。

```
$ podman login --authfile ~/.docker/config.json quay-server.example.com
```

出力例

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

4. 次のコマンドを入力して、更新された認可設定を確認します。

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

10.4. COSIGN のインストールと使用

Cosign を直接インストールするには、次の手順を実行します。

前提条件

- Go バージョン 1.16 以降がインストールされている。
- **config.yaml** ファイルで **FEATURE_GENERAL_OCI_SUPPORT** を **true** に設定している。

手順

1. 次の **go** コマンドを入力して、Cosign を直接インストールします。

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

出力例

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

2. 次のコマンドを入力して、Cosign 用のキーと値のペアを生成します。

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

出力例

```
Enter password for private key:
```

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

3. 次のコマンドを入力して、キーと値のペアに署名します。

```
$ cosign sign -key cosign.key quay-server.example.com/user1/busybox:test
```

出力例

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

認証に **~./docker/config.json** に依存していることが原因で起こる **error: signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[]** エラーが発生した場合は、次のコマンドを実行する必要がある場合があります。

```
$ podman login --authfile ~/.docker/config.json quay-server.example.com
```

出力例

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

4. 次のコマンドを入力して、更新された認可設定を確認します。

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

10.5. 他のアーティファクトタイプの使用

デフォルトでは、他のアーティファクトタイプは Red Hat Quay で使用できるように有効になっています。

以下の手順で OCI メディアタイプを追加します。

前提条件

- **config.yaml** ファイルで **FEATURE_GENERAL_OCI_SUPPORT** を **true** に設定している。

手順

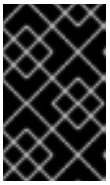
1. **config.yaml** ファイルに、**ALLOWED_OCI_ARTIFACT_TYPES** 設定フィールドを追加します。以下に例を示します。

```
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. 以下を **config.yaml** ファイルに追加して、目的のアーティファクトタイプ (Singularity Image Format (SIF) など) のサポートを追加します。

```
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

デフォルトで設定されていないアーティファクトタイプを追加する場合、Red Hat Quay 管理者は、必要に応じて cosign と Helm のサポートを手動で追加する必要もあります。

現在、ユーザーは Red Hat Quay レジストリーの SIF イメージにタグを付けることができます。

10.6. RED HAT QUAY での OCI アーティファクトの無効化

OCI アーティファクトのサポートを無効にするには、次の手順を実行します。

手順

- **config.yaml** ファイルで **FEATURE_GENERAL_OCI_SUPPORT** を **false** に設定して、OCI アーティファクトのサポートを無効にします。以下に例を示します。

```
FEATURE_GENERAL_OCI_SUPPORT = false
```

第11章 RED HAT QUAY クォータの管理と適用の概要

Red Hat Quay では、ユーザーは、設定されたストレージクォータ制限を確立することにより、ストレージ消費を報告し、レジストリーの増加を抑えることができます。オンプレミスの Red Hat Quay ユーザーには、環境の容量制限を管理するための次の機能が装備されるようになりました。

- **Quota reporting:** この機能を使用すると、スーパーユーザーはすべての組織のストレージ消費量を追跡できます。さらに、ユーザーは割り当てられた組織のストレージ消費量を追跡できます。
- **Quota management:** この機能を使用すると、スーパーユーザーは Red Hat Quay ユーザーのソフトチェックとハードチェックを定義できます。ソフトチェックは、組織のストレージ消費量が設定されたしきい値に達しているかどうかをユーザーに通知します。ハードチェックは、ストレージ消費量が設定された制限に達したときにユーザーがレジストリーにプッシュするのを防ぎます。

これらの機能を組み合わせることで、Red Hat Quay レジストリーのサービス所有者はサービスレベル契約を定義し、健全なリソース予算をサポートできるようになります。

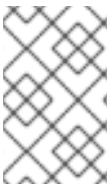
11.1. クォータ管理アーキテクチャー

クォータ管理機能を有効にすると、個々の BLOB サイズがリポジトリーレベルと名前空間レベルで合計されます。たとえば、同じリポジトリー内の 2 つのタグが同じ BLOB を参照している場合、その BLOB のサイズはリポジトリーの合計に対して 1 回だけカウントされます。さらに、マニフェストリストの合計もリポジトリーの合計にカウントされます。

重要

マニフェスト一覧の合計はリポジトリーの合計にカウントされるため、Red Hat Quay の以前のバージョンからアップグレードするときに消費される合計クォータは、Red Hat Quay 3.9 では大幅に異なる可能性があります。場合によっては、新しい合計がリポジトリーで以前に設定された制限を超える可能性があります。Red Hat Quay 管理者は、これらの変更を考慮して、リポジトリーに割り当てられたクォータを調整する必要がある場合があります。

クォータ管理機能は、バックフィルワーカーを使用して既存のリポジトリーと名前空間のサイズを計算し、その後プッシュまたはガベージコレクションされたすべてのイメージの合計を加算または減算することによって機能します。さらに、マニフェストがガベージコレクションされるときに、合計からの減算が行われます。



注記

マニフェストがガベージコレクションされるときに合計から減算が発生するため、ガベージコレクションが可能になるまでサイズの計算に遅れが生じます。ガベージコレクションの詳細は、Red Hat Quay ガベージコレクション を参照してください。

以下のデータベーステーブルには、組織内の Red Hat Quay リポジトリーのクォータリポジトリーサイズ、クォータ名前空間サイズ、およびクォータレジストリーサイズ (バイト単位) が保持されます。

- **QuotaRepositorySize**
- **QuotaNameSpaceSize**
- **QuotaRegistrySize**

組織のサイズは、重複しないようにバックフィル作業者によって計算されます。イメージプッシュが初期化されると、ユーザーの組織ストレージが検証され、設定されたクォータ制限を超えているかどうかがチェックされます。イメージプッシュが定義されたクォータ制限を超えると、ソフトチェックまたはハードチェックが発生します。

- ソフトチェックの場合は、ユーザーに通知されます。
- ハードチェックの場合は、プッシュが停止します。

ストレージ消費量が設定済みのクォータ制限内にある場合は、プッシュを続行できます。

イメージマニフェストの削除も同様のフローに従い、関連するイメージタグとマニフェストの間のリンクが削除されます。さらに、イメージマニフェストが削除された後、リポジトリーサイズが再計算され、**QuotaRepositorySize**、**QuotaNameSpaceSize**、および **QuotaRegistrySize** テーブルで更新されます。

11.2. クォータ管理の制限

クォータ管理は、組織がリソース消費を維持するのに役立ちます。クォータ管理の制限の 1 つは、プッシュでリソース消費を計算すると、計算がプッシュのクリティカルパスの一部になることです。これがないと、使用状況データがドリフトする可能性があります。

最大ストレージクォータサイズは、選択したデータベースによって異なります。

表11.1 ワーカー数の環境変数

変数	説明
Postgres	8388608 TB
MySQL	8388608 TB
SQL Server	16777216 TB

11.3. クォータ管理設定フィールド

表11.2 クォータ管理設定

フィールド	型	説明
FEATURE_QUOTA_MANAGEMENT	Boolean	クォータ管理機能の設定、キャッシュ、検証を有効にします。 **Default:** `False`
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	String	すべての組織に対してシステムのデフォルトクォータ拒否バイト許容量を有効にします。 デフォルトでは、制限は設定されていません。

フィールド	型	説明
QUOTA_BACKFILL	Boolean	クォータバックフィルワーカーが既存の BLOB のサイズを計算できるようにします。 **デフォルト**: **True**
QUOTA_TOTAL_DELAY_SECONDS	String	クォータバックフィルを開始するまでの遅延時間。ローリングデプロイメントでは、合計が不正確になる可能性があります。このフィールドは、ローリングデプロイメントが完了するまでにかかる時間よりも長い時間を設定する**必要があります**。 **デフォルト**: **1800**
PERMANENTLY_DELETE_TAGS	Boolean	タイムマシンウィンドウからのタグの削除に関連する機能を有効にします。 **デフォルト**:**False**
RESET_CHILD_MANIFEST_EXPIRATION	Boolean	子マニフェストを対象とする一時タグの有効期限をリセットします。この機能を **True** に設定すると、子マニフェストはすぐにガベージコレクションされます。 **デフォルト**:**False**

11.3.1. クォータ管理設定の例

次の YAML は、クォータ管理を有効にするときに推奨される設定です。

クォータ管理 YAML 設定

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

11.4. RED HAT QUAY API を使用したクォータの確立

組織が最初に作成されたとき、割り当ては適用されていません。**/api/v1/organization/{organization}/quota** エンドポイントを使用します。

サンプルコマンド

```
$ 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
```

出力例

```
[]
```

11.4.1. クォータの設定

組織の割り当てを設定するには、データを **/api/v1/organization/{orgname}/quota** エンドポイントに POST します。以下はコマンド例です。

```
$ 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
```

出力例

```
"Created"
```

11.4.2. クォータの表示

適用されたクォータを確認するには、**/api/v1/organization/{orgname}/quota** エンドポイントからデータを **GET** します。

サンプルコマンド

```
$ 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
```

出力例

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

11.4.3. クォータの変更

既存の割り当てを変更する (ここでは 10MB から 100MB に) には、データを **/api/v1/organization/{orgname}/quota/{quota_id}** エンドポイントに PUT します。

サンプルコマンド

```
$ 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
```

出力例

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

11.4.4. イメージのプッシュ

消費されたストレージを確認するには、さまざまなイメージを組織にプッシュします。

11.4.4.1. ubuntu:18.04 のプッシュ

コマンドラインから組織に ubuntu:18.04 をプッシュします。

サンプルコマンド

```
$ 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. API を使用してクォータの使用状況の表示

消費されたストレージを表示するには、**/api/v1/repository** エンドポイントからデータを **GET** します。

サンプルコマンド

```
$ 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&quota=true' | jq
```

出力例

```
{
  "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. 別のイメージをプッシュ

1. 2 番目のイメージをプル、タグ付け、プッシュします。たとえば、**nginx** です。

   サンプルコマンド

   ```
   $ 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. 組織内のリポジトリーのクォータレポートを表示するには、**/api/v1/repository** エンドポイントを使用します。

   サンプルコマンド

   ```
   $ 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&quota=true'
   ```

   出力例


   ```
   {
     "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. 組織の詳細でクォータ情報を表示するには、**/api/v1/organization/{orgname}** エンドポイントを使用します。

サンプルコマンド

```
$ 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
```

出力例


```
{
  "name": "testorg",
  ...
  "quotas": [
    {
      "id": 1,
      "limit_bytes": 104857600,
      "limits": []
    }
  ],
  "quota_report": {
    "quota_bytes": 87190725,
    "configured_quota": 104857600
  }
}
```


11.4.5. クォータ制限を使用してプッシュの拒否

イメージプッシュが定義されたクォータ制限を超えると、ソフトチェックまたはハードチェックが発生します。

- ソフトチェックまたは **警告** の場合は、ユーザーに通知されます。
- ハードチェックまたは **拒否** の場合、プッシュは終了します。

11.4.5.1. 拒否および警告の制限の設定

拒否 および **警告** の制限を設定するには、データを **/api/v1/organization/{orgname}/quota/{quota_id}/limit** エンドポイントに POST します。

サンプル拒否制限コマンド

```
$ 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
```

警告制限コマンドの例

```
$ 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. 拒否および警告の制限の表示

拒否 および **警告** の制限を表示するには、**/api/v1/Organization/{orgname}/quota** エンドポイントを使用します。

クォータ制限の表示

```
$  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
```

クォータ制限のサンプル出力

```
[
  {
    "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. 拒否制限を超えたときにイメージをプッシュ

この例では、拒否制限 (80%) が現在のリポジトリーサイズ (~83%) 未満に設定されているため、次のプッシュは自動的に拒否されます。

コマンドラインからサンプルイメージを組織にプッシュします。

サンプルイメージプッシュ

```
$ 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
```

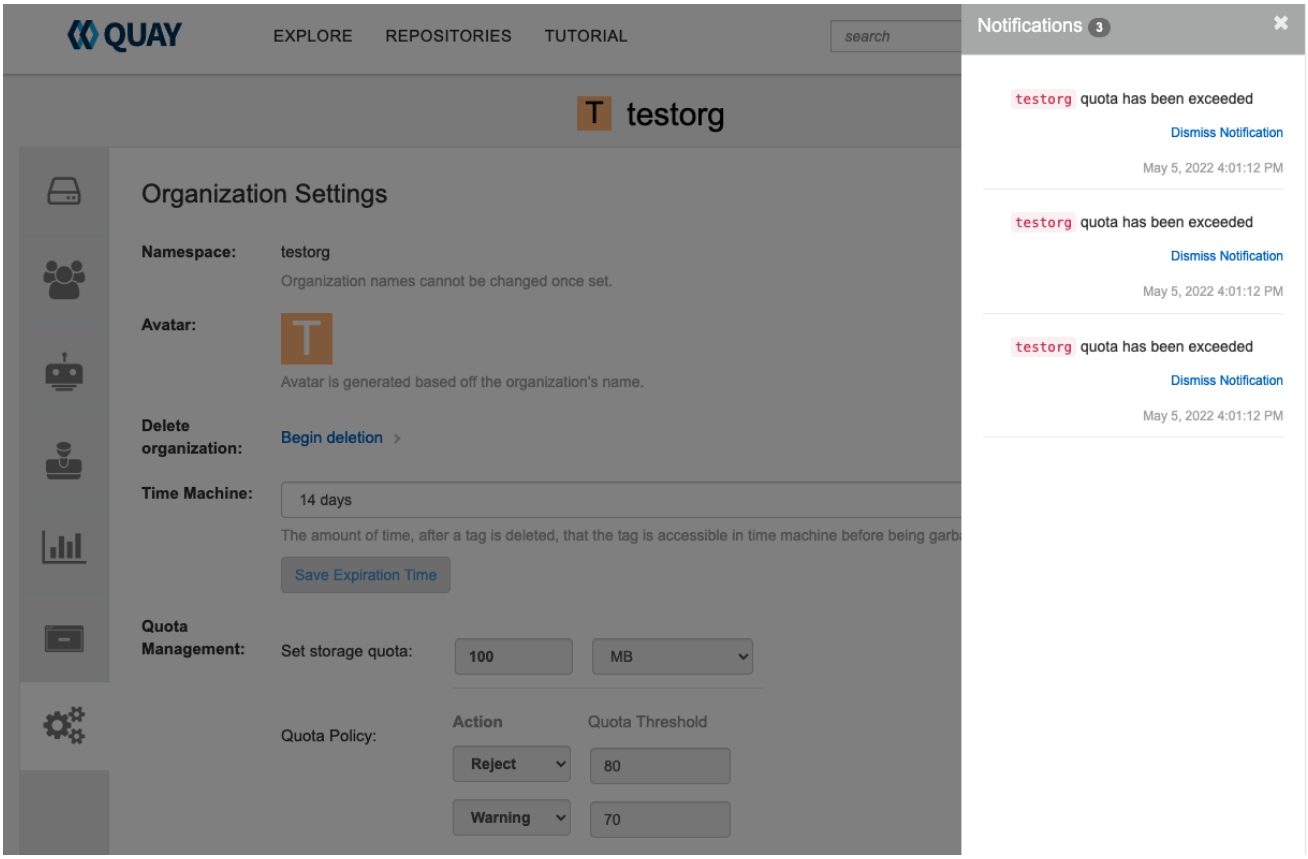
クォータを超えたときのサンプル出力

```
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. 制限を超えた場合の通知

制限を超えると、通知が表示されます。

クォータ通知


QUAY
EXPLORE
REPOSITORIES
TUTORIAL
search
T testorg
Organization Settings
Namespace:
testorg
Organization names cannot be changed once set.
Avatar:
T
Avatar is generated based off the organization's name.
Delete organization:
Begin deletion
Time Machine:
14 days
The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garb
Save Expiration Time
Quota Management:
Set storage quota:
100
MB
Quota Policy:
Action
Quota Threshold
Reject
80
Warning
70
Notifications 3
testorg quota has been exceeded
Dismiss Notification
May 5, 2022 4:01:12 PM
testorg quota has been exceeded
Dismiss Notification
May 5, 2022 4:01:12 PM
testorg quota has been exceeded
Dismiss Notification
May 5, 2022 4:01:12 PM

第12章 アップストリームレジストリーのプロキシーキャッシュとしての RED HAT QUAY

コンテナー開発の人気が高まるにつれ、お客様はサービスを稼働させるために Docker や Google Cloud Platform などのアップストリームレジストリーからのコンテナーイメージにますます依存するようになっています。現在、レジストリーにはレート制限があり、ユーザーがこれらのレジストリーからプルできる回数が制限されています。

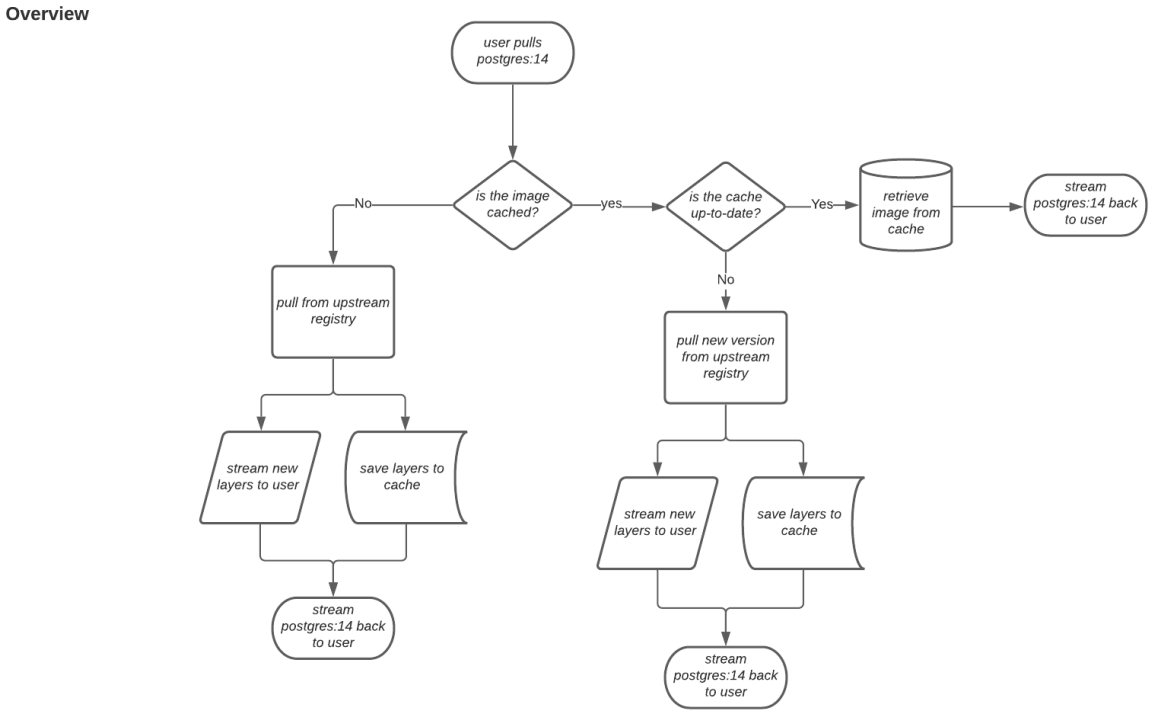
この機能により、Red Hat Quay は、アップストリームレジストリーからのプルレート制限を回避するためのプロキシーキャッシュとして機能します。キャッシュ機能を追加すると、アップストリームの依存関係ではなくキャッシュからイメージがプルされるため、プルのパフォーマンスも向上します。キャッシュされたイメージは、アップストリームのイメージダイジェストがキャッシュされたイメージと異なる場合にのみ更新され、レート制限と潜在的なスロットリングを削減します。

Red Hat Quay キャッシュプロキシーを使用すると、次の機能が利用可能になります。

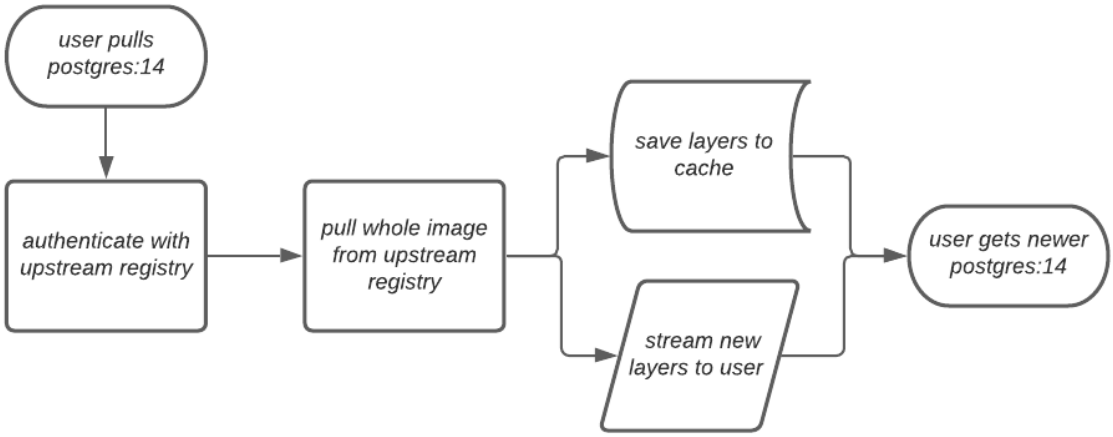
- 特定の組織は、アップストリームレジストリーのキャッシュとして定義できます。
- 特定のアップストリームレジストリーのキャッシュとして機能する Quay 組織の設定。このリポジトリーは、Quay UI を使用して定義でき、次の設定を提供します。
    - プライベートリポジトリーのアップストリームレジストリークレデンシャルまたはレート制限の強化。
    - キャッシュ組織のサイズを超えないようにするための有効期限タイマー。
- 設定アプリケーションを介して設定可能なグローバルオン/オフ。
- アップストリームレジストリー全体または単一の名前空間 (たとえば、すべての **docker.io** または単に **docker.io/library** のキャッシュ。
- すべてのキャッシュプルのログ。
- Clair によるキャッシュイメージのスキャン可能性。

12.1. プロキシーキャッシュアーキテクチャー

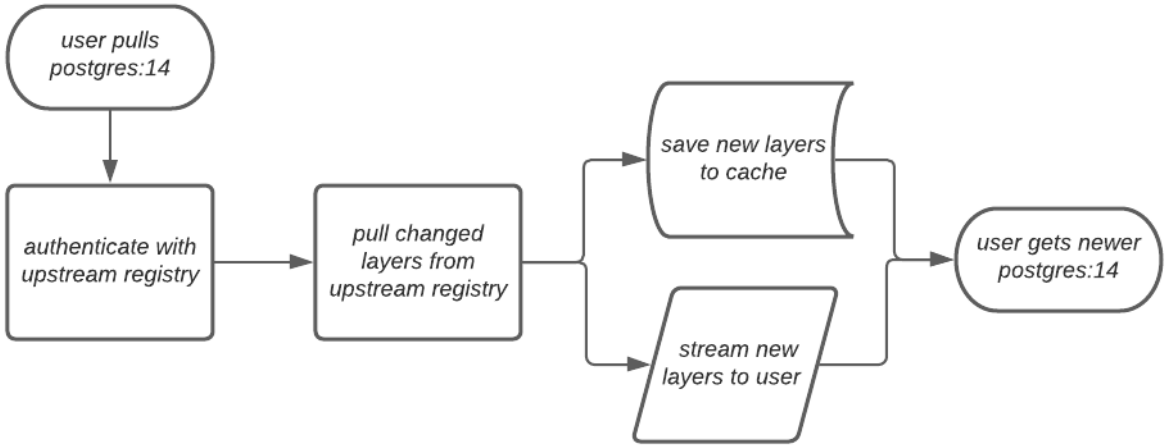
次のイメージは、プロキシーキャッシュ機能の予想される設計フローおよびアーキテクチャーを示しています。



ユーザーが Red Hat Quay のアップストリームリポジトリーからイメージ (**postgres:14** など) をプルすると、リポジトリーはイメージが存在するかどうかを確認します。イメージが存在しない場合は、新しいプルが開始します。プルされた後、イメージレイヤーはキャッシュに保存され、サーバーに並行してユーザーに提供されます。次のイメージは、このシナリオのアーキテクチャーの概要を示しています。

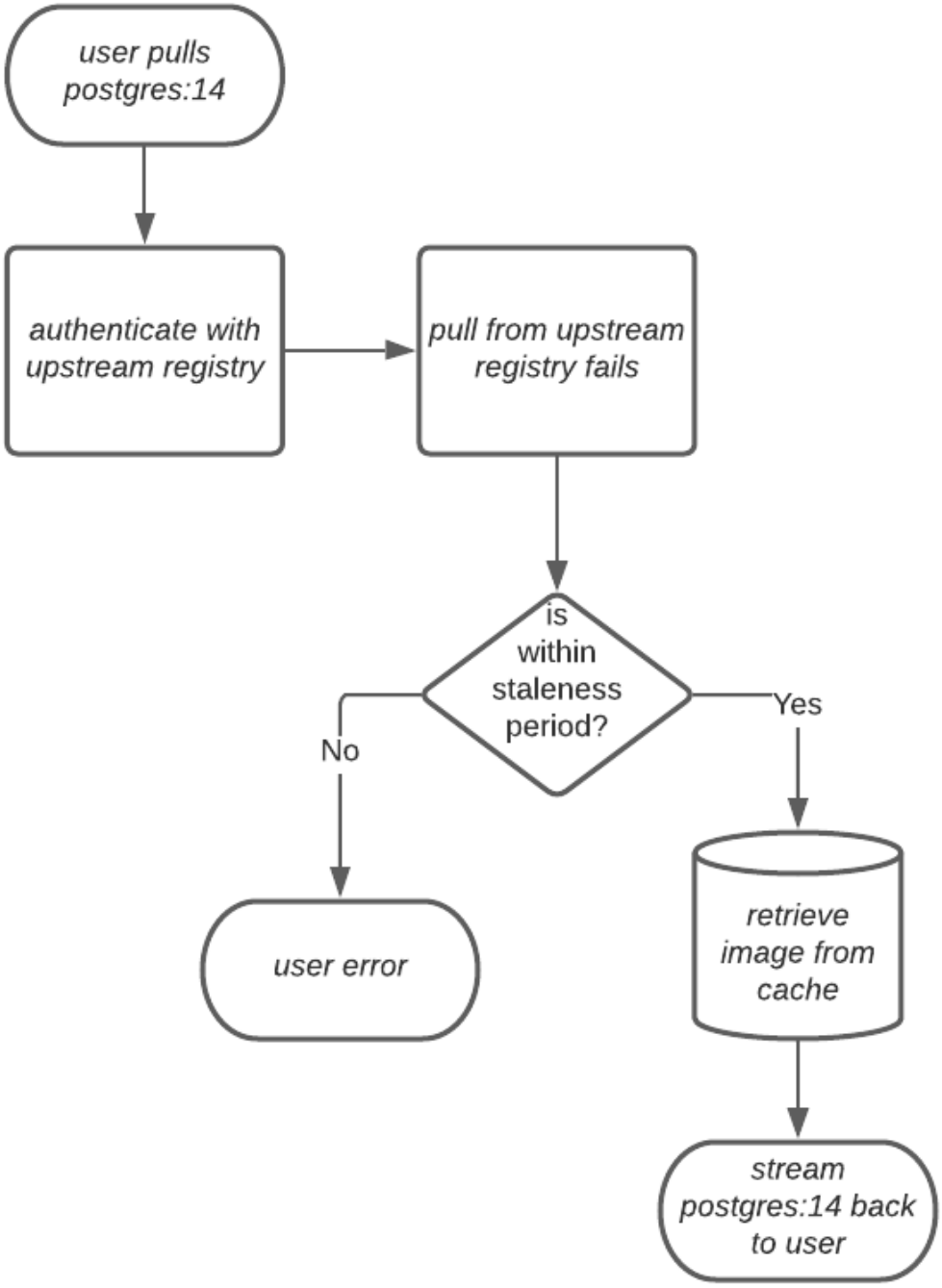


キャッシュ内のイメージが存在する場合、ユーザーは Quay のキャッシュを利用してアップストリームソースを最新の状態に保ち、キャッシュから新しいイメージが自動的にプルされるようにできます。これは、元のイメージのタグがアップストリームレジストリーで上書きされた場合に発生します。次のイメージは、アップストリームイメージとキャッシュされたバージョンのイメージが異なる場合に何が起こるかを示すアーキテクチャーの概要を示しています。



アップストリームイメージとキャッシュされたバージョンが同じである場合、レイヤーはプルされず、キャッシュされたイメージがユーザーに配信されます。

場合によっては、アップストリームレジストリーがダウンしたときにユーザーがプルを開始します。設定された失効期間でこれが発生した場合は、キャッシュに保存されたイメージが配信されます。設定された失効期間の後にプルが発生すると、エラーはユーザーに伝播されます。次のイメージは、設定された失効期間の後にプルが発生した場合のアーキテクチャーの概要を示しています。



Quay 管理者は、組織の設定可能なサイズ制限を利用してキャッシュサイズを制限できるため、バックエンドストレージの消費量を予測できます。これは、イメージが使用される頻度に応じてキャッシュからイメージを破棄することで実現されます。次のイメージは、このシナリオのアーキテクチャーの概要を示しています。

12.2. プロキシーキャッシュの制限

Red Hat Quay を使用したプロキシーキャッシングには、次の制限があります。

- プロキシーキャッシュには、キャッシュするイメージ以上のサイズ制限が必要です。たとえば、プロキシーキャッシュ組織の最大サイズが 500 MB で、ユーザーがプルしたいイメージが 700 MB の場合、イメージはキャッシュされますが、設定された制限を超えてオーバーフローします。

- キャッシュされたイメージは、Quay リポジトリー上のイメージが持つ必要があるのと同じプロパティーを持っている必要があります。

12.3. RED HAT QUAY を使用してリモートレジストリーをプロキシーする

次の手順では、Red Hat Quay を使用してリモートレジストリーをプロキシーする方法を説明します。この手順は、プロキシー quay.io に設定されています。これにより、ユーザーは **podman** を使用して、quay.io 上の任意の名前空間から任意のパブリックイメージをプルできます。

前提条件

- config.yaml の **FEATURE_PROXY_CACHE** が **true** に設定されている。
- **Member** チームのロールが割り当てられている。チームのロールの詳細は、Red Hat Quay のユーザーおよび組織 を参照してください。

手順

1. UI の Quay 組織 (たとえば **cache-quayio**) で、左側のペインの **Organization Settings** をクリックします。
2. オプション: **Add Storage Quota**をクリックして、組織のクォータ管理を設定します。クォータ管理の詳細は、クォータ管理 を参照してください。

注記

場合によっては、Podman を使用してイメージをプルすると、プル中にクォータ制限に達したときに **unable to pull image: Error parsing image configuration: Error fetching blob: invalid status code from registry 403 (Forbidden)** エラーが返されることがあります。エラー **403** は不正確であり、Podman が正しい API エラー **Quota has been exceeded on namespace** を非表示にしているために発生します。この既知の問題は、将来の Podman 更新で修正される予定です。

3. **Remote Registry** に、キャッシュするリモートレジストリーの名前 (**quay.io** など) を入力し、**Save** をクリックします。

注記

Remote Registry に名前空間 (たとえば **quay.io/<namespace>**) を追加すると、組織内のユーザーはその名前空間からのみプロキシーできるようになります。

4. オプション: **Remote Registry Username** および **Remote Registry Password** を追加します。

注記

Remote Registry Username および **Remote Registry Password** を設定しない場合は、プロキシーキャッシュを削除して新しいレジストリーを作成しない限り、パスワードを追加できません。

5. オプション: **Expiration** フィールドに時間を設定します。

注記

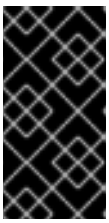
- プロキシー組織でキャッシュされたイメージのデフォルトのタグ **Expiration** フィールドは 86400 秒に設定されています。プロキシー組織では、タグがプルされるたびに、タグの有効期限が UI の **Expiration** フィールドに設定された値に更新されます。この機能は、Quay のデフォルトの 個別タグ有効期限 機能とは異なります。プロキシー組織では、個々のタグ機能をオーバーライドできます。これが発生すると、プロキシー組織の **Expiration** フィールドに従って、個々のタグの有効期限がリセットされます。

- 期限切れのイメージは、割り当てられた時間が経過すると消えますが、Quay に保存されます。イメージが完全に削除されるタイミング、またはコレクションされるタイミングは、組織の **Time Machine** の設定によって異なります。特に指定がない限り、ガベージコレクションのデフォルトの時間は 14 日です。

6. **Save** をクリックします。

7. CLI で、プロキシーキャッシュとして機能するパブリックイメージ (quay.io など) をレジストリーからプルします。

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



重要

組織がリモートレジストリー内の単一の名前空間からプルするように設定されている場合は、リモートレジストリーの名前空間を URL から省略する必要があります。たとえば、**podman pull <registry_url>/<organization_name>/<image_name>** です。

12.3.1. プロキシー組織でのストレージクォータ制限の活用

Red Hat Quay 3.8 では、プロキシーキャッシュ機能が強化され、タグ付きイメージの自動プルーニング機能が追加されました。イメージタグの自動プルーニングは、プロキシーされた名前空間にクォータ制限が設定されている場合にのみ使用できます。現在、イメージサイズが組織のクォータより大きい場合は、管理者が必要なスペースを作成するまで、イメージのアップロードはスキップされます。現在、割り当てられたスペースを超えるイメージがプッシュされると、自動プルーニングの機能強化により、使用頻度の最も低いタグが削除対象としてマークされます。その結果、新しいイメージタグが保存され、最も使用頻度の低いイメージタグが削除対象としてマークされます。

重要

- 自動プルーニング機能の一部として、削除対象としてマークされたタグは、最終的にガベージコレクター (gc) ワーカープロセスによってガベージコレクションされます。そのため、この期間中はクォータサイズの制限が完全には適用されません。

- 現在、名前空間クォータサイズの計算では、マニフェストの子のサイズは考慮されていません。これは既知の問題であり、Red Hat Quay の今後のバージョンで修正される予定です。

12.3.1.1. プロキシー組織でのストレージクォータ制限機能のテスト

次の手順を使用して、プロキシーキャッシュとストレージクォータ制限が有効になっている組織の自動プルーニング機能をテストします。

前提条件

- 組織がプロキシー組織として機能するように設定されている。次の例では、quay.io からプロキシーします。
- **config.yaml** ファイルの **FEATURE_PROXY_CACHE** が **true** に設定されている。
- **config.yaml** ファイルで **FEATURE_QUOTA_MANAGEMENT** が **true** に設定されている。
- 組織には **150 MB** などのクォータ制限が設定されている。

手順

1. プロキシー組織からリポジトリーにイメージをプルします。以下に例を示します。

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

2. リポジトリーに残っているスペースによっては、プロキシー組織から追加のイメージのプルが必要になる場合があります。以下に例を示します。

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

3. Red Hat Quay レジストリー UI で、リポジトリーの名前をクリックします。
   - ナビゲーションペインで **Tags** をクリックし、**quay:3.7.9** と **quay:3.6.2** がタグ付けされていることを確認します。
4. リポジトリーに割り当てられたクォータを超えるように、最後のイメージをプルします。次に例を示します。

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

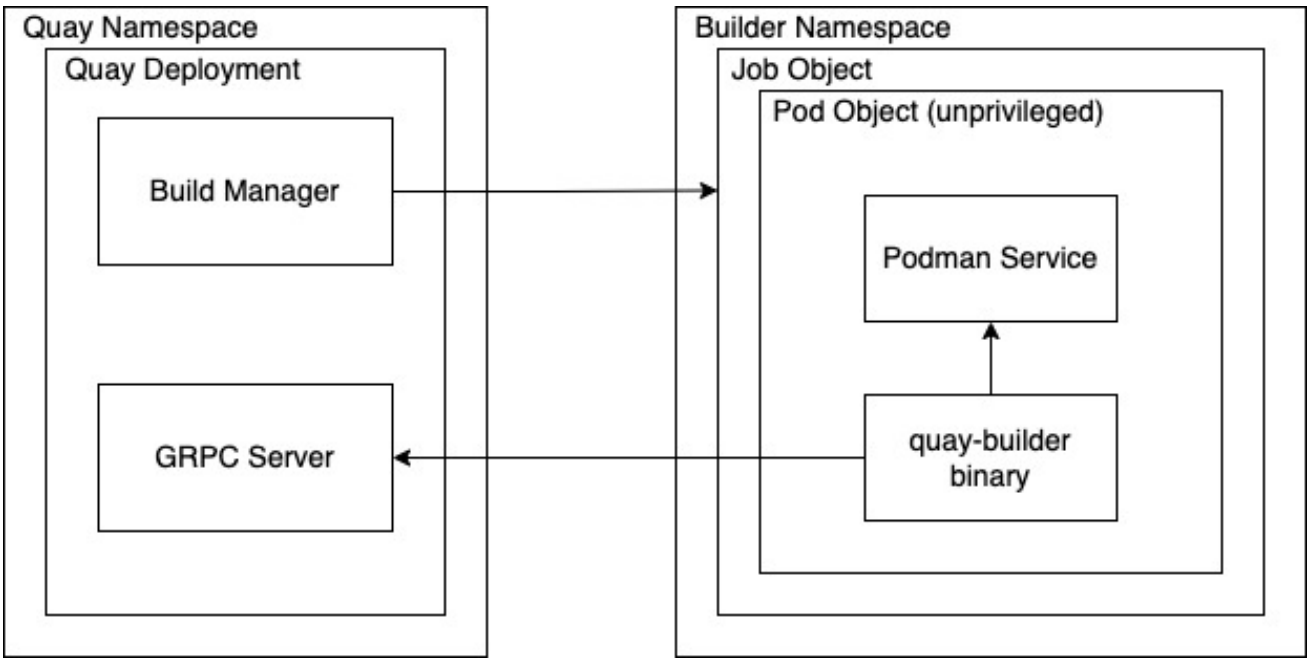
5. Red Hat Quay レジストリーの **Tags** ページを更新します。プッシュした最初のイメージ (**quay:3.7.9** など) は、自動プルーニングされているはずです。**Tags** ページには **quay:3.6.2** と **quay:3.5.1** が表示されるはずです。

第13章 RED HAT QUAY ビルドの機能強化

Red Hat Quay ビルドは仮想化プラットフォーム上で実行できます。以前のビルド設定を実行するための下位互換性も利用できます。

13.1. RED HAT QUAY の拡張ビルドアーキテクチャー

以下のイメージは、拡張ビルド機能の想定される設計フローとアーキテクチャーを示しています。



この機能拡張により、ビルドマネージャーは最初に **Job Object** を作成します。次に、**Job Object** は **quay-builder-image** を使用して Pod を作成します。 **quay-builder-image** には、**quay-builder binary** サービスおよび **Podman** サービスが含まれます。作成された Pod は **unprivileged** として実行されます。次に、**quay-builder binary** は、ステータスを伝達し、ビルドマネージャーからビルド情報を取得しながら、イメージをビルドします。

13.2. RED HAT QUAY ビルドの制限

特権のないコンテキストで Red Hat Quay でビルドを実行すると、以前のビルド戦略で機能していた一部のコマンドが失敗する可能性があります。ビルド戦略を変更しようとすると、ビルドのパフォーマンスの問題と信頼性の問題が発生する可能性があります。

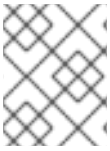
コンテナーでビルドを直接実行しても、仮想マシンを使用する場合と同じように分離されることはありません。ビルド環境を変更すると、以前は機能していたビルドが失敗する可能性もあります。

13.3. OPENSHIFT CONTAINER PLATFORM を使用した RED HAT QUAY ビルダー環境の作成

このセクションの手順では、OpenShift Container Platform で Red Hat Quay 仮想ビルダー環境を作成する方法を説明します。

13.3.1. OpenShift Container Platform TLS コンポーネント

tls コンポーネントを使用すると、TLS 設定を制御できます。



注記

TLS コンポーネントが Operator によって管理されている場合、Red Hat Quay 3.11 はビルダーをサポートしません。

tls を **unmanaged** に設定する場合は、独自の **ssl.cert** ファイルと **ssl.key** ファイルを提供します。このとき、クラスターでビルダーをサポートする場合は、Quay ルートとビルダールート名の両方を証明書の SAN リストに追加するか、ワイルドカードを使用する必要があります。

ビルダールートを追加するには、次の形式を使用します。

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

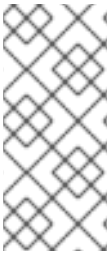
13.3.2. OpenShift Container Platform ビルダー向けの Red Hat Quay の使用

ビルダーには SSL/TLS 証明書が必要です。SSL/TLS 証明書の詳細は Red Hat Quay コンテナーへの TLS 証明書の追加 を参照してください。

Amazon Webservice (AWS) S3 ストレージを使用している場合は、ビルダーを実行する前に、AWS コンソールでストレージバケットを変更する必要があります。必要なパラメーターは、次のセクションの AWS S3 ストレージバケットの変更を参照してください。

13.3.2.1. 仮想ビルダー向けの OpenShift Container Platform の準備

以下の手順を使用して、Red Hat Quay 仮想ビルダー用に OpenShift Container Platform を準備します。



注記

- この手順は、クラスターがすでにプロビジョニングされており、Quay Operator が実行されていることを前提とします。
- この手順は、OpenShift Container Platform で仮想 namespace を設定するためのものです。

手順

1. クラスター管理者アカウントを使用して Red Hat Quay クラスターにログインします。
2. 次のコマンドを実行して、仮想ビルダーが実行される新しいプロジェクト (**virtual-builders** など) を作成します。

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

3. 次のコマンドを入力して、ビルドの実行に使用されるプロジェクトに **ServiceAccount** を作成します。

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

4. 作成したサービスアカウントに編集権限を付与して、ビルドを実行できるようにします。

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

5. 次のコマンドを入力して、Quay ビルダーに **anyuid scc** 権限を付与します。

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

注記

このアクションには、クラスター管理者特権が必要です。非特権ビルドまたはルートレスビルドを機能させるには、ビルダーを Podman ユーザーとして実行する必要があるため、これが必要です。

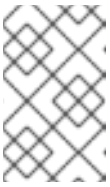
6. Quay ビルダーサービスアカウントのトークンを取得します。

   a. OpenShift Container Platform 4.10 以前のバージョンを使用している場合は、以下のコマンドを入力します。

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

   b. OpenShift Container Platform 4.11 以降を使用している場合は、以下のコマンドを入力します。

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



注記

トークンの有効期限が切れたら、新しいトークンを要求する必要があります。必要に応じて、カスタムの有効期限を追加することもできます。たとえば、トークンを 2 週間保持するには、**--duration 20160m** と指定します。

出力例

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJscldSdEUtYWsifQ...
```

7. 次のコマンドを入力してビルダールートを決定します。

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

出力例

```
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. 次のコマンドを入力して、拡張子が .crt の自己署名 SSL/TLS 証明書を生成します。

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

出力例

```
ca.crt
```

9. 次のコマンドを入力して、**ca.crt** ファイルの名前を **extra_ca_cert_build_cluster.crt** に変更します。

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

10. **Console** で設定バンドルのシークレットを見つけ、**Actions → Edit Secret** を選択して、適切なビルダー設定を追加します。

```
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. ビルドルートは、Open Shift Operators namespace の名前で **oc get route -n** を実行することにより取得されます。ルートの最後にポートを指定する必要があり、**quayregistry-cr-name-quay-builder-ocp-namespace.ocp-domain-name:443** の形式を使用する必要があります。
2. **JOB_REGISTRATION_TIMEOUT** パラメーターの設定が低すぎると、**failed to register job to build manager: rpc error: code = Unauthenticated desc = Invalid build token: Signature has expired** エラーが発生する可能性があります。このパラメーターは少なくとも 240 に設定することを推奨します。
3. Redis ホストにパスワードまたは SSL/TLS 証明書がある場合は、それに応じて更新する必要があります。
4. 仮想ビルダーの namespace の名前と一致するように設定します (例: **virtual-builders**)。
5. 早期アクセスの場合、**BUILDER_CONTAINER_IMAGE** は現在 **quay.io/projectquay/quay-builder:3.7.0-rc.2** です。ただし、早期アクセス期間中に変更される可能性があります。これが発生した場合は、顧客に警告が表示されます。
6. **K8S_API_SERVER** は、**oc cluster-info** を実行して取得します。
7. カスタム CA 証明書を手動で作成して追加する必要があります (例 **K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt**)。
8. 指定しないと、デフォルトは **5120Mi** です。
9. 仮想ビルドの場合は、クラスターに十分なリソースがあることを確認する必要があります。指定しないと、デフォルトは **1000m** です。
10. 指定しないと、デフォルトは **3968Mi** です。
11. 指定しないと、デフォルトは **500m** です。
12. **oc create sa** の実行時に取得します。

設定サンプル

```
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:
"eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJs
cldSdEUtYWsifQ"
```

13.3.2.2. SSL/TLS 証明書の手動追加

設定ツールの既知の問題のため、ビルダーを適切に実行するには、カスタム SSL/TLS 証明書を手動で追加する必要があります。次の手順を使用して、カスタム SSL/TLS 証明書を手動で追加します。

SSL/TLS 証明書の作成の詳細は、Red Hat Quay コンテナーへの TLS 証明書の追加 を参照してください。

13.3.2.2.1. 証明書の作成と署名

SSL/TLS 証明書を作成して署名するには、次の手順を実行します。

手順

- 認証局を作成し、証明書に署名します。詳細は、認証局の作成と証明書への署名 を参照してください。

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. Red Hat Quay レジストリーの URL の **alt_name** を含める必要があります。
2. **BUILDMAN_HOSTNAME** の **alt_name**

サンプルコマンド

```
$ 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. TLS の管理対象外への設定

次の手順を使用して、**king:tls** を管理対象外に設定します。

手順

1. Red Hat Quay Registry YAML で、**kind: tls** を **managed: false** に設定します。

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

2. **Events** ページでは、適切な **config.yaml** ファイルを設定するまで変更がブロックされます。以下に例を示します。

```
- 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. 一時的なシークレットの作成

次の手順を使用して、CA 証明書の一時的なシークレットを作成します。

手順

1. CA 証明書のデフォルトの namespace にシークレットを作成します。

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

2. **ssl.key** ファイルおよび **ssl.cert** ファイルのデフォルトの namespace にシークレットを作成します。

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

13.3.2.2.4. シークレットデータの設定 YAML へのコピー

次の手順を使用して、シークレットデータを **config.yaml** ファイルにコピーします。

手順

1. コンソール UI の **Workloads → Secrets** で新しいシークレットを見つけます。
2. シークレットごとに、YAML ビューを見つけます。

```
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. UI で Red Hat Quay レジストリー設定バンドルのシークレットを見つけるか、以下のようなコマンドを実行してコマンドラインから見つけます。

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

4. OpenShift Container Platform コンソールで、設定バンドルのシークレットの YAML タブを選択し、作成した 2 つのシークレットからデータを追加します。

```
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: >-
    RkVBVFVSRV9VU0VSX0lOSVRJQUxJWkU6IHRydWUKQlJ...
  extra_ca_cert_build_cluster.crt: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0ldw....
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

5. **Save** をクリックします。

6. 次のコマンドを入力して、Pod が再起動しているかどうかを確認します。

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

出力例

```
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. Red Hat Quay レジストリーが再設定されたら、次のコマンドを入力して、Red Hat Quay アプリの Pod が実行されているかどうかを確認します。

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

出力例

```
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. ブラウザーで、レジストリーエンドポイントにアクセスし、証明書が適切に更新されていることを確認します。以下に例を示します。

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

13.3.2.3. UI を使用してビルドトリガーを作成

UI を使用してビルドトリガーを作成するには、次の手順に従います。

手順

1. Red Hat Quay リポジトリーにログインします。
2. **Create New Repository** をクリックして、**testrepo** などの新しいレジストリーを作成します。
3. **Repositories** ページで、ナビゲーションペインの **Builds** タブをクリックします。または、対応する URL を直接使用します。

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

重要

場合によっては、ビルダーでホスト名の解決に問題が発生することがあります。この問題は、ジョブオブジェクトで **default** に設定されている **dnsPolicy** に関連している可能性があります。現在、この問題に対する回避策はありません。これは、Red Hat Quay の将来のバージョンで解決される予定です。

4. **Create Build Trigger→ Custom Git Repository Push** をクリックします。
5. Git リポジトリーのクローン作成に使用する HTTPS または SSH スタイルの URL を入力し、**Continue** をクリックします。以下に例を示します。

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

6. **Tag manifest with the branch or tag name**を確認し、**Continue** をクリックします。
7. トリガーが呼び出されたときにビルドする Dockerfile の場所 (たとえば **/Dockerfile**) を入力し、**Continue** をクリックします。
8. Docker ビルドのコンテキストの場所 (たとえば /) を入力し、**Continue** をクリックします。
9. 必要に応じて、ロボットアカウントを作成します。それ以外の場合は、**Continue** をクリックします。
10. **Continue** をクリックして、パラメーターを確認します。
11. **Builds** ページで、トリガー名の **Options** アイコンをクリックし、**Run Trigger Now** をクリックします。
12. Git リポジトリーからコミット SHA を入力し、**Start Build** をクリックします。

13. ビルドのステータスを確認するには、**Build History** ページで commit をクリックするか、**oc get pods -n virtual-builders** を実行します。以下に例を示します。

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

出力例

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

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

出力例

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

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

出力例

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

14. ビルドが完了したら、ナビゲーションペインのタグで **Tags** のステータスを確認できます。

注記

早期アクセスにより、完全なビルドログとビルドのタイムスタンプは現在利用できません。

13.3.2.4. AWS S3 ストレージバケットの変更

AWS S3 ストレージを使用している場合は、ビルダーを実行する前に、AWS コンソールでストレージバケットを変更する必要があります。

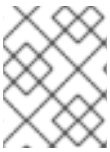
手順

1. s3.console.aws.com で AWS コンソールにログインします。
2. 検索バーで **S3** を検索し、**S3** をクリックします。
3. バケットの名前 (**myawsbucket** など) をクリックします。
4. **Permissions** タブをクリックします。
5. **Cross-origin resource sharing (CORS)**の下に、次のパラメーターを含めます。

```
[
    {
        "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. Google Cloud Platform オブジェクトバケットの変更



注記

現在、Google Cloud Platform オブジェクトバケットの変更は、IBM Power および IBM Z ではサポートされていません。

仮想ビルダーの Cross Origin Resource Sharing (CORS) を設定するには、次の手順を実行します。

注記

CORS 設定がないと、ビルド Dockerfile のアップロードは失敗します。

手順

1. 次のリファレンスを使用して、特定の CORS ニーズに合わせた JSON ファイルを作成します。以下に例を示します。

```
$ cat gcp_cors.json
```

出力例

```
[
  {
    "origin": ["*"],
    "method": ["GET"],
    "responseHeader": ["Authorization"],
```

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

2. 次のコマンドを入力して、GCP ストレージバケットを更新します。

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

出力例

```
Updating
  Completed 1
```

3. 次のコマンドを実行すると、GCP バケットの更新された CORS 設定を表示できます。

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

出力例

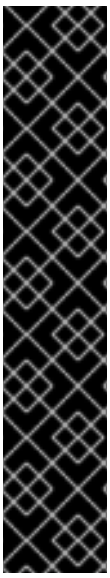
```
cors:
- maxAgeSeconds: 3600
  method:
  - GET
  origin:
  - '*'
  responseHeader:
  - Authorization
- maxAgeSeconds: 3600
  method:
  - PUT
  origin:
  - '*'
  responseHeader:
  - Content-Type
  - x-goog-acl
  - origin
```

第14章 V2 UI の使用

以下の手順を使用して、Red Hat Quay v2 UI を設定し、使用します。

14.1. V2 ユーザーインターフェイス設定

FEATURE_UI_V2 を有効にすると、現在のバージョンのユーザーインターフェイスと新しいバージョンのユーザーインターフェイスを切り替えることができます。



重要

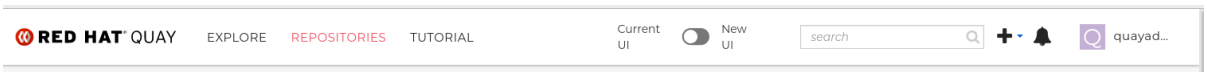
- この UI は現在ベータ版であり、変更される可能性があります。現在の状態では、ユーザーは組織、リポジトリー、およびイメージタグのみを作成、表示、および削除できます。
- 古い UI を使用している場合にセッションがタイムアウトになると、ユーザーはポップアップウィンドウでパスワードを再度入力する必要がありました。新しい UI では、ユーザーはメインページに戻り、ユーザー名とパスワードの認証情報を入力する必要があります。これは既知の問題であり、新しい UI の今後のバージョンで修正される予定です。
- 従来の UI と新しい UI の間で、イメージマニフェストのサイズが報告される方法に違いがあります。従来の UI では、イメージマニフェストはメビバイト単位で報告されていました。v2 UI では、メガバイト (MB) の標準定義を使用してイメージマニフェストのサイズが報告されます。

手順

1. デプロイメントの **config.yaml** ファイルで、**FEATURE_UI_V2** パラメーターを追加し、**true** に設定します。次に例を示します。

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

2. Red Hat Quay デプロイメントにログインします。
3. デプロイメントのナビゲーションペインに、**現在の UI** と **新しい UI** を切り替えるオプションが表示されます。切り替えボタンをクリックして新しい UI に設定し、次に **Use Beta Environment** をクリックします。次に例を示します。



14.1.1. v2 UI を使用した新しい組織の作成

前提条件

- v2 UI を使用するようにデプロイメントを切り替えている。

v2 UI を使用して組織を作成するには、次の手順を実行します。

手順

1. ナビゲーションペインで **Organization** をクリックします。
2. **Create Organization** をクリックします。
3. **Organization Name** (**testorg** など) を入力します。
4. **Create** をクリックします。

これで、サンプルの組織が **Organizations** ページの下に表示されます。

14.1.2. v2 UI を使用した組織の削除

v2 UI を使用して組織を削除するには、次の手順を実行します。

手順

1. **Organizations** ページで、削除する組織の名前 (**testorg** など) を選択します。
2. **More Actions** ドロップダウンメニューをクリックします。
3. **Delete** をクリックします。

注記

Delete ページには、**Search** 入力ボックスがあります。このボックスを使用すると、ユーザーは特定の組織を検索して、削除が適切にスケジュールされていることを確認できます。たとえば、ユーザーが 10 の組織を削除していて、特定の組織が削除されたことを確認したい場合、**Search** 入力ボックスを使用して、その組織が削除対象としてマークされていることを確認できます。

4. ボックスに **confirm** と入力して、組織を完全に削除することを確定します。
5. **Delete** をクリックします。
削除後、**Organizations** ページに戻ります。

注記

複数の組織を選択してから、**More Actions → Delete** をクリックすると、一度に複数の組織を削除できます。

14.1.3. v2 UI を使用した新しいリポジトリーの作成

v2 UI を使用してリポジトリーを作成するには、次の手順を実行します。

手順

1. ナビゲーションペインで **Repositories** をクリックします。
2. **Create Repository** をクリックします。
3. 名前空間 (例: **quayadmin**) を選択し、**リポジトリー名** (例: **testrepo**) を入力します。

重要

リポジトリー名には次の単語を使用しないでください。* **build** * **trigger** * **tag**

これらの単語をリポジトリー名に使用すると、ユーザーがリポジトリーにアクセスできなくなり、リポジトリーを完全に削除できなくなります。このようなリポジトリーを削除しようとすると、**Failed to delete repository <repository_name>, HTTP404 - Not Found.** というエラーが返されます。

4. Create をクリックします。
これで、サンプルリポジトリーが Repositories ページの下に表示されるはずです。

14.1.4. v2 UI を使用したリポジトリーの削除

前提条件

- リポジトリーを作成している。

手順

1. v2 UI の Repositories ページで、削除するイメージの名前 (**quay/admin/busybox** など) をクリックします。
2. More Actions ドロップダウンメニューをクリックします。
3. Delete をクリックします。

注記

必要に応じて、Make Public または Make Private をクリックできます。

4. ボックスに confirm と入力してから、Delete をクリックします。
5. 削除後、Repositories ページに戻ります。

14.1.5. v2 UI へのイメージのプッシュ

イメージを v2 UI にプッシュするには、次の手順を実行します。

手順

1. 外部レジストリーからサンプルイメージをプルします。

```
$ podman pull busybox
```

2. イメージにタグを付けます。

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

3. イメージをレジストリーにプッシュします。

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

4. v2 UI の **Repositories** ページに移動し、イメージが適切にプッシュされていることを確認します。
5. イメージタグを選択し、**Security Report** ページに移動すると、セキュリティーの詳細を確認できます。

14.1.6. v2 UI を使用したイメージの削除

v2 UI を使用してイメージを削除するには、次の手順を実行します。

前提条件

- イメージをレジストリーにプッシュしている。

手順

1. v2 UI の **Repositories** ページで、削除するイメージの名前 (**quay/admin/busybox** など) をクリックします。
2. **More Actions** ドロップダウンメニューをクリックします。
3. **Delete** をクリックします。

注記

必要に応じて、**Make Public** または **Make Private** をクリックできます。

4. ボックスに **confirm** と入力してから、**Delete** をクリックします。
5. 削除後、**Repositories** ページに戻ります。

14.1.7. Red Hat Quay v2 UI を使用した新しいチームの作成

Red Hat Quay v2 UI を使用して新しいチームを作成するには、次の手順を実行します。

前提条件

- リポジトリーを持つ組織を作成している。

手順

1. Red Hat Quay v2 UI で、組織の名前をクリックします。
2. 組織のページで、**Teams and membership** をクリックします。
3. **Create new team** ボックスをクリックします。
4. **Create team** ポップアップウィンドウで、新しいチームの名前を入力します。
5. オプション: 新しいチームの説明を入力します。
6. **Proceed** をクリックします。新しいポップアップウィンドウが表示されます。

7. オプション: このチームをリポジトリーに追加し、権限を **Read**、**Write**、**Admin**、または **None** のいずれかに設定します。

8. オプション: チームメンバーまたはロボットアカウントを追加します。チームメンバーを追加するには、Red Hat Quay アカウントの名前を入力します。

9. 情報を確認し、**Review and Finish** をクリックします。新しいチームが **Teams and membership** ページに表示されます。ここから、縦の省略記号メニューをクリックし、次のオプションのいずれかを選択できます。

    - **Manage Team Members**。このページでは、すべてのメンバー、チームメンバー、ロボットアカウント、または招待したユーザーを表示できます。**Add new member** をクリックして、新しいチームメンバーを追加することもできます。

    - **Set repository permissions**。このページでは、リポジトリーの権限を **Read**、**Write**、**Admin**、または **None** のいずれかに設定できます。

    - **Delete**。このポップアップウィンドウでは、**Delete** をクリックしてチームを削除できます。

10. オプション: 次のオプションのいずれかをクリックすると、チーム、メンバー、コラボレーターに関する詳細情報が表示されます。

    - **Team View**。このメニューには、すべてのチーム名、メンバーの数、リポジトリーの数、および各チームのロールが表示されます。

    - **Members View**。このメニューには、チームメンバーのすべてのユーザー名、メンバーが属しているチーム、ユーザーのリポジトリー権限が表示されます。

    - **Collaborators View**。このメニューにはリポジトリーのコラボレーターが表示されます。コラボレーターは、組織内のどのチームにも属さないが、組織に属する 1 つ以上のリポジトリーに対する直接権限を持つユーザーです。

14.1.8. v2 UI を使用したロボットアカウントの作成

v2 UI を使用してロボットアカウントを作成するには、次の手順を実行します。

手順

1. v2 UI で、**Organizations** をクリックします。

2. ロボットアカウントを作成する組織の名前をクリックします (例: **test-org**)。

3. **Robot accounts** タブ → **Create robot account** をクリックします。

4. **Provide a name for your robot account**ボックスに、**robot1** などの名前を入力します。

5. オプション: 必要に応じて、以下のオプションを使用できます。

    a. ロボットをチームに追加します。

    b. ロボットをリポジトリーに追加します。

    c. ロボットのパーミッションを調整します。

6. **Review and finish** ページで、入力した情報を確認してから、**Review and finish** をクリックします。**Successfully created robot account with robot name: <organization_name> + <robot_name>** というアラートが表示されます。
また、別のロボットアカウントと同じ名前でロボットアカウントを作成しようとすると、**Error creating robot account** というエラーメッセージが表示される場合があります。

7. オプション: **Expand** または **Collapse** をクリックすると、ロボットアカウントの説明情報を表示できます。

8. オプション: 縦の省略記号メニュー → **Set repository permissions** をクリックして、ロボットアカウントのパーミッションを変更できます。**Successfully updated repository permission** というメッセージが表示されます。

9. オプション: ロボットアカウントを削除するには、ロボットアカウントのボックスをオンにし、ゴミ箱アイコンをクリックします。ポップアップボックスが表示されます。テキストボックスに **confirm** と入力してから **Delete** をクリックします。または、縦の省略記号メニュー → **Delete** をクリックできます。**Successfully deleted robot account** というメッセージが表示されます。

14.1.8.1. Red Hat Quay v2 UI を使用したロボットアカウントのリポジトリーアクセスの一括管理

Red Hat Quay v2 UI を使用してロボットアカウントのリポジトリーアクセスを一括管理するには、次の手順を実行します。

前提条件

- ロボットアカウントを作成している。
- 1 つの組織に複数のリポジトリーを作成している。

手順

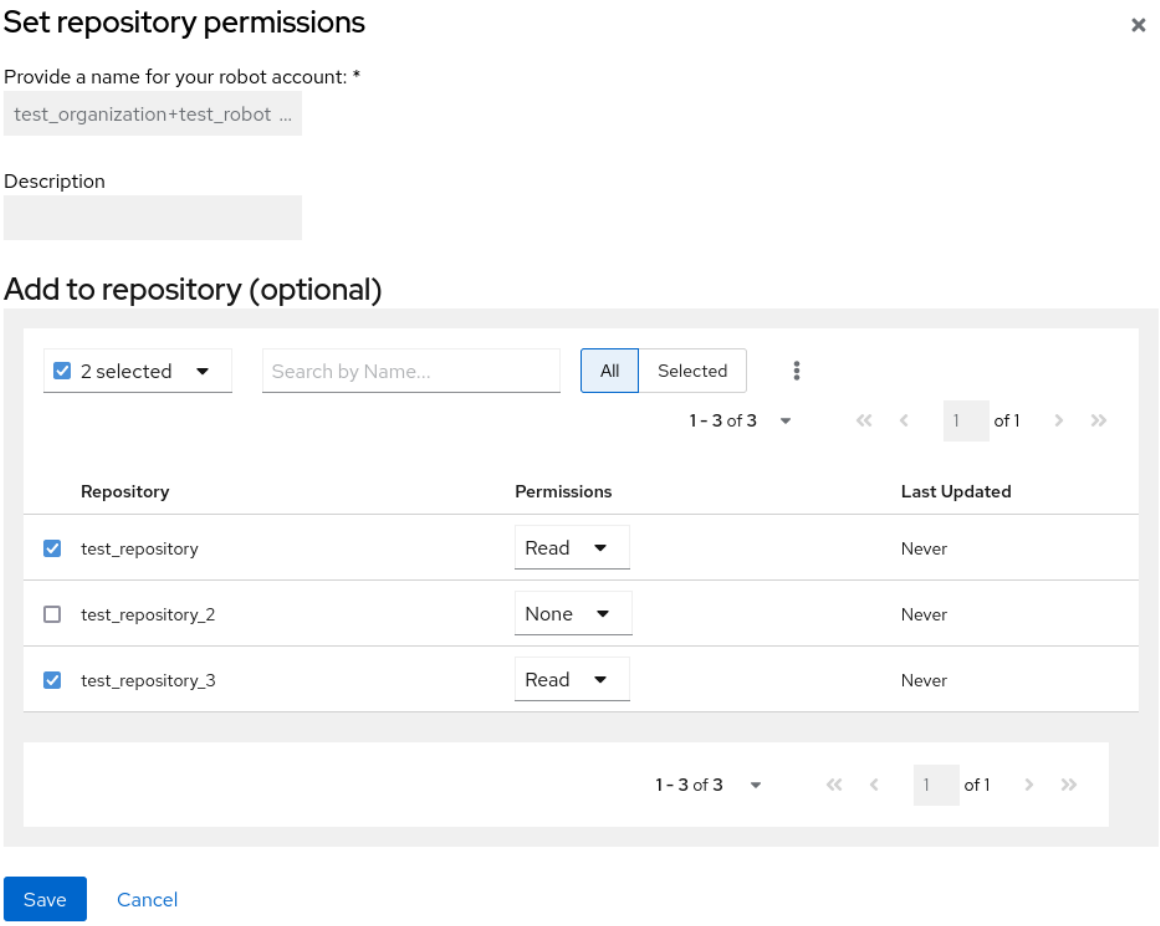
1. Red Hat Quay v2 UI ランディングページで、ナビゲーションペインの **Organizations** をクリックします。

2. **Organizations** ページで、複数のリポジトリーを持つ組織の名前を選択します。単一組織内のリポジトリーの数は、**Repo Count** 列で確認できます。

3. 組織のページで、**Robot accounts** をクリックします。

4. 複数のリポジトリーにロボットアカウントを追加する場合は、縦の省略記号アイコン → **Set repository permissions** をクリックします。

5. **Set repository permissions** ページで、ロボットアカウントを追加するリポジトリーのチェックボックスをオンにします。以下に例を示します。



6. ロボットアカウントの権限を設定します (例: **None**、**Read**、**Write**、**Admin**)。
7. **Save** をクリックします。**Success alert: Successfully updated repository permission**というアラートが **Set repository permissions** ページに表示され、変更が確認されます。
8. **Organizations** → **Robot accounts** ページに戻ります。これで、ロボットアカウントの **Repositories** 列に、ロボットアカウントが追加されたリポジトリーの数が表示されます。

14.1.9. Red Hat Quay v2 UI を使用したデフォルトの権限の作成

デフォルトの権限は、リポジトリーの作成者のデフォルトに加えて、リポジトリーの作成時に自動的に付与される権限を定義します。権限は、リポジトリーを作成したユーザーに基づいて割り当てられます。

Red Hat Quay v2 UI を使用してデフォルトの権限を作成するには、次の手順を実行します。

手順

1. 組織名をクリックします。
2. **Default permissions** をクリックします。
3. **create default permissions**をクリックします。切り替え式のドロワーが表示されます。
4. リポジトリーの作成時にデフォルトの権限を作成するには、**Anyone** または **Specific user** を選択します。
    a. **Anyone** を選択する場合は、次の情報を入力する必要があります。

- **Applied to**。ユーザー/ロボット/チームを検索、招待、または追加します。
- **Permission**。権限を **Read**、**Write**、または **Admin** のいずれかに設定します。

b. **Specific user** を選択する場合は、次の情報を指定する必要があります。

- **Repository creator**。ユーザーまたはロボットアカウントを指定します。
- **Applied to**。ユーザー名、ロボットアカウント、またはチーム名を入力します。
- **Permission**。権限を **Read**、**Write**、または **Admin** のいずれかに設定します。

5. **Create default permission** をクリックします。確認ボックスが表示され、**Successfully created default permission for creator** というアラートが返されます。

14.1.10. v2 UI での組織の設定

v2 UI を使用して組織の設定を変更するには、次の手順を実行します。

手順

1. v2 UI で、**Organizations** をクリックします。
2. ロボットアカウントを作成する組織の名前をクリックします (例: **test-org**)。
3. **Settings** タブをクリックします。
4. オプション: 組織に関連付けられたメールアドレスを入力します。
5. オプション: **Time Machine** 機能に割り当てられた時間を以下のいずれかに設定します。
   - **1 週間**
   - **1 カ月**
   - **1 年**
   - **なし**
6. **Save** をクリックします。

14.1.11. v2 UI を使用したイメージタグ情報の表示

v2 UI を使用してイメージタグ情報を表示するには、次の手順を実行します。

手順

1. v2 UI で、**Repositories** をクリックします。
2. リポジトリーの名前をクリックします (例: **quayadmin/busybox**)。
3. タグの名前をクリックします (例: **test**)。タグの **Details** ページに移動します。このページには、次の情報が表示されます。
   - 名前
   - リポジトリー

- ダイジェスト
- 脆弱性
- 作成
- 修正済み
- サイズ
- ラベル
- イメージタグの取得方法

4. オプション: **Security Report** をクリックして、タグの脆弱性を表示します。アドバイザリーの列を拡張して、CVE データを開くことができます。
5. オプション: **Packages** をクリックして、タグのパッケージを表示します。
6. リポジトリーの名前 (例: **busybox**) をクリックし、**Tags** ページに戻ります。
7. オプション: **Pull** アイコンの上にマウスをかざすと、タグの取得方法が表示されます。
8. タグのボックスまたは複数のタグにチェックを入れ、**Actions** ドロップダウンメニューをクリックしてから、**Delete** をクリックしてタグを削除します。ポップアップボックスの **Delete** をクリックして削除を確定します。

14.1.12. v2 UI を使用したリポジトリー設定の調整

v2 UI を使用してリポジトリーのさまざまな設定を調整するには、次の手順を実行します。

手順

1. v2 UI で、**Repositories** をクリックします。
2. リポジトリーの名前をクリックします (例: **quayadmin/busybox**)。
3. **Settings** タブをクリックします。
4. オプション: **User and robot permissions** をクリックします。**Permissions** のドロップダウンメニューオプションをクリックすると、ユーザーまたはロボットアカウントの設定を調整できます。設定を **Read**、**Write**、または **Admin** に変更できます。
5. オプション: **Events and notifications** をクリックします。**Create Notification** をクリックして、イベントおよび通知を作成できます。以下のイベントオプションを使用できます。
   - リポジトリーへのプッシュ
   - パッケージの脆弱性の検出
   - イメージビルドの失敗
   - イメージビルドのキューへの追加
   - イメージビルドの開始
   - イメージビルドの成功

- イメージビルドのキャンセル
  次に、通知を発行します。以下のオプションを設定できます。
- メール通知
- Flowdock チーム通知
- HipChat ルーム通知
- Slack 通知
- Webhook POST
  イベントオプションと通知方法を選択したら、**Room ID #**、**Room Notification Token** を追加し、**Submit** をクリックします。

6. オプション: **Repository visibility** をクリックします。**Make Public** をクリックして、リポジトリーをプライベートまたはパブリックにすることができます。
7. オプション: **Delete repository** をクリックします。**Delete Repository** をクリックして、リポジトリーを削除できます。

14.2. RED HAT QUAY タグ履歴の表示

Red Hat Quay v2 UI でタグ履歴を表示するには、次の手順を実行します。

手順

1. Red Hat Quay v2 UI ダッシュボードのナビゲーションペインで **Repositories** をクリックします。
2. イメージタグのあるリポジトリーの名前をクリックします。
3. **Tag History** をクリックします。このページでは、次の操作を実行できます。
   - タグ名での検索
   - 日付範囲の選択
   - タグの変更の表示
   - タグの変更日と変更時刻の表示

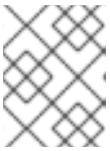
14.3. RED HAT QUAY V2 UI でのラベルの追加と管理

Red Hat Quay 管理者は、次の手順を使用してタグのラベルを追加および管理できます。

手順

1. Red Hat Quay v2 UI ダッシュボードのナビゲーションペインで **Repositories** をクリックします。
2. イメージタグのあるリポジトリーの名前をクリックします。
3. イメージの縦の省略記号メニューをクリックし、**Edit labels** を選択します。
4. **Edit labels** ウィンドウで、**Add new label** をクリックします。

5. **key=value** 形式を使用してイメージタグのラベルを入力します (例: **com.example.release-date=2023-11-14**)。



注記

key=value 形式の使用に失敗すると、**Invalid label format, must be key value separated by =** というエラーが返されます。

6. 空白のボックスをクリックしてラベルを追加します。

7. オプション: 2 番目のラベルを追加します。

8. **Save labels** をクリックして、ラベルをイメージタグに保存します。**Created labels successfully** という通知が返されます。

9. オプション: 同じイメージタグの縦の省略記号メニュー→ **Edit labels** → ラベルの **X** をクリックして削除します。または、テキストを編集することもできます。**Save labels** をクリックします。これでラベルが削除または編集されました。

14.4. RED HAT QUAY V2 UI でのタグの有効期限の設定

Red Hat Quay 管理者は、リポジトリー内の特定のタグの有効期限を設定できます。これにより、古いタグや未使用のタグのクリーンアップを自動化し、ストレージスペースを削減できます。

手順

1. Red Hat Quay v2 UI ダッシュボードのナビゲーションペインで **Repositories** をクリックします。

2. イメージタグのあるリポジトリーの名前をクリックします。

3. イメージの縦の省略記号メニューをクリックし、**Change expiration** を選択します。

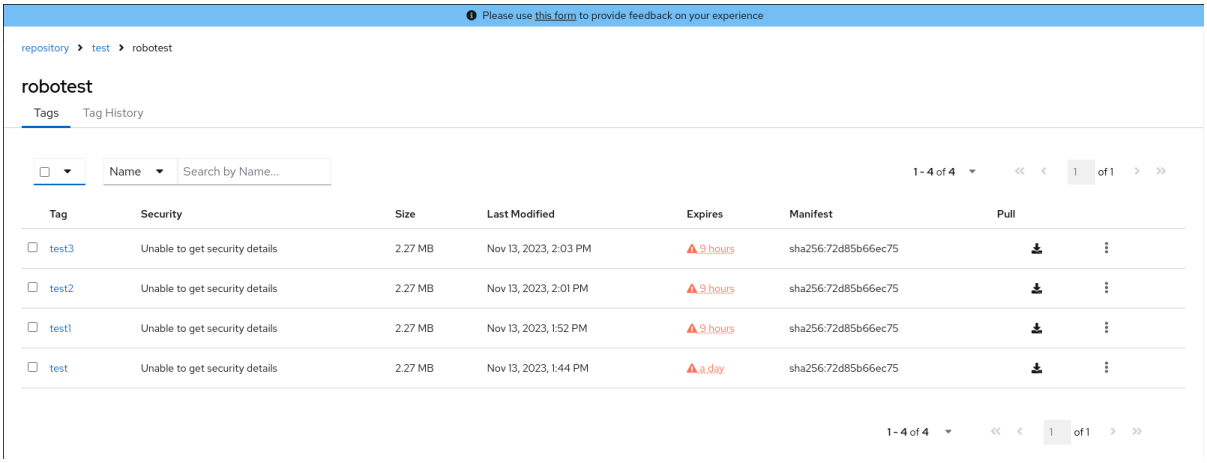
4. オプション: または、複数のタグのボックスをクリックし、**Actions** → **Set expiration** を選択して有効期限を一括追加することもできます。

5. **Change Tags Expiration** ウィンドウで、曜日、月、日、年を指定して有効期限を設定します。たとえば、**Wednesday, November 15, 2023** に設定します。または、カレンダーボタンをクリックして日付を手動で選択することもできます。

6. 時刻を **2:30 PM** などに設定します。

7. **Change Expiration** をクリックして日付と時刻を確認します。**Successfully set expiration for tag test to Nov 15, 2023, 2:26 PM** という通知が返されます。

8. Red Hat Quay v2 UI の **Tags** ページで、タグの有効期限の設定を確認できます。以下に例を示します。



14.5. RED HAT QUAY V2 UI でのカラーテーマ設定の選択

v2 UI を使用する場合、ユーザーはライトモードとダークモードの切り替えができます。この機能にはモードの自動選択も含まれ、ユーザーのブラウザーの設定に応じてライトモードまたはダークモードを選択します。

自動、ライト、およびダークモードを切り替えるには、次の手順を使用します。

手順

1. Red Hat Quay リポジトリーにログインします。
2. ナビゲーションペインで、ユーザー名 (**quayadmin** など) をクリックします。
3. **Appearance** で、**Light theme**、**Dark theme**、および **Device-based theme** のいずれかを選択します。デバイスベースのテーマは、ブラウザーのカラー設定に応じてモードを設定します。

14.6. RED HAT QUAY V2 UI での使用状況ログの表示

Red Hat Quay ログは、Red Hat Quay レジストリーの使用方法に関する貴重な情報を提供できます。ログは、以下の手順を使用して v2 UI の組織、リポジトリー、または名前空間で表示できます。

手順

1. Red Hat Quay レジストリーにログインします。
2. 自分が管理者である組織、リポジトリーまたは名前空間に移動します。
3. **Logs** をクリックします。

4. オプション: **From** ボックスと **To** ボックスに日付を追加して、ログエントリーを表示する日付範囲を設定します。

5. オプション: **Export** をクリックして、ログをエクスポートします。**http://** または**https://** で始まるメールアドレスまたは有効なコールバック URL を入力する必要があります。このプロセスは、ログの数に応じて 1 時間かかる場合があります。

14.7. レガシー UI の有効化

1. ナビゲーションペインに、**現在の UI** と **新しい UI** を切り替えるオプションが表示されます。切り替えボタンをクリックして、**Current UI** に設定します。

第15章 RED HAT QUAY API の使用

Red Hat Quay は、完全な OAuth 2 RESTful API を提供します。

- 各 Red Hat Quay インスタンスのエンドポイント (URL https://<yourquayhost>/api/v1) から利用できます。
- Swagger UI を有効にして、ブラウザー経由でエンドポイントに接続し、Red Hat Quay の設定を取得、削除、投稿、および配置できます。
- API 呼び出しを実行し、OAuth トークンを使用するアプリケーションからアクセスできます。
- JSON としてデータを送受信します。

以下のテキストは、Red Hat Quay API にアクセスし、API を使用して Red Hat Quay クラスターで設定を表示して変更する方法を説明します。次のセクションでは、API エンドポイントをリスト表示し、説明します。

15.1. QUAY.IO からの QUAY API へのアクセス

独自の Red Hat Quay クラスターがまだ実行されていない場合に、Web ブラウザーから Quay.io で利用可能な Red Hat Quay API を確認できます。

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

表示される API Explorer には Quay.io API エンドポイントが表示されます。Quay.io で有効でない Red Hat Quay 機能のスーパーユーザー API エンドポイントまたはエンドポイント (リポジトリーミラーリングなど) は表示されません。

API Explorer から、以下に関する情報を取得し、変更できます。

- 請求、サブスクリプション、およびプラン
- リポジトリービルドおよびビルドトリガー
- エラーメッセージおよびグローバルメッセージ
- リポジトリーイメージ、マニフェスト、パーミッション、通知、脆弱性、およびイメージの署名
- 使用状況に関するログ
- 組織、メンバー、および OAuth アプリケーション
- ユーザーとロボットアカウント
- その他

エンドポイントを選択して開き、エンドポイントの各部分のモデルスキーマを表示します。エンドポイントを開き、必要なパラメーター (リポジトリー名またはイメージなど) を入力し、**Try it out!** ボタンを選択して Quay.io エンドポイントに関連する設定を照会するか、変更します。

15.2. OAUTH アクセストークンの作成

OAuth アクセストークンは、保護されたリソースへのセキュアなアクセスを可能にする認証情報です。Red Hat Quay では、組織の API エンドポイントにアクセスする前に、OAuth アクセストークンを作成する必要があります。

OAuth アクセストークンを作成するには、次の手順を実行します。

前提条件

- Red Hat Quay に管理者としてログインしている。

手順

1. メインページで、Organization を選択します。
2. ナビゲーションペインで、**Applications** を選択します。
3. **Create New Application** をクリックし、新しいアプリケーション名を入力して、**Enter** を押します。
4. **OAuth Applications** ページで、アプリケーションの名前を選択します。
5. オプション: 以下の情報を入力します。
    a. **Application Name**
    b. **Homepage URL**
    c. **Description**
    d. **Avatar E-mail**
    e. **Redirect/Callback URL prefix**
6. ナビゲーションペインで、**Generate Token** を選択します。
7. 次のオプションのチェックボックスをオンにします。
    a. **Administer Organization**
    b. **Administer Repositories**
    c. **Create Repositories**
    d. **View all visible repositories**
    e. **Read/Write to any accessible repositories**
    f. **Super User Access**
    g. **Administer User**
    h. **Read User Information**
8. **Generate Access Token** をクリックします。新しいページにリダイレクトされます。
9. 許可する権限を確認し、**Authorize Application** をクリックします。**Authorize Application** をクリックして決定した内容を確定します。

10. **Access Token** ページにリダイレクトされます。アクセストークンをコピーして保存します。

重要

これは、アクセストークンをコピーして保存する唯一の機会です。このページを離れると再取得できません。

15.3. WEB ブラウザーからの QUAY API へのアクセス

Swagger を有効にし、Web ブラウザーを使用して独自の Red Hat Quay インスタンスの API にアクセスできます。この URL は、Red Hat Quay API を UI および以下の URL 経由で公開します。

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

この方法で API にアクセスしても、Red Hat Quay インストールで利用可能なスーパーユーザーエンドポイントにはアクセスできません。以下は、swagger-ui コンテナーイメージを実行してローカルシステムで実行されている Red Hat Quay API インターフェイスにアクセスする例です。

```
# 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
```

Swagger-ui コンテナーが実行された状態で、Web ブラウザーを localhost ポート 8888 で開き、swagger-ui コンテナー経由で API エンドポイントを表示します。

API calls must be invoked with an X-Requested-With header if called from a browser などのエラーを回避するには、以下の行をクラスター内の全ノードの **config.yaml** に追加し、Red Hat Quay を再起動します。

```
BROWSER_API_CALLS_XHR_ONLY: false
```

15.4. コマンドラインでの RED HAT QUAY API へのアクセス

curl コマンドを使用して、Red Hat Quay クラスターの API を使用して GET、PUT、POST、または DELETE 操作を実行できます。**<token>** は、以下の例の設定を取得または変更するために作成した OAuth アクセストークンに置き換えます。

15.4.1. スーパーユーザー情報の取得

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

以下に例を示します。

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

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

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


15.4.2. API を使用したスーパーユーザーの作成

- Quay のデプロイで説明されているようにスーパーユーザー名を設定します。
  - 設定エディター UI を使用します。または、
  - 設定 API を使用して更新された設定バンドルを検証 (およびダウンロード) して、**config.yaml** ファイルを直接編集します。
- スーパーユーザー名のユーザーアカウントを作成します。
  - 上記のように承認トークンを取得し、**curl** を使用してユーザーを作成します。

```
$ 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
```

  - 返されるコンテンツには、新規ユーザーアカウント用に生成されたパスワードが含まれます。


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


ユーザーのリストを要求すると、**quaysuper** がスーパーユーザーとして表示されます。

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

{
  "users": [
  {
    "kind": "user",
```

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


15.4.3. 使用状況ログのリスト表示

内部 API **/api/v1/superuser/logs** を使用して、現在のシステムの使用状況ログをリスト表示できます。結果はページネーションされます。以下の例では、20 以上のリポジトリーを作成し、複数の呼び出しを使用して結果セット全体にアクセスする方法を紹介しています。

15.4.3.1. ページネーションの例

最初の呼び出し

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

初期出力


```
{
  "start_time": "Sun, 12 Dec 2021 11:41:55 -0000",
  "end_time": "Tue, 14 Dec 2021 11:41:55 -0000",
  "logs": [
    {
      "kind": "create_repo",
      "metadata": {
        "repo": "t21",
```

```
      "namespace": "namespace1"
    },
    "ip": "10.131.0.13",
    "datetime": "Mon, 13 Dec 2021 11:41:16 -0000",
    "performer": {
      "kind": "user",
      "name": "user1",
      "is_robot": false,
      "avatar": {
        "name": "user1",
        "hash": "5d40b245471708144de9760f2f18113d75aa2488ec82e12435b9de34a6565f73",
        "color": "#ad494a",
        "kind": "user"
      }
    },
    "namespace": {
      "kind": "org",
      "name": "namespace1",
      "avatar": {
        "name": "namespace1",
        "hash": "6cf18b5c19217bfc6df0e7d788746ff7e8201a68cba333fca0437e42379b984f",
        "color": "#e377c2",
        "kind": "org"
      }
    }
  },
  {
    "kind": "create_repo",
    "metadata": {
      "repo": "t20",
      "namespace": "namespace1"
    },
    "ip": "10.131.0.13",
    "datetime": "Mon, 13 Dec 2021 11:41:05 -0000",
    "performer": {
      "kind": "user",
      "name": "user1",
      "is_robot": false,
      "avatar": {
        "name": "user1",
        "hash": "5d40b245471708144de9760f2f18113d75aa2488ec82e12435b9de34a6565f73",
        "color": "#ad494a",
        "kind": "user"
      }
    },
    "namespace": {
      "kind": "org",
      "name": "namespace1",
      "avatar": {
        "name": "namespace1",
        "hash": "6cf18b5c19217bfc6df0e7d788746ff7e8201a68cba333fca0437e42379b984f",
        "color": "#e377c2",
        "kind": "org"
      }
    }
  },
```

```
...

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


next_page を使用した 2 回目の呼び出し

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

2 回目の呼び出しからの出力


```
{
  "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. ディレクトリーの同期

LDAP の対応するグループ名が **ldapgroup** となる **testadminorg** 組織の **newteam** チームのディレクトリーの同期を有効にするには、以下を実行します。

```
$ 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
```

同じチームの同期を無効にするには、以下を実行します。

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

15.4.5. API を使用したリポジトリービルドの作成

指定の入力からリポジトリーをビルドし、ビルドにカスタムタグを付けるには、requestRepoBuild エンドポイントを使用できます。以下のデータを使用できます。

```
{
"docker_tags": [
```

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

archive_url パラメーターは、Dockerfile とビルドに必要な他のファイルが含まれる **tar** または **zip** アーカイブを参照する必要があります。**file_id** パラメーターは、以前のビルドシステムに含まれていましたが、これは今後使用できません。Dockerfile がサブディレクトリーにある場合は、これも指定する必要があります。

アーカイブは一般に公開されている必要があります。組織の管理者のみがロボットのアカウントトークンにアクセスできるため、OAuth アプリには Administer Organization スコープが必要です。そうしないと、誰かがロボットに対するビルドアクセスを割り当てるだけで、ロボットパーミッションを取得し、そのパーミッションを使用してイメージコンテンツを取得できるようになります。エラーが発生した場合は、返される json ブロックを確認し、アーカイブの場所、プルロボット、およびその他のパラメーターが正しく指定されていることを確認します。個別のビルドページの右上にある Download logs をクリックし、詳細なメッセージがないかログを確認します。

15.4.6. 組織のロボットの作成

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

15.4.7. ビルドのトリガー

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

要求のある 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. プライベートリポジトリーの作成

```
$ 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. ミラーリングされたリポジトリーの作成

最小設定

```
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
```

拡張設定

```
$ 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
```