Red Hat Quay 3.8

Red Hat Quay の使用

Red Hat Quay の使用

Last Updated: 2024-01-16

Red Hat Quay 3.8 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

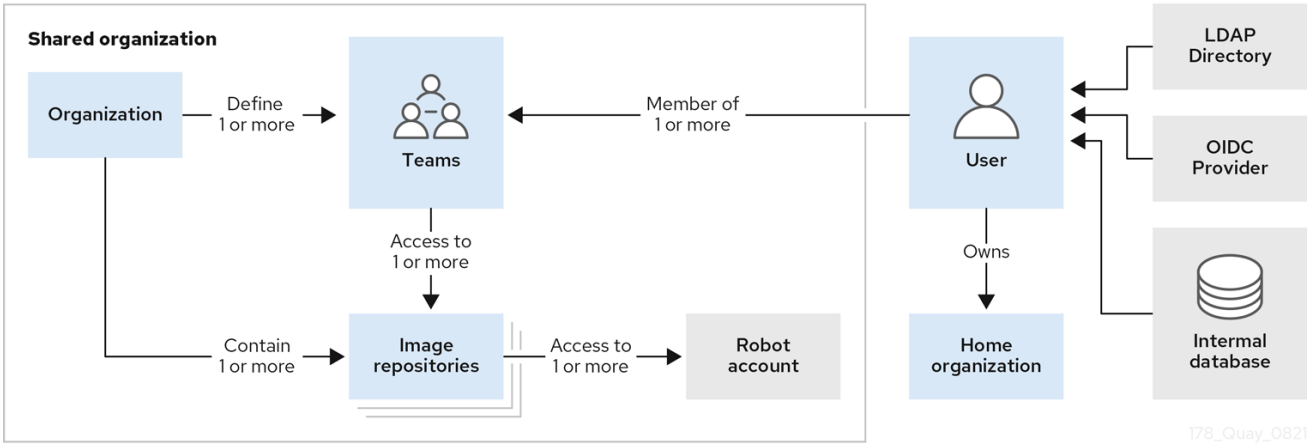
Red Hat Quay コンテナーイメージレジストリーでは、コンテナーイメージを中央の場所に保存できます。Red Hat Quay レジストリーの通常のユーザーとして、イメージを整理するためのリポジトリーを作成し、自分が管理するリポジトリーへの読み取り (プル) と書き込み (プッシュ) のアクセスを選択的に追加することができます。管理者権限を持つユーザーは、ユーザーの追加やデフォルト設定の制御など、より広範なタスクを実行することができます。

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 RED HAT QUAY のユーザーおよび組織

Red Hat Quay でコンテナーイメージを保持するためのリポジトリーの作成を開始する前に、そのリポジトリーをどのように整理するかを検討する必要があります。Red Hat Quay インスタンス内のすべてのリポジトリーは、組織またはユーザーのいずれかに関連付けられている必要があります。

1.1. RED HAT QUAY のテナントモデル



- **Organizations** は、共通の名前空間の下でリポジトリーを共有する方法を提供します。この名前空間は、1 人のユーザーに属するものではなく、(会社などの) 共有設定の多くのユーザーに属するものです。
- **Teams** は、組織が権限 (グローバルおよび特定のリポジトリーの両方) をユーザーのセットまたはグループに委譲する方法を提供します。
- **Users** は、Red Hat Quay の Web UI やクライアント (**podman login** など) を使用してレジストリーにログインできます。各ユーザーは自動的にユーザー namespace を取得します。たとえば、**quay-server.example.com/user/<username>** のようになります。
- **Super users** は、ユーザーインターフェイスの Super User Admin Panel や、通常のユーザーには見えないまたはアクセスできない Super User API コールを通じて、強化されたアクセスと権限を持っています。
- **Robot accounts** 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するもので、OpenShift のサービスアカウントと似た性質を持っています。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。

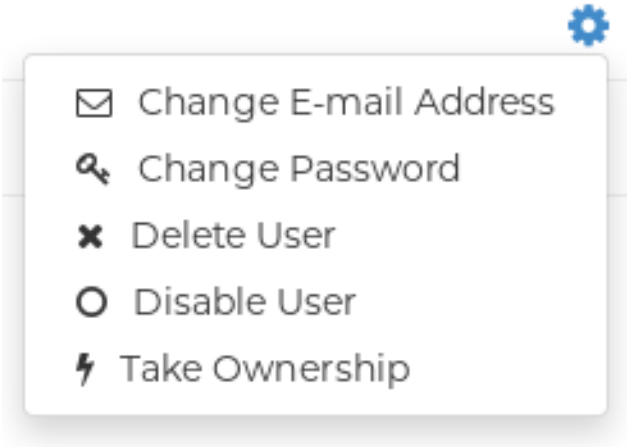
1.2. ユーザーアカウントの作成

Red Hat Quay インスタンスに新しいユーザーを作成するには、以下の手順に従います。

1. Red Hat Quay にスーパーユーザー (デフォルトでは quay) としてログインします。
2. ホームページの右上からアカウント名を選択し、Super User Admin Panel を選択します。
3. 左の列から Users アイコンを選択します。
4. Create User ボタンを選択します。

5. 新しいユーザーの Username と Email アドレスを入力し、続いて Create User ボタンを選択します。

6. Users ページに戻り、新しい Username の右側にある Options アイコンを選択します。下図のようなドロップダウンメニューが表示されます。



7. メニューから Change Password を選択します。

8. 新しいパスワードを追加して確認したら、Change User Password ボタンを選択します。

新しいユーザーは、そのユーザー名とパスワードを使用して、Web UI やコンテナークライアントを使用してログインできるようになります。

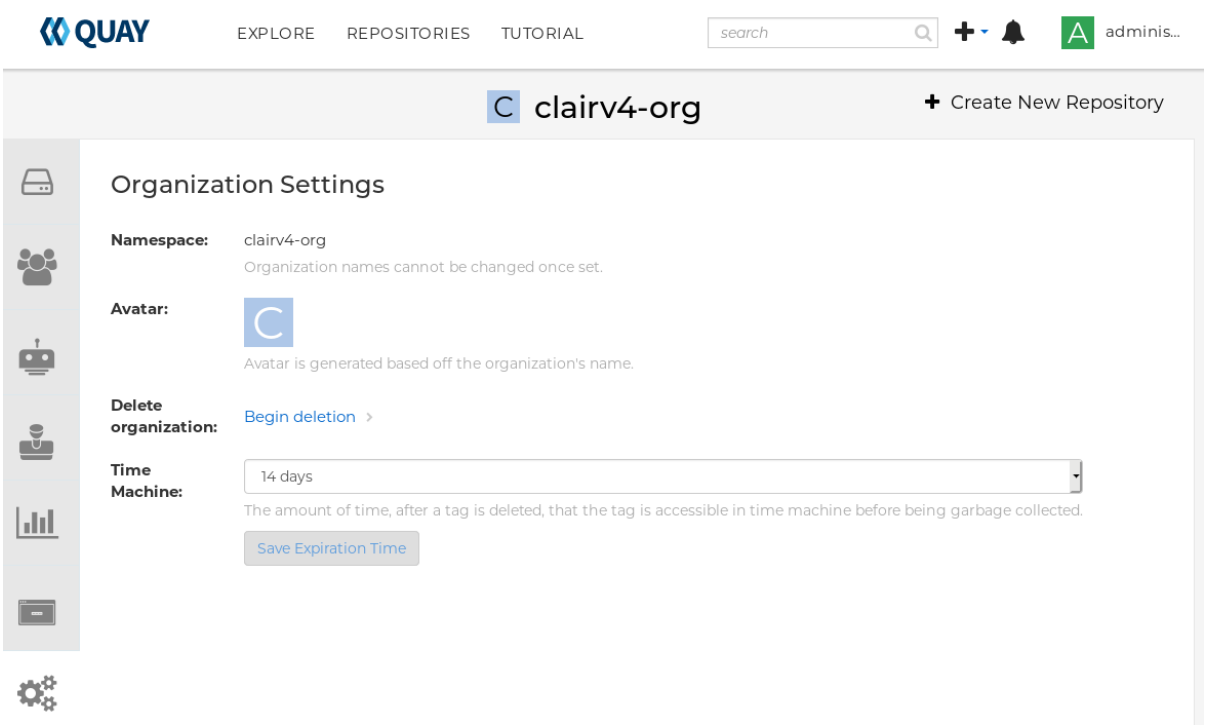
1.3. 組織アカウントの作成

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...
clairv4-org
Create New Repository
Organization Settings
Namespace:
clairv4-org
Organization names cannot be changed once set.
Avatar:
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

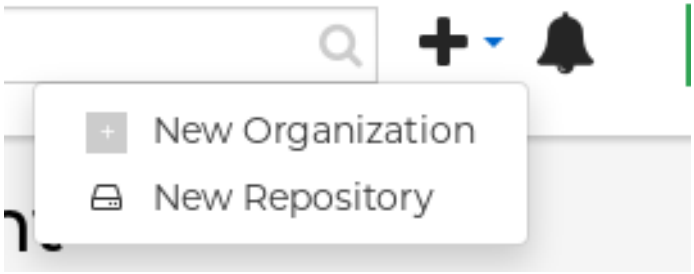
第2章 リポジトリーの作成

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保存する中心となる場所を提供します。Red Hat Quay でリポジトリーを作成するには、(**docker** や **podman** からの) プッシュによる方法と、Red Hat Quay の UI による方法の 2 つがあります。これらは基本的に同じで、Quay.io を使用するか、Red Hat Quay の独自のインスタンスを使用するかの違いです。

2.1. UI によるイメージリポジトリーの作成

ユーザーアカウントで Red Hat Quay UI でリポジトリーを作成するには、以下の手順に従います。Web UI からユーザーアカウントにログインします。次の図のように、ホームページ (またはユーザーに関連するその他のページ) のヘッダーの右上にある + アイコンをクリックし、New Repository を選択します。

+



1. 表示される Create New Repository ページで以下の操作を実施します。
   - ユーザー名に新しいリポジトリー名を追加する
   - Repository Description をクリックして、リポジトリーの説明を入力する
   - Repository Visibility で、リポジトリーを公開するか非公開にするかを選択する
   - Create Repository ボタンをクリックする

新しいリポジトリーが作成され、最初は空の状態です。このリポジトリー (イメージ名を除いたもの) からイメージをプルするのに使用できる docker pull コマンドが画面に表示されます。

組織の下で Red Hat Quay UI でリポジトリーを作成するには、以下の手順に従います。

1. 管理者権限または組織への書き込み権限を持つユーザーでログインします。
2. Repositories ビューで、Users and Organizations の右欄から組織名を選択します。図 2.x に示すような組織のページが表示されます。
3. ページの右上にある +Create New Repository をクリックします。
4. 表示される Create New Repository ページで以下の操作を実施します。
   - 組織名に新しいリポジトリー名を追加する
   - Repository Description をクリックして、リポジトリーの説明を入力する
   - Repository Visibility で、リポジトリーを公開するか非公開にするかを選択する
   - Create Repository ボタンをクリックする

新しいリポジトリーが作成され、最初は空の状態です。このリポジトリー (イメージ名を除いたもの) からイメージをプルするのに使用できる docker pull コマンドが画面に表示されます。

2.2. DOCKER または PODMAN によるイメージリポジトリーの作成

適切な認証情報があれば、Red Hat Quay インスタンスにまだ存在していないリポジトリーにイメージをプッシュすると、イメージをリポジトリーにプッシュする際にそのリポジトリーが作成されます。これらの例では、**docker** コマンドまたは **podman** コマンドのいずれかを使用できます。

1. イメージにタグを付けます。ローカルシステムで **docker** や **podman** から利用可能なイメージがあれば、そのイメージに新しいリポジトリー名とイメージ名をタグ付けします。ここでは、Quay.io や専用の Red Hat Quay セットアップ (たとえば reg.example.com) にイメージをプッシュする例を紹介します。この例では、namespace を Red Hat Quay のユーザー名または組織に、repo_name を作成するリポジトリーの名前に置き換えてください。

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

2. 適切なレジストリーにプッシュします。以下に例を示します。

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

注記

アプリケーションのリポジトリーを作成するには、コンテナーイメージのリポジトリーを作成したときと同じ手順で行います。

第3章 リポジトリーへのアクセス管理

Red Hat Quay のユーザーとして、専用のリポジトリーを作成し、Red Hat Quay インスタンスの他のユーザーがアクセスできるようにすることができます。別の方法として、組織を作成して、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両方で、ロボットアカウントに関連する認証情報を作成して、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により、Red Hat Quay のユーザーアカウントを持っていない様々なコンテナークライアント (docker や podman など) が、簡単にレポにアクセスすることができます。

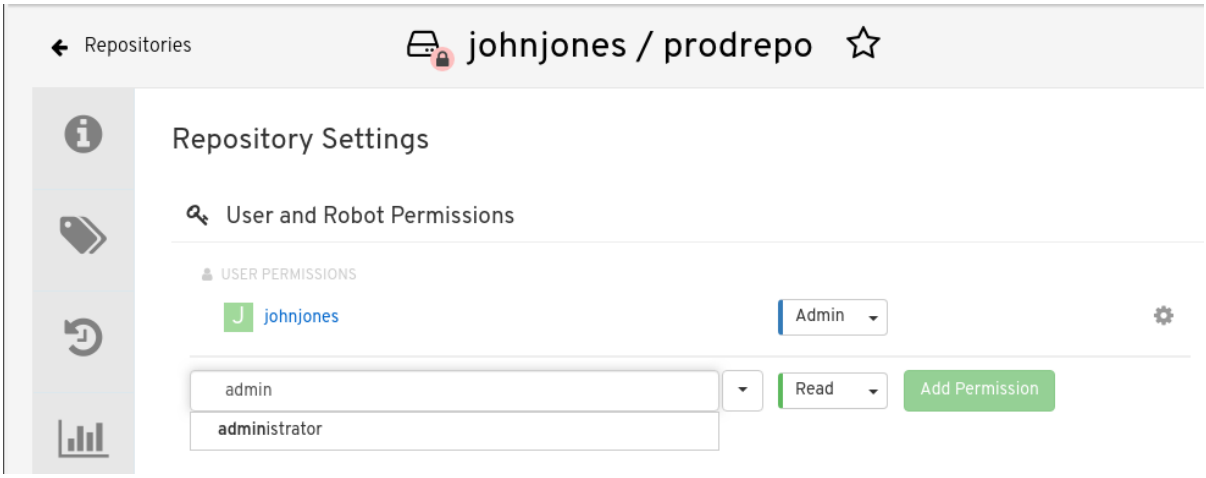
3.1. ユーザーリポジトリーへのアクセスの許可

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経由で追加できます。

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、以下のようにします。

1. Red Hat Quay のユーザーアカウントにログインします。
2. 自分のユーザー名前空間の下で、アクセスを共有したいリポジトリーを選択します。
3. 左列の Settings アイコンを選択します。
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。次の図のように、入力したユーザー名が表示されます。



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 - ユーザーがリポジトリーを表示し、そこからプルすることを許可します。
   - Write - ユーザーはリポジトリーを表示したり、リポジトリーからイメージをプルしたり、リポジトリーにイメージをプッシュしたりできます。
   - Admin - リポジトリーに対するすべての管理設定と、すべての読み取りおよび書き込み権限を許可します。
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。

リポジトリーに対するユーザーの権限を削除するには、ユーザーエントリーの右にある Options アイコンを選択し、Delete Permission を選択します。

3.2. ユーザーリポジトリーへのロボットアクセスの許可

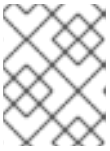
ロボットアカウントは、Red Hat Quay レジストリー内のリポジトリーへの自動アクセスを設定するのに使用されます。これらは、OpenShift のサービスアカウントに似ています。ロボットアカウントを設定する際に、以下を行います。

- ロボットアカウントに関連付けられる認証情報を生成する
- ロボットがイメージをプッシュまたはプルできるリポジトリーやイメージを特定する
- 生成された認証情報をコピー/ペーストして、異なるコンテナークライアント (Docker、podman、Kubernetes、Mesos など) で使用し、定義された各リポジトリーにアクセスする

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ることに留意してください。たとえば、ロボットは、ユーザー jsmith がアクセスできるすべてのリポジトリーへのアクセスを提供しますが、リポジトリーのユーザーリストにないユーザーにはアクセスを提供しません。

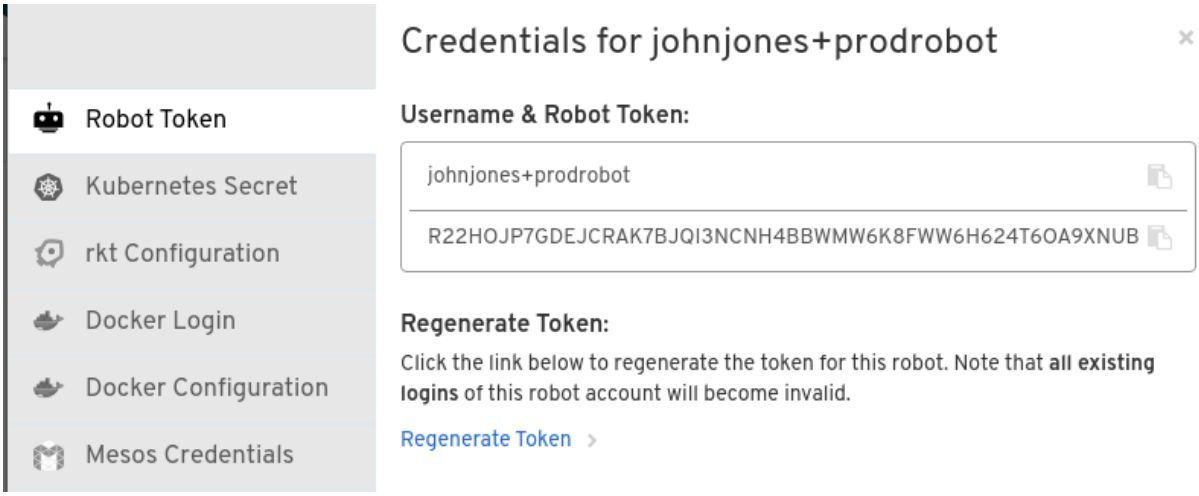
以下の手順では、リポジトリーへのアクセスを許可するロボットアカウントの設定方法を順を追って説明します。

1. Robot アイコンを選択します。Repositories ビューで、左の列から Robot アイコンを選択します。
2. ロボットアカウントを作成します。Create Robot Account ボタンを選択します。
3. ロボット名を設定します。名前と説明を入力し、Create robot account ボタンを選択します。ロボット名は、ご自分のユーザー名と、設定したロボット名の組み合わせになります (例: jsmith+myrobot)。
4. ロボットアカウントに権限を追加します。ロボットアカウントの Add permissions 画面から、ロボットにアクセスさせたいリポジトリーを以下のように定義します。
   - ロボットがアクセスできる各リポジトリーにチェックマークを入れます。
   - 各リポジトリーについて、以下のいずれかを選択し、Add permissions をクリックします。
     - None - ロボットにはリポジトリーに対する権限がありません。
     - Read - ロボットがリポジトリーを閲覧し、プルすることができます。
     - Write - ロボットは、リポジトリーからの読み込み (プル) と、リポジトリーへの書き込み (プッシュ) が可能です。
     - 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ことができます。
   - Add permissions ボタンを選択し、設定を適用します。
5. ロボットを使用してリポジトリーにアクセスするための認証情報を取得します。Robot Accounts ページに戻り、ロボットのアカウント名を選択すると、そのロボットの認証情報が表示されます。
6. トークンを取得します。次の図のように Robot Token を選択すると、ロボットに生成されたトークンが表示されます。トークンをリセットしたい場合は、Regenerate Token を選択します。



注記

トークンを再生成すると、そのロボットの過去のトークンが無効になることを理解しておくことが重要です。



7. 認証情報を取得します。生成されたトークンで問題なければ、以下の方法で認証情報を取得します。

    - Kubernetes Secret: Kubernetes プルシークレットの yaml ファイルの形で認証情報をダウンロードする場合は、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場合は、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場合は、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をダウンロードして、クライアントシステムに認証情報を恒久的に保存する場合は、これを選択します。
    - Mesos Credentials: Mesos 設定ファイルの uris フィールドで特定できる認証情報を提供する tarball をダウンロードする場合は、これを選択します。

3.3. 組織のリポジトリーへのアクセスの許可

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。その組織内のリポジトリーへのアクセスを追加するには、チーム (同じ権限を持つユーザーのセット) と個々のユーザーを追加します。基本的に、組織はユーザーと同じようにリポジトリーやロボットアカウントを作成する機能を持っていますが、組織は (チームまたは個人の) ユーザーのグループを通じて共有リポジトリーを設定することを目的としています。

その他、組織について知っておくべきこと:

- 他の組織の中に組織を持つことはできません。組織を細分化するには、チームを使用します。
- 組織に直接ユーザーを含めることはできません。まずチームを追加し、次に各チームに 1 人または複数のユーザーを追加する必要があります。
- チームは、組織の中で、レポジトリーや関連するイメージを使用するただのメンバーとして、または組織を管理する特別な権限を持つ管理者として設定できます。

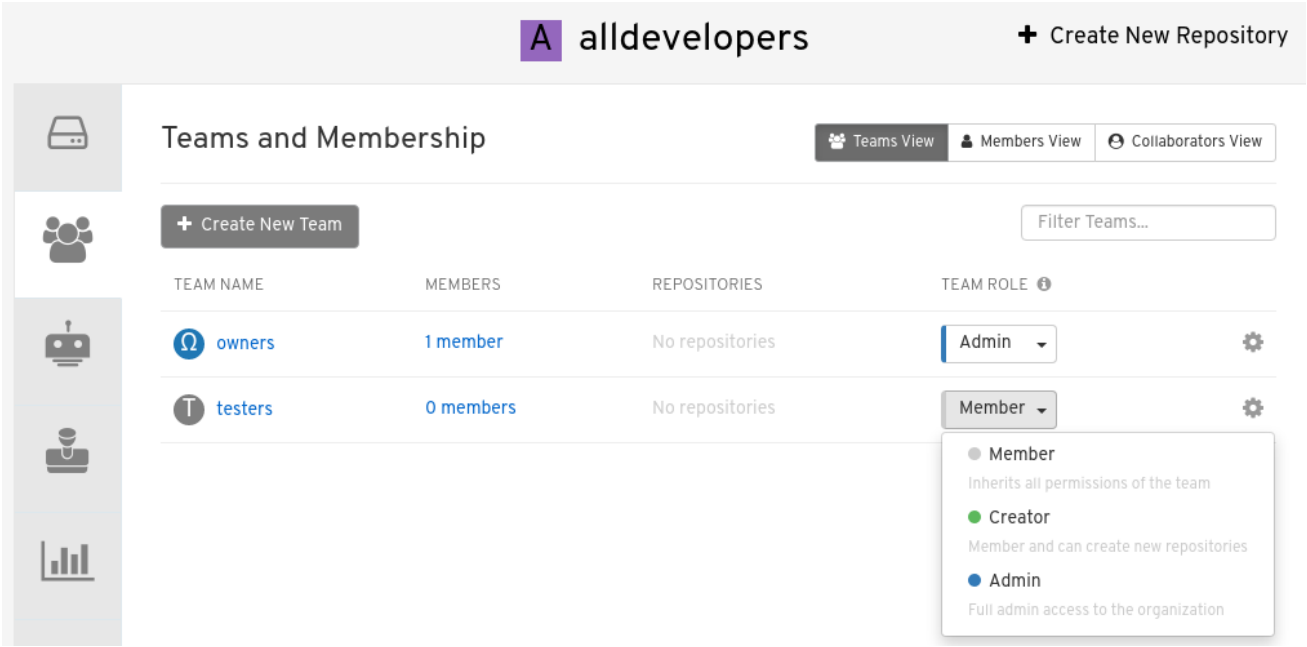
3.3.1. 組織へのチームの追加

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。

1. Organization ビューで、左側の列から Teams and Membership アイコンを選択します。組織を作成したユーザーの管理者権限を持つ所有者の Team が存在していることがわかります。
2. Create New Team を選択します。組織に関連付ける新しいチーム名の入力を求められます。チーム名を入力してください。チーム名は必ず小文字で始まり、残りの部分は小文字と数字の任意の組み合わせです (大文字や特殊文字は使用できません)。
3. Create team ボタンを選択します。Add permissions ウィンドウが表示され、組織内のリポジトリーの一覧が表示されます。
4. チームがアクセスできるようにしたい各リポジトリーにチェックを入れます。続いて、それぞれに以下の権限のいずれかを選択します。
   - Read - チームメンバーはイメージを閲覧し、プルすることができます。
   - Write - チームメンバーがイメージを閲覧、プル、プッシュことができます。
   - Admin - チームメンバーは完全な読み取り/書き込み権限に加えて、リポジトリーに関連する管理タスクを実行することができます。
5. 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。

3.3.2. 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することができるようになります。組織内の Teams and Membership 画面から、下図のように TEAM ROLE ドロップダウンメニューを選択します。



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持ちます。

- Admin - チームの作成、メンバーの追加、権限の設定機能など、組織への完全な管理アクセスがあります。

3.3.3. チームへのユーザーの追加

組織の管理者権限を持つ者として、ユーザーやロボットをチームに追加できます。ユーザーを追加すると、そのユーザーにメールが送信されます。そのユーザーが招待を受け入れるまで、保留状態が続きます。

ユーザーやロボットをチームに追加するには、組織の画面から以下の操作を行います。

1. ユーザーやロボットを追加したいチームを選択します。
2. Team Members ボックスに以下のいずれかを入力します。
   - Red Hat Quay 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電子メールアドレス
   - ロボットのアカウントの名前。名前は、組織名 + ロボット名の形式でなければなりません。
3. ロボットアカウントの場合は、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は、参加の招待状がユーザーに送付されます。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 の状態のままです。

次に、ユーザーはチームに参加する招待メールを受け入れます。ユーザーが次回 Red Hat Quay インスタンスにログインすると、ユーザーは組織の INVITED TO JOIN リストから MEMBERS リストに移動します。

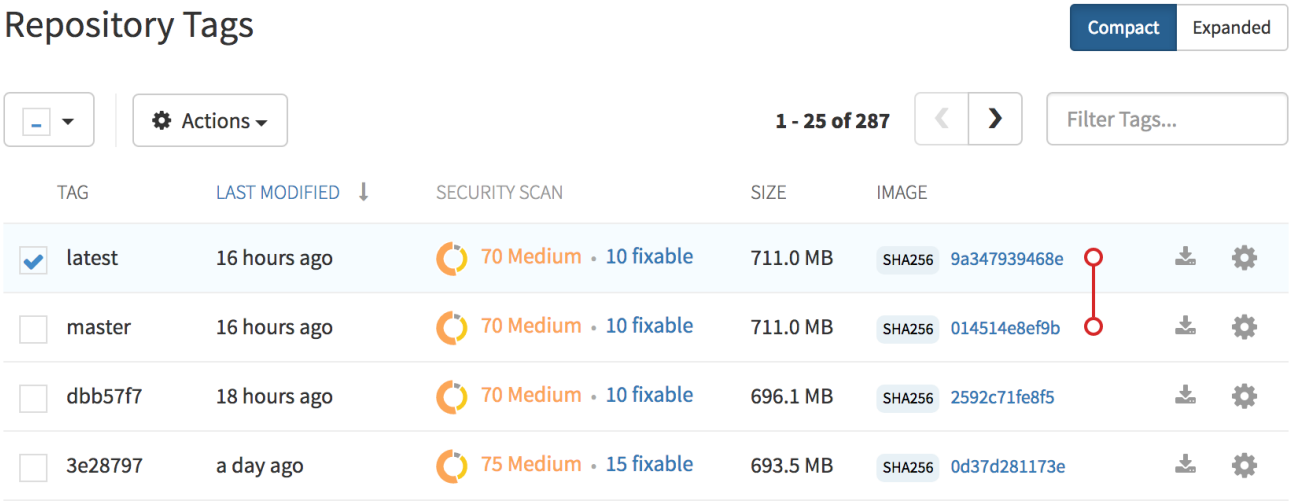
第4章 タグの使用

タグを使用すると、イメージのバージョンを識別できると同時に、同じイメージに異なる名前を付けることもできます。イメージのバージョン以外にも、イメージタグはその用途 (デビル、テスト用、プロダクションなど) や、最新バージョン (latest) であることを識別することができます。

イメージリポジトリーの **Tags** タブでは、タグの表示、変更、追加、移動、削除、履歴の確認ができます。また、さまざまなコマンドを使用して、特定のイメージを (その名前とタグに基づいて) ダウンロード (プル) するのに使用するコマンドラインを取得することができます。

4.1. タグの表示と変更

リポジトリーのタグは、**Tags** タブをクリックして表示されるリポジトリーページのタグパネルで表示および変更できます。



4.1.1. タグの付いたイメージへの新しいタグの追加

タグの付いたイメージに新しいタグを追加するには、タグの横にある歯車のアイコンをクリックして **Add New Tag** を選択します。Red Hat Quay は、イメージへの新しいタグの追加を確認します。

4.1.2. タグの移動

タグを別のイメージに移動させるには、新しいタグを追加するのと同じ操作を行います。ただし、既存のタグ名を指定します。Red Hat Quay は、タグを追加するのではなく、移動させることを確認します。

4.1.3. タグの削除

タグの歯車アイコンをクリックして **Delete Tag** を選択すると、特定のタグとそのイメージをすべて削除できます。これにより、タグと、そのタグに固有のイメージがすべて削除されます。イメージは、直接または親子関係で間接的に参照するタグがなくなるまで削除されません。

4.1.4. タグの履歴の表示および操作の取り消し

4.1.4.1. タグの履歴の表示

タグのイメージ履歴を表示するには、**Actions** メニューの下にある **View Tags History** メニュー項目をクリックします。表示されるページには、タグが過去にポイントした各イメージと、そのイメージをポイントした時期が表示されます。

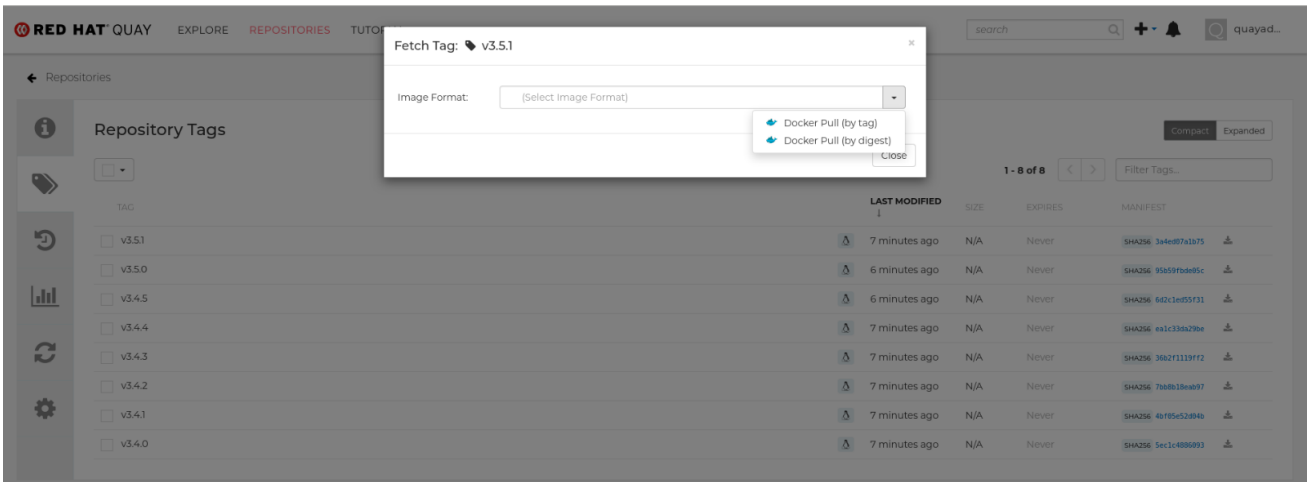
4.1.4.2. 操作の取り消し

タグを以前のイメージに戻すには、目的のイメージが上書きされた履歴行を探し、Restore のリンクをクリックします。

4.1.5. タグやダイジェストによるイメージの取得

Tags タブからは、イメージを使用する準備ができているクライアントからイメージをプルするさまざまな方法を見ることができます。

1. 特定のリポジトリー/イメージを選択します。
2. 左列の Tags を選択します。
3. 特定のイメージ/タグの組み合わせの Fetch Tag アイコンを選択します。
4. Fetch Tag ポップアップが表示された際に、Image format ボックスを選択すると、イメージをプルするためのさまざまな方法を示すドロップダウンメニューが表示されます。選択肢には、特定のコンテナーイメージをローカルシステムにプルするための完全なコマンドラインが用意されています。



docker コマンドを使用して、タグ名またはダイジェスト名で通常のイメージをプルできます。プルの種類を選び、続いて **Copy Command** を選択します。コマンドラインの全文がクリップボードにコピーされます。この 2 つのコマンドは、タグとダイジェストによる **docker pull** を示しています。

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

このコマンドを、**docker** コマンドおよびサービスが利用できるシステム上のコマンドラインシェルに貼り付けて、Enter キーを押します。この時点で、コンテナーイメージがローカルシステム上で実行できるようになります。

RHEL および Fedora システムでは、**podman** を **docker** の代わりに使用して、選択したイメージをプルして実行できます。

4.2. タグの有効期限

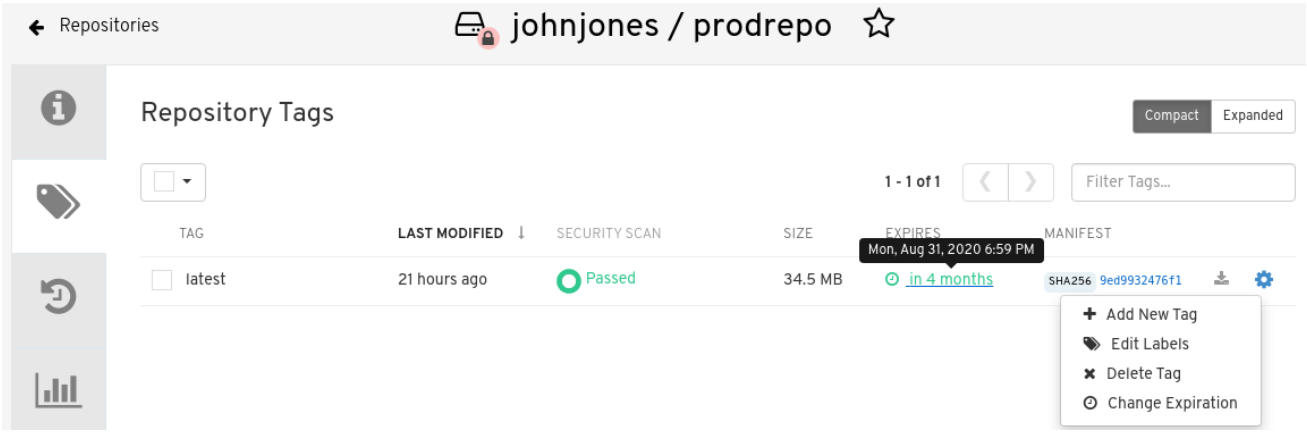
イメージは、**tag expiration** と呼ばれる機能を使用して、指定した日時に Red Hat Quay リポジトリーから期限切れになるように設定できます。ここでは、タグの有効期限について知っておきたいことをご紹介します。

- タグの有効期限が切れると、そのタグは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、そのイメージは削除されるように設定されています。
- 有効期限は、リポジトリー全体ではなく、タグごとに設定されます。
- タグの有効期限が切れたり、削除されたりしても、すぐにはレジストリーから削除されません。(User 設定の) Time Machine の値は、削除されたタグが実際に削除され、ガベージコレクションされるタイミングを定義します。デフォルトでは、その値は 14 日です。それまでは、期限切れのイメージや削除されたイメージにタグを再度参照できます。
- 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。有効期限や最終的な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。

タグの有効期限はさまざまな方法で設定できます。

- 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。
- リポジトリータグの EXPIRES 列から有効期限を選択し、有効期限の特定の日時を指定する方法。

次の図は、タグの有効期限を変更するための Options エントリーと、タグの有効期限を設定する EXPIRES フィールドを示しています。EXPIRES フィールドにカーソルを合わせると、現在設定されている有効期限の日時が表示されます。



4.2.1. Dockerfile からのタグの有効期限の設定

Dockerfile の LABEL コマンドで **quay.expires-after=20h** のようなラベルを追加すると、タグは指定された時間後に自動的に期限切れとなります。時間の値は、イメージがビルドされてからの時間、日、週をそれぞれ表す **1h**、**2d**、**3w** のようになります。

4.2.2. リポジトリーからのタグの有効期限の設定

Repository Tag ページには、タグの有効期限を示す **EXPIRES** というタイトルの UI カラムがあります。ユーザーは、有効期限が切れる時間をクリックするか、右の Settings ボタン (歯車のアイコン) をクリックして **Change Expiration** を選択することで、これを設定できます。

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。タグは、有効期限に達すると、リポジトリーから削除されるように設定されます。

4.3. セキュリティースキャン

タブの横に表示されている脆弱性や修正可能な数をクリックすると、そのタグのセキュリティースキャン情報にジャンプできます。このページでは、お客様のイメージがどの CVE に影響されやすいか、どのような修復オプションがあるかを確認できます。

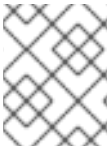
イメージスキャンでは、Clair イメージスキャナーで発見された脆弱性のみが掲載されていることに留意してください。発見された脆弱性に対してどのように対処するかは、それぞれのユーザーに委ねられています。Red Hat Quay のスーパーユーザーは、発見されたこれらの脆弱性に対処しません。

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 (ユーザーと組織) について収集されます。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 を利用してログエントリーにアクセスする

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。



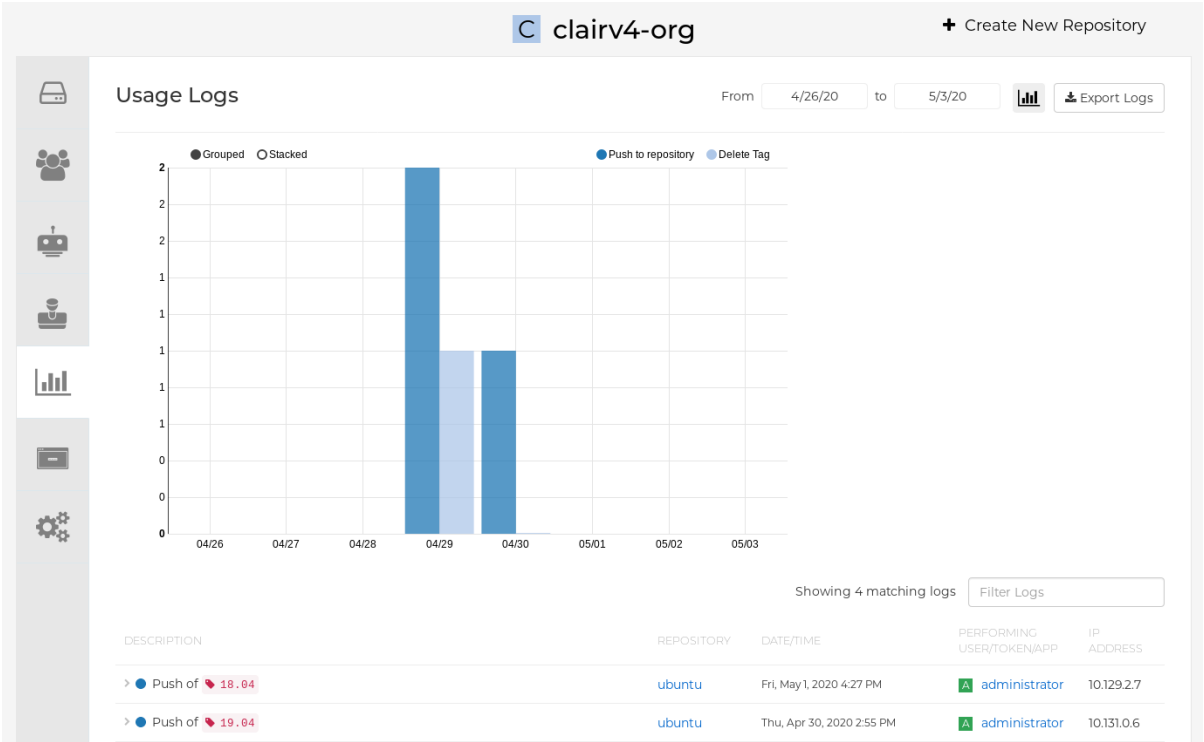
注記

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. ログの表示

Web UI からリポジトリーや名前空間のログエントリーを表示するには、次のようにします。

1. 管理者権限のあるリポジトリーや名前空間 (組織やユーザー) を選択します。
2. 左の列から Usage Logs アイコンを選択します。次の図のような Usage Logs 画面が表示されます。



3. Usage Logs ページでは、以下のことができます。
   - From および 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の範囲を設定。デフォルトでは、直近の 1 週間分のログエントリーが表示されます。
   - Filter Logs ボックスに文字列を入力して、指定した文字列が含まれるログエントリーを表示。

- 各ログエントリーの左にある矢印を切り替えて、そのログエントリーに関連するテキストの表示を増減。

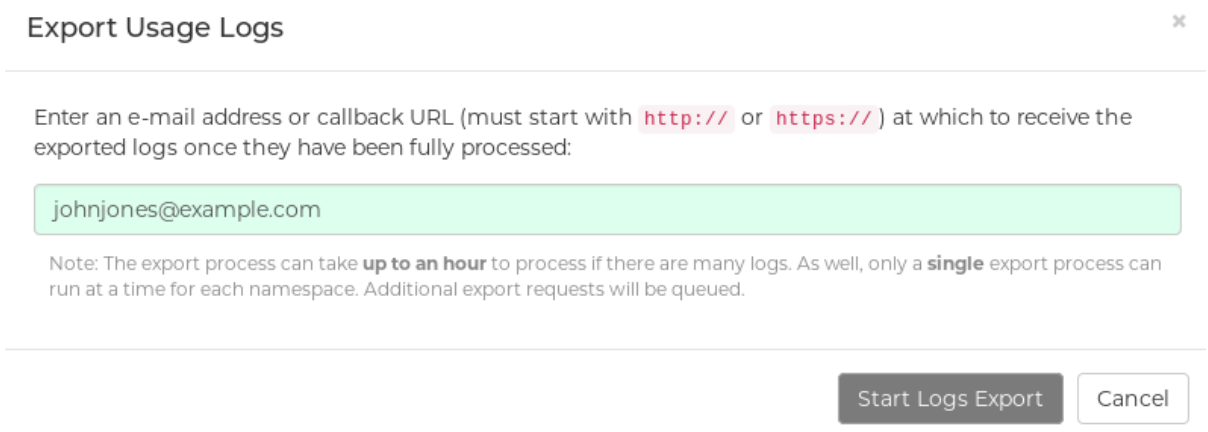
5.2. リポジトリーログのエクスポート

より多くのログファイルを取得して Red Hat Quay データベースの外に保存するには、ログのエクスポート機能を使用できます。ここでは、Export Logs を使用する上で知っておくべきことをいくつかご紹介します。

- 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。
- ログをメールに添付して送ったり、コールバック URL に誘導したり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権限が必要です。
- 一度に最大 30 日分のログデータをエクスポートできます。
- Export Logs では、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 インスタンスが外部ストレージ用に設定されている必要があります (ローカルストレージは使用できません)。
- ログが収集され利用できるようになったら、そのデータを保存したい場合はすぐにコピーしてください。デフォルトでは、データの有効期限は 1 時間となっています。

ログのエクスポート機能を使用するには、以下の手順を行います。

1. 管理者権限のあるリポジトリーを選択します。
2. 左の列から Usage Logs アイコンを選択します。Usage Logs 画面が表示されます。
3. 収集したいログエントリーの開始日と終了日の範囲を選択します。
4. Export Logs ボタンを選択します。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、webhook.site のような場所への URL を使用できます。
6. Start Logs Export を選択します。これにより、Red Hat Quay は選択したログエントリーの収集を開始します。収集するログデータの量にもよりますが、1 分から 1 時間程度で完了します。
7. ログのエクスポートが完了すると、以下のようになります。

- 要求したエクスポートされたログエントリーが利用可能になったことを通知するメールを受信します。

- webhook URL からログエクスポートのリクエストが成功したことを確認できます。エクスポートされたデータへのリンクが表示されるため、選択してログをダウンロードしてください。

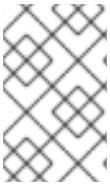
この URL は Red Hat Quay の外部ストレージの場所を指しており、1時間で期限切れになるように設定されていることに注意してください。そのため、エクスポートしたログを保存する場合は、その有効期限までにコピーしてください。

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ビルドできます。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 1 つまたは複数の OpenShift/K8s クラスターをセットアップする方法を、順を追って説明します。Red Hat Quay 3.4 では、Red Hat Quay の Python 2 から Python 3 への移行の一環として、基盤となる Build Manager が完全に書き直されました。その結果、Red Hat Quay 3.3 以前では継続的に動作していたビルダーノードが、Kubernetes のジョブとして動的に作成されるようになりました。これにより、Red Hat Quay がビルドを管理する方法が大幅に簡素化され、毎日何千ものコンテナーイメージビルドを処理するのに quay.io が利用しているのと同じメカニズムが提供されます。現在、静的なメカニズム (Red Hat Quay 3.3 での Enterprise ビルダー) を運用しているお客様は、Kubernetes ベースのビルドメカニズムに移行する必要があります。

6.1. アーキテクチャーの概要

Red Hat Quay Build システムはスケーラビリティーを考慮して設計されています (quay.io にすべてのビルドをホストするために使用されているため)。Red Hat Quay の Build Manager コンポーネントは、ビルド要求を追跡し、ビルドエクゼキューター (OpenShift/K8s クラスター) が各要求を実行することを保証するオーケストレーション層を提供します。各ビルドは Kubernetes ジョブによって処理されます。ジョブは、小さな仮想マシンを起動してイメージのビルドプロセスを完全に分離して格納します。これにより、コンテナーのビルドが相互に影響したり、基盤となるビルドシステムに影響を与えたりすることはありません。複数のエクゼキューターを設定することで、インフラストラクチャーに障害が発生した場合でも確実にビルドを実行できます。Red Hat Quay は、あるエクゼキューターが問題を抱えていることを検知すると、自動的に別のエクゼキュータにビルドを送ります。



注記

Red Hat Quay のアップストリームバージョンでは、AWS/EC2 ベースのエグゼキューターを設定する方法を説明します。この設定は、Red Hat Quay のお客様にはサポートされていません。

6.1.1. ビルドマネージャー

ビルドマネージャーは、スケジュールされたビルドのライフサイクルに責任を持ちます。ビルドキュー、ビルドフェーズ、実行中のジョブの状態を更新する必要がある操作は、ビルドマネージャーが行います。

6.1.2. ビルドワーカーのコントロールプレーン

ビルドのジョブは別々のワーカーノードで実行され、別々のコントロールプレーン (エグゼキューター) でスケジューリングされます。現在、Red Hat Quay は AWS と Kubernetes 上でのジョブの実行をサポートしています。ビルドは quay.io/quay/quay-builder を使用して実行されます。AWS では、EC2 インスタンス上でビルドがスケジューリングされます。k8s では、ビルドはジョブリソースとしてスケジューリングされます。

6.1.3. オーケストレーター

オーケストレーターは、現在実行中のビルドジョブの状態を保存し、ビルドマネージャーが消費するイベントを発行するために使用されます (例: 期限切れイベント)。現在、サポートされているオーケストレーターのバックエンドは Redis です。

6.2. OPENSHIFT の要件

Red Hat Quay のビルドは、Kubernetes と OpenShift 4.5 以降でサポートされています。ビルド Pod には kvm 仮想化を実行する機能が必要なため、ベアメタル (非仮想化) ワーカーノードが必要です。各ビルドはエフェメラル仮想マシンで行われ、ビルド実行中の完全な分離性とセキュリティーが確保されます。さらに、OpenShift クラスターでは、特権コンテナーをサポートするのに必要な SecurityContextConstraint で実行するために、Red Hat Quay のビルドに関連付けられた ServiceAccount を許可する必要があります。

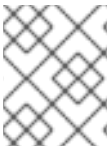
6.3. オーケストレーターの要件

Red Hat Quay のビルドでは、ビルドのステータス情報を追跡するために Redis インスタンスへのアクセスが必要です。Red Hat Quay のインストール時に既にデプロイされている同じ Redis インスタンスを使用しても構いません。すべてのビルドキューは Red Hat Quay データベースで管理されるため、高可用性のある Redis インスタンスは必要ありません。

6.4. RED HAT QUAY ビルダーと OPENSHIFT のセットアップ

6.4.1. OpenShift 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と、Red Hat Quay 3.7 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]
```

6.4.2. Red Hat Quay のビルド用 OpenShift の準備

OpenShift クラスターが Red Hat Quay からのビルドを受け入れる前に、OpenShift クラスター上で必要なアクションがいくつかあります。

1. ビルドを実行するプロジェクトを作成します (例: 'builder')。

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

2. ビルドの実行に使用する **ServiceAccount** をこの **Project** に作成します。**Jobs** や **Pod** を作成するのに十分な権限を持っていることを確認してください。後で使用するために、**ServiceAccount** のトークンをコピーします。

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

3. OpenShift クラスターの API サーバーの URL を特定します。これは、OpenShift Console から確認できます。

4. **Jobs** のビルドをスケジューリングする際に使用するワーカーノードのラベルを特定します。ビルド Pod 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。

5. クラスターが自己署名証明書を使用している場合は、kube apiserver の CA を取得して Red Hat Quay の追加証明書に追加します。

   a. CA が含まれる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 のコンソールで、secret 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. ConfigTool を使用して Red Hat Quay に CA をインポートします。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てください。

6. **ServiceAccount** に必要なセキュリティーコンテキスト/ロールバインディングを作成します。

```
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
---
```

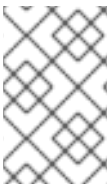
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: quay-builder-scc
  namespace: builder
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - quay-builder
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: 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
```

6.4.3. ビルダーの有効化および Red Hat Quay 設定バンドルへのビルド設定の追加

1. Red Hat Quay の設定で Builds が有効になっていることを確認してください。

```
FEATURE_BUILD_SUPPORT: True
```

1. Red Hat Quay の設定バンドルに以下を追加し、各値をインストールに固有の値で置き換えます。



注記

現在、Red Hat Quay Config Tool を介して有効にできるのは Build 機能自体のみです。Build Manager と Executor の実際の設定は、config.yaml ファイルで手動で行う必要があります。

```
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
```

各設定項目の説明は以下のとおりです。

ALLOWED_WORKER_COUNT

Red Hat Quay Pod ごとにインスタンス化される Build Worker の数を定義します。通常、これは '1' です。

ORCHESTRATOR_PREFIX

すべての Redis キーに追加される一意の接頭辞を定義します (Orchestrator の値を他の Redis キーから分離するのに便利です)。

REDIS_HOST

Redis サービスのホスト名。

REDIS_PASSWORD

Redis サービスに認証されるためのパスワード。

REDIS_SSL

Redis の接続に SSL を使用するかどうかを定義します。

REDIS_SKIP_KEYSPACE_EVENT_SETUP

デフォルトでは、Red Hat Quay はランタイム時のキーイベントに必要なキースペースイベントを設定しません。これを行うには、REDIS_SKIP_KEYSPACE_EVENT_SETUP を **false** に設定します。

EXECUTOR

このタイプのエグゼキュータの定義を開始します。有効な値は 'kubernetes' および 'ec2' です。

BUILDER_NAMESPACE

Red Hat Quay のビルドが行われる Kubernetes 名前空間。

K8S_API_SERVER

ビルドが行われる OpenShift クラスターの API サーバーのホスト名。

K8S_API_TLS_CA

API 呼び出しの実行時に Quay アプリケーションが信頼するビルドクラスターの CA 証明書の **Quay** コンテナーのファイルパス。

KUBERNETES_DISTRIBUTION

使用している Kubernetes の種類を示します。有効な値は'openshift' および 'k8s' です。

CONTAINER_*

各ビルド Pod のリソース要求および制限を定義します。

NODE_SELECTOR_*

ビルド Pod がスケジューリングされるノードセレクターラベル名と値のペアを定義します。

CONTAINER_RUNTIME

ビルダーが **docker** と **podman** のどちらを実行するかを指定します。Red Hat の **quay-builder** イメージを使用しているお客様は、これを **podman** に設定してください。

SERVICE_ACCOUNT_NAME/SERVICE_ACCOUNT_TOKEN

ビルド Pod で使用されるサービスアカウント名/トークンを定義します。

QUAY_USERNAME/QUAY_PASSWORD

WORKER_IMAGE フィールドで指定された Red Hat Quay ビルドワーカーイメージをプルするために必要なレジストリー認証情報を定義します。お客様は、registry.redhat.io に対して https://access.redhat.com/RegistryAuthentication の記事のレジストリーサービスアカウントの作成セクションで定義されている Red Hat Service Account の認証情報を提供する必要があります。

WORKER_IMAGE

Red Hat Quay ビルダーイメージのイメージ参照 (registry.redhat.io/quay/quay-builder)。

WORKER_TAG

希望するビルダーイメージのタグ。最新バージョンは v3.4.0 です。

BUILDER_VM_CONTAINER_IMAGE

各 Red Hat Quay ビルドの実行に必要な内部仮想マシンを保持するコンテナーイメージの完全な参照 (**registry.redhat.io/quay/quay-builder-qemu-rhcos:v3.4.0**)。

SETUP_TIME

ビルドがまだ Build Manager に登録されていない場合に、タイムアウトする秒数を指定します (デフォルトは 500 秒)。タイムアウトしたビルドは、3 回再起動が試みられます。3 回試してもビルドが登録されない場合は、失敗とみなされます。

MINIMUM_RETRY_THRESHOLD

この設定は、複数のエグゼキューターで使用されます。別のエグゼキューターが選択されるまでに、ビルドの開始を何回再試行するかを示します。0 に設定すると、ビルドジョブの試行回数に制限はありません。この値は意図的に小さく (3 以下) しておくことで、インフラストラクチャーに障害が発生した際にも迅速にフェイルオーバーを行うことができます。この設定には値を指定する必要があります。たとえば、Kubernetes を第 1 のエグゼキューター、EC2 を第 2 のエグゼキューターとして設定します。ジョブ実行の最後の試行を常に Kubernetes ではなく EC2 で実行したい場合は、Kubernetes のエグゼキューターの **MINIMUM_RETRY_THRESHOLD** を 1 に、EC2 の **MINIMUM_RETRY_THRESHOLD** を 0 に設定します (設定していない場合はデフォルトで 0 になります)。この場合、kubernetes の **MINIMUM_RETRY_THRESHOLD** > retries_remaining(1) は False と評価され、設定された 2 番目のエクゼキュータにフォールバックされます。

SSH_AUTHORIZED_KEYS

ignition 設定でのブートストラップする ssh キーのリスト。これにより、他の鍵を使用して EC2 インスタンスや QEMU 仮想マシンに ssh 接続できます。

6.5. OPENSHIFT ルートの制限

注記

このセクションは、マネージド **route** コンポーネントを持つ OpenShift 上で Quay Operator を使用している場合にのみ適用されます。

OpenShift **Routes** では、単一のポートにしかトラフィックを提供できないという制限があるため、ビルドの設定には追加の手順が必要です。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て、**QuayRegistry** が存在することを確認します (ビルダーが動作するベアメタルクラスターと同じで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。
- Quay Operator は、既存の Quay Pod 内で稼働しているビルドマネージャーサーバーに gRPC トラフィックを誘導する **Route** を作成します。カスタムホスト名 (**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** の **status.ingress[0].host** を指定する CNAME レコードを DNS プロバイダーで作成するようにしてください。

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

- OpenShift UI または CLI を使用して、**QuayRegistry** の **spec.configBundleSecret** で参照される **Secret** をビルドクラスター CA 証明書で更新し (キーに **extra_ca_cert_build_cluster.cert** という名前を付けます)、**config.yaml** エントリーを、**BUILDMAN_HOSTNAME** フィールドと上記のビルダー設定で参照される正しい値で更新します (ビルドエクゼキューターに依存します)。

```
BUILDMAN_HOSTNAME: <build-manager-hostname>
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
    ...
```

追加の設定項目の説明は以下のとおりです。

BUILDMAN_HOSTNAME

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** (例: Openshift Route の場合は **somehost:443**)。ビルドマネージャーとの通信に使用される gRPC クライアントは、ポートを省略すると推測しないためです。

6.6. 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/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。つまり、ビルダー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。

6.6.1. DEBUG 設定フラグ

DEBUG フラグを設定することで、完了/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。そのためには、目的のエクゼキューターの設定で、DEBUG を true に設定します。以下に例を示します。

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

DEBUG を true に設定すると、quay-builder サービスが終了した後や失敗した後にビルドノードがシャットダウンするのを防ぎ、ビルドマネージャーがインスタンスをクリーンアップする (EC2 インスタンスの終了や k8s ジョブの削除) のを防ぐことができます。これはビルダーノードの問題をデバッグするためのもので、本番環境で **設定すべきではありません**。ライフタイムサービスは引き続き存在します。つまり、インスタンスはそれでも約 2 時間後にシャットダウンします (EC2 インスタンスが終了し、k8s ジョブが完了する)。EBUG を設定すると、終了していないインスタンスやジョブが稼働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。

以下の手順で行います。

1. ゲスト仮想マシンは、その SSH ポート (22) をホスト (Pod) のポート 2222 に転送します。ビルダー Pod のポート 2222 を、ローカルホストのポートにポートフォワードします。

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

2. SSH_AUTHORIZED_KEYS のキーセットを使用して、コンテナー内で動作している仮想マシンに SSH 接続します。

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

3. quay-builder のサービスログを取得します。

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

   - ステップ 2 ~ 3 は、1 つの SSH コマンドで行うこともできます。

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

6.7. GITHUB ビルドの設定 (オプション)

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 DOCKERFILE のビルド

Red Hat Quay は、当社のビルドフリート上で Dockerfile をビルドし、出来上がったイメージをリポジトリーにプッシュする機能をサポートしています。

7.1. ビルドの表示および管理

リポジトリーのビルドは、**Repository View** の Builds タブをクリックして表示および管理できます。

7.2. 手動でのビルド開始

リポジトリーのビルドを手動で開始するには、任意のリポジトリーページのヘッダーの右上にある **+** アイコンをクリックし、**New Dockerfile Build** を選択します。ビルドには、アップロードされた **Dockerfile**、**.tar.gz**、またはいずれかへの HTTP URL を使用できます。

注記

手動でビルドを開始する際に、Docker ビルドコンテキストを指定することはできません。

7.3. ビルドトリガー

リポジトリーのビルドは、SCM (GitHub、BitBucket、GitLab) へのプッシュや webhook への呼び出しなどのイベントによって自動的にトリガーされることもあります。

7.3.1. 新しいビルドトリガーの作成

ビルドトリガーを設定するには、Builds view ページの **Create Build Trigger** ボタンをクリックし、ダイアログの指示に従います。トリガーをセットアップするには、Red Hat Quay にリポジトリーへのアクセス権を与える必要があり、アカウントには **SCM リポジトリーの管理者アクセスが必要** です。

7.3.2. ビルドトリガーの手動起動

ビルドトリガーを手動で起動するには、ビルドトリガーの横にあるアイコンをクリックし、**Run Now** を選択します。

7.3.3. ビルドコンテキスト

Docker 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。Red Hat Quay で行われるビルドは、自分のマシンで **docker build** を実行するのと変わらないため、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 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、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。

```
example
├── .git
```

```
├── Dockerfile
├── file
└── subdir
    └── Dockerfile
```

上の例が、"example" という名前の GitHub リポジトリーのディレクトリー構造だと想像してみてください。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や、手動でビルドを開始した場合は、example ディレクトリーでビルドを行います。

ビルドトリガーの設定でサブディレクトリーとして **subdir** 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、**.dockerignore** ファイルに表示されてはいけません。

第8章 カスタム GIT トリガーの設定

カスタム Git トリガーは、あらゆる git サーバーがビルドトリガーとして機能するための汎用的な方法です。SSH キーと webhook のエンドポイントのみに依存しており、それ以外はすべてユーザーが実装することになります。

8.1. 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、いくつかの微妙な違いがあります。

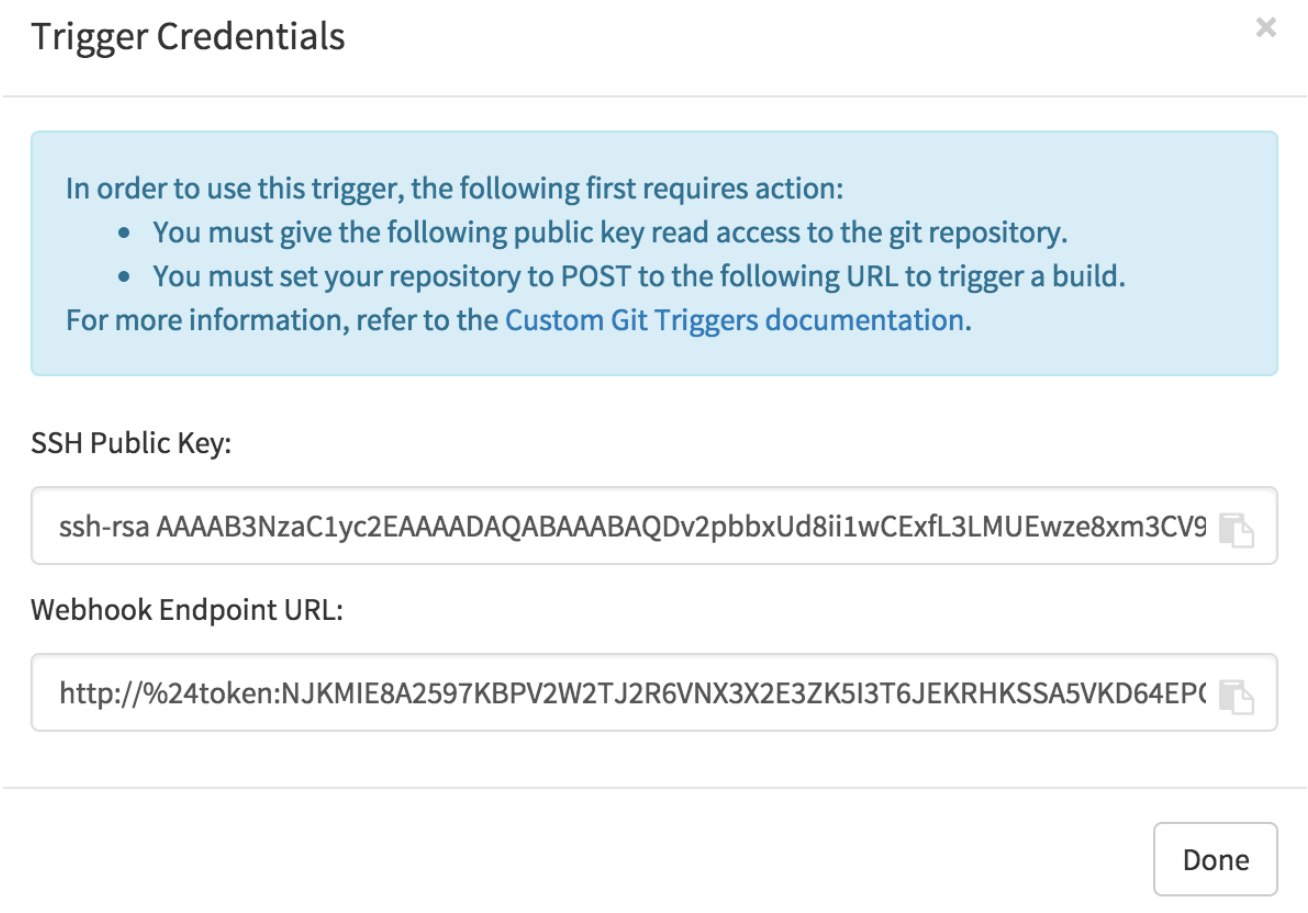
- Red Hat Quay は、トリガーで使用する適切なロボットアカウントを自動的に検出することはできません。これは、作成時に手動で行う必要があります。
- トリガーを使用するには、トリガーの作成後に追加の手順を実施する必要があります。その手順は以下のとおりです。

8.2. トリガー作成後の設定

トリガーを作成した後、トリガーを使用する前に **2 つの追加ステップが必要です**。

- 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。
- 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。

このキーと URL は、トリガーリストにあるギアから **View Credentials** を選択することで、どちらもいつでも確認できます。



8.2.1. SSH 公開鍵へのアクセス

Git サーバーのセットアップに応じて、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** に追加するだけで、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、通常は **Deploy Keys** と呼ばれるキーを入力する場所があります。

8.2.2. Webhook

ビルドを自動的にトリガーするには、以下のフォーマットの JSON ペイロードを webhook URL に POST する必要があります。

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

注記

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。

繰り返しになりますが、これはサーバーの設定により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 受信後の git フック で行うことができます。

第9章 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。

第10章 GITHUB のビルドトリガータグの設定

Red Hat Quay は、イメージをビルドするトリガーとして GitHub または GitHub Enterprise の使用をサポートしています。まだ実行していない場合は、Red Hat Quay でビルドサポートを有効化 してください。

10.1. ビルドトリガーのタグ命名規則について

Red Hat Quay 3.3 より前は、ビルドトリガーから作成されたイメージの名前の付け方には制限がありました。ビルドトリガーで作られたイメージは、以下の情報と共に名前が付けられていました。

- その変化によってトリガーが呼び出されたブランチやタグ
- デフォルトのブランチを使用していたイメージの **latest** タグ

Red Hat Quay 3.3 以降では、イメージタグの設定方法がより柔軟になりました。まず、カスタムタグを入力して、任意の文字列を各ビルドイメージのタグとして割り当てます。しかし、別の方法として、以下のタグテンプレートを使用して、イメージを各コミットの情報にタグ付けすることもできます。

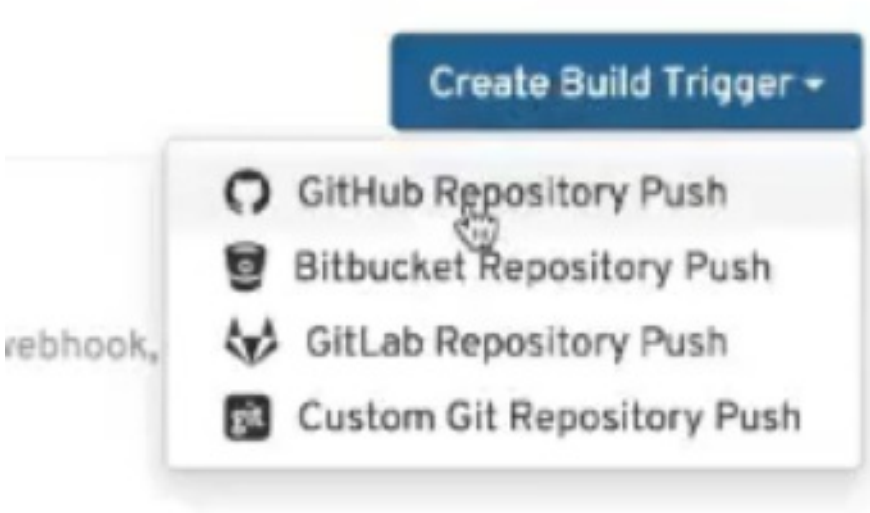
- **${commit_info.short_sha}**: コミットの短い SHA
- **${commit_info.date}**: コミットのタイムスタンプ
- **${commit_info.author}**: コミットの作成者
- **${commit_info.committer}**: コミットのコミッター
- **${parsed_ref.branch}**: ブランチ名

以下の手順では、ビルドトリガーのタグ付けを設定する方法を説明します。

10.2. ビルドトリガーのタグ名の設定

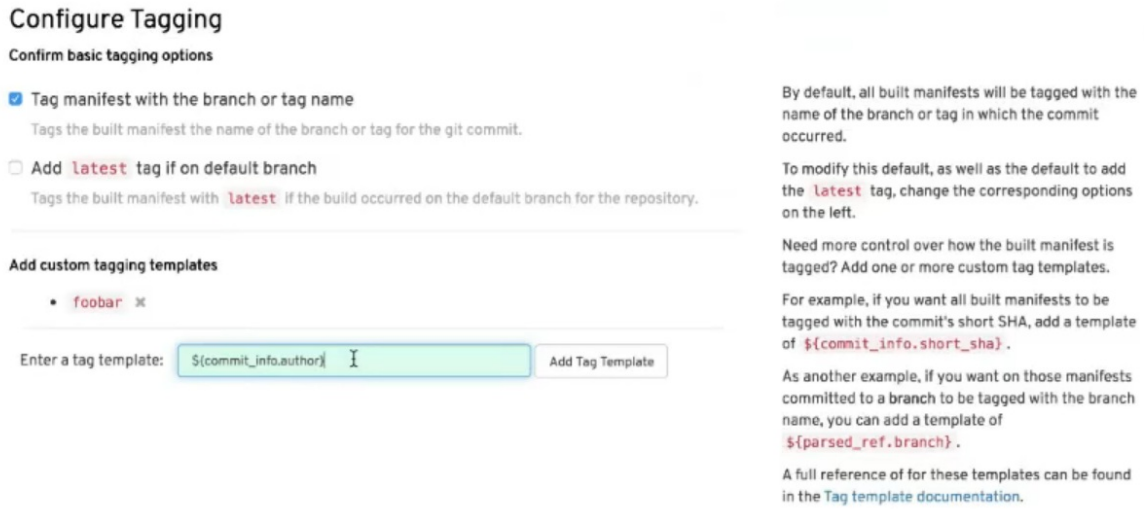
以下の手順で、ビルドトリガー用のカスタムタグを設定します。

1. リポジトリービューで、左のナビゲーションから Builds アイコンを選択します。
2. Create Build Trigger メニューを選択し、必要なリポジトリープッシュの種類 (GitHub、Bitbucket、GitLab、Custom Git リポジトリープッシュ) を選択します。この例では、下図のように **GitHub Repository Push** を選択しています。



3. **Setup Build Trigger** ページが表示されたら、トリガーをセットアップするリポジトリーと名前空間を選択します。

4. Configure Trigger で、**Trigger for all branches and tags** または **Trigger only on branches and tags matching a regular expression** のいずれかを選択します。その後、Continue を選択します。次の図のように、Configure Tagging セクションが表示されます。



5. **Configure Tagging** までスクロールダウンし、以下のオプションから選択します。

   - **Tag manifest with the branch or tag name** このチェックボックスをオンにすると、コミットが発生したブランチまたはタグの名前が、イメージで使用されるタグとして使用されます。これはデフォルトで有効になっています。

   - **Add latest tag if on default branch** リポジトリーのデフォルトブランチにある場合、イメージに **latest** タグを使用するには、このチェックボックスをオンにします。これはデフォルトで有効になっています。

   - **Add custom tagging templates** カスタムタグやテンプレートを **Enter a tag template** ボックスに入力します。ここに入力できるタグのテンプレートは、このセクションの前半で説明したように、複数あります。その中には、コミットの短い SHA、タイムスタンプ、作成者名、コミッター、ブランチ名などをタグとして使用する方法も含まれています。

6. Continue を選択します。Docker ビルド用のディレクトリービルドコンテキストを選択するプロンプトが表示されます。ビルドコンテキストディレクトリーは、ビルドがトリガーされると

きに必要な他のファイルとともに、Dockerfile が含まれるディレクトリーの場所を特定します。Dockerfile が git リポジトリーのルートにある場合は "/" を入力します。

7. Continue を選択します。オプションのロボットアカウントを追加するプロンプトが表示されます。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する場合は、この設定を行います。ロボットアカウントには、ビルドへのアクセスが必要です。

8. Continue を選択すると、ビルドトリガーの設定が完了します。

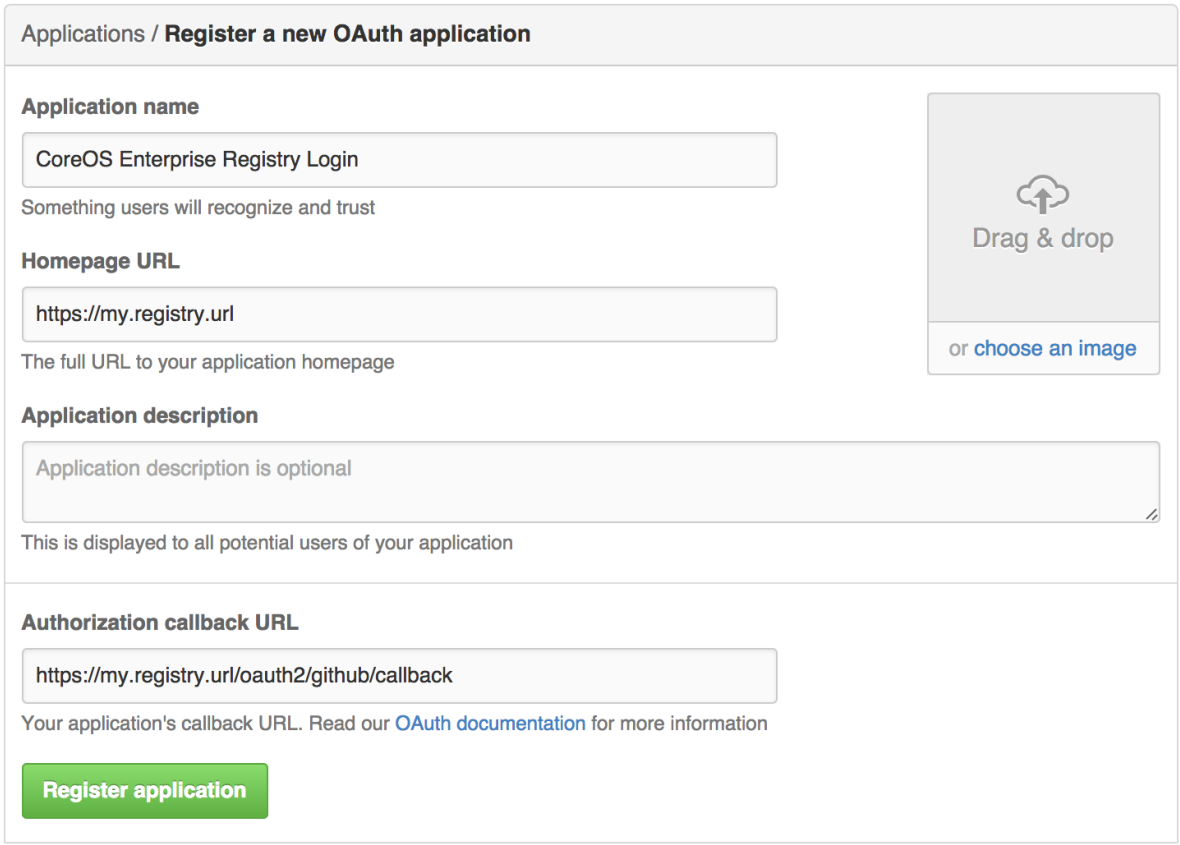
リポジトリーの Repository Builds ページに戻ると、設定したビルドトリガーが Build Triggers の見出しの下に表示されます。

第11章 GITHUB での OAUTH アプリケーションの作成

レジストリーを GitHub OAuth アプリケーションとして登録して、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。

11.1. GITHUB アプリケーションの新規作成

1. GitHub (Enterprise) にログインします。
2. 組織の設定にある Applications のページにアクセスします。
3. Register New Application をクリックします。以下の **Register a new OAuth application** 設定画面が表示されます。



4. ホームページの URL を設定します。Quay Enterprise の URL を **Homepage URL** として入力してください。

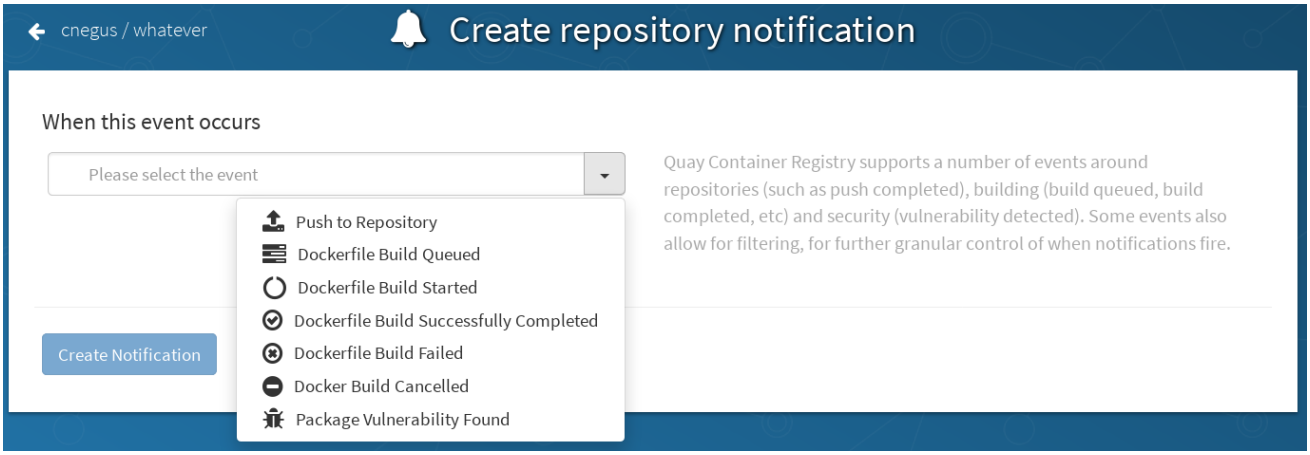
注記

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なければなりません。その URL が内部用のままである可能性があります。

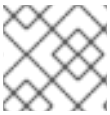
5. 承認のコールバック URL を設定します。Authorization callback URL として、https://{$RED_HAT_QUAY_URL}/oauth2/github/callback を入力します。
6. Register application ボタンをクリックして、設定を保存します。新しいアプリケーションの概要が表示されます。
7. 新しいアプリケーションに表示されるクライアント ID およびクライアントシークレットを記録します。

第12章 リポジトリー通知

Quay 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に対して、リポジトリーへの **通知** の追加をサポートしています。通知を追加するには、リポジトリーの表示中に **Settings** タブをクリックし、**Create Notification** を選択します。**When this event occurs** フィールドから、通知を受け取りたいアイテムを選択します。



イベントを選択したら、そのイベントの通知方法を追加してさらに設定します。



注記

通知の追加には、**リポジトリーの管理者権限** が必要です。

リポジトリーイベントの例を以下に示します。

12.1. リポジトリーイベント

12.1.1. 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"
  ]
}
```

12.1.2. Dockerfile ビルドのキューへの追加

以下は、Dockerfile のビルドがビルドシステムにキューに追加されたことに対するレスポンスのサンプルです。オプション属性の使用により、レスポンスが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"
}
```

12.1.3. Dockerfile ビルドの開始

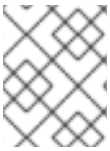
ここでは、ビルドシステムによって Dockerfile のビルドが開始された例を紹介します。一部の属性がオプションであることに基づいて、レスポンスが異なる場合があります。

```
{
  "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"
}
```

12.1.4. Dockerfile ビルドの正常な完了

ここでは、ビルドシステムによって正常に完了した Dockerfile のビルドに対する応答例を示します。



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@sha256:2a7af5265344cc3704d5d47c4604b1efcbd227a7a6a6ff73d6e4e08a27f
d7d99",

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

12.1.5. Dockerfile ビルドの失敗

Dockerfile のビルドに失敗しました。

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

```
    "message": "failed build test",
    "committer": {                                          //Optional
      "username": "web-flow",
      "url": "https://github.com/web-flow",                 //Optional
      "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"     //Optional
    },
    "author": {                                             //Optional
      "username": "dgangaia",
      "url": "https://github.com/dgangaia",                 //Optional
      "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
    }
  }
 },
 "homepage": "https://quay.io/repository/dgangaia/test/build/5346a21d-3434-4764-85be-
5be1296f293c"
}
```

12.1.6. Dockerfile ビルドのキャンセル

Dockerfile のビルドがキャンセルされました。

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

```
  "homepage": "https://quay.io/repository/dgangaia/test/build/cbd534c5-f1c0-4816-b4e3-
55446b851e70"
}
```

12.1.7. 脆弱性の検出

リポジトリーに脆弱性が検出されました。

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

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

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

12.2. 通知アクション

12.2.1. Quay 通知

Quay.io の通知領域に通知が追加されます。通知エリアは、Quay.io のページの右上にあるベルのアイコンをクリックすると表示されます。

Quay.io 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。

12.2.2. 電子メール

指定したアドレスに、発生したイベントを記載したメールが送信されます。

注記

すべての電子メールアドレスは、**リポジトリーごとに** 確認する必要があります。

12.2.3. Webhook POST

指定された URL に、イベントのデータ (各イベントのデータ形式は上記参照) を含む HTTP POST コールが行われます。

URL が HTTPS の場合、呼び出しには Quay.io 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、Quay.io 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 通知の再試行となります。

12.2.4. Flowdock 通知

Flowdock にメッセージを投稿します。

12.2.5. Hipchat 通知

HipChat にメッセージを投稿します。

12.2.6. Slack 通知

Slack にメッセージを投稿します。

第13章 OCI サポートおよび RED HAT QUAY

Red Hat Quay などのコンテナーレジストリーは、当初は Docker イメージ形式で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す。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を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 3.6 での OCI ベースのアーティファクトのサポートは、Helm のみから拡張され、デフォルトで cosign および ztsd 圧縮スキームが含まれるようになりました。そのため、**FEATURE_HELM_OCI_SUPPORT** は非推奨になりました。

OpenShift Operator を使用して Red Hat Quay 3.6 をデプロイすると、**FEATURE_GENERAL_OCI_SUPPORT** 設定で Helm および OCI アーティファクトのサポートがデフォルトで有効になります。機能を明示的に有効にする必要がある場合 (機能が無効にされている場合や、デフォルトで有効にされていないバージョンからアップグレードした場合など) は、OCI および Helm サポートの明示的な有効化 セクションを参照してください。

13.1. HELM および OCI の前提条件

- **信頼される証明書:** Helm クライアントと Quay 間の通信は HTTPS 経由で行われ、Helm 3.5 の時点では、サポートは信頼される証明書を使用して HTTPS で通信するレジストリーについてのみ利用できます。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。今後の Helm リリースでのサポートにより、リモートレジストリーとの非セキュアな通信が可能になります。これを念頭に置いて、オペレーティングシステムが Quay で使用される証明書を信頼するように設定されていることを確認します。以下はその例です。

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

- **一般提供:** Helm 3.8 の時点で、チャートの OCI レジストリーサポートが一般提供されています。

- **Helm クライアントをインストールする:** Helm リリース ページから目的のバージョンをダウンロードします。これを展開し、helm バイナリーをその必要な宛先に移動します。

```
$ tar -zxvf helm-v3.8.2-linux-amd64.tar.gz
$ mv linux-amd64/helm /usr/local/bin/helm
```

- **Quay での組織の作成:** Quay レジストリー UI を使用し、Helm チャートを保存するために新しい組織を作成します。たとえば、**helm** という名前の組織を作成します。

13.2. RED HAT QUAY を使用した HELM チャート

Helm は、Cloud Native Computing Foundation (CNCF) から進展したプロジェクトとしてアプリケーションのパッケージ化およびデプロイを単純化する、Kubernetes の事実上のパッケージマネージャー

です。Helm は、アプリケーションを表す Kubernetes リソースが含まれる Chart というパッケージ形式を使用します。Chart は、リポジトリーでの一般的なディストリビューションや消費用に利用できます。Helm リポジトリーは、**index.yaml** メタデータファイルと、オプションでパッケージ化されたチャートのセットを提供する HTTP サーバーです。Helm バージョン 3 以降、従来のリポジトリーの代わりとして OCI レジストリーでチャートを提供するためのサポートが利用できるようになりました。

13.2.1. Red Hat Quay での 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。

手順

1. 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. 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. リポジトリーからチャートをダウンロードします。

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

4. 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 Quay リポジトリーにログインします。

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

6. **helm push** コマンドを使用して、グラフを Quay 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
```

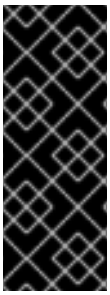
13.3. OCI および HELM 設定フィールド

Helm のサポートが **FEATURE_GENERAL_OCI_SUPPORT** プロパティーでサポートされるようになりました。機能を明示的に有効にする必要がある場合 (機能が無効にされている場合や、デフォルトで有効にされていないバージョンからアップグレードした場合など) は、OCI アーティファクトの使用を有効にするために 2 つのプロパティーを Quay 設定に追加する必要があります。

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

表13.1 OCI および Helm 設定フィールド

フィールド	タイプ	説明
FEATURE_GENERAL_OCI_SUPPORT	ブール値	OCI アーティファクトのサポートを有効にします **デフォルト:** True
FEATURE_HELM_OCI_SUPPORT	ブール値	Helm アーティファクトのサポートを有効にします **デフォルト:** True



重要

Red Hat Quay 3.6 の時点で、**FEATURE_HELM_OCI_SUPPORT** は非推奨になり、Red Hat Quay の今後のバージョンで削除される予定です。Red Hat Quay 3.6 では、Helm アーティファクトがデフォルトでサポートされ、**FEATURE_GENERAL_OCI_SUPPORT** プロパティーに含まれています。ユーザーは、サポートを有効にするために config.yaml ファイルを更新する必要がなくなりました。

13.4. RED HAT QUAY との COSIGN OCI 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

13.5. QUAY と COSIGN の使用

Go 1.16+ を使用している場合は、次のコマンドを使用して cosign を直接インストールできます。

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

次に、キーペアを生成します。

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

次のコマンドでキーペアに署名します。

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

一部のユーザーは、次のエラーが発生する可能性があります。

```
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[]
```

cosign は認証を ~/.docker/config.json に依存しているため、次のコマンドの実行が必要になる場合があります。

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

次のコマンドを使用して、更新された許可設定を確認できます。

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

13.6. その他の OCI メディアタイプの QUAY への追加

Helm、cosign、および ztsd 圧縮スキームアーティファクトはデフォルトで Red Hat Quay 3.6 に組み込まれています。デフォルトでサポートされていない他の OCI メディアタイプでは、以下の形式を使用して Quay の config.yaml の **ALLOWED_OCI_ARTIFACT_TYPES** 設定に追加できます。

```
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>
...
```

たとえば、以下を config.yaml に追加して Singularity (SIF) サポートを追加できます。

```
...
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
...
```

注記

デフォルトで設定されていない OCI メディアタイプを追加する場合、ユーザーは必要に応じて cosign と Helm のサポートも手動で追加する必要があります。ztsd 圧縮スキームはデフォルトでサポートされているため、ユーザーはサポートを有効にするためにその OCI メディアタイプを config.yaml に追加する必要はありません。

13.7. 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場合は、config.yaml で **FEATURE_GENERAL_OCI_SUPPORT** を **False** に設定できます。

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

第14章 RED HAT QUAY のクォータ管理および施行

Red Hat Quay 3.7 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 Quay ユーザーには、環境の容量制限を管理するために以下の機能が追加されました。

- **クォータレポート:** 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、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ます。

14.1. クォータ管理アーキテクチャー

RepositorySize データベーステーブルは、組織内の Red Hat Quay リポジトリーのストレージ消費量をバイト単位で保持します。組織のすべてのリポジトリーサイズの合計は、Red Hat Quay 組織の現在のストレージサイズを定義します。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、**RepositorySize** テーブルで更新されます。

14.2. 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14.1 ワーカー数の環境変数

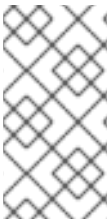
変数	説明
Postgres	8388608 TB
MySQL	8388608 TB

変数	説明
SQL Server	16777216 TB

14.3. クォータ管理設定

クォータ管理は **FEATURE_QUOTA_MANAGEMENT** プロパティーでサポートされるようになり、デフォルトでオフになっています。クォータ管理を有効にするには、**config.yaml** の機能フラグを **true** に設定します。

```
FEATURE_QUOTA_MANAGEMENT: true
```



注記

Red Hat Quay 3.7 では、クォータを作成、更新、削除するにはスーパーユーザー権限が必要です。クォータはユーザーと組織に設定できますが、Red Hat Quay UI を使用して **ユーザー** クォータを再設定することはできず、代わりに API を使用する必要があります。

14.3.1. デフォルトのクォータ

すべての組織およびユーザーに適用されるシステム全体のデフォルトのストレージクォータを指定するには、**DEFAULT_SYSTEM_REJECT_QUOTA_BYTES** 設定フラグを使用します。

表14.2 デフォルトのクォータ設定

フィールド	タイプ	説明
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	String	すべての組織およびユーザーに適用するクォータサイズ デフォルトでは、制限は設定されていません。

組織またはユーザーに特定のクォータを設定してから、そのクォータを削除すると、システム全体のデフォルトのクォータが設定されている場合に適用されます。同様に、組織またはユーザーに特定のクォータを設定してから、システム全体のデフォルトクォータを変更した場合、更新されたシステム全体のデフォルトは特定の設定を上書きします。

14.4. 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

```
[]
```

14.4.1. 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"
```

14.4.2. 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
  }
]
```

14.4.3. 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
}
```

14.4.4. 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。

14.4.4.1. 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
```

14.4.4.2. 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
    }
  ]
}
```

14.4.4.3. 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
  }
}
```


14.4.5. 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。

14.4.5.1. 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
```

14.4.5.2. 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
  }
]
```

14.4.5.3. 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
```

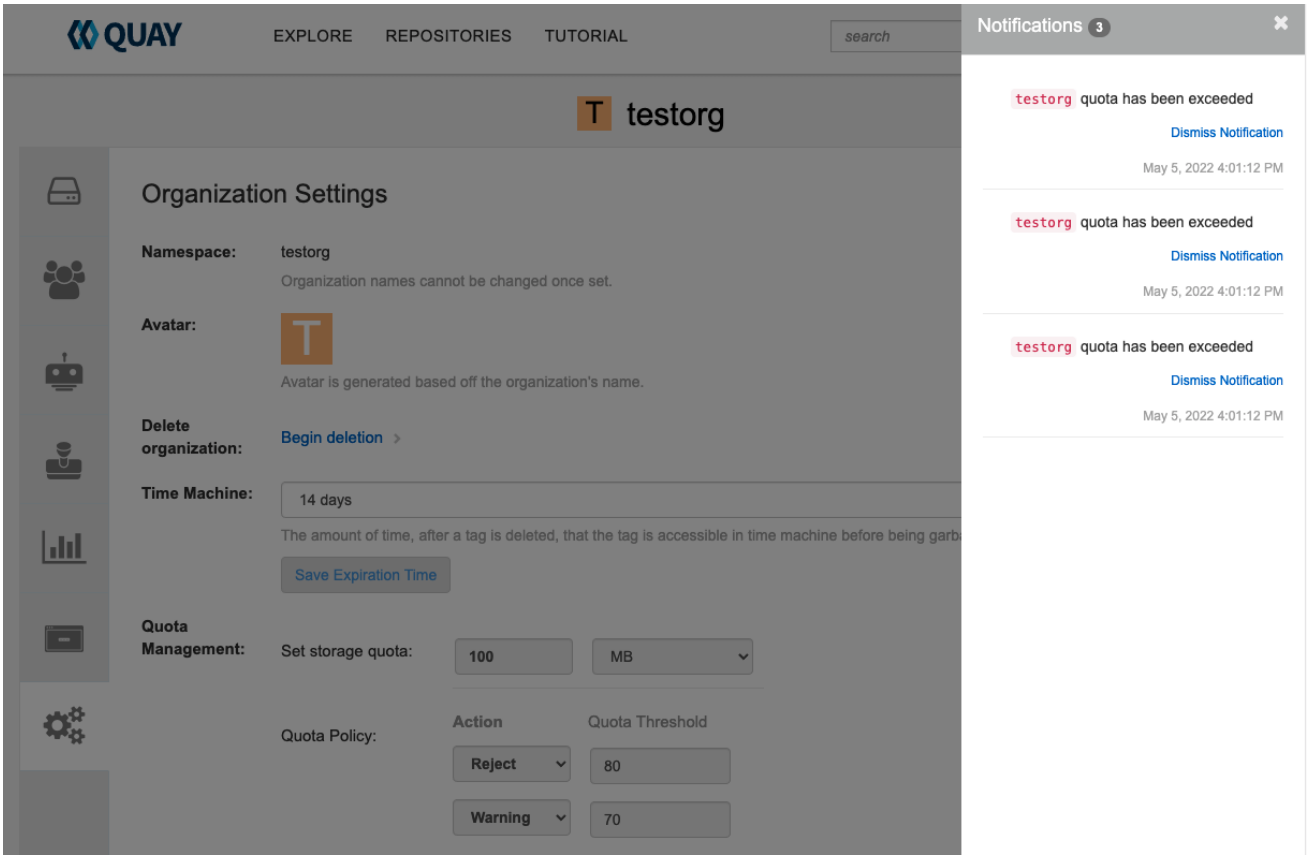
クォータを超えたときのサンプル出力

```
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
```

14.4.5.4. 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15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。

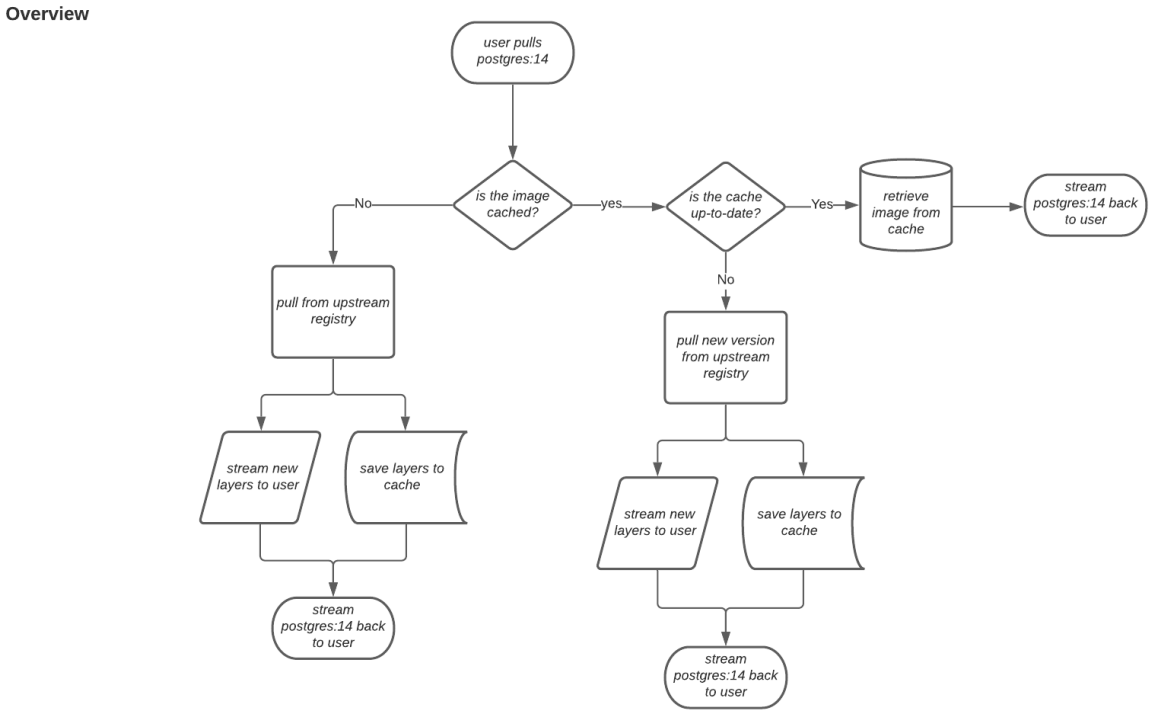
この機能により、Red Hat Quay は、アップストリームレジストリーからのプルレート制限を回避するためのプロキシーキャッシュとして機能します。キャッシュ機能を追加すると、アップストリームの依存関係ではなくキャッシュからイメージがプルされるため、プルのパフォーマンスも向上します。キャッシュされたイメージは、アップストリームのイメージダイジェストがキャッシュされたイメージと異なる場合にのみ更新され、レート制限と潜在的なスロットリングを削減します。

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。

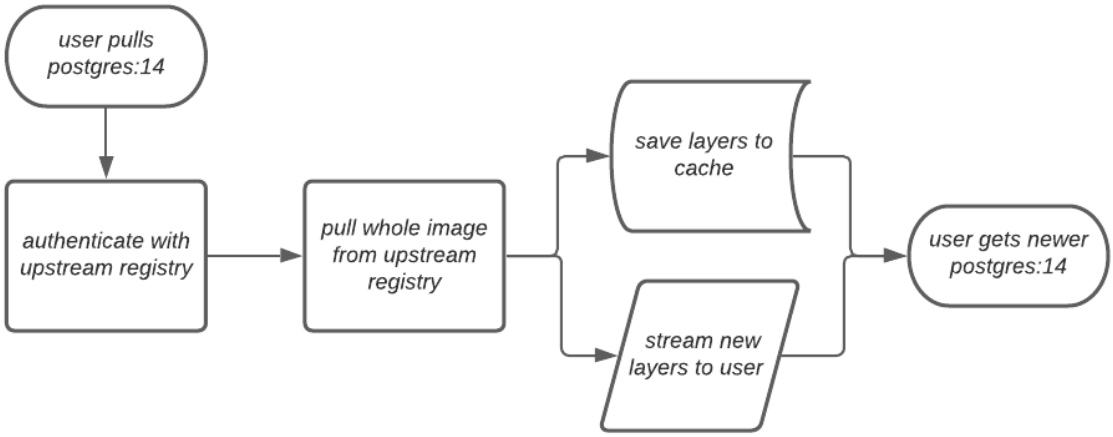
- 特定の組織は、アップストリームレジストリーのキャッシュとして定義できます。
- 特定のアップストリームレジストリーのキャッシュとして機能する Quay 組織の設定。このリポジトリーは、Quay UI を使用して定義でき、次の設定を提供します。
    - プライベートリポジトリーのアップストリームレジストリークレデンシャルまたはレート制限の強化。
    - キャッシュ組織のサイズを超えないようにするための有効期限タイマー。
- 設定アプリケーションを介して設定可能なグローバルオン/オフ。
- アップストリームレジストリー全体または単一の名前空間 (たとえば、すべての **docker.io**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

15.1. プロキシーキャッシュアーキテクチャー

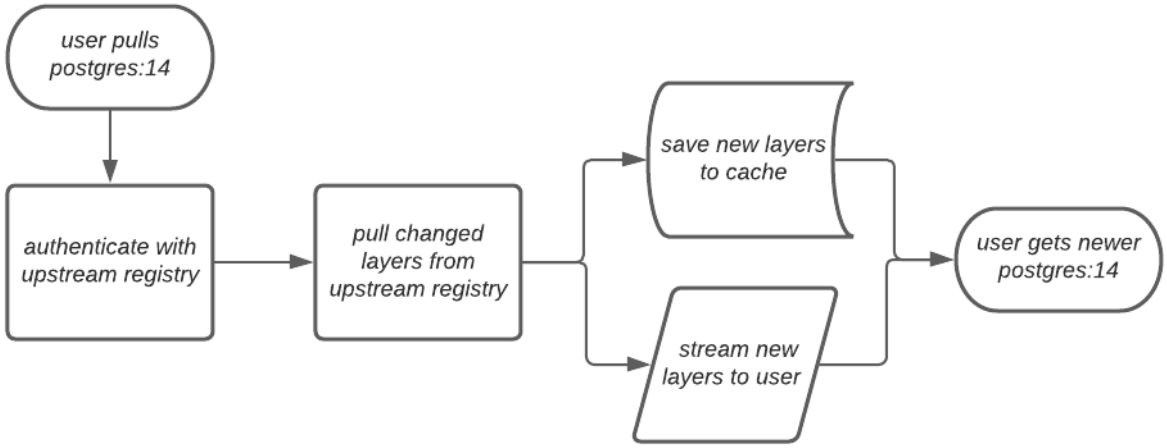
次のイメージは、プロキシーキャッシュ機能の予想される設計フローおよびアーキテクチャーを示しています。



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。

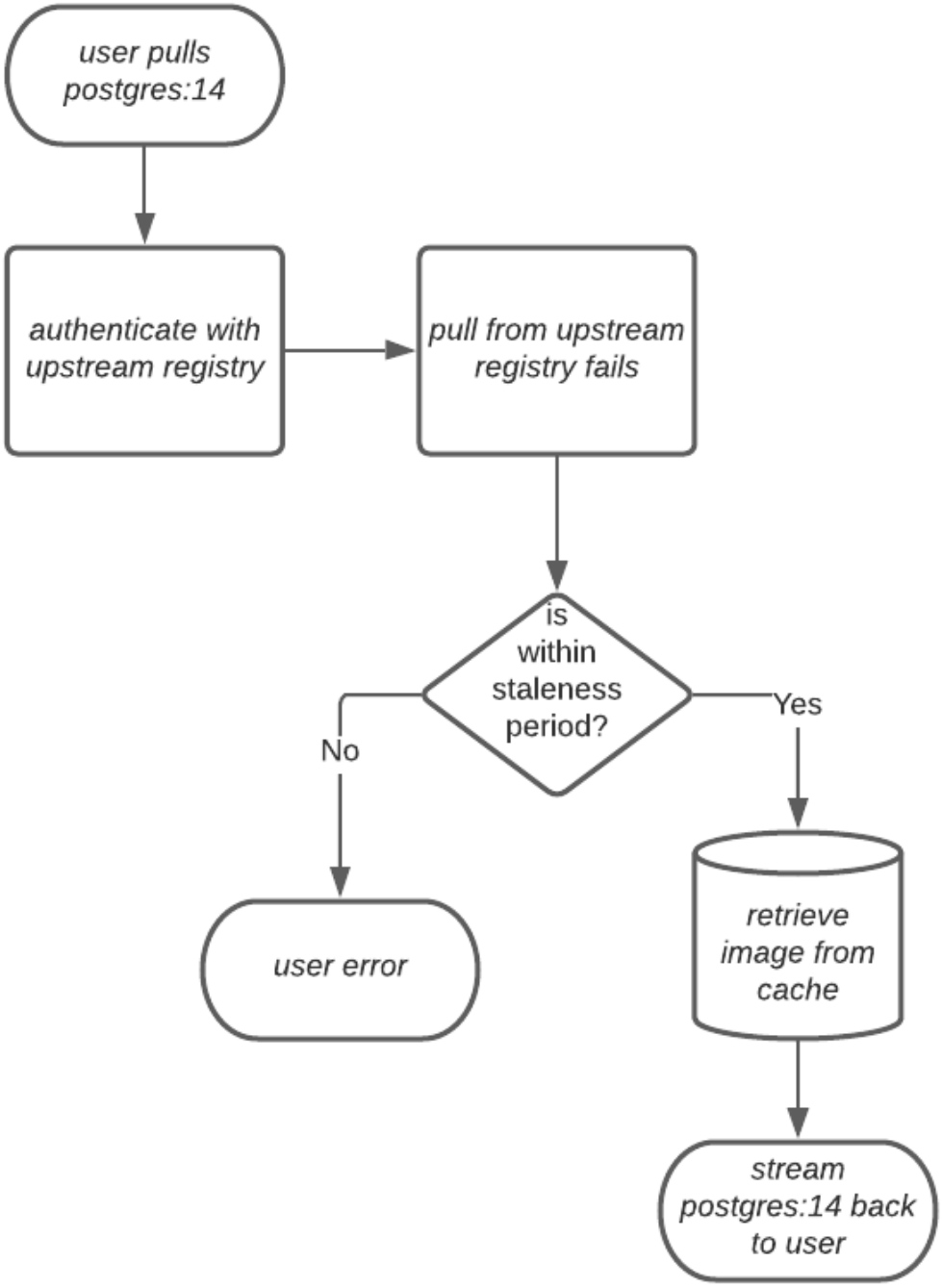


キャッシュ内のイメージが存在する場合、ユーザーは Quay のキャッシュを利用してアップストリームソースを最新の状態に保ち、キャッシュから新しいイメージが自動的にプルされるようにできます。これは、元のイメージのタグがアップストリームレジストリーで上書きされた場合に発生します。次のイメージは、アップストリームイメージとキャッシュされたバージョンのイメージが異なる場合に何が起こるかを示すアーキテクチャーの概要を示しています。



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。

15.2. 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。

15.3. 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** に設定されています。
- **Member** チームのロールを割り当てられました。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。

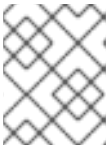
手順

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、**保存** 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

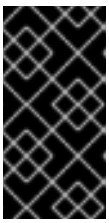
- プロキシー組織でキャッシュされたイメージのデフォルトのタグ **Expiration** フィールドは 86400 秒に設定されています。プロキシー組織では、タグがプルされるたびに、タグの有効期限が UI の **Expiration** フィールドに設定された値に更新されます。この機能は、Quay のデフォルトの 個別タグ有効期限 機能とは異なります。プロキシー組織では、個々のタグ機能をオーバーライドできます。これが発生すると、プロキシー組織の **Expiration** フィールドに従って、個々のタグの有効期限がリセットされます。

- 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>
```

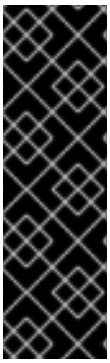


重要

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。

15.3.1. 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、ネームスペース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。

15.3.1.1. 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. リポジトリーが、割り当てられたクォータを超えることになる最後の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:3.5.1
```

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。

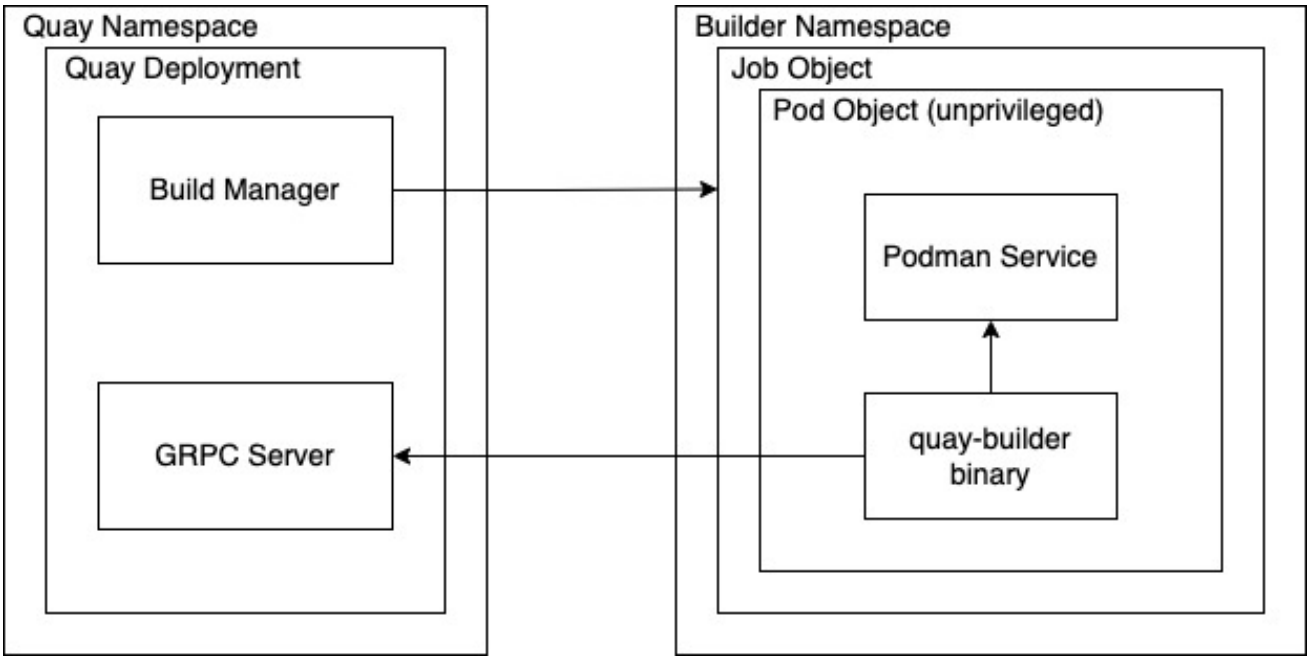
第16章 RED HAT QUAY ビルドの機能強化

Red Hat Quay 3.7 より前は、Quay は Pod によって起動された仮想マシンで **podman** コマンドを実行していました。仮想プラットフォームでビルドを実行するには、ネストされた仮想化を有効にする必要があります。これは、Red Hat Enterprise Linux または OpenShift Container Platform では機能しません。その結果、ビルドはベアメタルクラスターで実行する必要があり、これはリソースの非効率的な使用です。

Red Hat Quay 3.7 では、仮想マシンレイヤーを含まないビルドオプションを追加することで、ビルドの実行に必要なベアメタル制約が削除されました。その結果、ビルドは仮想化されたプラットフォームで実行できます。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。

16.1. RED HAT QUAY の拡張ビルドアーキテクチャー

前のイメージは、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。

16.2. 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。

16.3. 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。

16.3.1. 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と、Red Hat Quay 3.8 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
```

16.3.2. 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。

16.3.2.1. 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、クラスターが既にプロビジョニングされており、Quay Operator が実行し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。

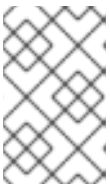
```
$ oc create sa -n virtual-builders quay-builder
```

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
```

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 に設定することをお勧めします。

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。これが発生すると、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"
```

16.3.2.2. 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。

16.3.2.2.1. 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、次の手順を使用します。

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
```

16.3.2.2.2. 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'
```

16.3.2.2.3. 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** 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
```

16.3.2.2.4. 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
```

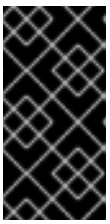
16.3.2.3. UI を使用してビルドトリガーを作成

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

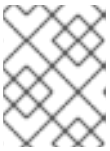
```
NAME                                     READY   STATUS       RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Terminating   0          9s
```

```
$ 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。

16.3.2.4. 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
    }
]
```

16.3.2.5. Google Cloud Platform オブジェクトバケットの変更

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

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17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 エンドポイントを一覧表示し、説明します。

17.1. 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。

17.2. OAUTH アクセストークンの作成

組織の API にアクセスできるように OAuth アクセストークンを作成するには、以下を実行します。

1. Red Hat Quay にログインし、組織を選択します (または新規の組織を作成します)。
2. 左側のナビゲーションからアプリケーションアイコンを選択します。
3. Create New Application を選択し、プロンプトが表示されたら、新規アプリケーションに名前を指定します。
4. 新規アプリケーションを選択します。
5. 左側のナビゲーションから Generate Token を選択します。
6. チェックボックスを選択してトークンのスコープを設定し、Generate Access Token を選択します。
7. 許可しているパーミッションを確認し、Authorize Application を選択してこれを承認します。
8. API へのアクセスに使用する新規生成されたトークンをコピーします。

17.3. 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
```

17.4. 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。

17.4.1. 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
    }
  ]
}
```

17.4.2. 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="
    }
    ```

ユーザーの一覧を要求すると、**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
    }
  ]
}
```

17.4.3. 使用ログの一覧表示

現在のシステムの使用ログを一覧表示するには、内部 API **/api/v1/superuser/logs** を使用できます。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。

17.4.3.1. 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_q3e3I0agCvB2VPQ8rY75WeaiUzM3rQlMc4i6EIR78t8oUxVfNp1RMPIRQYYZyXP9h6E8LZZhq
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_q3e3I0agCvB2VPQ8rY75WeaiUzM3rQlMc4i6EIR78t8oUxVfNp1RMPIRQYYZyXP9h
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"
        }
      }
    },
    ...
  ]
}
```


17.4.4. 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
```

17.4.5. 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。

17.4.6. 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>"
```

17.4.7. 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)
```

17.4.8. 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
```

17.4.9. 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
```