



Red Hat Quay 3.4

Red Hat Quayの使用

Red Hat Quayの使用

Red Hat Quay 3.4 Red Hat Quayの使用

Red Hat Quayの使用

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2021 | You need to change the HOLDER entity in the en-US/Use_Red_Hat_Quay.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

Red Hat Quayの使用法を学ぶ

目次

前書き	4
第1章 RED HAT QUAY のユーザーと組織の作成	5
1.1. ユーザーアカウントの作成	5
1.2. 組織アカウントの作成	6
第2章 リポジトリの作成	7
2.1. UIによるイメージリポジトリの作成	7
2.2. DOCKER または PODMANによるイメージリポジトリの作成	8
第3章 リポジトリへのアクセス管理	9
3.1. ユーザーリポジトリへのアクセスの許可	9
3.1.1. ユーザーリポジトリへのユーザーアクセスの許可	9
3.2. ユーザーリポジトリへのロボットアクセスの許可	10
3.3. 組織のリポジトリへのアクセスの許可	11
3.3.1. 組織へのチームの追加	11
3.3.2. チームロールの設定	12
3.3.3. チームへのユーザーの追加	12
第4章 タグの使用	14
4.1. タグの表示と変更	14
4.1.1. タグの付いたイメージへの新しいタグの追加	14
4.1.2. タグの移動	14
4.1.3. タグの削除	14
4.1.4. タグの履歴の表示および操作の取り消し	14
4.1.4.1. タグの履歴の表示	14
4.1.4.2. 操作の取り消し	15
4.1.5. タグやダイジェストによるイメージの取得	15
4.2. タグの有効期限	15
4.2.1. Dockerfileからのタグの有効期限の設定	16
4.2.2. リポジトリからのタグの有効期限の設定	16
4.3. セキュリティスキャン	17
第5章 ログの表示とエクスポート	18
5.1. ログの表示	18
5.2. リポジトリログのエクスポート	19
第6章 ビルドワーカーを使用した DOCKERFILE の自動ビルド	21
6.1. アーキテクチャーの概要	21
6.1.1. ビルドマネージャー	21
6.1.2. ビルドワーカーのコントロールプレーン	21
6.1.3. オーケストレーター	21
6.2. OPENSIFTの要件	22
6.3. オーケストレーターの要件	22
6.4. RED HAT QUAY ビルダーと OPENSIFT のセットアップ	22
6.4.1. Red Hat Quayのビルド用OpenShiftの準備	22
6.4.2. 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	24
6.5. OPENSIFTルートの制限	26
6.6. ビルドのトラブルシューティング	27
6.6.1. DEBUG設定フラグ	27
6.7. GITHUBビルドの設定（オプション）	28
第7章 DOCKERFILEのビルド	29

7.1. ビルドの表示と管理	29
7.2. 手動でのビルド開始	29
7.3. ビルドトリガー	29
7.3.1. 新しいビルドトリガーの作成	29
7.3.2. ビルドトリガーの手動起動	29
7.3.3. ビルドコンテキスト	29
第8章 カスタムGITトリガーの設定	31
8.1. トリガーの作成	31
8.2. トリガー作成後の設定	31
8.2.1. SSH公開鍵へのアクセス	32
8.2.2. Webhook	32
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	33
第10章 GITHUBのビルドトリガータグの設定	34
10.1. ビルドトリガーのタグ命名規則について	34
10.2. ビルドトリガーのタグ名の設定	34
第11章 GITHUBでのOAUTHアプリケーションの作成	37
11.1. GITHUBアプリケーションの新規作成	37
第12章 リポジトリ通知	38
12.1. リポジトリイベント	38
12.1.1. リポジトリプッシュ	38
12.1.2. Dockerfile ビルドのキューへの追加	38
12.1.3. Dockerfileビルドの開始	39
12.1.4. Dockerfileビルドの正常な完了	40
12.1.5. Dockerfileビルドの失敗	41
12.1.6. Dockerfileビルドのキャンセル	42
12.1.7. 脆弱性の検出	43
12.2. 通知アクション	43
12.2.1. Quay通知	43
12.2.2. 電子メール	43
12.2.3. Webhook POST	43
12.2.4. Flowdock通知	44
12.2.5. Hipchat通知	44
12.2.6. Slack通知	44
第13章 RED HAT QUAY API の使用	45
13.1. QUAY.IO からの QUAY API へのアクセス	45
13.2. OAUTH アクセストークンの作成	45
13.3. WEB ブラウザーからの QUAY API へのアクセス	46
13.4. コマンドラインでの RED HAT QUAY API へのアクセス	46
13.4.1. スーパーユーザー情報の取得	46
13.4.2. API を使用したスーパーユーザーの作成	47
13.4.3. API を使用したリポジトリビルドの作成	48
13.4.4. 組織のロボットの作成	49
13.4.5. ビルドのトリガー	49
13.4.6. プライベートリポジトリの作成	49
追加リソース	49

前書き

Red Hat Quay コンテナイメージレジストリーでは、コンテナイメージを中央の場所に保存することができます。Red Hat Quay レジストリーの通常ユーザーとして、イメージを整理するためのリポジトリを作成し、自分が管理するリポジトリへの読み取り（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 のユーザーと組織の作成

Red Hat Quay でコンテナイメージを保持するためのリポジトリの作成を開始する前に、そのリポジトリ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 インスタンスの各リポジトリは、以下のいずれかに関連付けられている必要があります。

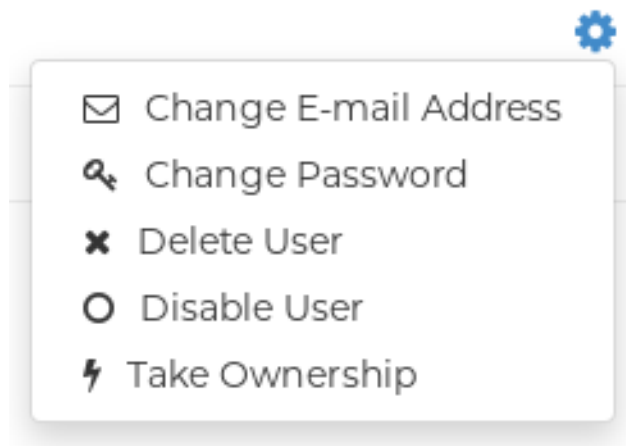
- **ユーザー**：ユーザーアカウントは、Web UIまたはコンテナクライアント（**docker login**など）を介して、Red Hat Quayインスタンスにログインするためのものです。レポジトリを作成すると、その名前は`myquay.example.com/<user>/<repo>`のように、ご自分のアカウントと関連付けられます。そのレポジトリを作成したら、他のRed Hat Quayユーザーアカウントやロボットアカウントと呼ばれるものにレポジトリへのアクセスを許可することができます。
- **組織**：ユーザーアカウントを作成するにはスーパーユーザーの権限が必要ですが、ユーザーのグループ間でリポジトリを効率的に共有するために、どのユーザーでも組織を作成することができます。組織とは、ユーザーアカウントのようなもので、他のユーザーアカウントやロボットアカウントにレポジトリへのアクセスを定義することができます。しかし、**チーム**と呼ばれるユーザーのセットにアクセス権を追加することもできます。ある組織の下で作成されたリポジトリは、`myquay.example.com/<org>/<repo>`のようになります。

以下のセクションでは、Red Hat Quay にユーザーアカウントと組織を作成する方法について説明します。ユーザーアカウントの作成には、スーパーユーザーの権限が必要です。

1.1. 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。



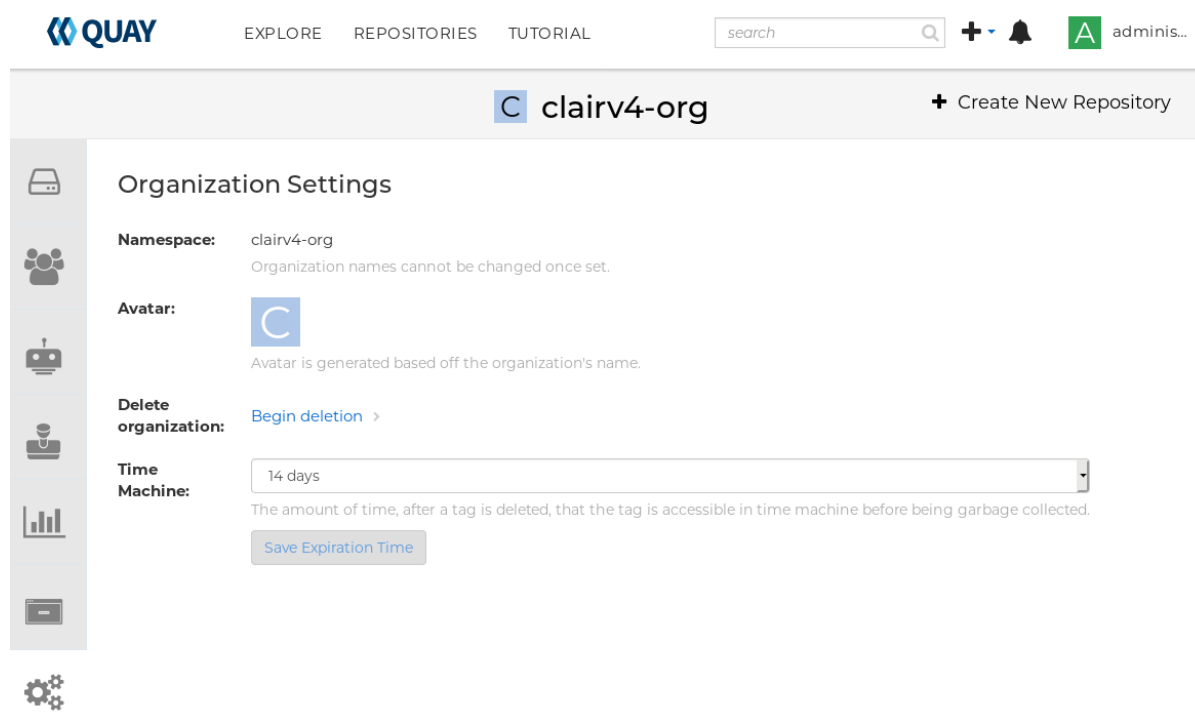
7. メニューからChange Passwordを選択します。
8. 新しいパスワードを追加して確認した後、Change User Passwordボタンを選択します。

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

1.2. 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、コンテナイメージのリポジトリ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、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することができます。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。

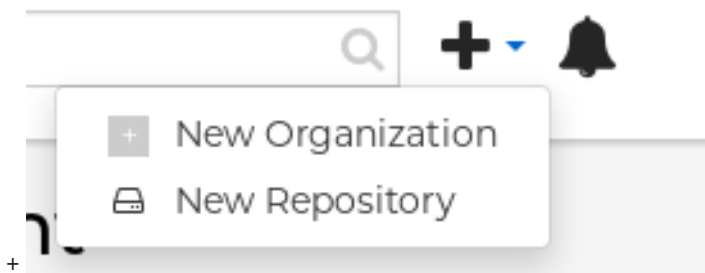


第2章 リポジトリの作成

リポジトリ（名前空間とも呼ばれる）は、関連するコンテナイメージのセットを一元的に保存するための場所を提供します。Red Hat Quayでリポジトリを作成するには、（**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の作成

ユーザーアカウントで Red Hat Quay UI でリポジトリを作成するには、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名を追加する
 - Repository Descriptionをクリックして、リポジトリの説明を入力する
 - Repository Visibilityで、リポジトリ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

新しいリポジトリが作成され、最初は空の状態です。このリポジトリ（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名を追加する
 - Repository Descriptionをクリックして、リポジトリの説明を入力する
 - Repository Visibilityで、リポジトリ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

新しいリポジトリが作成され、最初は空の状態です。このリポジトリ（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にイメージをプッシュすると、イメージをリポジトリにプッシュする際にそのリポジトリが作成されます。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（例えばreg.example.com）にイメージをプッシュする例を紹介します。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. 適切なレジストリにプッシュします。例:

```
# 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を作成するには、コンテナイメージのリポジトリ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と組織リポジトリの両方で、ロボットアカウントに関連する認証情報を作成することで、それらのリポジトリへのアクセスを許可することができます。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。

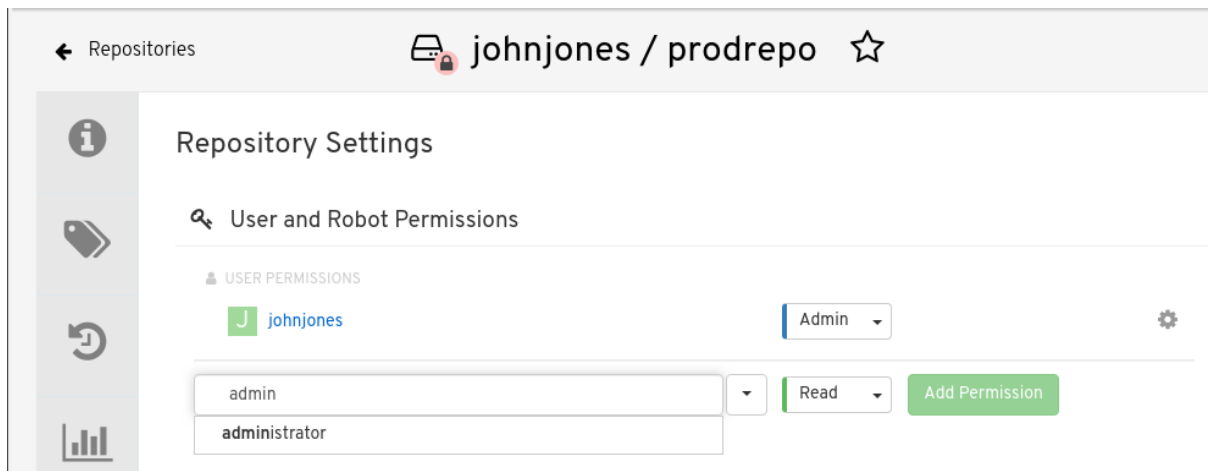
3.1. ユーザーリポジトリへのアクセスの許可

ユーザー名前空間にリポジトリを作成すると、そのリポジトリ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することができます。

3.1.1. ユーザーリポジトリ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へのアクセスを許可するには、以下のようになります。

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を表示したり、リポジトリからイメージをプルしたり、リポジトリにイメージをプッシュしたりすることができます。
 - Admin - リポジトリ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

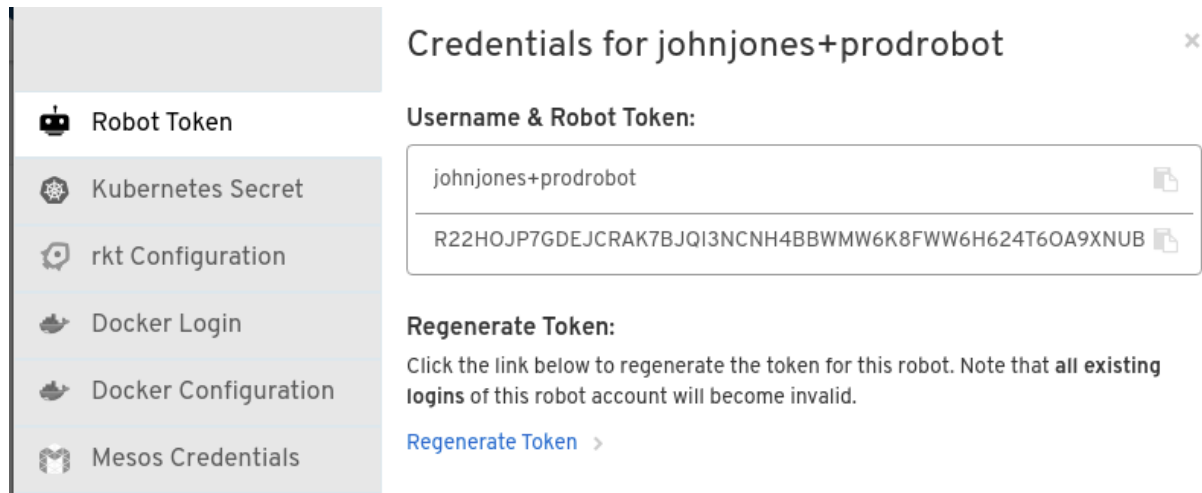
ロボットアカウントは、Red Hat Quay レジストリ内のリポジトリへの自動アクセスを設定するために使用されます。これらは、OpenShiftのサービスアカウントに似ています。ロボットアカウントを設定する際に、以下を行います。

- 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。例えば、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へのアクセスを提供しますが、リポジトリ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にチェックマークを入れます。
 - 各リポジトリについて、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に対する権限がありません。
 - Read - ロボットがリポジトリを閲覧し、プルすることができます。
 - Write - ロボットは、リポジトリからの読み込み（プル）と、リポジトリへの書き込み（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へのフルアクセスに加えて、リポジトリ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. ロボットを使ってリポジトリ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。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. 認証情報を取得する。生成されたトークンに満足したら、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 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. 組織のリポジトリへのアクセスの許可

組織を作成すると、リポジトリ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することができます。基本的に、組織はユーザーと同じようにリポジトリ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、チームを使います。
- 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することができます。

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は、チーム名を選択し、チームが利用できるリポジトリを選択し、チームのアクセスレベルを決定することができます。

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オーナーズチーム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

チームを追加した後、そのチームの組織内でのロールを設定することができます。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。

The screenshot shows the 'Teams and Membership' page for the 'alldevelopers' organization. The page has a sidebar with navigation icons and a main content area. The main content area has a 'Teams and Membership' header with tabs for 'Teams View', 'Members View', and 'Collaborators View'. Below the header is a table with the following columns: TEAM NAME, MEMBERS, REPOSITORIES, and TEAM ROLE. The table lists two teams: 'owners' (1 member) and 'testers' (0 members). The 'owners' team has an 'Admin' role, and the 'testers' team has a 'Member' role. A dropdown menu is open for the 'testers' team, showing three options: 'Member' (inherited), 'Creator' (member and can create new repositories), and 'Admin' (full admin access).

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することができます。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 レジストリ上のアカウントからのユーザー名
 - レジストリ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。画像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って、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

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

Repository Tags

CompactExpanded

Actions

1 - 25 of 287

Filter Tags...

TAG	LAST MODIFIED ↓	SECURITY SCAN	SIZE	IMAGE
<input checked="" type="checkbox"/> latest	16 hours ago	<div><div></div>70 Medium • 10 fixable</div>	711.0 MB	<div>SHA2569a347939468e</div> <div></div>
<input type="checkbox"/> master	16 hours ago	<div><div></div>70 Medium • 10 fixable</div>	711.0 MB	<div>SHA256014514e8ef9b</div> <div></div>
<input type="checkbox"/> dbb57f7	18 hours ago	<div><div></div>70 Medium • 10 fixable</div>	696.1 MB	<div>SHA2562592c71fe8f5</div> <div></div>
<input type="checkbox"/> 3e28797	a day ago	<div><div></div>75 Medium • 15 fixable</div>	693.5 MB	<div>SHA2560d37d281173e</div> <div></div>

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、既存のタグ名前を指定します。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。

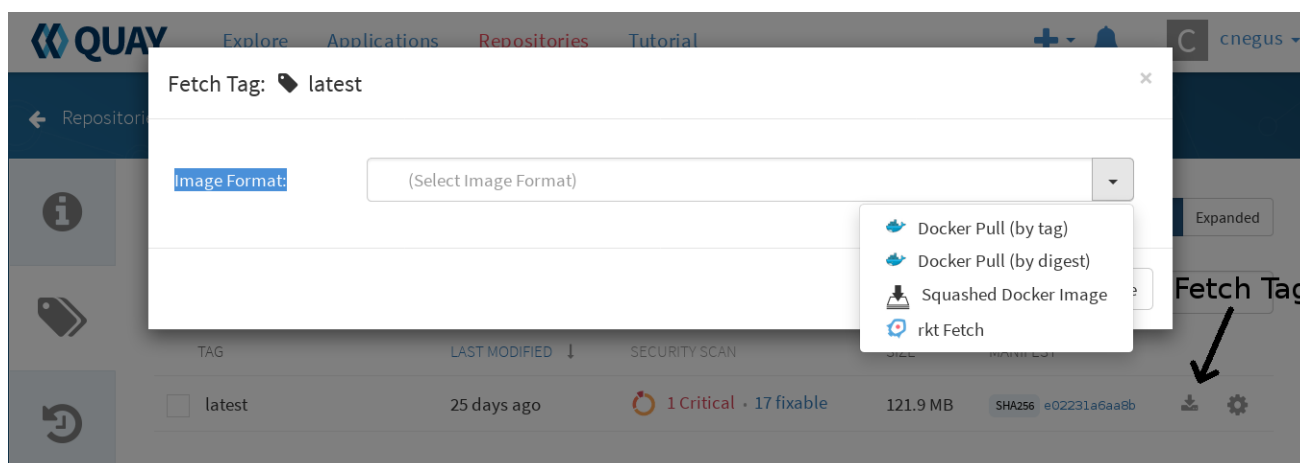
4.1.4.2. 操作の取り消し

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/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コマンドを使って、タグ名またはダイジェスト名で通常のイメージをプルすることができます。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**コマンドと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することができます。

4.2. タグの有効期限

イメージは、**タグの有効期限**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ピントすることができます。
- Red Hat Quay のスーパーユーザーには、ユーザーリポジトリ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を設定することで。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ことで。

次の図は、タグの有効期限を変更するためのOptionsエン트리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。

The screenshot displays the 'Repository Tags' interface for the 'johnjones / prodrepo' repository. It features a table with the following data:

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
latest	21 hours ago	Passed	34.5 MB	in 4 months	SHA256 9ed9932476f1

A tooltip for the 'EXPIRES' column shows the date and time: 'Mon, Aug 31, 2020 6:59 PM'. A context menu is visible over the 'in 4 months' link, with the following options:

- + Add New Tag
- Edit Labels
- Delete Tag
- Change Expiration

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、リポジトリから削除されるように設定されます。

4.3. セキュリティスキャン

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

アクティビティログは、Red Hat Quay のすべてのリポジトリと名前空間（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。



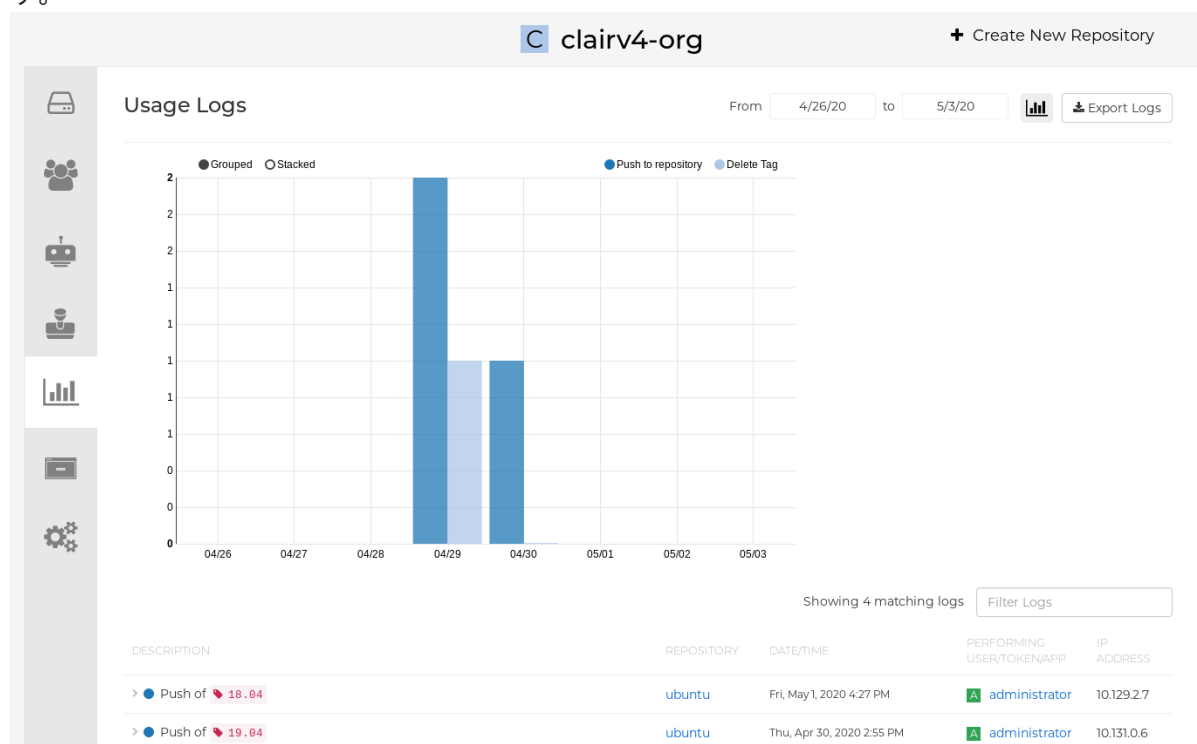
注記

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ト機能を使う必要があります。

5.1. ログの表示

Web UIからリポジトリや名前空間のログエントリを表示するには、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、ログエントリを表示する日付の範囲を設定する。デフォルトでは、直近の1週間分のログエントリが表示されます。
 - 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を表示する。

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

5.2. リポジトリ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。

- リポジトリから収集するログの日付の範囲を選択することができます。
- 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することができます。
- ログをエクスポートするには、リポジトリ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することができる
- 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うには、以下の手順を実施します。

1. 管理者権限のあるリポジトリを選択します。
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。

Export Usage Logs ✕

Enter an e-mail address or callback URL (must start with **http://** or **https://**) at which to receive the exported logs once they have been fully processed:

johnjones@example.com

Note: The export process can take **up to an hour** to process if there are many logs. As well, only a **single** export process can run at a time for each namespace. Additional export requests will be queued.

Start Logs Export

Cancel

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を使うことができます。
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ので、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ポイントし、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をビルドすることができます。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、ビルドエグゼキュータ（OpenShift/K8sクラスタ）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。複数のエグゼキュータを設定することで、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することができます。Red Hat Quay は、あるエグゼキュータが問題を抱えていることを検知すると、自動的に別のエグ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を設定する方法についての説明があります。この構成は、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）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 を使って実行されます。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. OPENSIFT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が確保されます。さらに、OpenShift クラスタでは、特権コンテナをサポートするために必要な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. オーク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 OPENSIFT のセットアップ

6.4.1. Red Hat Quayのビルド用OpenShiftの準備

OpenShift クラスタが Red Hat Quay からのビルドを受け入れる前に、OpenShift クラスタ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：'ビルダー'）。

```
$ 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。**ジョブ**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. ビルド**ジョブ**のスケジューリング時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については、クラスタ管理者に確認してください。
5. クラスタ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を使って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

```

6.4.2. 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_cert_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プレフィックスを定義します（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'と'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クラスタのAPIサーバのホスト名

K8S_API_TLS_CA

QuayアプリケーションがAPIコールを行う際に信頼する、ビルドクラスタのCA証明書のQuayコンテナ内のファイルパス。

KUBERNETES_DISTRIBUTION

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

CONTAINER_*

各ビルドPodのリソース要求と制限を定義します。

NODE_SELECTOR_*

ビルドPodがスケジューリングされるノードセクタ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イメージの完全な参照 (**registry.redhat.io/quay/quay-builder-gemu-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。例：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。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（設定され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番目のエグゼキューター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、他の鍵を使ってEC2インスタンスやQEMU 仮想マシンにssh接続することができます。

6.5. OPENSIFTルートの制限



注記

このセクションは、管理対象の**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。**kubectl**または**oc** CLIツールが、Quay Operatorがインストールされているクラスターで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**をポイントする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**エントリを、**BUILDMAN_HOSTNAME**フィールドとともに、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（ビルドエグゼキュータに依存します）。

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

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**、またはカスタムホスト名を使用している場合はCNAMEエントリ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、目的のエグゼキュータの設定で、DEBUGをtrueに設定します。例:

```
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、本番環境では**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するためには、ALLOWED_WORKER_COUNTに達した場合、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のキーセットを使って、コンテナ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. ビルドの表示と管理

リポジトリのビルドは、**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

リポジトリのビルドを手動で開始するには、任意のリポジトリ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

リポジトリのビルドは、SCM（GitHub、BitBucket、GitLab）へのプッシュや[webhookの呼び出し](#)などのイベントによって自動的に開始することもできます。

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ためには、Red Hat Quay にリポジトリ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。

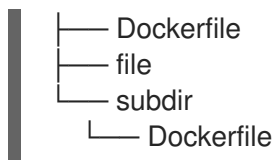
7.3.3. ビルドコンテキスト

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ので、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**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。例:

```
example
└── .git
```



上の例が、"example"という名前のGitHubリポジトリのディレクトリ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として**subdir**を指定した場合、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**コマンドを使って**ファイル**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ためには、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することができます。

Trigger Credentials



In order to use this trigger, the following first requires action:

- You must give the following public key read access to the git repository.
- You must set your repository to POST to the following URL to trigger a build.

For more information, refer to the [Custom Git Triggers documentation](#).

SSH Public Key:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDv2pbbxUd8ii1wCExfL3LMUEwze8xm3CV9

Webhook Endpoint URL:

<http://%24token:NJKMIE8A2597KBPV2W2TJ2R6VNX3X2E3ZK5I3T6JEKRHKSSA5VKD64EP>

Done

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について説明しています。この場合、鍵を`$HOME/.ssh/authorize_keys`に追加するだけで、ビルダーがリポジトリ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ためには、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ってさまざまな方法で行うことができますが、ほとんどの場合は[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にしてください](#)。

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**最新タグ**

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って、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、必要なりポジトリプッシュの種類（GitHub、Bitbucket、GitLab、Custom Git リポジトリプッシュ）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。

Configure Tagging

Confirm basic tagging options

☒ **Tag manifest with the branch or tag name**

Tags the built manifest the name of the branch or tag for the git commit.

☐ **Add latest tag if on default branch**

Tags the built manifest with **latest** if the build occurred on the default branch for the repository.

Add custom tagging templates

• foobar ✕

Enter a tag template:

By default, all built manifests will be tagged with the name of the branch or tag in which the commit occurred.

To modify this default, as well as the default to add the **latest** tag, change the corresponding options on the left.

Need more control over how the built manifest is tagged? Add one or more custom tag templates.

For example, if you want all built manifests to be tagged with the commit's short SHA, add a template of `${commit_info.short_sha}`.

As another example, if you want on those manifests committed to a branch to be tagged with the branch name, you can add a template of `${parsed_ref.branch}`.

A full reference of for these templates can be found in the [Tag template documentation](#).

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:** このボックスをチェックすると、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、このボックスをチェックします。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共に、Dockerfileが含まれるディレクトリの場所を特定します。Dockerfileがgitリポジトリのルートにある場合は"/"を入力します。

- 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したい場合は、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。

- 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。

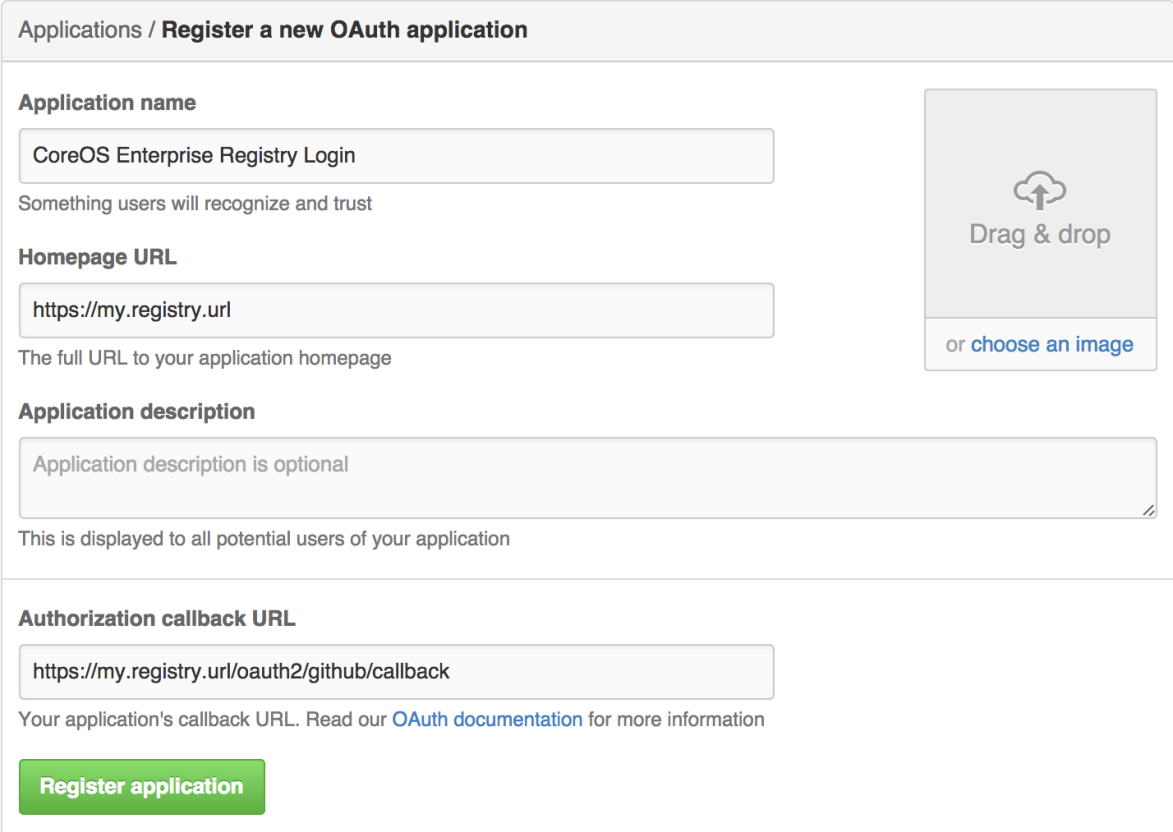
Build Triggers						Create Build Trigger +
TRIGGER NAME	DOCKERFILE LOCATION	CONTEXT LOCATION	BRANCHES/TAGS	PULL ROBOT	TAGGING OPTIONS	
 Push to GitHub repository dongboyan77/ruby-hello-world	/Dockerfile	/	All	(None)	Branch/tag name foo	
 Push to GitHub repository dongboyan77/ruby-hello-world	/Dockerfile	/	All	(None)	latest if default branch latest	
 Push to repository git@github.com:dongboyan77/ruby-hello-world.git	/Dockerfile	/	All	(None)	Branch/tag name latest if default branch	

第11章 GITHUBでのOAUTHアプリケーションの作成

レジストリをGitHub OAuthアプリケーションとして登録することで、GitHubアカウントとそのリポジトリへのアクセスを許可することができます。

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。



Applications / **Register a new OAuth application**

Application name

CoreOS Enterprise Registry Login

Something users will recognize and trust

Homepage URL

https://my.registry.url

The full URL to your application homepage

Application description

Application description is optional

This is displayed to all potential users of your application

Authorization callback URL

https://my.registry.url/oauth2/github/callback

Your application's callback URL. Read our [OAuth documentation](#) for more information

Register application

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。それはまだ内部のURL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](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とクライアントシークレットを記録します。

第12章 リポジトリ通知

Quayは、リポジトリのライフサイクルで発生する様々なイベントに対して、リポジトリへの**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。

The screenshot shows the 'Create repository notification' page. On the left, there's a dropdown menu titled 'When this event occurs' with the placeholder text 'Please select the event'. The dropdown is open, showing a list of events: 'Push to Repository', 'Dockerfile Build Queued', 'Dockerfile Build Started', 'Dockerfile Build Successfully Completed', 'Dockerfile Build Failed', 'Docker Build Cancelled', and 'Package Vulnerability Found'. Below the dropdown is a blue button labeled 'Create Notification'. On the right, there's a text box explaining that Quay Container Registry supports various events around repositories, building, and security, and that some events allow for filtering.

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、**リポジトリの管理者権限**が必要です。

リポジトリイベントの例を以下に示します。

12.1. リポジトリイベント

12.1.1. リポジトリプッシュ

1つまたは複数のイメージのリポジトリ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。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": "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": {
  "commit": "50bc5996d4587fd4b2d8edc4af652d4cec293c42",
  "ref": "refs/heads/master",
  "default_branch": "master",
  "git_url": "git@github.com:dgangaia/test.git",
  "commit_info": {
    "url": "https://github.com/dgangaia/test/commit/50bc5996d4587fd4b2d8edc4af652d4cec293c42",
    "date": "2019-03-06T14:10:14+11:00",
    "message": "test 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/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",
  "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": "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": {
        "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
        "date": "2019-03-06T12:48:24+11:00",
        "message": "adding 5",
        "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/296ec063-5f86-4706-a469-f0a400bf9df2",
    "manifest_digests": [
        "quay.io/dgangaia/test@sha256:2a7af5265344cc3704d5d47c4604b1efcbd227a7a6a6ff73d6e4e08a27fd7d99",
        "quay.io/dgangaia/test@sha256:569e7db1a867069835e8e97d50c96eccafde65f08ea3e0d5deba16e2545d9d1"
    ]
}

```

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",
    "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",
    "docker_tags": [
        "master",
        "latest"
    ],
    "build_name": "6ae9a86",
    "trigger_metadata": {
        "commit": "6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
        "ref": "refs/heads/master",
        "default_branch": "master",
        "git_url": "git@github.com:dgangaia/test.git",
        "commit_info": {
            "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

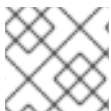
12.2.1. Quay通知

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 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、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。付録 A は API エンドポイントを一覧表示し、説明します。

13.1. 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、または変更します。

13.2. 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。

13.3. 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 インターフェー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 (API 呼び出しはブラウザーから呼び出される場合に X-Requested-With ヘッダーをつけて呼び出す必要があります)」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
```

13.4. 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。

13.4.1. 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/"
```

例:

```
$ curl -X GET -H "Authorization: Bearer mFCdgS7SAIoMcnTsHCGx23vcNsTgziAa4CmmHlsg"
http://quay-server:8080/api/v1/superuser/users/ | jq

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

13.4.2. 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せん。

```
{
  "username": "quaysuper",
  "email": "quaysuper@example.com",
  "password": "EH67NB3Y6PTBED8H0HC6UVHGGGA3ODSE",
  "encrypted_password":
  "fn37AZAUQH0PTsU+vIO9IS0QxPW9A/boXL4ovZjIFtUPrBz9i4j9UDOqMjuxQ/0HTfy38go
KEpG8zYXVeQh3IOFzuOjSvKic2Vq7xdtQsU="
}
```

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 mFCdgS7SAIoMcNtsHCGx23vcNsTgziAa4CmmHlsg"
http://quay-server:8080/api/v1/superuser/users/ | jq
```

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

13.4.3. API を使用したリポジトリビルドの作成

指定の入力からリポジトリをビルドし、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

カイク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アプリのスコープは「Administer Organization」でなければなりません。そうでない場合には、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には、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。

13.4.4. 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>'
```

13.4.5. ビルドのトリガー

```
$ curl -X POST https://quay.io/api/v1/repository/YOURORNAME/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)
```

13.4.6. プライベートリポジトリーの作成

```
$ curl -X POST https://quay.io/api/v1/repository \
-d '{"namespace":"yournamespace","name":"reponame","description":"description of your
repo","visibility":"private"}' -H 'Authorization: Bearer {token}'
```

追加リソース