Red Hat Quay 3.7

Red Hat Quay の設定

設定オプションを使用した Red Hat Quay のカスタマイズ

Last Updated: 2023-08-23

Red Hat Quay 3.7 Red Hat Quay の設定

設定オプションを使用した Red Hat Quay のカスタマイズ

法律上の通知

概要

Red Hat Quay の設定

目次

第1章 RED HAT QUAY 設定の開始

Red Hat Quay は、独立したスタンドアロン設定によって、または OpenShift Container Platform Red Hat Quay Operator を使用してデプロイできます。

Red Hat Quay 設定を作成、取得、更新、および検証する方法は、使用しているデプロイメントのタイプによって異なります。ただし、コア設定オプションはどちらの展開タイプでも同じです。コア設定は、次のオプションのいずれかで設定できます。

- **config.yaml** ファイルを編集して直接実行する。詳しくは、設定ファイルの編集 を参照してください。
- プログラムでコンフィグレーション API を使用する。詳しくは、設定 API の使用 を参照してください。
- 視覚的に設定ツール UI を使用する。詳しくは、設定ツールの使用 を参照してください。

Red Hat Quay のスタンドアロンデプロイメントの場合、レジストリーを開始する前に、最低限必要な設定パラメーターを指定する必要があります。Red Hat Quay レジストリーを開始するための最低限の要件は、現在の設定の取得 セクションに記載されています。

Red Hat Quay Operator を使用して OpenShift Container Platform に Red Hat Quay をインストールする場合、Red Hat Quay Operator はレジストリーをデプロイするためのデフォルト情報を提供するため、設定パラメーターを指定する必要はありません。

目的の設定で Red Hat Quay をデプロイしたら、デプロイから完全な設定を取得して保存する必要があります。完全な設定には、システムの再始動またはアップグレード時に必要になる可能性がある追加の生成値が含まれています。

1.1. QUAY 3.7 の設定更新

1.1.1. Red Hat Quay 3.7.7 の新規設定フィールド

フィールド	タイプ	説明
REPO_MIRROR_ROLLBACK	ブール値	**true** に設定されている場合、リポジトリーはミラー試行の失敗後にロールバックします。 **デフォルト:false**

1.1.2. 新規設定フィールド

以下の設定フィールドが Red Hat Quay 3.7 で導入されました。

パラメーター	説明

パラメーター	説明
FEATURE_QUOTA_MANAGEMENT	クォータ管理がサポートされるようになりました。この機能により、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。クォータ管理の詳細については、Red Hat Quay クォータの管理と実施 を参照してください。
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	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クォータ管理の詳細については、Red Hat Quay クォータの管理と実施 を参照してください。
FEATURE_PROXY_CACHE	Red Hat Quay を使用してリモート組織をプロキシーすることがサポートされるようになりました。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。クォータ管理の詳細については、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 を参照してください。

1.2. RED HAT QUAY 3.6 の設定の更新

1.2.1. 新規設定フィールド

以下の設定フィールドが Red Hat Quay 3.6 で導入されました。

パラメーター	説明
FEATURE_EXTENDED_REPOSITORY_NAMES	ネストされたリポジトリーと拡張リポジトリー名のサポートが追加されました。この変更により、特定の OpenShift Container Platform ユースケースに必要なリポジトリー名で / を使用できるようになります。詳細は、ネストされたリポジトリーの設定 を参照してください。
FEATURE_USER_INITIALIZE	true に設定すると、API **/api/v1/user/initialize** によって最初の **User** アカウントを作成できます。詳細は、自動化のための Red Hat Quay の事前設定 を参照してください。
ALLOWED_OCI_ARTIFACT_TYPES	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その他の Open Container Initiative (OCI) メディアタイプについては、Quay の **config.yaml** の **ALLOWED_OCI_ARTIFACT_TYPES** 設定に追加できます。詳細については、他の OCI メディアタイプを Quay に追加する を参照してください。

パラメーター	説明
CREATE_PRIVATE_REPO_ON_PUSH	レジストリーユーザーは、セキュリティーのニーズに応じて、**config.yaml** の **CREATE_PRIVATE_REPO_ON_PUSH** を **True** または **False** に設定できるようになりました。
CREATE_NAMESPACE_ON_PUSH	存在しない組織にプッシュした場合に、組織を自動的に作成するように設定できるようになりました。

1.2.2. 非推奨の設定フィールド

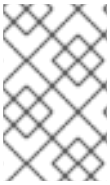
以下の設定フィールドは、Red Hat Quay 3.6 で廃止されました。

パラメーター	説明
FEATURE_HELM_OCI_SUPPORT	このオプションは廃止され、Red Hat Quay の将来のバージョンでは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。

1.3. 設定ファイルの編集

Red Hat Quay のスタンドアロンインスタンスをデプロイするには、最小限の設定情報を提供する必要があります。最小設定の要件については、Red Hat Quay の最小設定 を参照 してください。

必須フィールドに入力したら、設定を検証できます。問題がある場合は、強調表示されます。



注記

コンフィグレーション API を使用して設定を検証することもできますが、これには Quay コンテナーを設定モードで起動する必要があります。詳しくは、設定ツールの使用 を参照してください。

変更を有効にするには、レジストリーを再起動する必要があります。

1.4. スタンドアロンデプロイメントにおける設定ファイルの場所

Red Hat Quay のスタンドアロンデプロイメントの場合、Red Hat Quay レジストリーを開始するときに **config.yaml** ファイルを指定する必要があります。このファイルは設定ボリュームにあります。たとえば、次のコマンドで Red Hat Quay をデプロイする場合、設定ファイルは **$QUAY/config/config.yaml** にあります。

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
   --name=quay \
```

```
    -v $QUAY/config:/conf/stack:Z \
    -v $QUAY/storage:/datastorage:Z \
    {productrepo}/{quayimage}:{productminv}
```

1.5. 最小設定

Red Hat Quay のスタンドアロンデプロイメントには、以下の設定オプションが必要です。

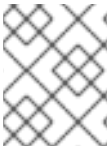
- サーバーのホスト名
- HTTP または HTTPS
- 認証タイプ (データベースや LDAP (Lightweight Directory Access Protocol) など)
- データ暗号化用の秘密鍵
- イメージのストレージ
- メタデータ用のデータベース
- ビルドログおよびユーザーイベント用の Redis
- タグの有効期限オプション

1.5.1. 最小設定ファイルの例

次の例は、イメージにローカルストレージを使用する最小限の設定ファイルの例を示しています。

```
AUTHENTICATION_TYPE: Database
BUILDLOGS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379
    ssl: false
DATABASE_SECRET_KEY: 0ce4f796-c295-415b-bf9d-b315114704b8
DB_URI: postgresql://quayuser:quaypass@quay-server.example.com:5432/quay
DEFAULT_TAG_EXPIRATION: 2w
DISTRIBUTED_STORAGE_CONFIG:
    default:
        - LocalStorage
        - storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
PREFERRED_URL_SCHEME: http
SECRET_KEY: e8f9fe68-1f84-48a8-a05f-02d72e6eccba
SERVER_HOSTNAME: quay-server.example.com
SETUP_COMPLETE: true
TAG_EXPIRATION_OPTIONS:
    - 0s
    - 1d
    - 1w
    - 2w
    - 4w
USER_EVENTS_REDIS:
```

```
    host: quay-server.example.com
    port: 6379
    ssl: false
```



注記

SETUP_COMPLETE フィールドは、設定が検証されたことを示します。レジストリーを起動する前に、設定エディターツールを使用して設定を検証する必要があります。

1.5.2. ローカルストレージ

イメージへのローカルストレージの使用は、概念実証の目的のためにレジストリーをデプロイする場合に限り推奨されます。

ローカルストレージを設定する場合、レジストリーの起動時にコマンドラインでストレージを指定します。次のコマンドは、ローカルディレクトリー **$QUAY/storage** をコンテナー内の **datastorage** ストレージパスにマップします。

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  {productrepo}/{quayimage}:{productminv}
```

1.5.3. クラウドストレージ

ストレージの設定は、イメージストレージ セクションを参照してください。一部のユーザーにとっては、Google Cloud Platform とローカルストレージ設定の違いを比較すると役立つ場合があります。たとえば、次の YAML は Google Cloud Platform のストレージ設定を表しています。

$QUAY/config/config.yaml

```
DISTRIBUTED_STORAGE_CONFIG:
    default:
        - GoogleCloudStorage
        - access_key: GOOGQIMFB3ABCDEFGHIJKLMN
          bucket_name: quay_bucket
          secret_key: FhDAYe2HeuAKfvZCAGyOioNaaRABCDEFGHIJKLMN
          storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
```

クラウドストレージを使用してレジストリーを起動する場合は、コマンドラインでの設定が必要ありません。以下に例を示します。

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  {productrepo}/{quayimage}:{productminv}
```

第2章 設定フィールド

このセクションでは、Red Hat Quay のデプロイ時に必須および任意の設定フィールドの両方について説明します。

2.1. 必須の設定フィールド

Red Hat Quay の設定で必須のフィールドは、以下のセクションで説明されています。

- 一般的な必須フィールド
- イメージのストレージ
- メタデータ用のデータベース
- ビルドログおよびユーザーイベント用の Redis
- タグの有効期限オプション

2.2. 自動化オプション

以下のセクションでは、Red Hat Quay デプロイメントで利用可能な自動化オプションについて説明します。

- 自動化のための Red Hat Quay の事前設定
- API を使用した最初のユーザーの作成

2.3. 任意の設定フィールド

Red Hat Quay の任意のフィールドについては、以下のセクションで説明します。

- 基本設定
- SSL
- LDAP
- リポジトリーのミラーリング
- セキュリティースキャナー
- OCI および Helm
- アクションログ
- ビルドログ
- Dockerfile ビルド
- OAuth
- ネストされたリポジトリーの設定
- その他の OCI メディアタイプの Quay への追加

- mail
- User
- reCAPTCHA
- ACI
- JWT
- アプリケーショントークン
- その他
- レガシーオプション

2.4. 一般的な必須フィールド

以下の表は、Red Hat Quay デプロイメントの必須設定フィールドについて説明しています。

表2.1 一般的な必須フィールド

フィールド	タイプ	説明
AUTHENTICATION_TYPE (必須)	文字列	認証情報の認証に使用する認証エンジン。 **値:** **Database**、**LDAP**、**JWT**、**Keystone**、**OIDC** のいずれか。 **デフォルト: Database**
PREFERRED_URL_SCHEME (必須)	文字列	Red Hat Quay へのアクセスに使用する URL スキーム。 **値:** **http**, **https** のいずれか。 **デフォルト: http**
SERVER_HOSTNAME (必須)	文字列	スキームなしで Red Hat Quay にアクセスできる URL。 **例:** **quay-server.example.com**

フィールド	タイプ	説明
DATABASE_SECRET_KEY (必須)	文字列	データベース内で機密フィールドを暗号化するのに使用されるキー。この値は、一旦設定したら変更しないでください。変更すると、リポジトリーのミラーユーザー名やパスワード設定など、すべての信頼できるフィールドが無効になります。
SECRET_KEY (必須)	文字列	データベース内および実行時に機密フィールドを暗号化するために使用されるキー。この値は一度設定すると決して変更しないでください。そうしないと、暗号化されたパスワード資格情報などのすべての依存フィールドが無効になります。
SETUP_COMPLETE (必須)	ブール値	これは、以前のバージョンのソフトウェアからそのまま残っているアーティファクトで、現時点では値を **true** に指定する **必要があります**。

2.5. データベースの設定

このセクションでは、Red Hat Quay デプロイメントで利用可能なデータベース設定フィールドについて説明します。

2.5.1. データベース URI

Red Hat Quay では、必要な **DB_URI** フィールドを使用してデータベースへの接続を設定します。

以下の表は **DB_URI** 設定フィールドについて説明しています。

表2.2 データベース URI

フィールド	タイプ	説明
DB_URI (必須)	文字列	認証情報を含む、データベースにアクセスするための URI。 **DB_URI** フィールドの例: **postgresql://quayuser:quaypass@quay-server.example.com:5432/quay**

2.5.2. データベース接続引数

オプションの接続引数は、**DB_CONNECTION_ARGS** パラメーターで設定されます。**DB_CONNECTION_ARGS** で定義されたキーと値のペアの一部は汎用的なものも、データベース固有のものもあります。

以下の表は、データベース接続引数について説明しています。

表2.3 データベース接続引数

フィールド	タイプ	説明
DB_CONNECTION_ARGS	オブジェクト	タイムアウトや SSL などのデータベースの任意の接続引数。
.autorollback	ブール値	スレッドローカル接続を使用するかどうか。 常に **true**である必要があります。
.threadlocals	ブール値	自動ロールバック接続を使用するかどうか。 常に **true**である必要があります。

2.5.2.1. PostgreSQL SSL 接続引数

SSL では、設定はデプロイするデータベースによって異なります。以下の例は、PostgreSQL SSL 設定を示しています。

```
DB_CONNECTION_ARGS:
  sslmode: verify-ca
  sslrootcert: /path/to/cacert
```

sslmode オプションは、セキュアな SSL/IP 接続がサーバーにネゴシエートされるかどうか、その優先度を決定します。モードは 6 つあります。

表2.4 SSL オプション

Mode	説明
disable	設定は SSL 以外の接続のみを試みます。
allow	設定は、SSL 以外の接続を最初に試行します。障害が発生したときに、SSL 接続を試行します。
prefer (デフォルト)	設定は最初に SSL 接続を試みます。障害が発生したときに、SSL 以外の接続を試みます。
require	設定は SSL 接続のみを試みます。ルート CA ファイルが存在する場合は、verify-ca が指定されているのと同じ方法で証明書を検証します。

Mode	説明
verify-ca	設定は SSL 接続のみを試行し、サーバー証明書が信頼できる認証局 (CA) によって発行されたことを確認します。
verify-full	SSL 接続のみを試行し、信頼された CA によりサーバー証明書が発行され、要求されたサーバーのホスト名が証明書と一致することを確認します。

PostgreSQL の有効な引数の詳細は、Database Connection Control Functions を参照してください。

2.5.2.2. MySQL SSL 接続引数

以下の例は、MySQL SSL 設定の例を示しています。

```
DB_CONNECTION_ARGS:
  ssl:
    ca: /path/to/cacert
```

MySQL の有効な接続引数に関する情報は、Connecting to the Server Using URI-Like Strings or Key-Value Pairs を参照してください。

2.6. イメージストレージ

このセクションでは、Red Hat Quay で利用可能なイメージストレージ機能と設定フィールドについて説明します。

2.6.1. イメージストレージ機能

以下の表は、Red Hat Quay のイメージストレージ機能について説明しています。

表2.5 ストレージ設定機能

フィールド	タイプ	説明
FEATURE_REPO_MIRROR	ブール値	true に設定されている場合、リポジトリーのミラーリングを有効にします。 **デフォルト:false**
FEATURE_PROXY_STORAGE	ブール値	NGINX を使用してストレージ内のすべての直接ダウンロード URL をプロキシーするかどうか。 **デフォルト:false**

フィールド	タイプ	説明
FEATURE_STORAGE_REPLICATION	ブール値	ストレージエンジン間で自動的にレプリケートするかどうか **のデフォルト:false**

2.6.2. イメージストレージ設定フィールド

以下の表は、Red Hat Quay のイメージストレージ設定フィールドについて説明しています。

表2.6 ストレージ設定フィールド

フィールド	タイプ	説明
DISTRIBUTED_STORAGE_CONFIG (必須)	オブジェクト	Red Hat Quay で使用するストレージエンジンの設定。各キーは、ストレージエンジンの一意の ID を表します。この値は、ストレージエンジンパラメーターを記述するオブジェクトのタプル (キー、値) で設定されます。 **デフォルト:[]**
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS (必須)	文字列の配列	イメージをデフォルトで他のすべてのストレージエンジンに対して完全にレプリケートする必要のあるストレージエンジンの一覧 (**DISTRIBUTED_STORAGE_CONFIG** の ID 別)。
DISTRIBUTED_STORAGE_PREFERENCE (必須)	文字列の配列	使用する優先ストレージエンジン（**DISTRIBUTED_STORAGE_CONFIG**の ID 別）。優先エンジンとは、プルする必要がないかを先にチェックしてから、イメージをプッシュするという意味です。 **デフォルト:false**
MAXIMUM_LAYER_SIZE	文字列	イメージレイヤーの最大許容サイズ。 **パターン**: **^[0-9]+(G|M)$** **例**: **100G** **デフォルト:20G**

2.6.3. ローカルストレージ

以下の YAML は、ローカルストレージを使用した設定のサンプルを示しています。

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - LocalStorage
    - storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
```

2.6.4. OCS/NooBaa

以下の YAML は、Open Container Storage/NooBaa インスタンスを使用した設定例を示しています。

```
DISTRIBUTED_STORAGE_CONFIG:
  rhocsStorage:
    - RHOCSStorage
    - access_key: access_key_here
      secret_key: secret_key_here
      bucket_name: quay-datastore-9b2108a3-29f5-43f2-a9d5-2872174f9a56
      hostname: s3.openshift-storage.svc.cluster.local
      is_secure: 'true'
      port: '443'
      storage_path: /datastorage/registry
```

2.6.5. Ceph / RadosGW Storage / Hitachi HCP:

以下の YAML は、Ceph/RadosGW および Hitachi HCP ストレージを使用した設定例を示しています。

```
DISTRIBUTED_STORAGE_CONFIG:
  radosGWStorage:
    - RadosGWStorage
    - access_key: access_key_here
      secret_key: secret_key_here
      bucket_name: bucket_name_here
      hostname: hostname_here
      is_secure: 'true'
      port: '443'
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
```

2.6.6. AWS S3 ストレージ

以下の YAML は、AWS S3 ストレージを使用した設定のサンプルを示しています。

```
DISTRIBUTED_STORAGE_CONFIG:
  s3Storage:
    - S3Storage
    - host: s3.us-east-2.amazonaws.com
```

```
      s3_access_key: ABCDEFGHIJKLMN
      s3_secret_key: OL3ABCDEFGHIJKLMN
      s3_bucket: quay_bucket
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - s3Storage
```

2.6.7. Google Cloud Storage

以下の YAML は、Google Cloud Storage を使用した設定例を示しています。

```
DISTRIBUTED_STORAGE_CONFIG:
    googleCloudStorage:
        - GoogleCloudStorage
        - access_key: GOOGQIMFB3ABCDEFGHIJKLMN
          bucket_name: quay-bucket
          secret_key: FhDAYe2HeuAKfvZCAGyOioNaaRABCDEFGHIJKLMN
          storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - googleCloudStorage
```

2.6.8. Azure Storage

以下の YAML は、Azure Storage を使用した設定のサンプルを示しています。

```
DISTRIBUTED_STORAGE_CONFIG:
  azureStorage:
    - AzureStorage
    - azure_account_name: azure_account_name_here
      azure_container: azure_container_here
      storage_path: /datastorage/registry
      azure_account_key: azure_account_key_here
      sas_token: some/path/
      endpoint_url: https://[account-name].blob.core.usgovcloudapi.net (1)
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - azureStorage
```

(1) Azure ストレージの **endpoint_url** パラメーターは任意であり、Microsoft Azure Government (MAG) エンドポイントで使用できます。空白のままにすると、**endpoint_url** は通常の Azure リージョンに接続します。

Red Hat Quay 3.7 以降では、MAG Blob サービスのプライマリーエンドポイントを使用する必要があります。MAG Blob サービスのセカンダリーエンドポイントを使用すると、**AuthenticationErrorDetail:Cannot find the claimed account when trying to GetProperties for the account whusc8-secondary** エラーが発生します。

2.6.9. Swift ストレージ:

以下の YAML は、Swift ストレージを使用した設定のサンプルを示しています。

-

```
DISTRIBUTED_STORAGE_CONFIG:
  swiftStorage:
    - SwiftStorage
    - swift_user: swift_user_here
      swift_password: swift_password_here
      swift_container: swift_container_here
      auth_url: https://example.org/swift/v1/quay
      auth_version: 1
      ca_cert_path: /conf/stack/swift.cert"
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - swiftStorage
```

2.7. REDIS 設定フィールド

本セクションでは、Redis デプロイメントで利用可能な設定フィールドについて説明します。

2.7.1. ビルドログ

Redis デプロイメントには、以下のビルドログ設定フィールドを利用できます。

表2.7 ビルドログの設定

フィールド	タイプ	説明
BUILDLOGS_REDIS (必須)	オブジェクト	ビルドログキャッシュ用の Redis 接続の詳細
.host (必須)	文字列	Redis にアクセスできるホスト名。 **例:** **quay-server.example.com**
.port (必須)	数値	Redis にアクセスできるポート。 **例:** **6379**
.password	文字列	Redis にアクセスできるポート **例:** **strongpassword**
.port (必須)	数値	Redis にアクセスできるポート。 **例:** **6379**
ssl	ブール値	Redis と Quay 間の TLS 通信を有効にするかどうか。デフォルトは false です。

2.7.2. ユーザーイベント

Redis デプロイメントには、以下のユーザーイベントフィールドを使用できます。

表2.8 ユーザーイベント設定

フィールド	タイプ	説明
USER_EVENTS_REDIS (必須)	オブジェクト	ユーザーイベント処理の Redis 接続の詳細
.host (必須)	文字列	Redis にアクセスできるホスト名。 **例:** **quay-server.example.com**
.port (必須)	数値	Redis にアクセスできるポート。 **例:** **6379**
.password	文字列	Redis にアクセスできるポート **例:** **strongpassword**
ssl	ブール値	Redis と Quay 間の TLS 通信を有効にするかどうか。デフォルトは false です。

2.7.3. redis の設定例

以下の YAML は、Redis を使用した設定例を示しています。

```
BUILDLOGS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379
    ssl: true

USER_EVENTS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379
    ssl: true
```

注記

デプロイで Azure Cache for Redis を使用し、**ssl** が **true** に設定されている場合、ポートは既定で **6380** になります。

2.8. MODELCACHE 設定オプション

以下のオプションは、ModelCache を設定するために Red Hat Quay で利用できます。

2.8.1. memcache 設定オプション

memcache は、デフォルトの ModelCache 設定オプションです。memcache を使用すると、追加の設定は必要ありません。

2.8.2. 単一の Redis 設定オプション

以下の設定は、オプションの読み取り専用レプリカを持つ単一の Redis インスタンス用です。

```
DATA_MODEL_CACHE_CONFIG:
  engine: redis
  redis_config:
    primary:
       host: <host>
       port: <port>
       password: <password if ssl is true>
      ssl: <true | false >
    replica:
       host: <host>
       port: <port>
       password: <password if ssl is true>
      ssl: <true | false >
```

2.8.3. クラスター化された Redis 設定オプション

クラスター化された Redis インスタンスには、次の設定を使用します。

```
DATA_MODEL_CACHE_CONFIG:
  engine: rediscluster
  redis_config:
    startup_nodes:
      - host: <cluster-host>
        port: <port>
    password: <password if ssl: true>
    read_from_replicas: <true|false>
    skip_full_coverage_check: <true | false>
    ssl: <true | false >
```

2.9. タグの有効期限の設定フィールド

以下のタグの有効期限設定フィールドは Red Hat Quay で利用できます。

表2.9 タグの有効期限の設定フィールド

フィールド	タイプ	説明
FEATURE_GARBAGE_COLLECTION	ブール値	リポジトリーのガベージコレクションを有効にするかどうか。 **デフォルト:** True

フィールド	タイプ	説明
TAG_EXPIRATION_OPTIONS (必須)	文字列の配列	有効にすると、名前空間内のタグの有効期限についてユーザーが選択できるオプション。 **パターン:** **^[0-9]+(w|m|d|h|s)$**
DEFAULT_TAG_EXPIRATION (必須)	文字列	タイムマシンのデフォルトの設定可能なタグの有効期限。 **パターン:** **^[0-9]+(w|m|d|h|s)$** **デフォルト 2w**
FEATURE_CHANGE_TAG_EXPIRATION	ブール値	ユーザーおよび組織が namespace のタグの有効期限を変更できるかどうか。 **デフォルト:** True

2.9.1. タグの有効期限の設定例

以下の YAML は、タグの有効期限の設定例を示しています。

```
DEFAULT_TAG_EXPIRATION: 2w
TAG_EXPIRATION_OPTIONS:
  - 0s
  - 1d
  - 1w
  - 2w
  - 4w
```

2.10. 自動化のための RED HAT QUAY の事前設定

Red Hat Quay には、自動化をサポートする複数の設定オプションがあります。これらのオプションはデプロイメントの前に設定でき、ユーザーインターフェイスとの対話の必要性を最小限に抑えることができます。

2.10.1. API による最初のユーザー作成の許可

/api/v1/user/initialize API を使用して最初のユーザー を作成するには、**FEATURE_USER_INITIALIZE** パラメーターを **true** に設定します。既存の組織の OAuth アプリケーションによって生成された OAuth トークンを必要とする他のすべてのレジストリー API 呼び出しとは異なり、API エンドポイントには認証は必要ありません。

Red Hat Quay のデプロイ後に、API を使用して他のユーザーがすでに作成されていない限り、**quayadmin** などのユーザーを作成できます。詳細は、API を使用した最初のユーザーの作成 を参照してください。

2.10.2. API 一般アクセスの有効化

Red Hat Quay レジストリー API の一般的なアクセスを許可するには、設定オプション **BROWSER_API_CALLS_XHR_ONLY** を **false** に設定します。

2.10.3. スーパーユーザーの追加

Red Hat Quay のデプロイ後に、ユーザーを作成できます。最初のユーザーには、全パーミッションが割り当てられた管理者権限を付与することをお勧めします。すべてのパーミッションは、**SUPER_USER** 設定オブジェクトを使用して事前に設定できます。以下に例を示します。

```
...
SERVER_HOSTNAME: quay-server.example.com
SETUP_COMPLETE: true
SUPER_USERS:
  - quayadmin
...
```

2.10.4. ユーザー作成の制限

スーパーユーザーを設定したら、新しいユーザーをスーパーユーザーグループに作成する機能を制限できます。ユーザー作成を制限するには、**FEATURE_USER_CREATION** を **false** に設定します。以下に例を示します。

```
...
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
...
```

2.10.5. 新機能の有効化

新規の Red Hat Quay 3.7 機能を使用するには、以下の機能の一部またはすべてを有効にします。

```
...
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: true
FEATURE_PROXY_CACHE: true
FEATURE_STORAGE_REPLICATION: true
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
...
```

2.10.6. 自動化の推奨設定

自動化には、以下の **config.yaml** パラメーターが推奨されます。

```
...
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
...
```

2.10.7. 初期設定を使用した Red Hat Quay Operator のデプロイ

以下の手順に従って、初期設定を使用して OpenShift Container Platform に Red Hat Quay をデプロイします。

前提条件

- **oc** CLI がインストールされている。

手順

1. 設定ファイルを使用してシークレットを作成します。

```
$ oc create secret generic -n quay-enterprise --from-file config.yaml=./config.yaml init-config-bundle-secret
```

2. **quayregistry.yaml** ファイルを作成します。以下のように、管理対象外のコンポーネントを特定し、作成されたシークレットを参照します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: init-config-bundle-secret
```

3. Red Hat Quay レジストリーをデプロイします。

```
$ oc create -n quay-enterprise -f quayregistry.yaml
```

次のステップ

- API を使用した最初のユーザーの作成

2.10.8. API を使用した Red Hat Quay のデプロイ

本セクションでは、API を使用して Red Hat Quay をデプロイする方法について説明します。

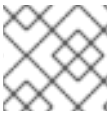
前提条件

- 設定オプション **FEATURE_USER_INITIALIZE** は **true** に設定する。
- データベースにユーザーが存在していない。

Red Hat Quay デプロイメントを事前に設定する方法は、自動化のための Red Hat Quay の設定 セクションを参照してください。

2.10.8.1. API を使用した最初のユーザーの作成

以下の手順に従って、Red Hat Quay 組織で最初のユーザーを作成します。



注記

この手順では、**"access_token": true** を指定して OAuth トークンを要求します。

- **status.registryEndpoint** URL を使用して、以下のコマンドを入力し、**/api/v1/user/initialize** API を呼び出してユーザー名、パスワード、およびメールアドレスを渡します。

```
$  curl -X POST -k  https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/user/initialize --header 'Content-Type: application/json' --data '{ "username": "quayadmin", "password":"quaypass123", "email": "quayadmin@example.com", "access_token": true}'
```

成功すると、このコマンドはユーザー名、メール、および暗号化されたパスワードが含まれるオブジェクトを返します。以下に例を示します。

```
{"access_token":"6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED", "email":"quayadmin@example.com","encrypted_password":"1nZMLH57RIE5UGdL/yYpDOHLqiNCgimb6W9kfF8MjZ1xrfDpRyRs9NUnUuNuAitW","username":"quayadmin"}
```

データベースにユーザーが存在している場合は、エラーが返されます。

```
{"message":"Cannot initialize user in a non-empty database"}
```

パスワードが 8 文字以上でない場合や、空白が含まれている場合には、エラーが返されます。

```
{"message":"Failed to initialize user: Invalid password, password must be at least 8 characters and contain no whitespace."}
```

2.10.8.2. OAuth トークンの使用

API の呼び出し後に、返される OAuth コードを指定して残りの Red Hat Quay API を呼び出すことができます。

前提条件

- **/api/v1/user/initialize** API を呼び出し、ユーザー名、パスワード、およびメールアドレスに渡している。

手順

- 以下のコマンドを入力して、現在のユーザーの一覧を取得します。

```
$ curl -X GET -k -H "Authorization: Bearer 6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/superuser/users/
```

出力例:

```
{
   "users": [
      {
         "kind": "user",
         "name": "quayadmin",
         "username": "quayadmin",
         "email": "quayadmin@example.com",
         "verified": true,
         "avatar": {
            "name": "quayadmin",
            "hash":
"3e82e9cbf62d25dec0ed1b4c66ca7c5d47ab9f1f271958298dea856fb26adc4c",
            "color": "#e7ba52",
            "kind": "user"
         },
         "super_user": true,
         "enabled": true
      }
   ]
}
```

このインスタンスでは、これまで作成した唯一のユーザーであるため、**quayadmin** ユーザーの詳細が返されます。

2.10.8.3. API を使用した組織の作成

以下の手順では、API を使用して Red Hat Quay 組織を作成する方法を説明します。

前提条件

- **/api/v1/user/initialize** API を呼び出し、ユーザー名、パスワード、およびメールアドレスに渡している。
- 返された OAuth コードを指定して、残りの Red Hat Quay API を呼び出している。

手順

1. 組織を作成するには、**api/v1/organization/** エンドポイントへの POST 呼び出しを使用します。

```
$ curl -X POST -k --header 'Content-Type: application/json' -H "Authorization: Bearer
6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://example-registry-quay-
quay-enterprise.apps.docs.quayteam.org/api/v1/organization/ --data '{"name": "testorg",
"email": "testorg@example.com"}'
```

出力例:

```
"Created"
```

2. 以下のコマンドを入力して、作成した組織の詳細を取得できます。

```
$ curl -X GET -k --header 'Content-Type: application/json' -H "Authorization: Bearer
6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://min-registry-quay-quay-
enterprise.apps.docs.quayteam.org/api/v1/organization/testorg
```

出力例:

```
{
    "name": "testorg",
    "email": "testorg@example.com",
    "avatar": {
        "name": "testorg",
        "hash": "5f113632ad532fc78215c9258a4fb60606d1fa386c91b141116a1317bf9c53c8",
        "color": "#a55194",
        "kind": "user"
    },
    "is_admin": true,
    "is_member": true,
    "teams": {
        "owners": {
            "name": "owners",
            "description": "",
            "role": "admin",
            "avatar": {
                "name": "owners",
                "hash":
"6f0e3a8c0eb46e8834b43b03374ece43a030621d92a7437beb48f871e90f8d90",
                "color": "#c7c7c7",
                "kind": "team"
            },
            "can_view": true,
            "repo_count": 0,
            "member_count": 1,
            "is_synced": false
        }
    },
    "ordered_teams": [
        "owners"
    ],
    "invoice_email": false,
    "invoice_email_address": null,
    "tag_expiration_s": 1209600,
    "is_free_account": true
}
```

2.11. 基本設定フィールド

表2.10 基本設定

フィールド	タイプ	説明

フィールド	タイプ	説明
REGISTRY_TITLE	文字列	指定されている場合は、レジストリーの長いタイトル **デフォルト:** **Quay Enterprise**
REGISTRY_TITLE_SHORT	文字列	指定されている場合は、レジストリーの短縮型のタイトル **デフォルト:** **Quay Enterprise**
BRANDING	オブジェクト	Red Hat Quay UI のロゴおよび URL のカスタムブランディング
.logo (必須)	文字列	主なロゴイメージ URL。 **例:** **/static/img/quay-horizontal-color.svg**
.footer_img	文字列	UI フッターのロゴ。 **例:** **/static/img/RedHat.svg**
.footer_url	文字列	フッターイメージへのリンク。 **例:** **https://redhat.com**
CONTACT_INFO	文字列の配列	指定されている場合は、連絡先ページに表示される連絡先情報。連絡先が 1 つしか指定されていない場合は、連絡先のフッターが直接リンクされます。
[0]	文字列	電子メールを送信するためのリンクを追加します。 **パターン:** **^mailto:(.)+$** **例:** **mailto:support@quay.io**

フィールド	タイプ	説明
[1]	文字列	IRC チャットルームにアクセスするためのリンクを追加します。 **パターン:** **^irc://(.)+$** **例:** **irc://chat.freenode.net:6665/quay**
[2]	文字列	電話番号を呼び出すためのリンクを追加します。+ **パターン:** **^tel:(.)+$** **例:** **tel:+1-888-930-3475**
[3]	文字列	定義された URL へのリンクを追加します。 **パターン:** **^http(s)?://(.)+$** Example: **https://twitter.com/quayio**

2.12. SSL 設定フィールド

表2.11 SSL 設定

フィールド	タイプ	説明

フィールド	タイプ	説明
PREFERRED_URL_SCHEME	文字列	**http** または **https** のいずれか。クライアントから Quay への通信パスに TLS 暗号化がない場合に限り、ユーザーは **PREFERRED_URL_SCHEME** を **http** に設定することに注意してください。 + TLS を終了するロードバランサー、リバースプロキシー (Nginx など) を使用する場合、またはカスタム SSL 証明書で Quay を直接使用する場合、ユーザーは **PREFERRED_URL_SCHEME `を`https** に設定する必要があります。ほとんどの場合、**PREFERRED_URL_SCHEME** は **https** である必要があります。 **デフォルト:http**
SERVER_HOSTNAME (必須)	文字列	スキームなしで Red Hat Quay にアクセスできる URL **例:** **quay-server.example.com**
SSL_CIPHERS	文字列の配列	指定されている場合、有効化および無効化する SSL 暗号の nginx 定義の一覧 **例:** [**CAMELLIA, !3DES**]
SSL_PROTOCOLS	文字列の配列	指定されている場合、nginx は、一覧で定義される SSL プロトコルの一覧を有効にするように設定されます。リストから SSL プロトコルを削除すると、Red Hat Quay の起動時にそのプロトコルが無効になります。 **例:** **['TLSv1','TLSv1.1','TLSv1.2', `TLSv1.3**]`

フィールド	タイプ	説明
SESSION_COOKIE_SECURE	ブール値	セッションクッキーに **secure** プロパティーを設定するべきであるかどうか。 **デフォルト:** False **推奨:** SSL を使用するインストールの場合はすべて、True に n 設定。

2.12.1. SSL の設定

1. 証明書ファイルとプライマリーキーファイルを設定ディレクトリーにコピーして、それぞれ **ssl.cert** と **ssl.key** という名前が付けられていることを確認します。

```
$ cp ~/ssl.cert $QUAY/config
$ cp ~/ssl.key $QUAY/config
$ cd $QUAY/config
```

2. **config.yaml** ファイルを編集し、Quay で TLS を処理できるように指定します。

   config.yaml

```
...
SERVER_HOSTNAME: quay-server.example.com
...
PREFERRED_URL_SCHEME: https
...
```

3. **Quay** コンテナーを停止し、レジストリーを再起動します。

2.13. RED HAT QUAY コンテナーへの TLS 証明書の追加

カスタム TLS 証明書を Red Hat Quay に追加するには、Red Hat Quay の config ディレクトリーの下に **extra_ca_certs/** という名前の新しいディレクトリーを作成します。必要なサイト固有の TLS 証明書をこの新しいディレクトリーにコピーします。

2.13.1. TLS 証明書を Red Hat Quay に追加

1. コンテナーに追加される証明書を表示します。

```
$ cat storage.crt
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
[...]
-----END CERTIFICATE-----
```

2. certs ディレクトリーを作成し、そこに証明書をコピーします。

```
$ mkdir -p quay/config/extra_ca_certs
$ cp storage.crt quay/config/extra_ca_certs/
$ tree quay/config/
├── config.yaml
├── extra_ca_certs
│   ├── storage.crt
```

3. **Quay** コンテナーの **CONTAINER ID** を **podman ps** で取得します。

```
$ sudo podman ps
CONTAINER ID  IMAGE                           COMMAND               CREATED
STATUS             PORTS
5a3e82c4a75f  <registry>/<repo>/quay:v3.7.10 "/sbin/my_init"  24 hours ago  Up
18 hours  0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 443/tcp  grave_keller
```

4. その ID でコンテナーを再起動します。

```
$ sudo podman restart 5a3e82c4a75f
```

5. コンテナーの名前空間にコピーされた証明書を調べます。

```
$ sudo podman exec -it 5a3e82c4a75f cat /etc/ssl/certs/storage.pem
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
```

2.14. LDAP 設定フィールド

表2.12 LDAP の設定

フィールド	タイプ	説明
AUTHENTICATION_TYPE (必須)	文字列	**LDAP** に設定する必要があります。
FEATURE_TEAM_SYNCING	ブール値	認証エンジン (LDAP または Keystone) のバッキンググループからチームメンバーシップを同期するかどうか。 **デフォルト: true**
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP	ブール値	有効にすると、スーパーユーザー以外のユーザーは LDAP を使用してチームの同期を設定できます **デフォルト値: false**
LDAP_ADMIN_DN	文字列	LDAP 認証の管理 DN。
LDAP_ADMIN_PASSWD	文字列	LDAP 認証の管理パスワード。

フィールド	タイプ	説明
LDAP_ALLOW_INSECURE_FALLBACK	ブール値	LDAP 認証で SSL の非セキュアなフォールバックを許可するかどうか。
LDAP_BASE_DN	文字列の配列	LDAP 認証のベース DN。
LDAP_EMAIL_ATTR	文字列	LDAP 認証のメール属性。
LDAP_UID_ATTR	文字列	LDAP 認証の uid 属性。
LDAP_URI	文字列	LDAP URI。
LDAP_USER_FILTER	文字列	LDAP 認証のユーザーフィルター。
LDAP_USER_RDN	文字列の配列	LDAP 認証のユーザー RDN。
TEAM_RESYNC_STALE_TIME	文字列	チームの同期が有効になっている場合、必要に応じてメンバーシップを確認して再同期する頻度 Pattern: **^[0-9]+(w|m|d|h|s)$** **例:** **2h** **デフォルト:** **30m**

2.14.1. LDAP 設定の例

$QUAY/config/config.yaml

```
AUTHENTICATION_TYPE: LDAP
...
LDAP_ADMIN_DN: uid=testuser,ou=Users,o=orgid,dc=jumpexamplecloud,dc=com
LDAP_ADMIN_PASSWD: samplepassword
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
    - o=orgid
    - dc=example
    - dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://ldap.example.com:389
LDAP_USER_RDN:
    - ou=Users
```

2.15. 設定フィールドのミラーリング

表2.13 ミラーリング設定

フィールド	タイプ	説明
FEATURE_REPO_MIRROR	ブール値	リポジトリーミラーリングを有効化または無効化します **デフォルト:false**
REPO_MIRROR_INTERVAL	数値	次にリポジトリーミラー候補をチェックするまでの秒数。 **デフォルト:**30
REPO_MIRROR_SERVER_HOSTNAME	文字列	**SERVER_HOSTNAME** は、ミラーリングの宛先に置き換えます。 **デフォルト:**None **例**: **openshift-quay-service**
REPO_MIRROR_TLS_VERIFY	ブール値	HTTPS を必要とし、ミラー時に Quay レジストリーの証明書を検証します。 **デフォルト:false**
REPO_MIRROR_ROLLBACK	ブール値	**true** に設定されている場合、リポジトリーはミラー試行の失敗後にロールバックします。 **デフォルト**:**false**

2.16. セキュリティースキャナー設定フィールド

表2.14 セキュリティースキャナーの設定

フィールド	タイプ	説明
FEATURE_SECURITY_SCANNER	ブール値	セキュリティースキャナー **デフォルト:false**

フィールド	タイプ	説明
FEATURE_SECURITY_NOTIFICATIONS	ブール値	セキュリティースキャナーが有効になっている場合、セキュリティー通知を有効にするか、オフにします **デフォルト値: false**
SECURITY_SCANNER_V4_REINDEX_THRESHOLD	文字列	このパラメーターは、最終のインデックス作成から状態が変更されたか、以前に失敗したマニフェストのインデックスをもう一度作成するまで待機する最小時間を判断するのに使用します。データは manifestsecuritystatus テーブルの **last_indexed datetime** から計算されます。インデックス作成実行時に必ず、失敗したマニフェストのインデックスを再度作成しないように、このパラメーターを使用します。再インデックスのデフォルト時間は 300 秒です。
SECURITY_SCANNER_V4_ENDPOINT	文字列	V4 セキュリティースキャナーのエンドポイント **パターン:** **^http(s)?://(.)+$** **例:** **http://192.168.99.101:6060**
SECURITY_SCANNER_V4_PSK	文字列	Clair 用に生成された共有キー (PSK)
SECURITY_SCANNER_INDEXING_INTERVAL	数値	セキュリティースキャナーのインデックス作成の間隔 (秒単位) **デフォルト**: 30
SECURITY_SCANNER_ENDPOINT	文字列	V2 セキュリティースキャナーのエンドポイント **パターン:** **^http(s)?://(.)+$** **例:** **http://192.168.99.100:6060**

フィールド	タイプ	説明
SECURITY_SCANNER_INDEXING_INTERVAL	文字列	このパラメーターは、セキュリティースキャナーのインデックス作成の間隔 (秒単位) を決定するために使用されます。インデックスがトリガーされると、Red Hat Quay は、Clair でインデックス化する必要のあるマニフェストに対してそのデータベースをクエリーします。これには、インデックス化されていないマニフェストや、以前にインデックスに失敗したマニフェストが含まれます。

以下は、インデックス変更の特別なケースです。

Clair v4 がマニフェストをインデックス化する場合は、結果として、決定論的なものである必要があります。たとえば、同じマニフェストが同じインデックスレポートを生成する必要があります。これは、異なるスキャナーを使用するとレポートで返される特定のマニフェストに関連して異なる情報を生成するため、スキャナーが変更されるまで True となります。そのため、Clair v4 はインデックスエンジン (**/indexer/api/v1/index_state**) の状態表現を公開し、スキャナー設定が変更されたかどうかを判別します。

Red Hat Quay は、Quay のデータベースの解析時にこれをインデックスレポートに保存し、このインデックス状態を活用します。以前にスキャンされてからこの状態が変更された場合、Quay は定期的なインデックスプロセス時にマニフェストの再作成を試行します。

デフォルトでは、このパラメーターは 30 秒に設定されています。インデックス作成のプロセスをより頻繁に実行する場合は、時間を短縮します。たとえば、新規タグをプッシュしてから、30 秒待たずに、UI でセキュリティースキャンの結果を表示する場合などです。また、ユーザーは要求パターンを Clair に制御し、Quay データベースで実行されるデータベース操作のパターンをより詳細に制御する必要がある場合にパラメーターを変更することもできます。

2.17. OCI および HELM 設定フィールド

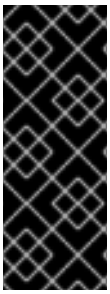
Helm のサポートが **FEATURE_GENERAL_OCI_SUPPORT** プロパティーでサポートされるようになりました。機能を明示的に有効にする必要がある場合 (機能が無効にされている場合や、デフォルトで有効にされていないバージョンからアップグレードした場合など) は、OCI アーティファクトの使用を有効にするために 2 つのプロパティーを Quay 設定に追加する必要があります。

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

表2.15 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

フィールド	タイプ	説明
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。

2.18. アクションログ設定フィールド

2.18.1. アクションログストレージ設定

表2.16 アクションログストレージ設定

フィールド	タイプ	説明
FEATURE_LOG_EXPORT	ブール値	アクションログのエクスポートを許可するかどうか **デフォルト: True**
LOGS_MODEL	文字列	セキュリティースキャナーを有効または無効にします。 **値: database**, **transition_reads_both_writes_es**, **elasticsearch** のいずれか。 **デフォルト: database**
LOGS_MODEL_CONFIG	オブジェクト	アクションログのログモデル設定

- LOGS_MODEL_CONFIG [オブジェクト]: アクションログ用のログモデル設定
  - **elasticsearch_config** [オブジェクト]: Elasticsearch クラスターの設定
    - **access_key** [文字列] :Elasticsearch のユーザー (AWS ES の場合は IAM キー)
      - **例**: **some_string**
    - **ホスト** [文字列]: Elasticsearch クラスターのエンドポイント
      - **例**: **host.elasticsearch.example**

- **index_prefix** [文字列]。Elasticsearch のインデックスの接頭辞
  - **例**: **logentry_**
- **index_settings** [オブジェクト]: Elasticsearch のインデックス設定
- **use_ssl** [ブール値]。Elasticsearch に ssl を使用します。デフォルトは true です。
  - **例**: **True**
- **secret_key** [文字列] :Elasticsearch のパスワード (AWS ES の場合は IAM シークレット)
  - **例**: **some_secret_string**
- **aws_region** [文字列]: Amazon Web サービスの地域
  - **例**: **us-east-1**
- **port** [番号]: Elasticsearch クラスターのエンドポイントポート
  - **例**: **1234**

- **kinesis_stream_config** [オブジェクト]: AWS Kinesis ストリームの設定
  - **aws_secret_key** [文字列]: AWS の秘密鍵
    - **例**: **some_secret_key**
  - **stream_name** [文字列]: アクションログの送信先となる Kinesis ストリーム
    - **例**: **logentry-kinesis-stream**
  - **aws_access_key** [文字列]: AWS アクセスキー
    - **例**: **some_access_key**
  - **retries** [番号]: 一回のリクエストに対する最大試行回数
    - **例**: **5**
  - **read_timeout** [番号]: 接続の読み込み時にタイムアウトするまでの秒数
    - **例**: **5**
  - **max_pool_connections** [番号]: コネクションプールに保持するコネクションの最大数
    - **例**: **10**
  - **aws_region** [文字列]: AWS のリージョン
    - **例**: **us-east-1**
  - **connect_timeout** [番号]: 接続を試みる際のタイムアウトまでの秒数
    - **例**: **5**
- **producer** [文字列]: Elasticsearch にロギングする場合は、producer を記録します。
  - **enum**: kafka、elasticsearch、kinesis_stream

- **例**: **kafka**

- kafka_config [オブジェクト]: Kafka クラスターの設定
  - topic [文字列]: ログエントリーを公開する Kafka トピック
    - **例**: **logentry**
  - bootstrap_servers [配列]: クライアントをブートストラップさせる Kafka ブローカーのリスト
  - max_block_seconds [番号]: **send()** の実行中に、バッファーがいっぱいになったり、メタデータが利用できないなどの理由でブロックする最大秒数
    - **例**: **10**

2.18.2. アクションログのローテーションおよびアーカイブ設定

表2.17 アクションログのローテーションおよびアーカイブ設定

フィールド	タイプ	説明
FEATURE_ACTION_LOG_ROTATION	ブール値	ログローテーションおよび archival を有効にすると、30 日以上経過したすべてのログをストレージに移動します。 **デフォルト**: **false**
ACTION_LOG_ARCHIVE_LOCATION	文字列	アクションログのアーカイブが有効な場合、アーカイブされたデータを配置するストレージエンジン **例**: **s3_us_east**
ACTION_LOG_ARCHIVE_PATH	文字列	アクションログのアーカイブが有効な場合、アーカイブされたデータを配置するストレージのパス **例**: **archives/actionlogs**
ACTION_LOG_ROTATION_THRESHOLD	文字列	ログをローテーションする間隔。 **例**: **30d**

2.19. ビルドログ設定フィールド

表2.18 ビルドログ設定フィールド

フィールド	タイプ	説明

フィールド	タイプ	説明
FEATURE_READER_BUILD_LOGS	ブール値	true に設定されている場合、書き込みアクセスや管理アクセスがある場合だけでなく、リポジトリーへの読み取りアクセスを持つユーザーがビルドログを読み取ることができます。 **デフォルト:False**
LOG_ARCHIVE_LOCATION	文字列	アーカイブされたビルドログを配置する、DISTRIBUTED_STORAGE_CONFIG で定義されたストレージの場所 **例:s3_us_east**
LOG_ARCHIVE_PATH	文字列	アーカイブされたビルドログを JSON 形式で配置する、設定されたストレージエンジンの下のパス **例:archives/buildlogs**

2.20. DOCKERFILE ビルドトリガーフィールド

表2.19 Dockerfile ビルドのサポート

フィールド	タイプ	説明
FEATURE_BUILD_SUPPORT	ブール値	Dockerfile ビルドをサポートするかどうか。 **初期値:** False
SUCCESSIVE_TRIGGER_FAILURE_DISABLE_THRESHOLD	数値	None ではない場合に、ビルドトリガーが自動的に無効にされる前に、連続で失敗できる回数。 **デフォルト:**100
SUCCESSIVE_TRIGGER_INTERNAL_ERROR_DISABLE_THRESHOLD	数値	None ではない場合に、ビルドトリガーが自動的に無効にされる前に、連続で発生可能な内部エラーの回数。 **デフォルト:**5

2.20.1. GitHub ビルドトリガー

表2.20 GitHub ビルドトリガー

フィールド	タイプ	説明
FEATURE_GITHUB_BUILD	ブール値	GitHub ビルドトリガーをサポートしているかどうか **デフォルト:** False
GITHUB_TRIGGER_CONFIG	オブジェクト	ビルドトリガーに GitHub (Enterprise) を使用するための設定
.GITHUB_ENDPOINT (必須)	文字列	GitHub (Enterprise) のエンドポイント **例:** **https://github.com/**
.API_ENDPOINT	文字列	使用する GitHub (Enterprise) API のエンドポイント。**github.com** に対して上書きする必要があります。 **例**: **https://api.github.com/**
.CLIENT_ID (必須)	文字列	この Red Hat Quay インスタンスの登録されたクライアント ID。GITHUB_LOGIN_CONFIG と共有にすることはできません。
.CLIENT_SECRET (必須)	文字列	この Red Hat Quay インスタンスの登録されたクライアントシークレット。

2.20.2. Bitbucket ビルドトリガー

表2.21 Bitbucket ビルドトリガー

フィールド	タイプ	説明
FEATURE_BITBUCKET_BUILD	ブール値	Bitbucket ビルドトリガーをサポートしているかどうか **デフォルト:** False

フィールド	タイプ	説明
BITBUCKET_TRIGGER_CONFIG	オブジェクト	ビルドトリガーに BitBucket を使用するための設定
.CONSUMER_KEY (必須)	文字列	この Quay インスタンスの登録されたコンシューマーキー (クライアント ID)
.CONSUMER_SECRET (必須)	文字列	この Quay インスタンスの、登録されたコンシューマーシークレット (クライアントシークレット)

2.20.3. GitLab ビルドトリガー

表2.22 GitLab ビルドトリガー

フィールド	タイプ	説明
FEATURE_GITLAB_BUILD	ブール値	GitLab ビルドトリガーをサポートしているかどうか **デフォルト:** False
GITLAB_TRIGGER_CONFIG	オブジェクト	ビルドトリガーに Gitlab を使用するための設定
.GITLAB_ENDPOINT (必須)	文字列	Gitlab (Enterprise) が実行されているエンドポイント
.CLIENT_ID (必須)	文字列	この Quay インスタンスの、登録されたクライアント ID
.CLIENT_SECRET (必須)	文字列	この Quay インスタンスの、登録されたクライアントシークレット

2.21. OAUTH 設定フィールド

表2.23 OAuth フィールド

フィールド	タイプ	説明
DIRECT_OAUTH_CLIENTID_WHITELIST	文字列の配列	ユーザーの承認なしに直接 OAuth 承認を実行できる **Red Hat Quay 管理** アプリケーションのクライアント ID の一覧。

2.21.1. GitHub OAuth 設定フィールド

表2.24 GitHub OAuth フィールド

フィールド	タイプ	説明
FEATURE_GITHUB_LOGIN	ブール値	GitHub ログインがサポートされるかどうか **デフォルト: **False**
GITHUB_LOGIN_CONFIG	オブジェクト	外部ログインプロバイダーとして GitHub (Enterprise) を使用するための設定
.ALLOWED_ORGANIZATIONS	文字列の配列	ORG_RESTRICT オプションを使用するためにホワイトリスト化された GitHub (Enterprise) 組織の名前
.API_ENDPOINT	文字列	使用する GitHub (Enterprise) API のエンドポイント。github.com **例: https://api.github.com/**
.CLIENT_ID (必須)	文字列	この Red Hat Quay インスタンスの登録されたクライアント ID。GITHUB_TRIGGER_CONFIG **例: 0e8dbe15c4c7630b6780**
.CLIENT_SECRET (必須)	文字列	Red Hat Quay インスタンスの登録クライアントシークレット。 **例: e4a58ddd3d7408b7aec109e85564a0d153d3e846**
.GITHUB_ENDPOINT (必須)	文字列	GitHub(Enterprise) のエンドポイント **例:https://github.com/**
.ORG_RESTRICT	ブール値	true の場合、このプロバイダーを使用してログインできるのは組織のホワイトリスト内のユーザーのみです。

2.21.2. Google OAuth 設定フィールド

表2.25 Google OAuth フィールド

フィールド	タイプ	説明
FEATURE_GOOGLE_LOGIN	ブール値	Google ログインがサポートされるかどうか **デフォルト: **False**
GOOGLE_LOGIN_CONFIG	オブジェクト	外部認証に Google を使用するための設定
.CLIENT_ID (必須)	文字列	この Red Hat Quay インスタンスの登録されたクライアント ID **例: 0e8dbe15c4c7630b6780**
.CLIENT_SECRET (必須)	文字列	Red Hat Quay インスタンスの登録クライアントシークレット **例:** e4a58ddd3d7408b7aec109e85564a0d153d3e846

2.22. ネストされたリポジトリー設定フィールド

Red Hat Quay 3.6 では、ネストされたリポジトリーパス名のサポートが **FEATURE_EXTENDED_REPOSITORY_NAMES** プロパティーに追加されました。このオプションの設定は、デフォルトで config.yaml に追加されます。有効にすると、リポジトリー名で / を使用できます。

```
FEATURE_EXTENDED_REPOSITORY_NAMES: true
```

表2.26 OCI およびネストされたリポジトリー設定フィールド

フィールド	タイプ	説明
FEATURE_EXTENDED_REPOSITORY_NAMES	ブール値	ネストされたリポジトリーのサポートを有効にする **デフォルト:** True

2.23. その他の OCI メディアタイプの QUAY への追加

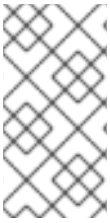
Helm、cosign、および ztsd 圧縮スキームアーティファクトはデフォルトで Red Hat Quay 3.6 に組み込まれています。デフォルトでサポートされていない他の OCI メディアタイプでは、以下の形式を使用して Quay の config.yaml の **ALLOWED_OCI_ARTIFACT_TYPES** 設定に追加できます。

```
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。

2.24. メール設定フィールド

表2.27 メール設定フィールド

フィールド	タイプ	説明
FEATURE_MAILING	ブール値	メールが有効かどうか **デフォルト:False**
MAIL_DEFAULT_SENDER	文字列	指定されている場合、Red Hat Quay がメールを送信する際の **from** として使用されるメールアドレス。何も指定されていない場合は、**support@quay.io** にデフォルト設定されます。**例: support@example.com**
MAIL_PASSWORD	文字列	メールの送信時に使用する SMTP パスワード。
MAIL_PORT	数値	使用する SMTP ポート。指定されていない場合は、デフォルトの 587 になります。

フィールド	タイプ	説明
MAIL_SERVER	文字列	メールの送信に使用する SMTP サーバー。FEATURE_MAILING が true に設定されている場合にのみ必要です。 **例: smtp.example.com**
MAIL_USERNAME	文字列	メールの送信時に使用する SMTP ユーザー名
MAIL_USE_TLS	ブール値	指定されている場合、電子メールの送信に TLS を使用するかどうか **デフォルト: True**

2.25. ユーザー設定フィールド

表2.28 ユーザー設定フィールド

フィールド	タイプ	説明
FEATURE_SUPER_USERS	ブール値	スーパーユーザーがサポートされるかどうか **デフォルト: true**
FEATURE_USER_CREATION	ブール値	ユーザーを作成するかどうか (スーパーユーザー以外) **デフォルト: true**
FEATURE_USER_LAST_ACCESSED	ブール値	ユーザーが最後にアクセスした時間を記録するかどうか **デフォルト: true**
FEATURE_USER_LOG_ACCESS	ブール値	true に設定すると、ユーザーは namespace の監査ログにアクセスできます **デフォルト: false**
FEATURE_USER_METADATA	ブール値	ユーザーメタデータを収集してサポートするかどうか **デフォルト: false**

フィールド	タイプ	説明
FEATURE_USERNAME_CONFIRMATION	ブール値	true に設定すると、生成されたユーザー名を確認できます **デフォルト:true**
FEATURE_USER_RENAME	ブール値	true に設定されている場合、ユーザーは独自の namespace の名前を変更できます。 **デフォルト:false**
FEATURE_INVITE_ONLY_USER_CREATION	ブール値	作成するユーザーは別のユーザーから招待を受ける必要があります。 **デフォルト:false**
FRESH_LOGIN_TIMEOUT	文字列	新規ログイン時にユーザーがパスワードの再入力を要求されるまでの時間 **例**: **5m**
USERFILES_LOCATION	文字列	ユーザーがアップロードしたファイルを配置するストレージエンジンの ID。 **例**:**s3_us_east**
USERFILES_PATH	文字列	ユーザーがアップロードしたファイルを配置するストレージの下のパス。 **例**:**userfiles**
USER_RECOVERY_TOKEN_LIFETIME	文字列	ユーザーアカウントを復元するためのトークンが有効な期間 **パターン**:**^[0-9]+(w|m|d|h|s)$** **デフォルト:30m**

2.26. RECAPTCHA 設定フィールド

表2.29 reCAPTCHA 設定フィールド

フィールド	タイプ	説明
FEATURE_RECAPTCHA	ブール値	ユーザーログインおよびリカバリーに Recaptcha が必要かどうか **デフォルト:** False
RECAPTCHA_SECRET_KEY	文字列	recaptcha が有効にされている場合は、Recaptcha サービスのシークレットキー
RECAPTCHA_SITE_KEY	文字列	recaptcha が有効にされている場合は、Recaptcha サービスのサイトキー

2.27. ACI 設定フィールド

表2.30 ACI 設定フィールド

フィールド	タイプ	説明
FEATURE_ACI_CONVERSION	ブール値	ACI への変換を有効にするかどうか。 **デフォルト:** False
GPG2_PRIVATE_KEY_FILENAME	文字列	ACI の復号化に使用される秘密鍵のファイル名
GPG2_PRIVATE_KEY_NAME	文字列	ACI に署名するために使用されるプライベートキーの名前
GPG2_PUBLIC_KEY_FILENAME	文字列	ACI の暗号化に使用する公開鍵のファイル名

2.28. JWT 設定フィールド

表2.31 JWT 設定フィールド

フィールド	タイプ	説明
JWT_AUTH_ISSUER	文字列	JWT ユーザーのエンドポイント **パターン**:**^http(s)?://(.)+$** **例**:**http://192.168.99.101:6060**

フィールド	タイプ	説明
JWT_GETUSER_ENDPOINT	文字列	JWT ユーザーのエンドポイント **パターン**:**^http(s)?://(.)+$** **例**:**http://192.168.99.101:6060**
JWT_QUERY_ENDPOINT	文字列	JWT クエリーのエンドポイント **パターン**:**^http(s)?://(.)+$** **例**:**http://192.168.99.101:6060**
JWT_VERIFY_ENDPOINT	文字列	JWT 検証のエンドポイント **パターン**:**^http(s)?://(.)+$** **例**:**http://192.168.99.101:6060**

2.29. アプリケーショントークン設定フィールド

表2.32 アプリケーショントークン設定フィールド

フィールド	タイプ	説明
FEATURE_APP_SPECIFIC_TOKENS	ブール値	有効な場合には、ユーザーは Docker CLI で使用するトークンを作成できます。 **デフォルト**: True
APP_SPECIFIC_TOKEN_EXPIRATION	文字列	外部アプリトークンの有効期限。 **デフォルト** なし **パターン**: **^[0-9]+(w|m|d|h|s)$**
EXPIRED_APP_SPECIFIC_TOKEN_GC	文字列	期限切れとなった外部アプリケーションがガベージコレクションが行われるまでに留まる期間。 **デフォルト**: 1d

2.30. その他の設定フィールド

表2.33 その他の設定フィールド

フィールド	タイプ	説明

フィールド	タイプ	説明
ALLOW_PULLS_WITHOUT_STRICT_LOGGING	文字列	true に指定すると、プルの監査ログのエントリーに書き込みできない場合でも、プルは成功します。これは、データベースが読み取り専用の状態にフォールバックし、その間プルを続行する必要がある場合に便利です。 **初期値:** False
AVATAR_KIND	文字列	表示する avatar のタイプ。インライン (ローカル) または Gravatar (gravatar)。 **値:** local, gravatar
BROWSER_API_CALLS_XHR_ONLY	ブール値	有効にされている場合には、ブラウザーから XHR による呼び出しとしてマークが付けられた API のみが許可されます。 **デフォルト:** True
DEFAULT_NAMESPACE_MAXIMUM_BUILD_COUNT	数値	namespace でキューに入れることができるデフォルトの最大ビルド数です。 **デフォルト:** None
ENABLE_HEALTH_DEBUG_SECRET	文字列	指定されている場合は、スーパーユーザーとして認証されていない場合に詳細なデバッグ情報を表示するために正常性エンドポイントに指定できるシークレット
EXTERNAL_TLS_TERMINATION	ブール値	TLS がサポートされているが、Quay の前の層で終了する場合は **true** に設定します。Quay が独自の SSL 証明書を使用して実行されており、TLS トラフィックを直接受信している場合は、**false** に設定します。
FRESH_LOGIN_TIMEOUT	文字列	新規ログイン時にユーザーがパスワードの再入力を要求されるまでの時間 **例: 5m**

フィールド	タイプ	説明
HEALTH_CHECKER	文字列	設定済みのヘルスチェック **例: ('RDSAwareHealthCheck', {'access_key': 'foo', 'secret_key': 'bar'})**
PROMETHEUS_NAMESPACE	文字列	公開されているすべての Prometheus メトリクスに適用される接頭辞 **デフォルト:quay**
PUBLIC_NAMESPACES	文字列の配列	namespace がパブリック namespace 一覧に定義されている場合に、それはユーザーが namespace のメンバーであるかどうかに関係なく、**すべての** ユーザーのリポジトリー一覧ページに表示されます。一般的には、企業のお客様がよく知られた名前空間のセットを設定する際に使用されます。
REGISTRY_STATE	文字列	レジストリーの状態 **値: normal** または **read-only**
SEARCH_MAX_RESULT_PAGE_COUNT	数値	ユーザーが検索で表示できる最大ページ数。 **デフォルト:** 10
SEARCH_RESULTS_PER_PAGE	数値	検索ページでページごとに返される結果数 **デフォルト:** 10
V1_PUSH_WHITELIST	文字列の配列	FEATURE_RESTRICTED_V1_PUSH が true に設定されている場合に V1 push をサポートする namespace 名の配列。
V2_PAGINATION_SIZE	数値	V2 レジストリー API において、1 ページあたりに返される結果の数 **デフォルト:** 50

フィールド	タイプ	説明
WEBHOOK_HOSTNAME_BLACKLIST	文字列の配列	検証時に、ローカルホスト以外に Webhook から禁止するホスト名のセット
CREATE_PRIVATE_REPO_ON_PUSH	ブール値	プッシュで作成された新規リポジトリーがプライベート表示に設定されているかどうか **デフォルト:** True
CREATE_NAMESPACE_ON_PUSH	ブール値	既存の組織への新規プッシュで namespace を作成するかどうか **デフォルト:** False
NON_RATE_LIMITED_NAMESPACES	文字列の配列	**FEATURE_RATE_LIMITS** を使用してレートの制限が有効で、特定の namespace で無制限のアクセス権が必要な場合に、オーバーライドできます。

2.31. レガシー設定フィールド

一部のフィールドは非推奨または廃止されています。

表2.34 レガシー設定フィールド

フィールド	タイプ	説明
FEATURE_BLACKLISTED_EMAILS	ブール値	true に設定すると、メールドメインがブラックリストに指定されている場合には、新しいユーザーアカウントが作成されません。
BLACKLISTED_EMAIL_DOMAINS	文字列の配列	FEATURE_BLACKLISTED_EMAILS が true に設定されている場合に使用されるメールアドレスドメインの一覧 **例**: **"example.com", "example.org"**
BLACKLIST_V2_SPEC	文字列	Red Hat Quay が V2 は **サポート対象外** であることを示す応答を返す Docker CLI バージョン。 **例**: **<1.8.0** **デフォルト**: **<1.6.0**

フィールド	タイプ	説明
DOCUMENTATION_ROOT	文字列	ドキュメントリンクのルート URL
SECURITY_SCANNER_V4_NAMESPACE_WHITELIST	文字列	セキュリティースキャナーを有効にする namespace

第3章 環境変数

Red Hat Quay は、動的に設定する多数の環境変数をサポートします。

3.1. GEO レプリケーション

ストレージバックエンド以外のすべてのリージョンで同じ設定を使用する必要があります。これは、**QUAY_DISTRIBUTED_STORAGE_PREFERENCE** 環境変数を使用して明示的に設定できます。

表3.1 Geo レプリケーションの設定

変数	タイプ	説明
QUAY_DISTRIBUTED_STORAGE_PREFERENCE	文字列	使用する優先されるストレージ (DISTRIBUTED_STORAGE_CONFIG の ID 別)

3.2. データベース接続プール

Red Hat Quay は、すべてが同じコンテナー内で実行する多くの異なるプロセスで設定されています。これらのプロセスの多くは、データベースと連動しています。

有効にすると、データベースと対話する各プロセスには、コネクションプールが含まれます。これらのプロセスごとのコネクションプールは、最大 20 個の接続を維持するように設定されています。高負荷時には、Red Hat Quay コンテナー内のすべてのプロセスの接続プールを満たすことが可能です。特定の展開や負荷の下では、Red Hat Quay がデータベースの設定された最大接続数を超えないようにするための分析が必要になる場合があります。

時間が経つと、接続プールはアイドル接続を解放します。すべての接続をすぐに解除するには、Red Hat Quay の再起動が必要です。

データベース接続プールは、環境変数 DB_CONNECTION_POOLING={true|false} を設定して切り替えることができます。

表3.2 データベース接続プールの設定

変数	タイプ	説明
DB_CONNECTION_POOLING	ブール値	データベース接続プールの有効化または無効化

データベース接続プーリングが有効な場合は、接続プールの最大サイズを変更することができます。これは、以下の config.yaml オプションを使用して実行できます。

config.yaml

```
...
DB_CONNECTION_ARGS:
  max_connections: 10
...
```

3.3. HTTP 接続回数

環境変数を使用して、HTTP の同時接続数を指定することができます。これらは、全体として、または特定のコンポーネントに対して指定することができます。それぞれのデフォルトは、1 プロセスあたり 50 並列接続です。

表3.3 HTTP 接続数の設定

変数	タイプ	説明
WORKER_CONNECTION_COUNT	数値	同時 HTTP 接続 **デフォルト:** 50
WORKER_CONNECTION_COUNT_REGISTRY	数値	レジストリーの同時 HTTP 接続 **デフォルト:** WORKER_CONNECTION_COUNT
WORKER_CONNECTION_COUNT_WEB	数値	Web UI の同時 HTTP 接続 **デフォルト:** WORKER_CONNECTION_COUNT
WORKER_CONNECTION_COUNT_SECSCAN	数値	Clair の同時 HTTP 接続 **デフォルト:** WORKER_CONNECTION_COUNT

3.4. ワーカーカウント変数

表3.4 ワーカーカウント変数

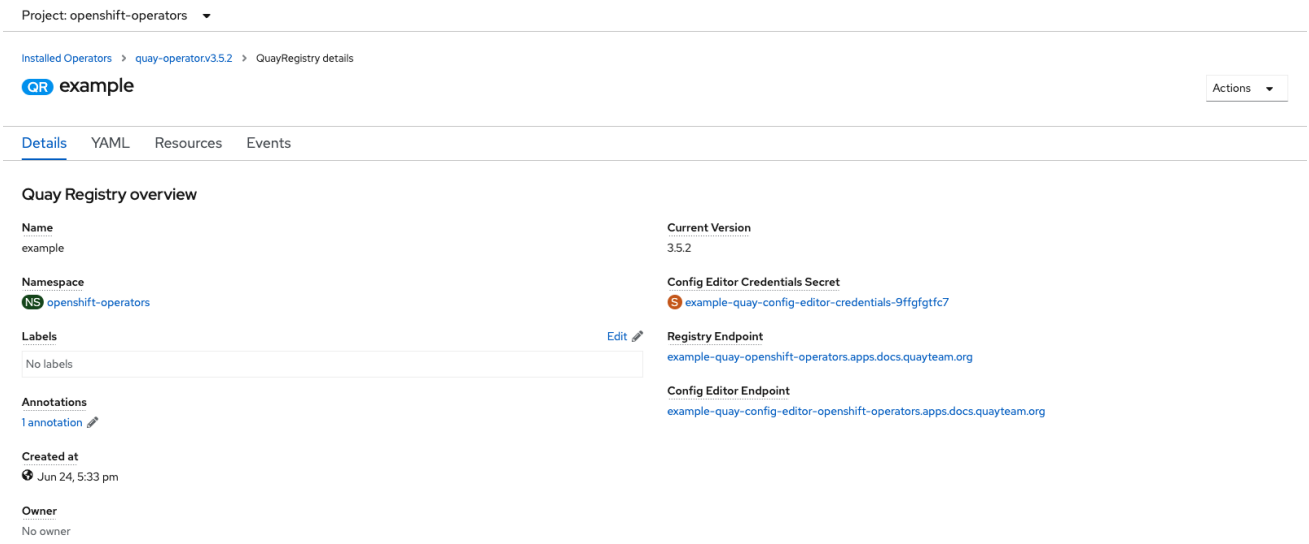
変数	タイプ	説明
WORKER_COUNT	数値	プロセス数の汎用上書き
WORKER_COUNT_REGISTRY	数値	**Quay** コンテナー内のレジストリー要求を処理するプロセス数を指定します **値:** 8 から 64 までの整数。
WORKER_COUNT_WEB	数値	コンテナー内の UI/Web リクエストを処理するプロセス数を指定します **値:** 2 から 32 までの整数。

変数	タイプ	説明
WORKER_COUNT_SECSCAN	数値	コンテナー内のセキュリティースキャン (Clair など) の統合を処理するプロセス数を指定します。 **値:** 2 から 4 までの整数。

第4章 設定ツールを使用した OPENSHIFT における QUAY の再設定

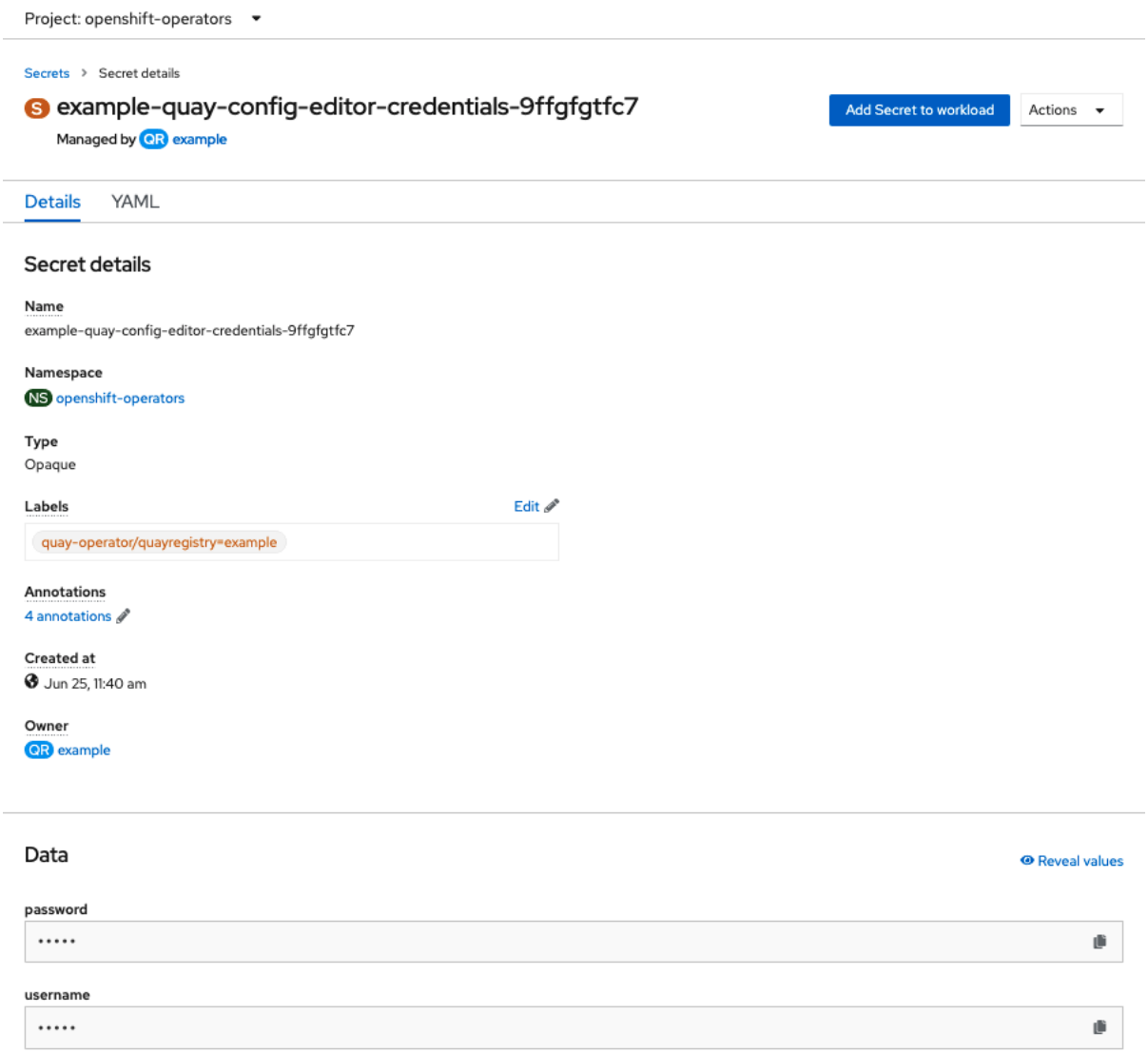
4.1. 設定エディターへのアクセス

QuayRegistry 画面の Details セクションには、設定エディターのエンドポイントと、設定エディターへのログインに使用する認証情報などのシークレットのリンクがあります。

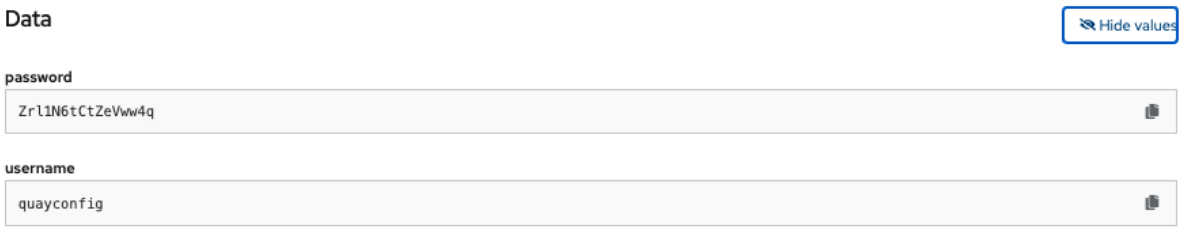


4.1.1. 設定エディターの認証情報の取得

1. 設定エディターシークレットのリンクをクリックします。

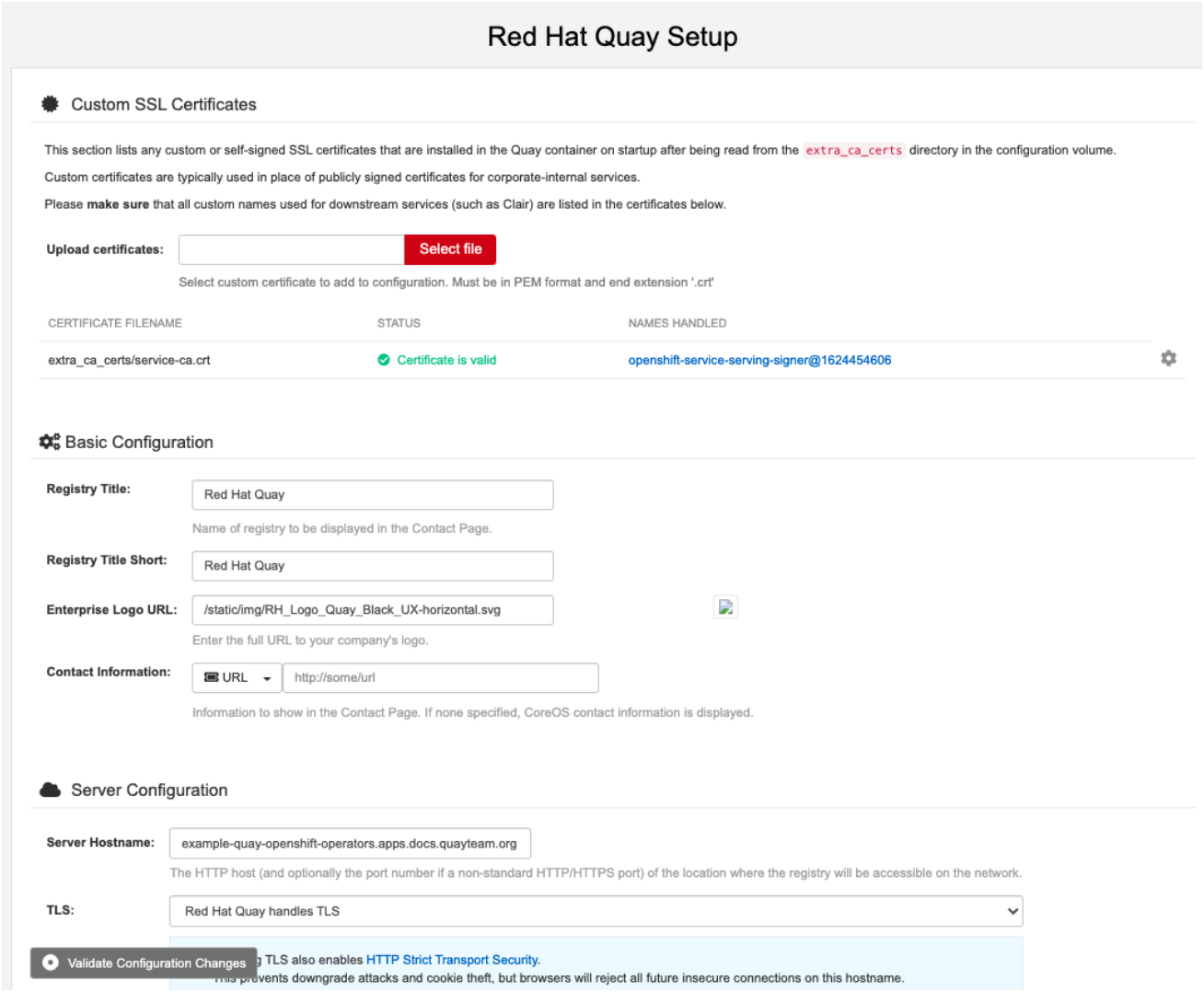


2. Secret details 画面の Data セクションで、**Reveal values** をクリックし、設定エディターへのログインに使用する認証情報を表示します。



4.1.2. 設定エディターへのログイン

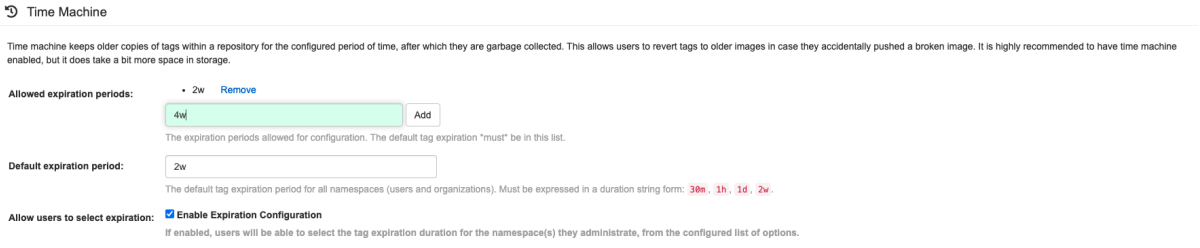
設定エディターエンドポイントを参照し、設定ツールにアクセスする時に使用するユーザー名 (通常は **quayconfig**)、および対応するパスワードを入力します。



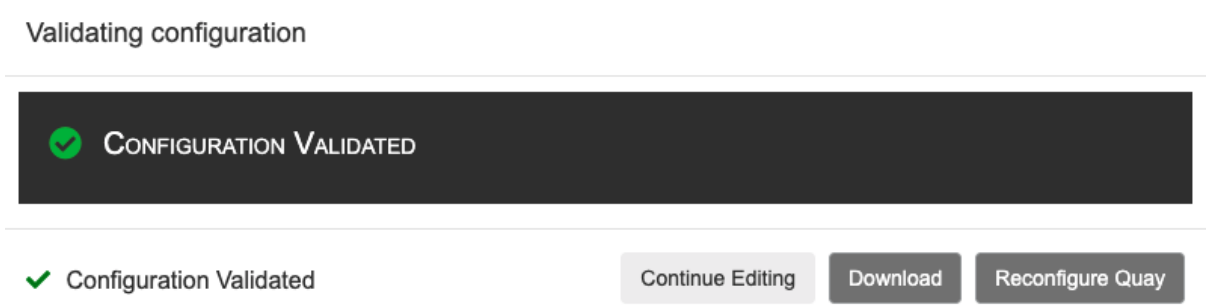
4.1.3. 設定の変更

設定の更新例では、設定エディターツールを使用してスーパーユーザーを追加しています。

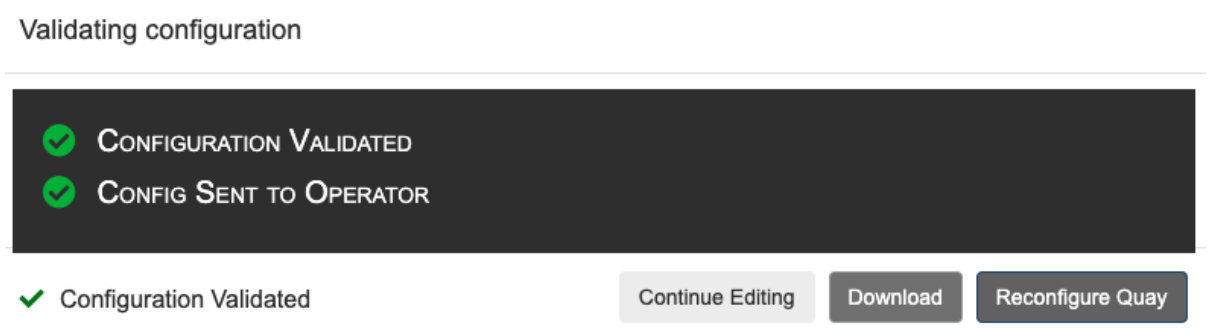
1. 時間マシン機能に関する有効期限 (**4w** など) を追加します。



2. **Validate Configuration Changes** を選択して、変更が有効であることを確認します。
3. **Reconfigure Quay** ボタンを押して変更を適用します。



4. 設定ツールは、変更が Quay に送信されていることを通知します。



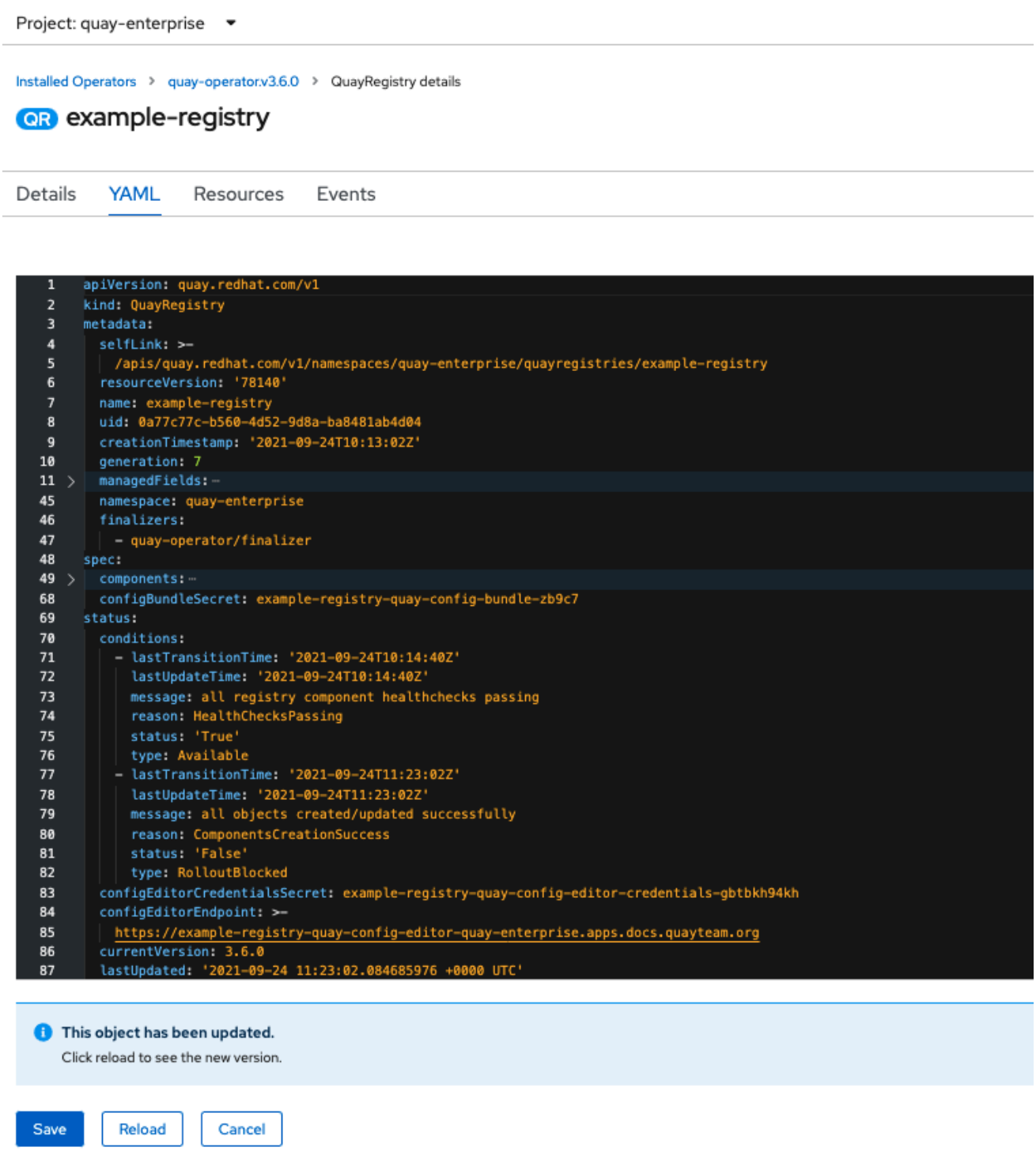
注記

設定ツール UI を使用して Red Hat Quay を再設定すると、更新された設定が適用されている間にレジストリーが短期間利用できなくなる可能性があります。

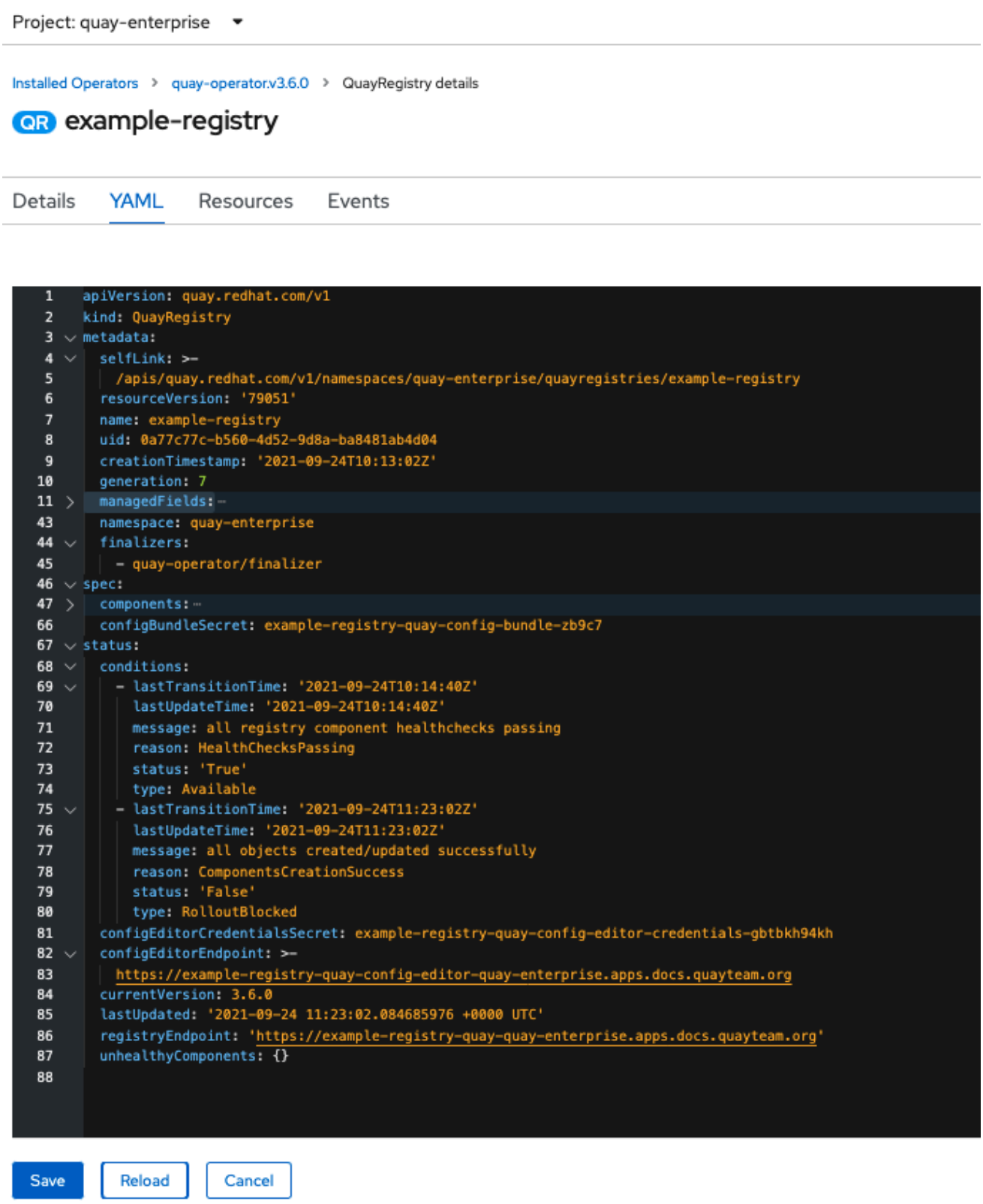
4.2. UI での再設定の監視

4.2.1. QuayRegistry リソース

Operator の再設定後に、QuayRegistry の特定インスタンスの YAML タブで再デプロイの進捗を追跡できます (この場合は **example-registry**)。

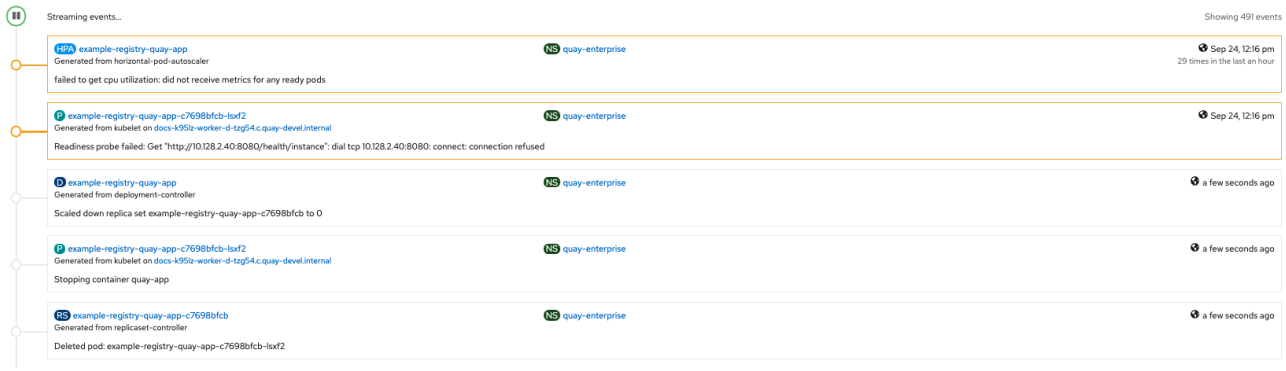


ステータスが変わるたびに、データをリロードして更新されたバージョンを表示するように求められます。最終的に、Operator は変更を調整し、正常でないコンポーネントが報告されることはありません。

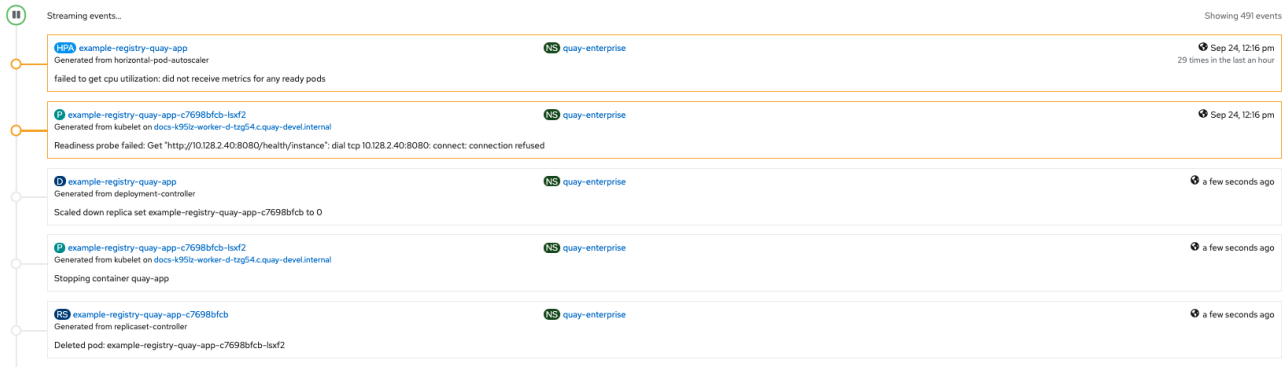


4.2.2. イベント

QuayRegistry の Events タブには、再デプロイに関連するイベントが表示されます。



再設定の影響を受ける namespace のすべてのリソースのストリーミングイベントは、Home → Events の OpenShift コンソールで利用できます。



4.3. 再設定後に更新された情報へのアクセス

4.3.1. UI で更新された設定ツールの認証情報へのアクセス

Red Hat Quay 3.7 では、UI を介して Quay を再設定しても、新しいログインパスワードが生成されなくなりました。パスワードは 1 回だけ生成され、**QuayRegistry** オブジェクトを調整した後も同じままです。

4.3.2. UI で更新された config.yaml へのアクセス

設定バンドルを使用して、更新された **config.yaml** ファイルにアクセスします。

1. QuayRegistry の詳細画面で、Config Bundle Secret をクリックします。
2. Secret の詳細画面の Data セクションで、Reveal values をクリックし、**config.yaml** ファイルを表示します。
3. 変更が適用されていることを確認します。この場合、**4w** は **TAG_EXPIRATION_OPTIONS** の一覧に存在するはずです。

```
...
SERVER_HOSTNAME: example-quay-openshift-operators.apps.docs.quayteam.org
SETUP_COMPLETE: true
SUPER_USERS:
- quayadmin
TAG_EXPIRATION_OPTIONS:
- 2w
- 4w
...
```

第5章 QUAY OPERATOR コンポーネント

Quay は強力なコンテナーレジストリープラットフォームであるため、多くの依存関係が存在します。これらには、データベース、オブジェクトストレージ、Redis などが含まれます。Quay Operator は、Kubernetes 上で Quay とその依存関係に指向したデプロイメントを管理します。これらの依存関係は **コンポーネント** として処理され、**QuayRegistry** API で設定されます。

QuayRegistry カスタムリソースでは、**spec.components** フィールドでコンポーネントを設定します。各コンポーネントには、**kind** (コンポーネントの名前) と **managed** (コンポーネントのライフサイクルを Operator が処理するかどうかを示すブール値) の 2 つのフィールドがあります。(このフィールドを省略する) デフォルトでは、すべてのコンポーネントが管理され、調整時に表示できるように自動的に入力されます。

```
spec:
  components:
    - kind: quay
      managed: true
    - kind: postgres
      managed: true
    - kind: clair
      managed: true
    - kind: redis
      managed: true
    - kind: horizontalpodautoscaler
      managed: true
    - kind: objectstorage
      managed: true
    - kind: route
      managed: true
    - kind: mirror
      managed: true
    - kind: monitoring
      managed: true
    - kind: tls
      managed: true
    - kind: clairpostgres
      managed: true
```

5.1. 管理コンポーネントの使用

QuayRegistry カスタムリソースを指定しないと、Operator は以下の管理コンポーネントについてデフォルトを使用します。

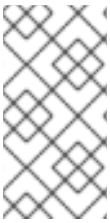
- **quay:** 環境変数やレプリカの数など、Quay デプロイメントのオーバーライドを保持します。このコンポーネントは Red Hat Quay 3.7 の新機能であり、アンマネージドに設定することはできません。
- **postgres:** レジストリーメタデータを保存するには、Software Collections から Postgres 10 のバージョンを使用します。
- **clair:** イメージの脆弱性スキャンを提供します。
- **redis:** Quay ビルダーの調整および一部の内部ロギングを処理します。
- **horizontalpodautoscaler:** メモリー/CPU の消費に応じて Quay Pod 数を調整します。

- **ObjectStorage:** イメージレイヤー Blob を格納するには、Noobaa/RHOCS によって提供される **ObjectBucketClaim** Kubernetes API を使用します。
- **route:** OpenShift の外部から Quay レジストリーへの外部エントリーポイントを提供します。
- **mirror:** リポジトリーミラーワーカーを設定します (オプションのリポジトリーミラーリングをサポートするため)。
- **monitoring:** Grafana ダッシュボード、個別のメトリクスへのアクセス、Quay Pod が頻繁に再起動されていることを通知するアラートなどが含まれます。
- **tls:** Red Hat Quay または OpenShift が TLS を処理するかどうかを設定します。
- **clairpostgres:** 管理された Clair データベースを設定します

Operator は Red Hat Quay が管理コンポーネントを使用するために必要な設定およびインストール作業を処理します。Quay Operator によって実行される事前に設定されたデプロイメントがお使いの環境に適さない場合、以下のセクションで説明されているように Operator に **unmanaged** のリソース (オーバーライド) を指定できます。

5.2. 依存関係向けの管理対象外コンポーネントの使用

Quay で使用する Postgres、Redis、またはオブジェクトストレージなどの既存のコンポーネントがある場合は、まず Quay 設定バンドル (**config.yaml**) 内でそれらを設定し、**QuayRegistry** でバンドルを参照します (Kubernetes **Secret**)。これは、非管理対象のコンポーネントを示します。



注記

Quay 設定エディターを使用して、既存の設定バンドルを作成または変更したり、Kubernetes **Secret** の更新プロセスを単純化したりできます。Quay の設定が設定エディターで変更され、Operator に送信されると、Quay デプロイメントは新規の設定を反映するように更新されます。

5.2.1. 既存の Postgres データベースの使用

1. 必要なデータベースフィールドを使用して設定ファイル **config.yaml** を作成します。

   config.yaml:

   ```
   DB_URI: postgresql://test-quay-database:postgres@test-quay-database:5432/test-quay-database
   ```

2. 設定ファイルを使用してシークレットを作成します。

   ```
   $ kubectl create secret generic --from-file config.yaml=./config.yaml config-bundle-secret
   ```

3. **postgres** コンポーネントを管理対象外としてマークし、作成された Secret を参照する QuayRegistry YAML ファイル **quayregistry.yaml** を作成します。

   quayregistry.yaml

   ```
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
   ```

```
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: postgres
      managed: false
```

4. 以下のセクションで説明されているようにレジストリーをデプロイします。

5.2.2. NooBaa アンマネージドストレージ

1. Storage → Object Bucket Claims のコンソールで NooBaa Object Bucket Claim を作成します。
2. アクセスキー、バケット名、エンドポイント (ホスト名)、およびシークレットキーを含む Object Bucket Claim データの詳細を取得します。
3. Object Bucket Claim (オブジェクトバケット要求) の情報を使用して **config.yaml** 設定ファイルを作成します。

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - RHOCSStorage
    - access_key: WmrXtSGk8B3nABCDEFGH
      bucket_name: my-noobaa-bucket-claim-8b844191-dc6c-444e-9ea4-87ece0abcdef
      hostname: s3.openshift-storage.svc.cluster.local
      is_secure: true
      port: "443"
      secret_key: X9P5SDGJtmSuHFCMSLMbdNCMfUABCDEFGH+C5QD
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - default
```

5.2.3. Horizontal Pod Autoscaler の無効化

HorizontalPodAutoscalers が Clair、Quay、Mirror Pod に追加され、負荷の急上昇時に自動的にスケーリングされるようになりました。

HPA はデフォルトで **managed** に設定され、Quay の Pod 数、Clair およびリポジトリーのミラーリングは 2 に設定されます。これにより、Operator 経由で Quay を更新/再設定する際や、イベントの再スケジュール時にダウンタイムを回避しやすくなります。

自動スケーリングを無効にするか、独自の **HorizontalPodAutoscaler** を作成する場合は、コンポーネントを単純に **QuayRegistry** インスタンスでアンマネージドとして指定します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  components:
    - kind: horizontalpodautoscaler
      managed: false
```

5.3. KUBERNETES へのデプロイ時に証明書を追加

Kubernetes にデプロイすると、Red Hat Quay は config アセットを保存するボリュームとしてシークレットにマウントします。残念ながら、これは現在、スーパーユーザーパネルの証明書のアップロード機能に干渉します。

このエラーを回避するには、Red Hat Quay が展開された **後に** base64 エンコードされた証明書をシークレットに追加します。以下に、実行する方法を説明します。

1. まず、証明書の内容を Base64 エンコードします。

```
$ cat ca.crt
-----BEGIN CERTIFICATE-----
MIIDljCCAn6gAwIBAgIBATANBgkqhkiG9w0BAQsFADA5MRcwFQYDVQQKDA5MQUlu
TElCQ09SRS5TTzEeMBwGA1UEAwwVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTE2
MDExMjA2NTkxMFoXDTM2MDExMjA2NTkxMFowOTEXMBUGA1UECgwOTEFCLkxJQkNP
UkUuU08xHjAcBgNVBAMMFUNlcnRpZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZI
[...]
-----END CERTIFICATE-----

$ cat ca.crt | base64 -w 0
[...]
c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

2. **kubectl** ツールを使用して、quay-enterprise-config-secret を編集します。

```
$ kubectl --namespace quay-enterprise edit secret/quay-enterprise-config-secret
```

3. 証明書のエントリーを追加し、エントリーの下に base64 エンコードされた文字列を完全に貼り付けます。

```
  custom-cert.crt:
c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

4. 最後に、すべての Red Hat Quay Pod をリサイクルします。**kubectl delete** を使用して、すべての Red Hat Quay Pod を削除します。Red Hat Quay Deployment は、新しい証明書データを使用して交換用 Pod を自動的にスケジュールします。

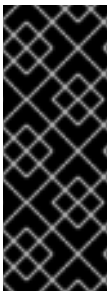
5.4. OPERATOR を使用した OCI および HELM の設定

Quay の設定のカスタマイズは、設定バンドルを含むシークレットで提供できます。以下のコマンドを実行して、**quay-config-bundle** という新規シークレットを適切な namespace に作成します。これには、OCI サポートを有効にするために必要なプロパティーが含まれます。

quay-config-bundle.yaml

```
apiVersion: v1
stringData:
  config.yaml: |
    FEATURE_GENERAL_OCI_SUPPORT: true
```

```
    FEATURE_HELM_OCI_SUPPORT: true
kind: Secret
metadata:
  name: quay-config-bundle
  namespace: quay-enterprise
type: Opaque
```



重要

Red Hat Quay 3.7 の時点で、**FEATURE_HELM_OCI_SUPPORT** は非推奨になり、Red Hat Quay の将来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。

シークレットを適切な namespace に作成します (この例では **quay-enterprise** です)。

```
$ oc create -n quay-enterprise -f quay-config-bundle.yaml
```

spec.configBundleSecret フィールドのシークレットを指定します。

quay-registry.yaml

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: quay-config-bundle
```

指定された設定でレジストリーを作成します。

```
$ oc create -n quay-enterprise -f quay-registry.yaml
```

5.5. ボリュームサイズのオーバーライド

Red Hat Quay v3.6.2 以降、管理対象コンポーネントにプロビジョニングされるストレージリソースの必要なサイズを指定できます。Clair および Quay PostgreSQL データベースのデフォルトサイズは **50Gi** です。パフォーマンス上の理由がある場合や、ストレージバックエンドにサイズ変更機能がない場合など、十分な容量を事前に選択できるようになりました。

以下の例では、Clair および Quay PostgreSQL データベースのボリュームサイズは **70Gi** に設定されています。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay-example
  namespace: quay-enterprise
spec:
  configBundleSecret: config-bundle-secret
```

```
components:
  - kind: objectstorage
    managed: false
  - kind: route
    managed: true
  - kind: tls
    managed: false
  - kind: clair
    managed: true
    overrides:
      volumeSize: 70Gi
  - kind: postgres
    managed: true
    overrides:
      volumeSize: 70Gi
```

第6章 コンフィグレーション API の利用

コンフィグレーションツールは、設定の構築、検証、バンドル、およびデプロイに使用できる 4 つのエンドポイントを示します。config-tool の API は、https://github.com/quay/config-tool/blob/master/pkg/lib/editor/API.md に記載されています。ここでは、API を使用して現在の設定を取得する方法と、変更した内容を検証する方法について説明します。

6.1. 初期設定値の取得

初めてコンフィグレーションツールを実行するときに、既存のコンフィグレーションがない場合は、デフォルトのコンフィグレーションを取得することができます。コンテナーをコンフィグモードで起動します。

```
$ sudo podman run --rm -it --name quay_config \
  -p 8080:8080 \
  registry.redhat.io/quay/quay-rhel8:v3.7.10 config secret
```

デフォルトを取得するには、コンフィグレーション API の **config** エンドポイントを使用します。

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config  | jq
```

返される値は、JSON 形式によるデフォルト設定です。

```
{
  "config.yaml": {
    "AUTHENTICATION_TYPE": "Database",
    "AVATAR_KIND": "local",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DEFAULT_TAG_EXPIRATION": "2w",
    "EXTERNAL_TLS_TERMINATION": false,
    "FEATURE_ACTION_LOG_ROTATION": false,
    "FEATURE_ANONYMOUS_ACCESS": true,
    "FEATURE_APP_SPECIFIC_TOKENS": true,
    ....
  }

}
```

6.2. 現在の設定の取得

すでに Quay レジストリーを設定してデプロイしている場合は、コンテナーを停止してコンフィグレーションモードで再起動し、既存のコンフィグレーションをボリュームとして読み込みます。

```
$ sudo podman run --rm -it --name quay_config \
  -p 8080:8080 \
  -v $QUAY/config:/conf/stack:Z \
  registry.redhat.io/quay/quay-rhel8:v3.7.10 config secret
```

現在の設定を取得するには、API の **config** エンドポイントを使用します。

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config  | jq
```

返される値は、データベースと Redis の設定データを含む、JSON 形式の現在の設定です。

```
{
  "config.yaml": {
    ....
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ....


  }

}
```

6.3. API による設定の検証

設定を検証するには、**config/validate** エンドポイントに設定を投稿します。

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
    ....
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ....

  }

} http://quay-server:8080/api/v1/config/validate | jq
```

返される値は、設定で見つかったエラーを含む配列です。設定が有効であれば、空の配列 **[]** が返されます。

6.4. 必須項目の決定

空の設定構造を **config/validate** エンドポイントに投稿することで、必須フィールドを決定することができます。

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
  }

} http://quay-server:8080/api/v1/config/validate | jq
```

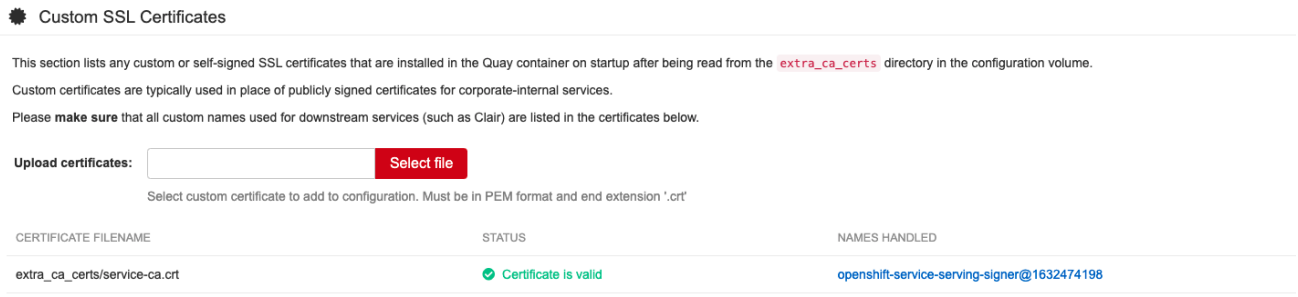
返される値は、どのフィールドが必須であるかを示す配列です。

```
[
  {
    "FieldGroup": "Database",
    "Tags": [
      "DB_URI"
    ],
    "Message": "DB_URI is required."
  },
  {
    "FieldGroup": "DistributedStorage",
    "Tags": [
      "DISTRIBUTED_STORAGE_CONFIG"
    ],
    "Message": "DISTRIBUTED_STORAGE_CONFIG must contain at least one storage location."
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME is required"
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME must be of type Hostname"
  },
  {
    "FieldGroup": "Redis",
    "Tags": [
      "BUILDLOGS_REDIS"
    ],
    "Message": "BUILDLOGS_REDIS is required"
  }
]
```

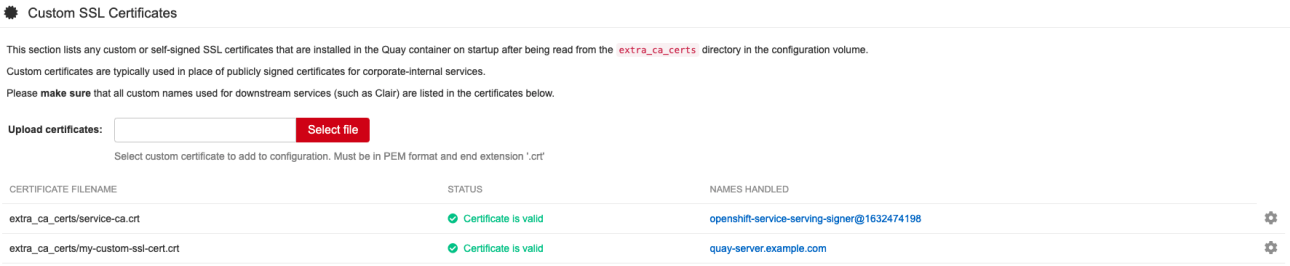
第7章 コンフィグレーションツールの使用

7.1. カスタム SSL 証明書 UI

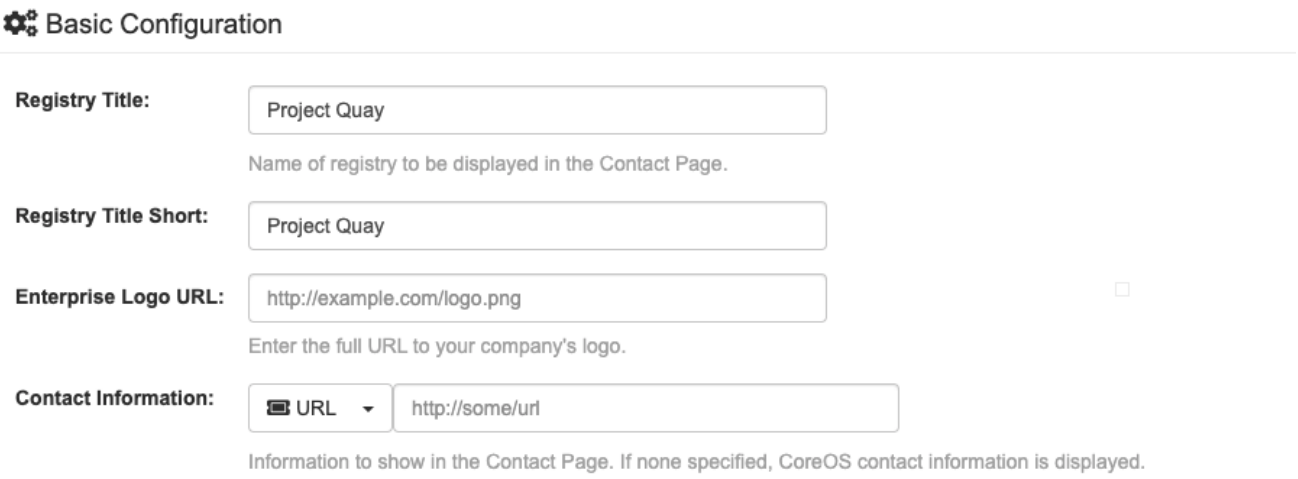
コンフィグツールを使用してカスタム証明書を読み込み、外部データベースなどのリソースへのアクセスを容易にします。アップロードするカスタム証明書を選択し、拡張子 **.crt** を使用して PEM 形式のものであることを確認します。



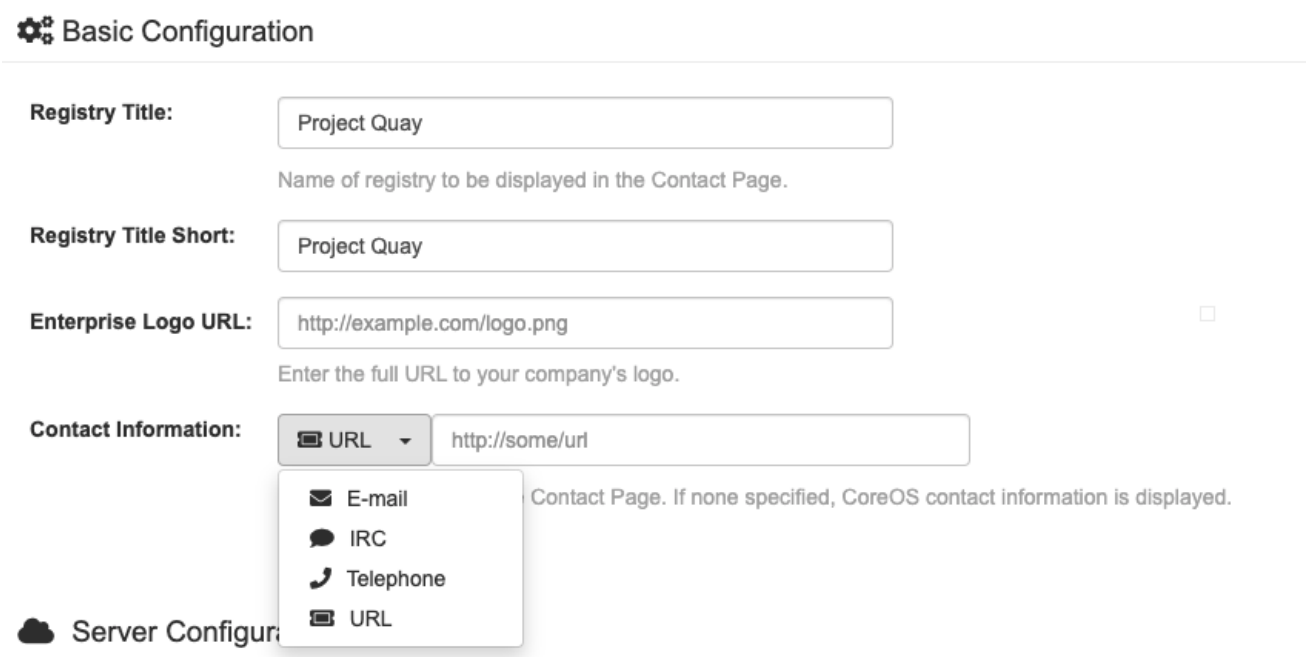
設定ツールには、アップロードされた証明書の一覧が表示されます。カスタムの SSL 証明書をアップロードすると、その証明書が一覧に表示されます。



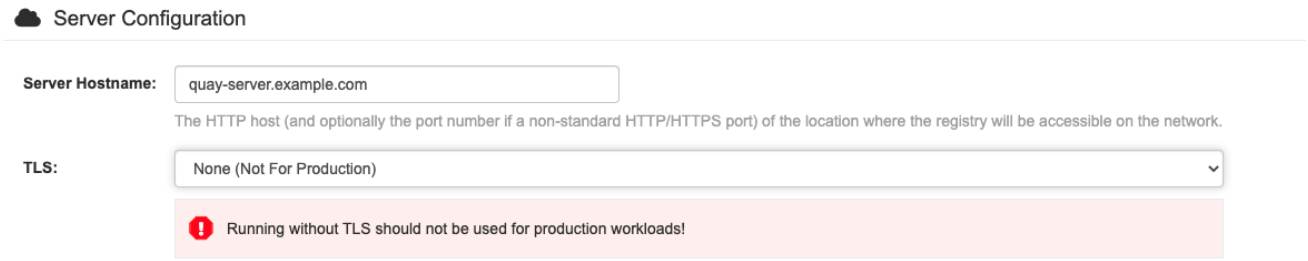
7.2. 基本設定



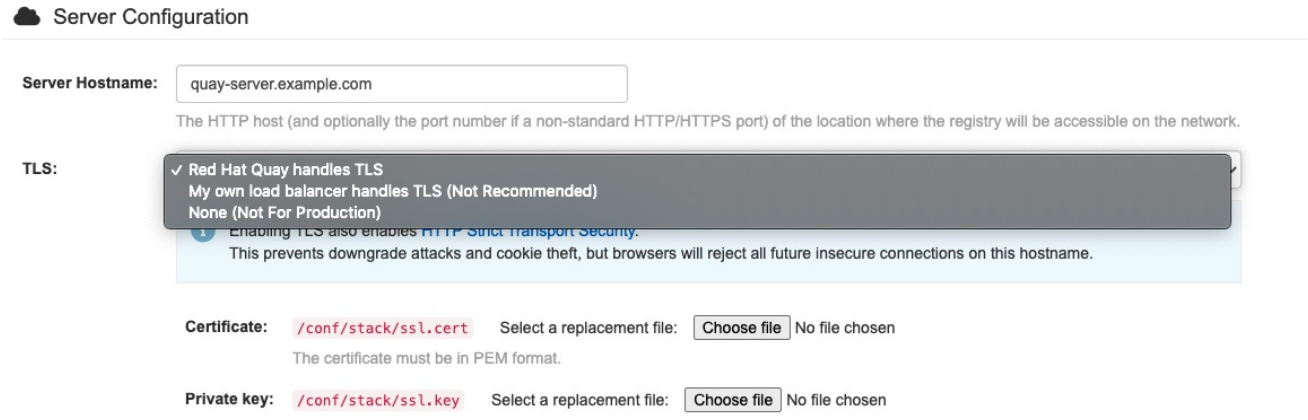
7.2.1. お問い合わせ先



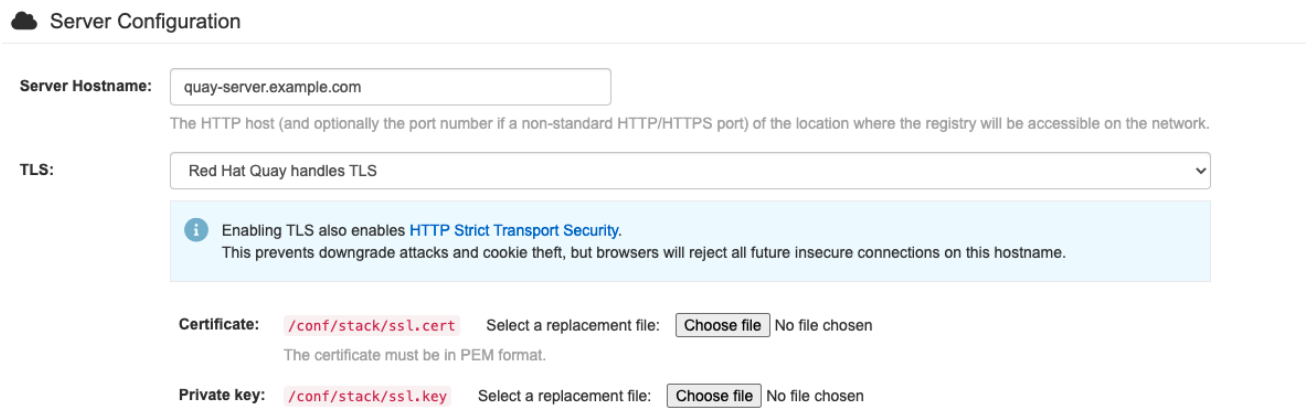
7.3. サーバー設定



7.3.1. サーバー設定の選択肢

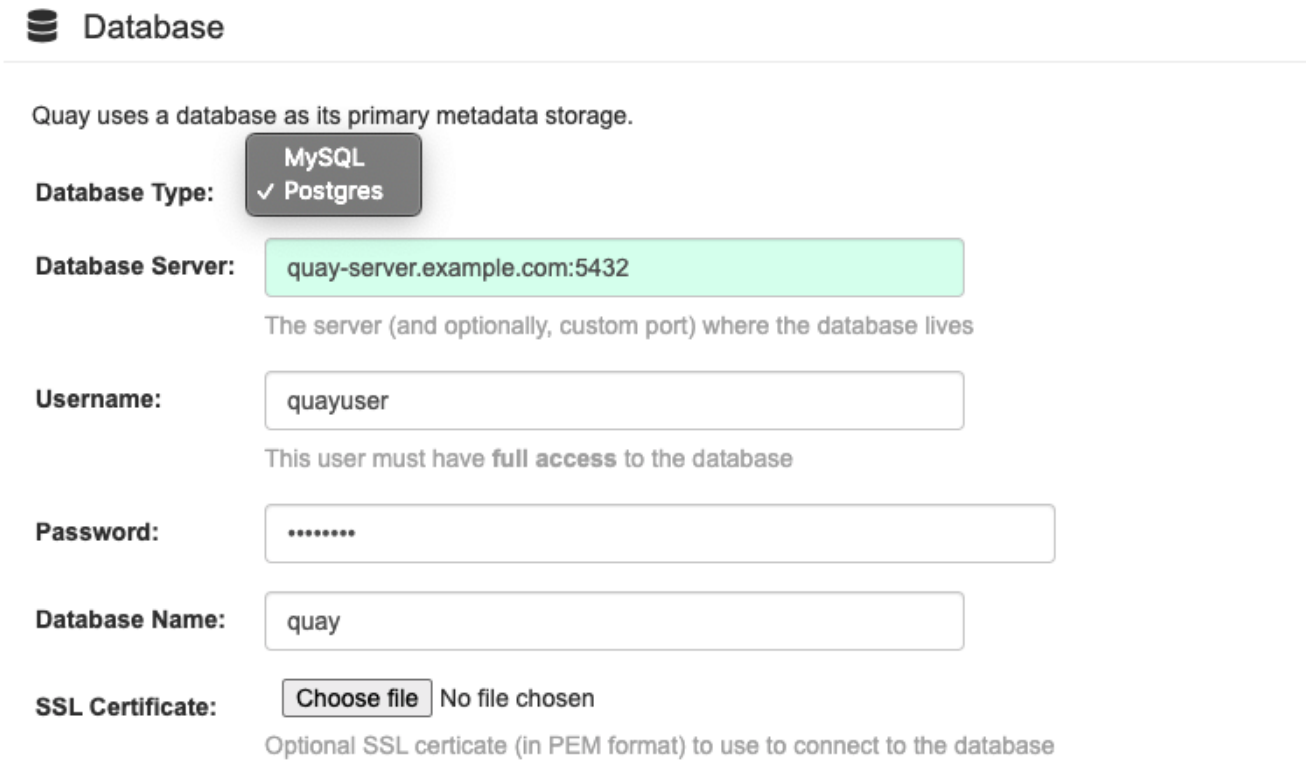


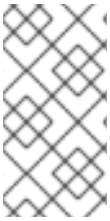
7.3.2. TLS 設定



7.4. データベースの設定

PostGreSQL と MySQL のいずれかを選択できます。



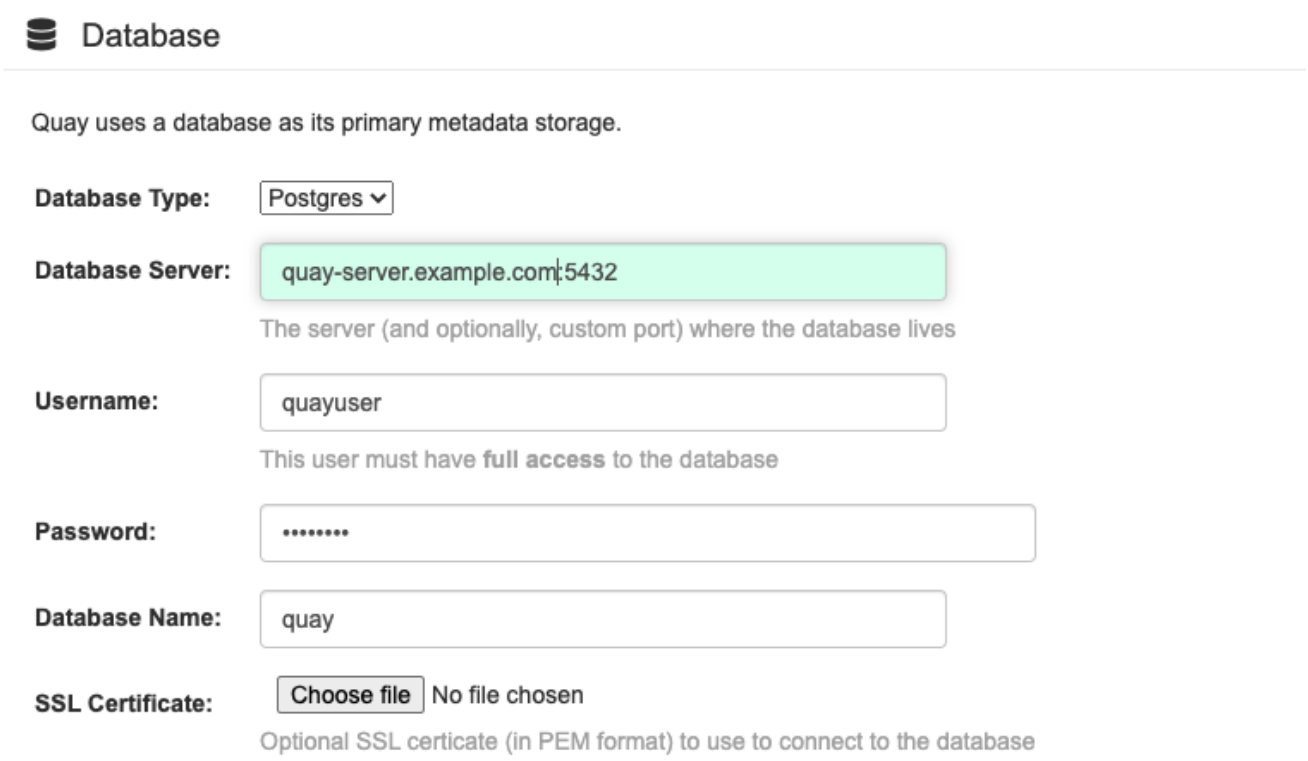


注記

MySQL および MariaDB データベースは、Red Hat Quay 3.6 で非推奨となりました。これらのデータベースのサポートは、Red Hat Quay の将来のバージョンで削除される予定です。Red Hat Quay の新規インストールを開始する場合は、Postgre SQL を使用することを強くお勧めします。

7.4.1. PostgreSQL の設定

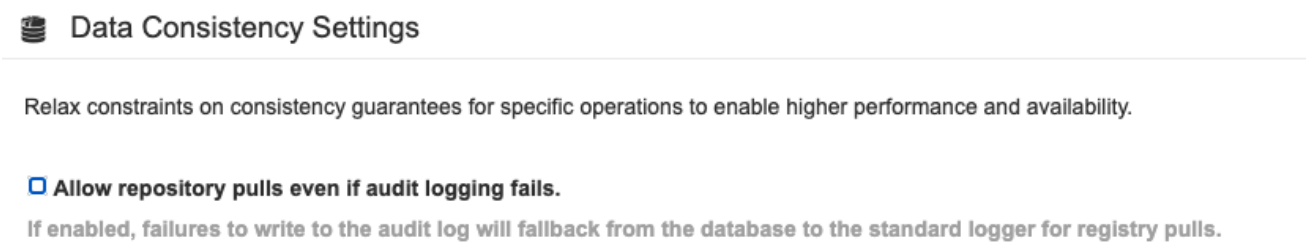
データベースへの接続情報を入力します。



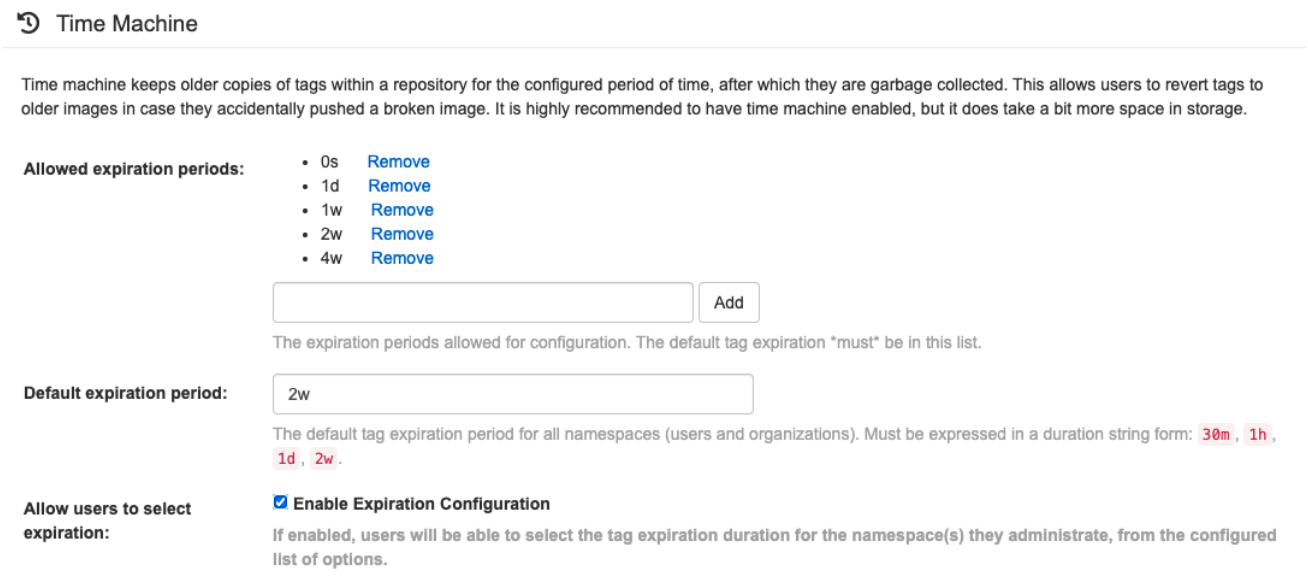
これにより、**postgresql://quayuser:quaypass@quay-server.example.com:5432/quay** 形式の DB_URI フィールドが生成されます。

接続引数を詳細に制御する必要がある場合は、設定ガイドのデータベース接続引数のセクションを参照してください。

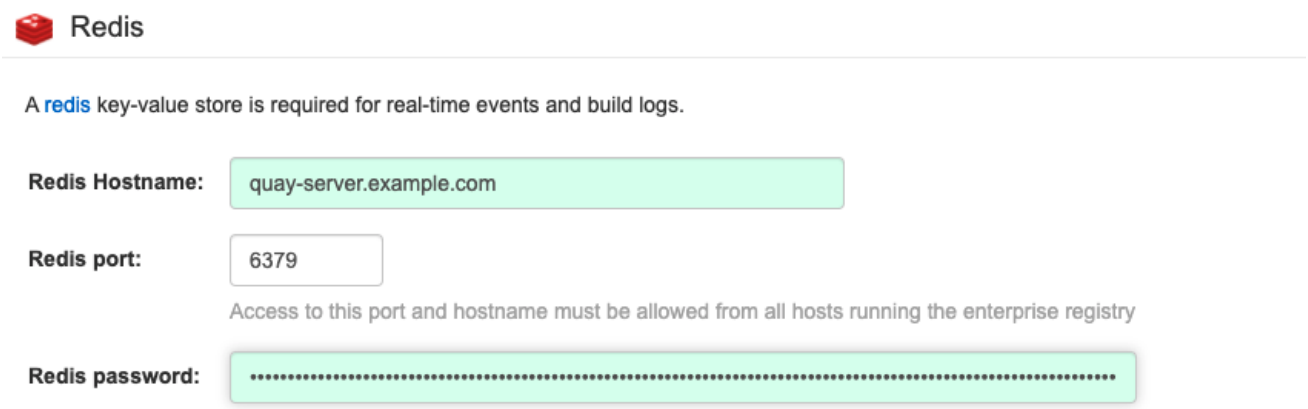
7.5. データの整合性



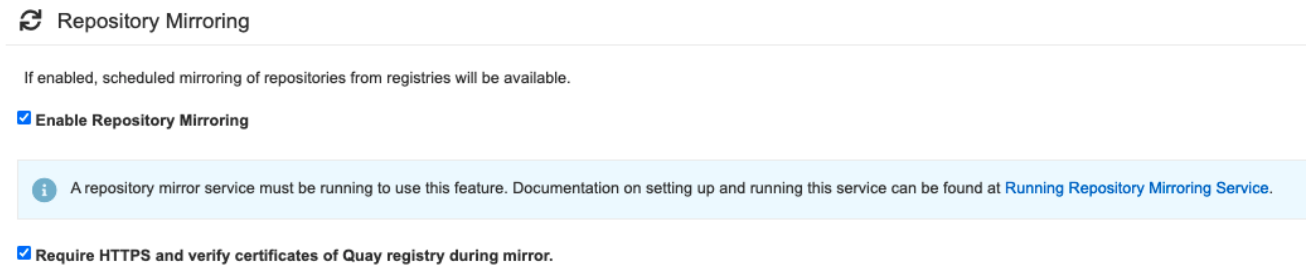
7.6. タイムマシン設定



7.7. REDIS の設定



7.8. リポジトリーのミラーリングの設定



7.9. レジストリーのストレージ設定

- プロキシーストレージ
- ストレージの Geo レプリケーション
- ストレージエンジン

7.9.1. ストレージレプリケーションを有効にする - スタンドアロン Quay

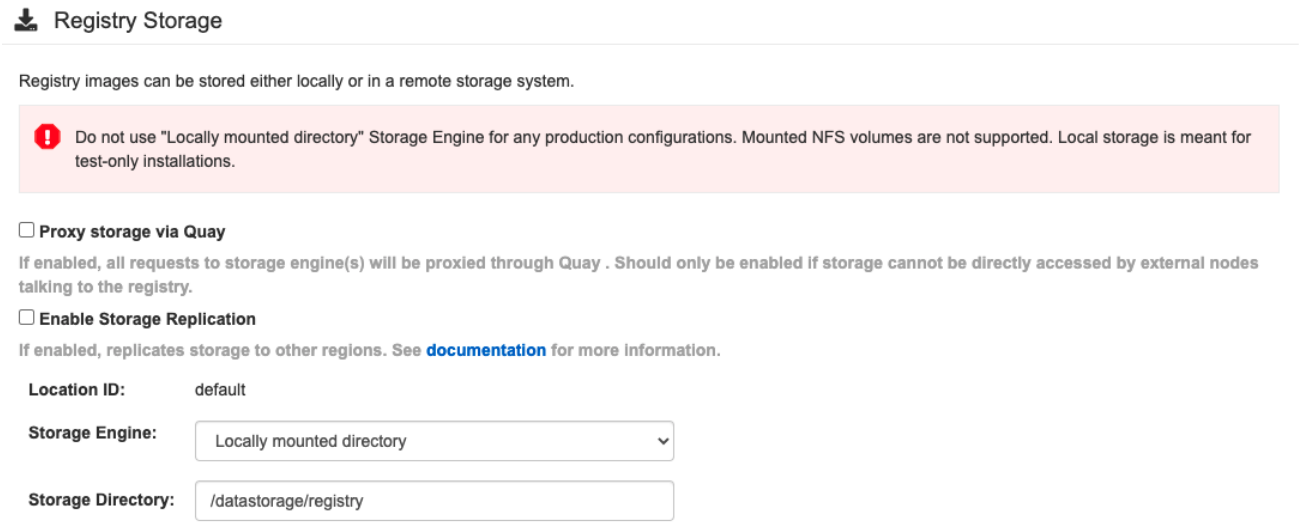
1. スクロールダウンして、**Registry Storage** というセクションに進みます。
2. **Enable Storage Replication** をクリックします。
3. データを複製するストレージエンジンをそれぞれ追加します。使用するすべてのストレージエンジンをリストに載せる必要があります。
4. すべてのイメージをすべてのストレージエンジンに完全にレプリケートする必要がある場合は、各ストレージエンジンの設定の下で **Replicate to storage engine by default** をクリックします。これにより、すべてのイメージがそのストレージエンジンにレプリケートされます。代わりに名前空間ごとのレプリケーションを有効にするには、サポートにお問い合わせください。
5. 完了したら、**Save Configuration Changes** をクリックします。設定変更は、Red Hat Quay が次回再起動したときに有効になります。
6. ストレージを追加し、Geo レプリケーションの Replicate to storage engine by default を有効にした後、既存のイメージデータをすべてのストレージで同期する必要があります。そのためには、コンテナーに **oc exec** (または docker/kubectl exec) して実行する必要があります。

```
# scl enable python27 bash
# python -m util.backfillreplication
```

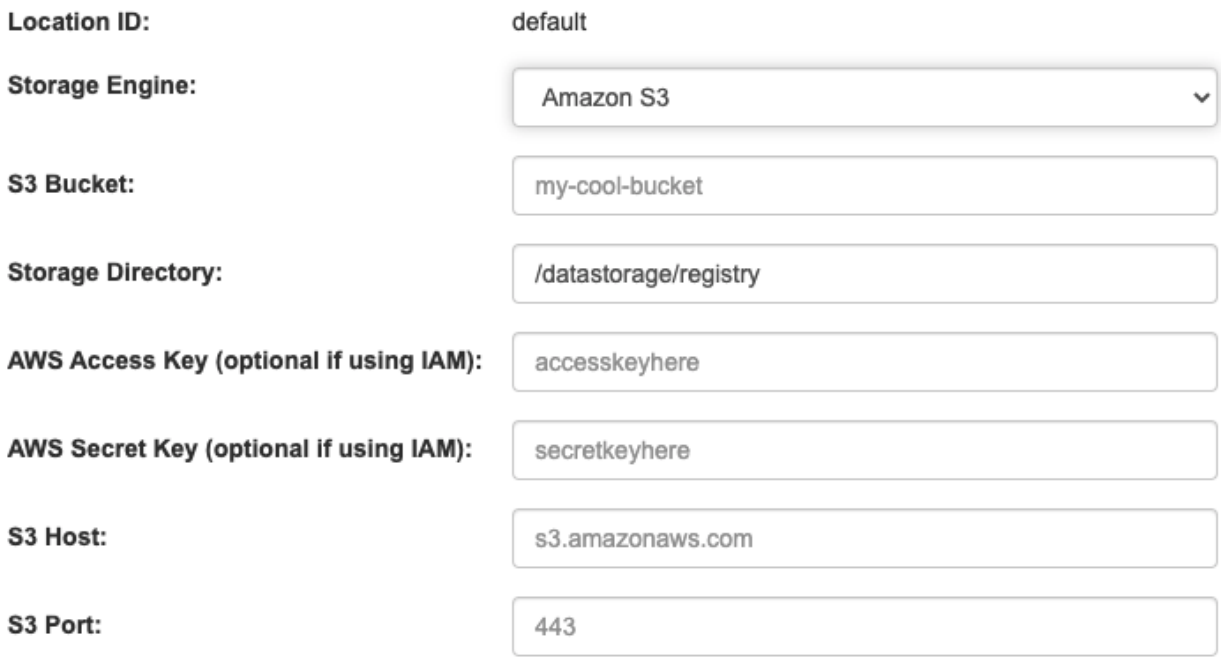
この操作は、新しいストレージを追加した後にコンテンツを同期するための 1 回限りの操作です。

7.9.2. ストレージエンジン

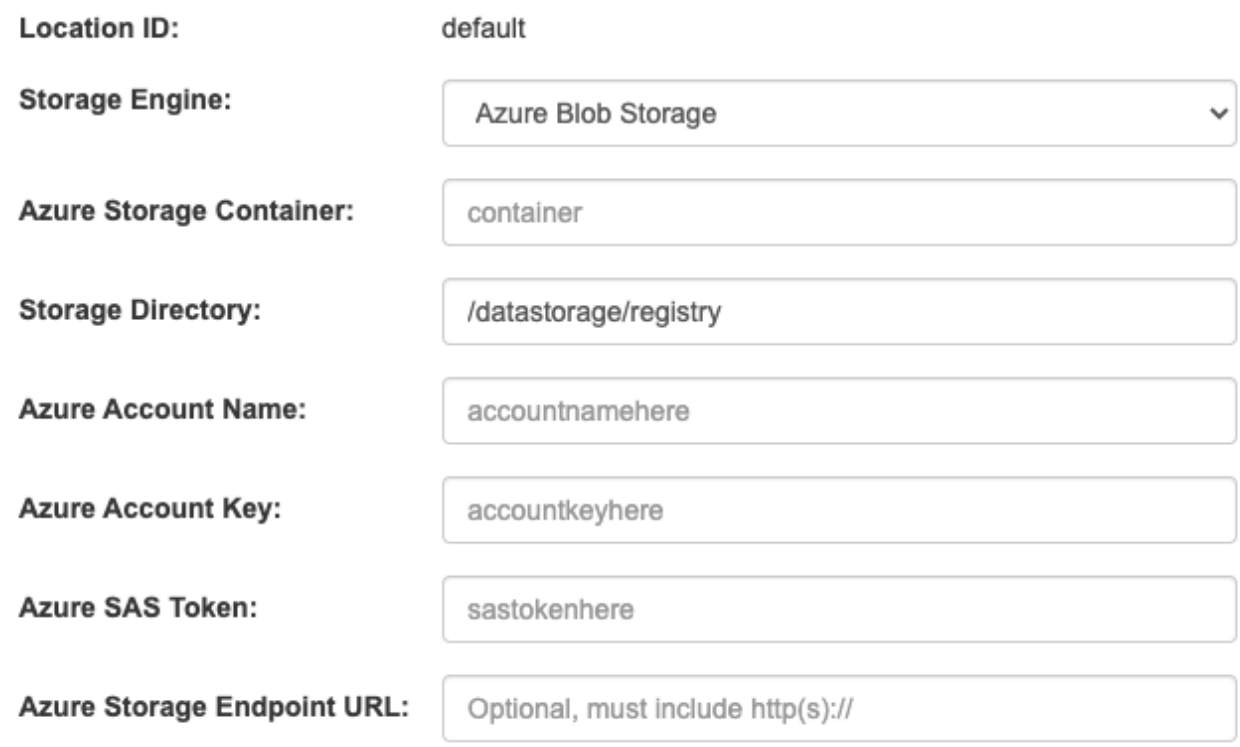
7.9.2.1. ローカルストレージ



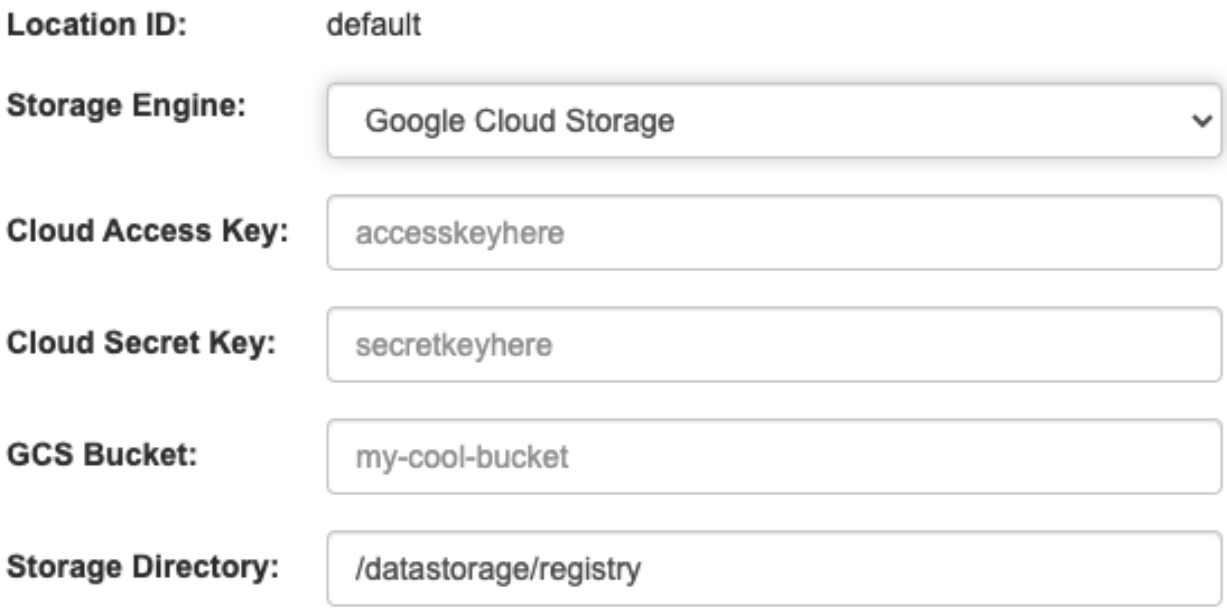
7.9.2.2. Amazon S3 ストレージ



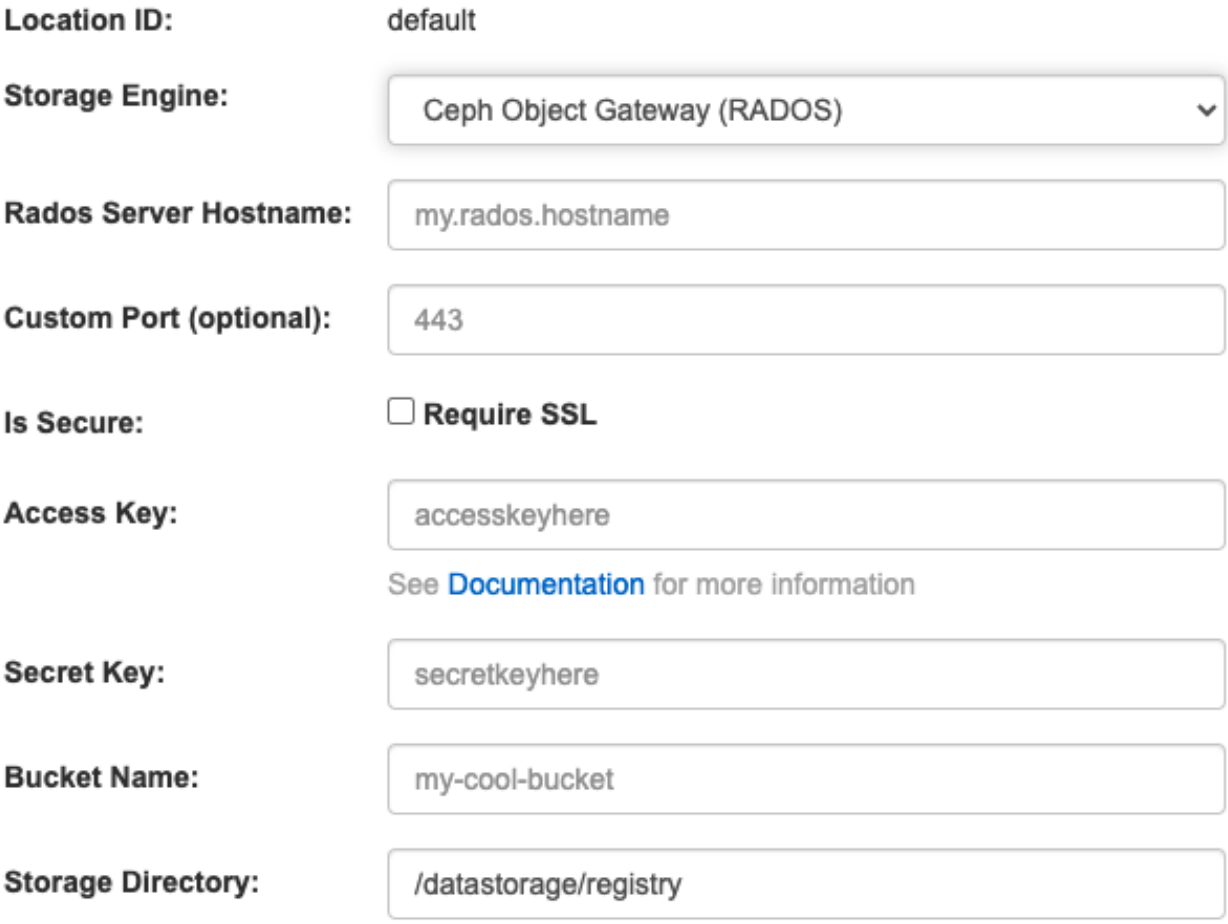
7.9.2.3. Azure Blob ストレージ



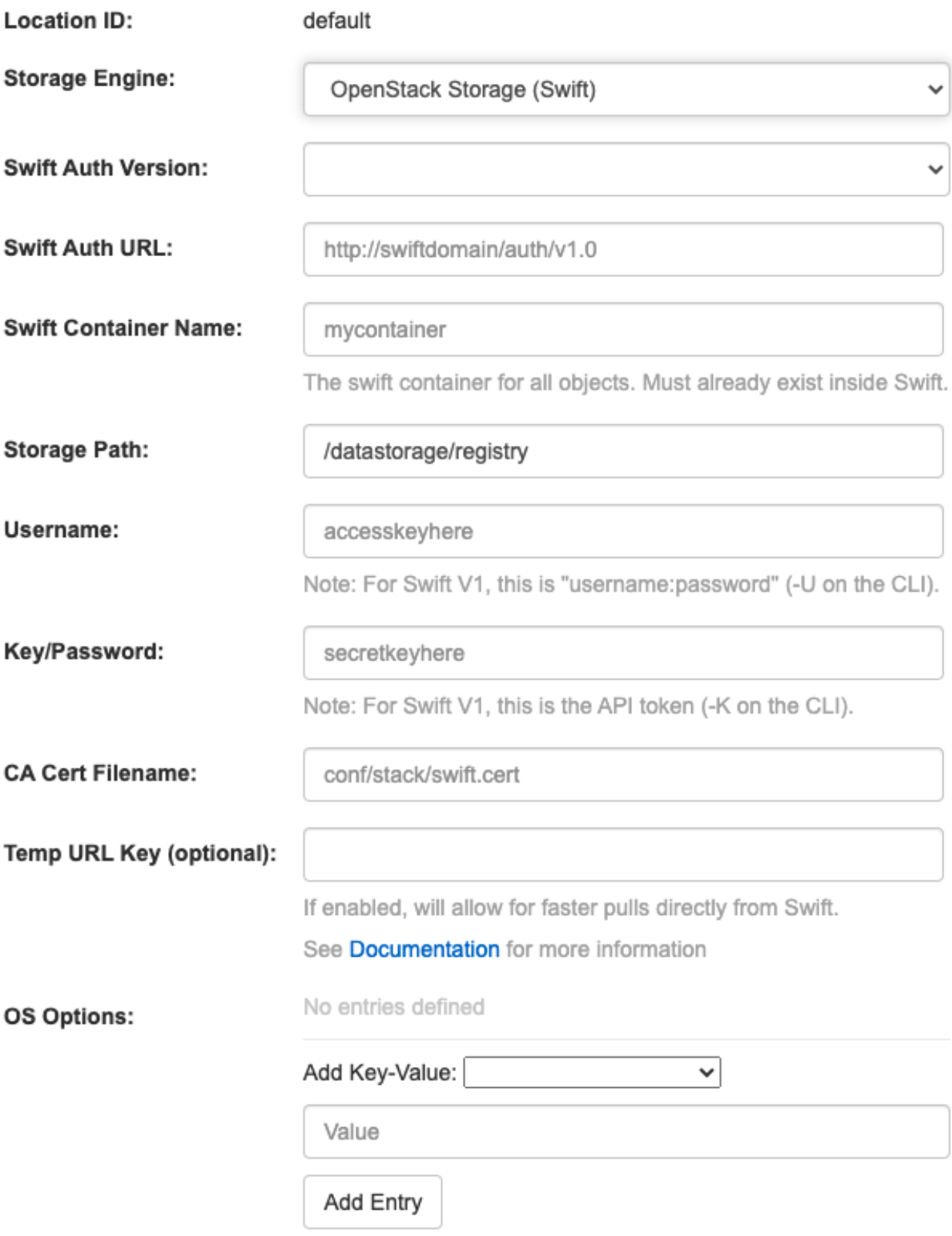
7.9.2.4. Google クラウドストレージ



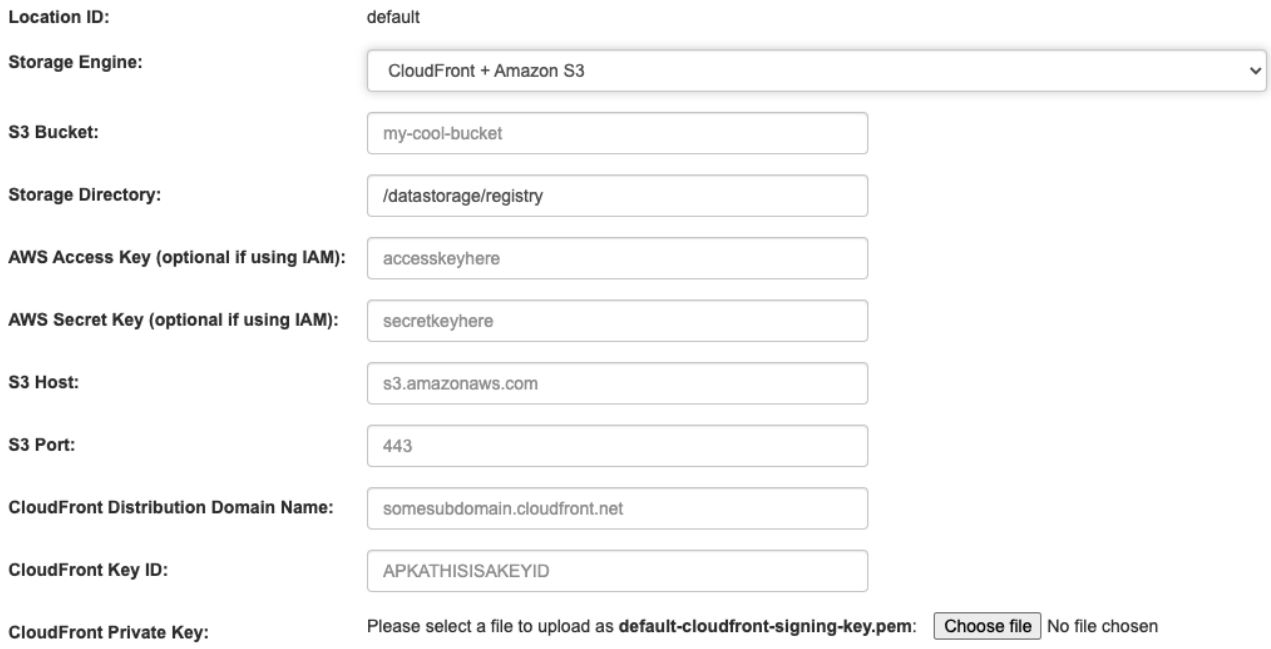
7.9.2.5. Ceph オブジェクトゲートウェイ (RADOS) ストレージ



7.9.2.6. OpenStack (Swift) ストレージ設定



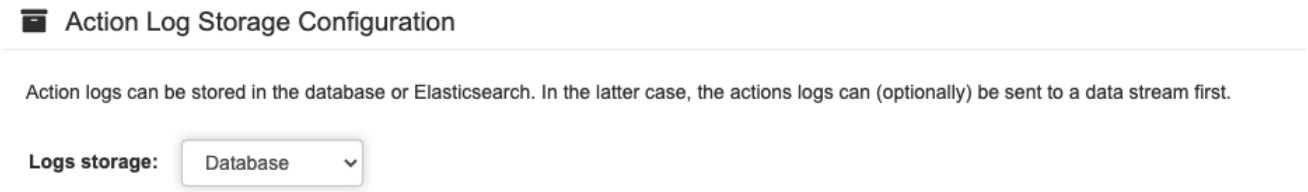
7.9.2.7. CloudFront + Amazon S3 ストレージ設定



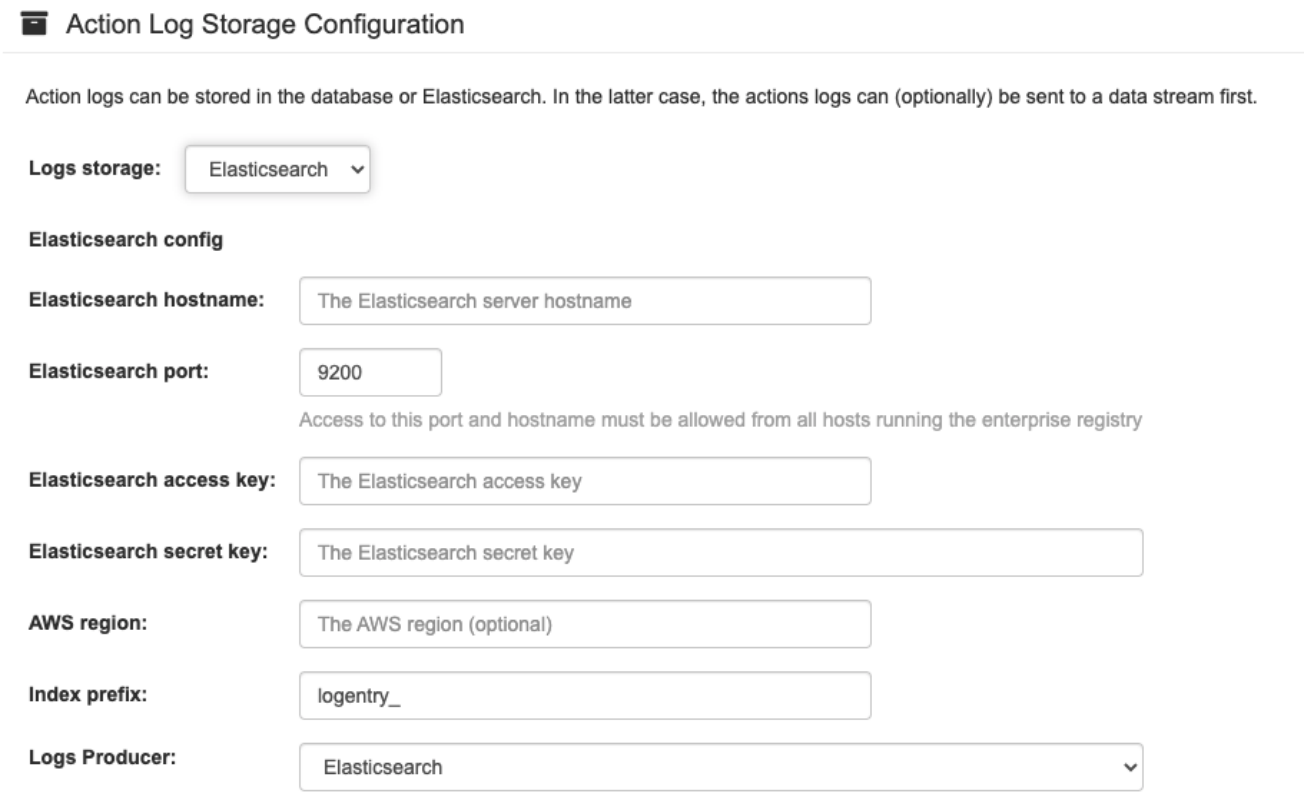
7.10. アクションログの設定

7.10.1. アクションログストレージ設定

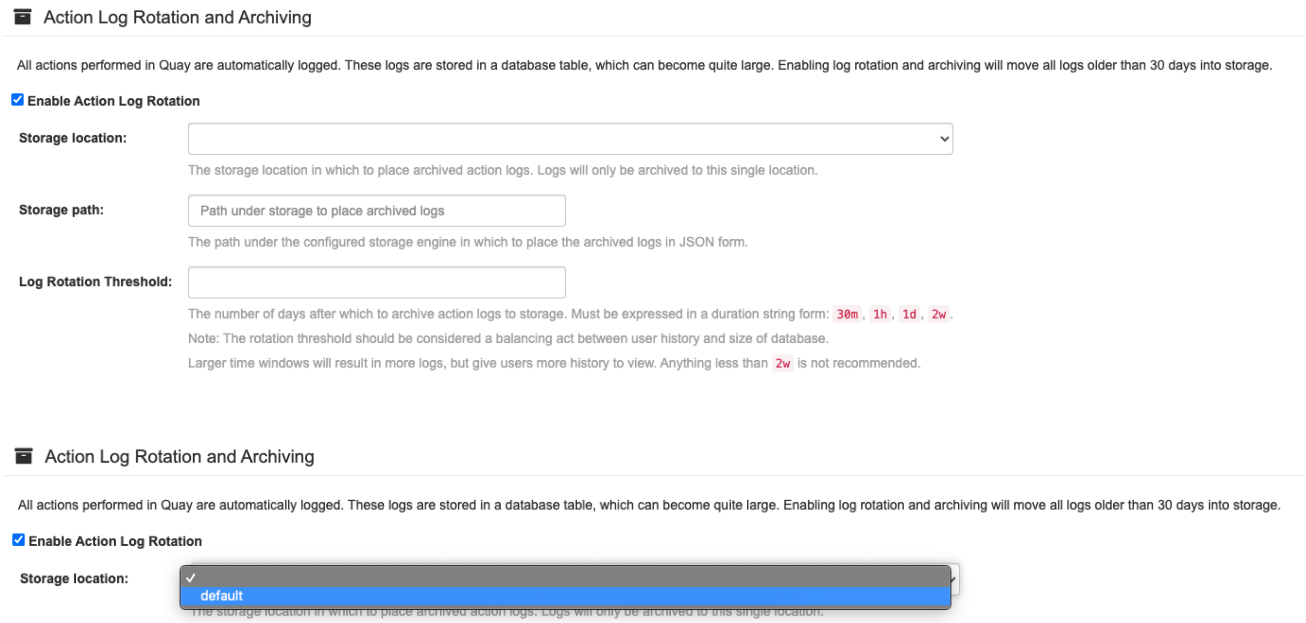
7.10.1.1. データベースアクションログストレージ



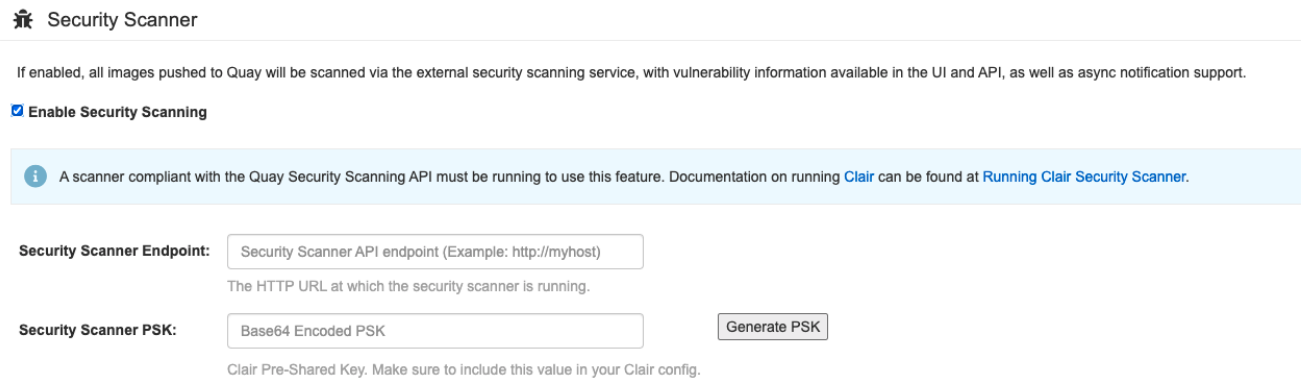
7.10.1.2. Elasticsearch アクションログストレージ



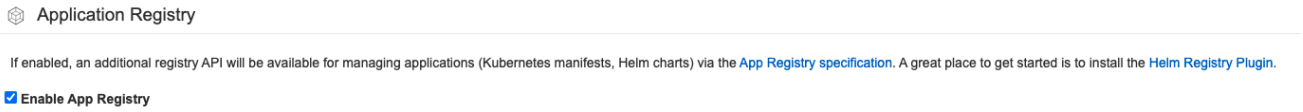
7.10.2. アクションログのローテーションおよびアーカイブ



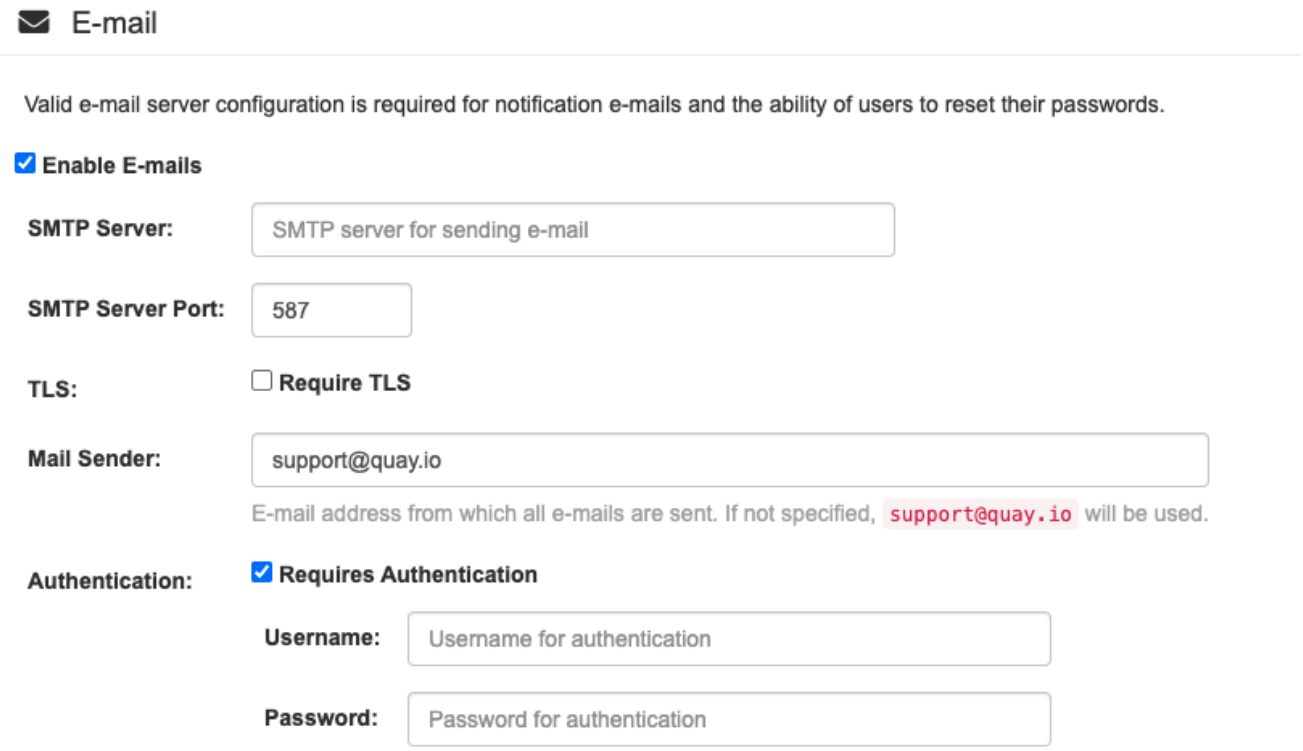
7.11. セキュリティースキャナーの設定



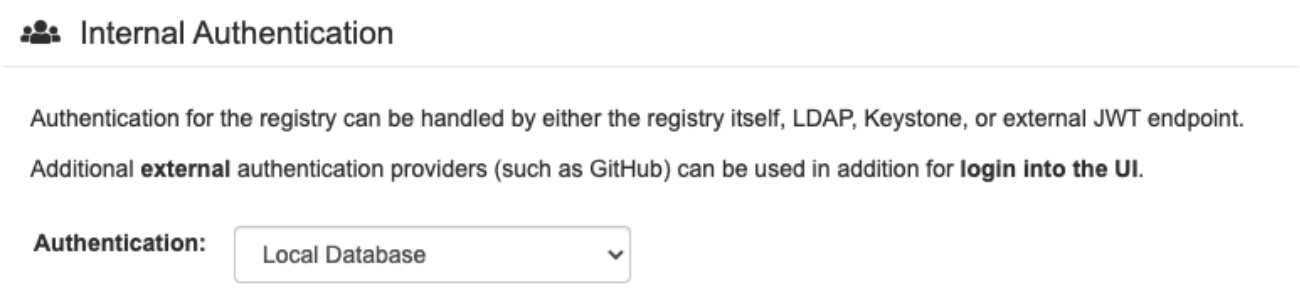
7.12. アプリケーションレジストリーの設定

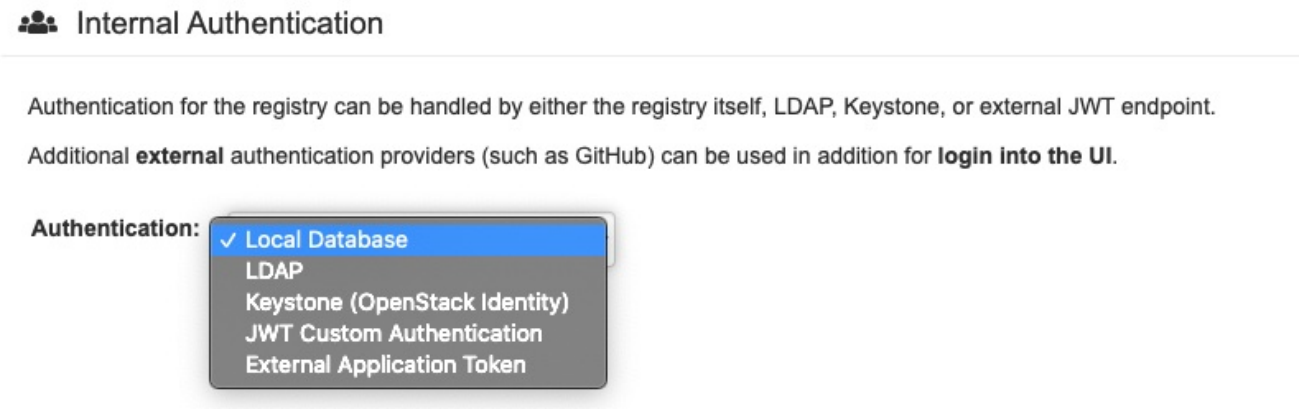


7.13. メール設定

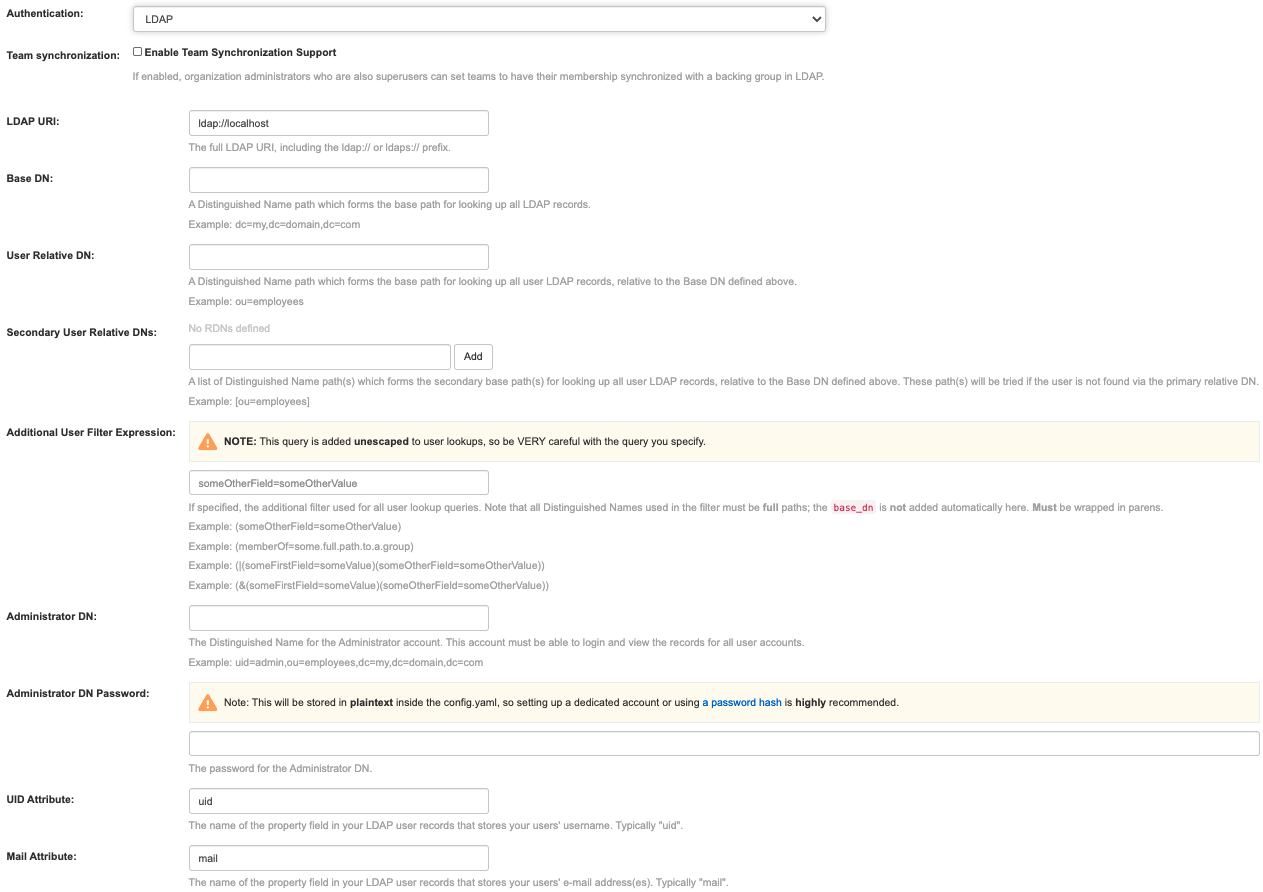


7.14. 内部認証設定

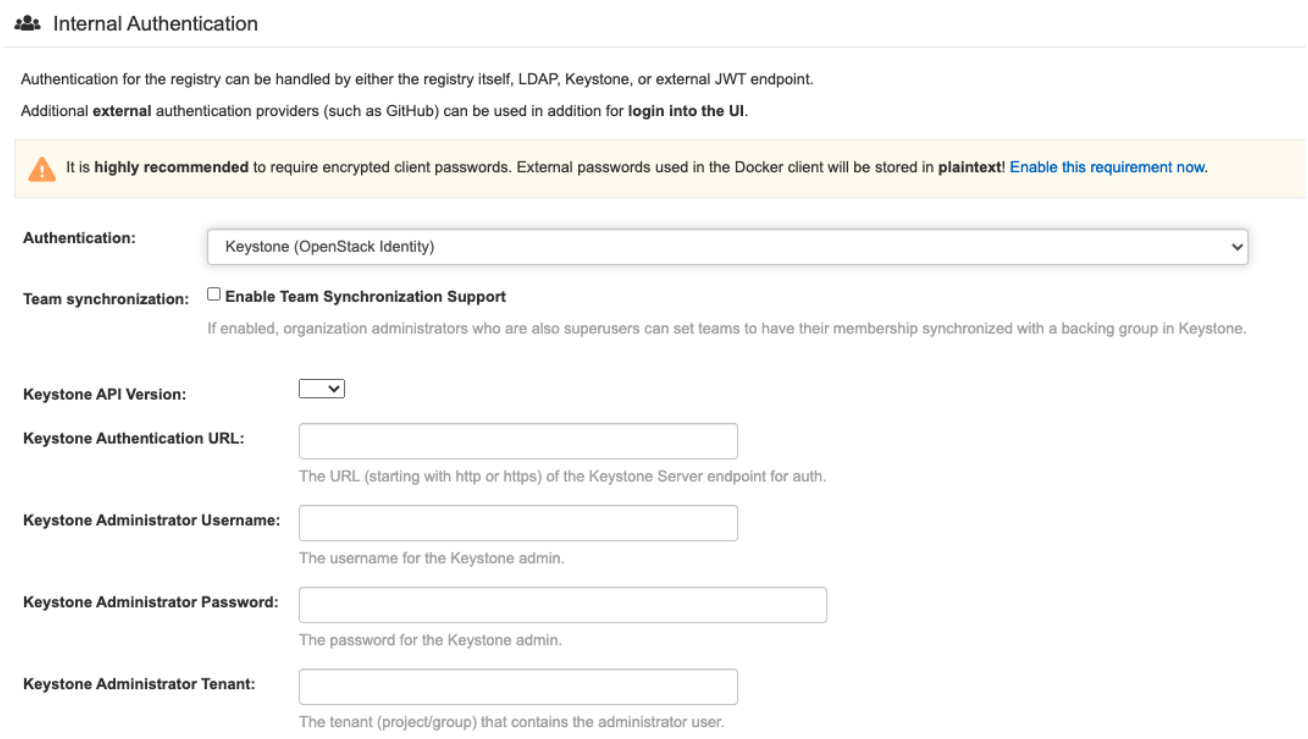




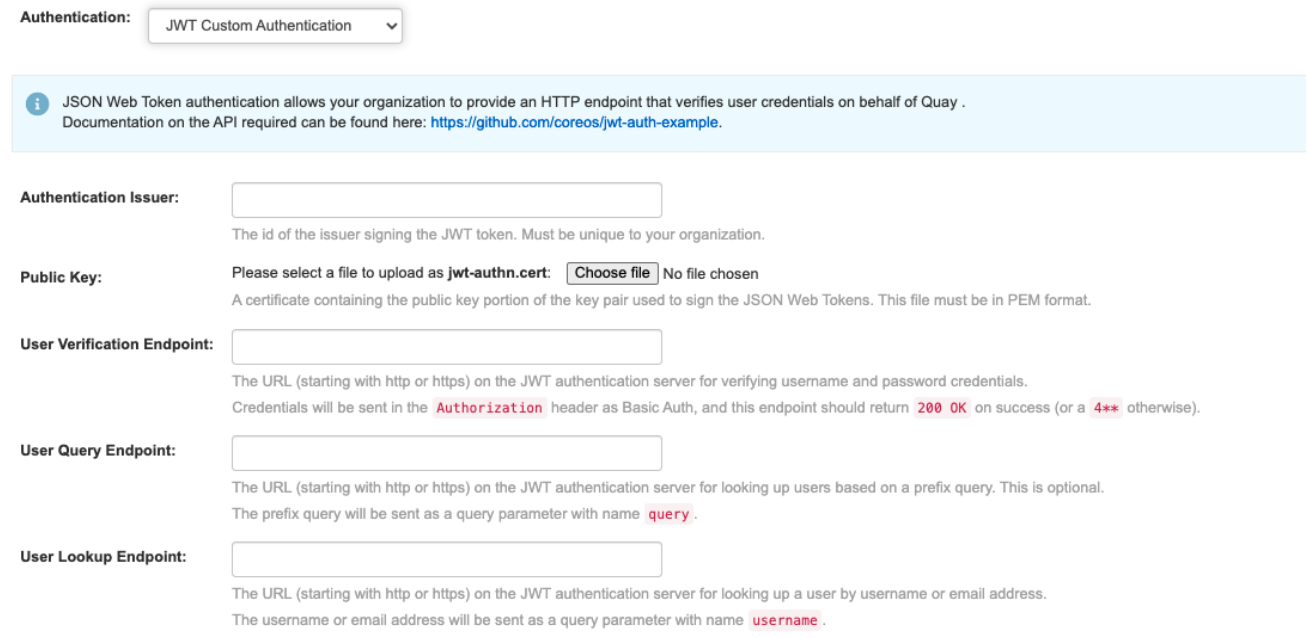
7.14.1. LDAP



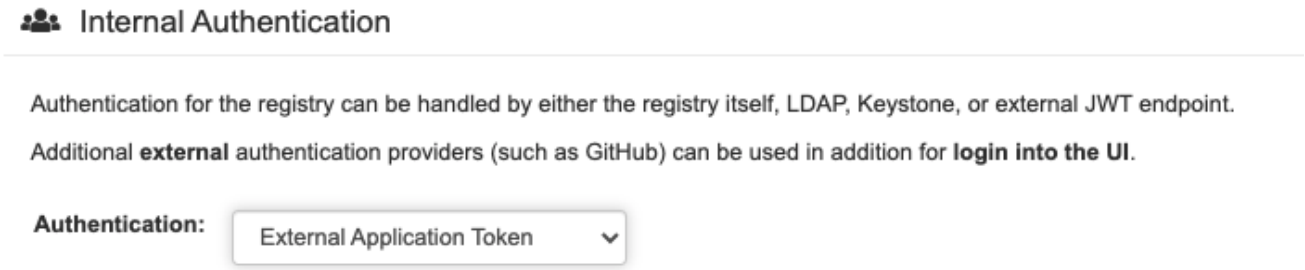
7.14.2. Keystone (OpenStack identity)



7.14.3. JWT カスタム認証

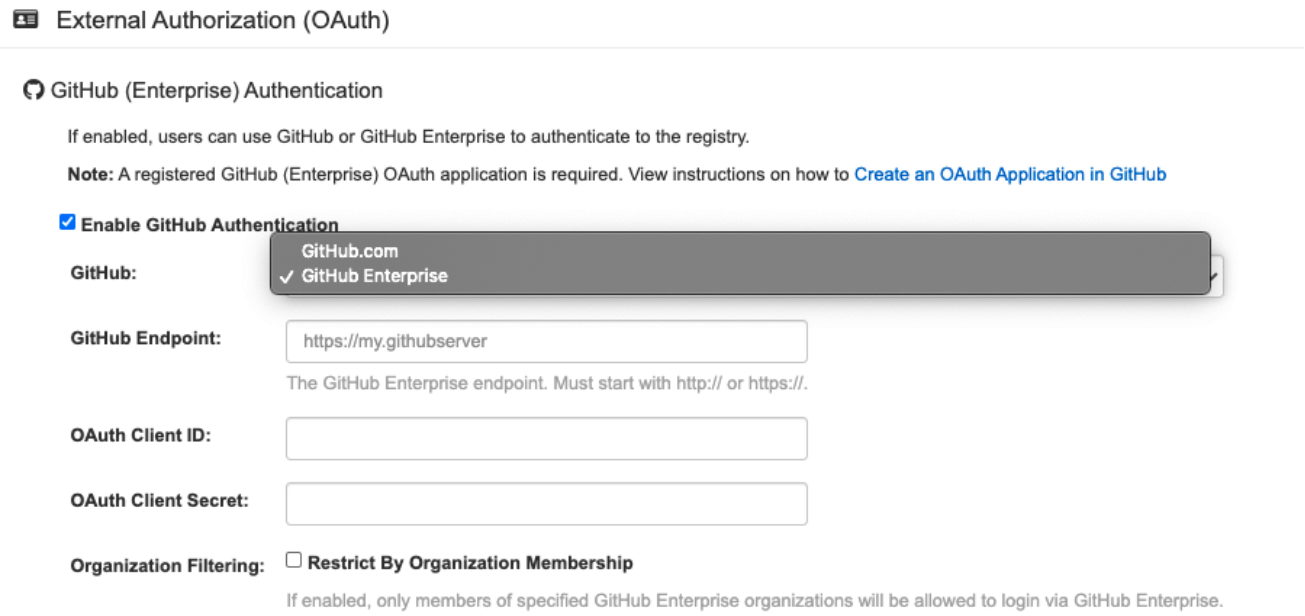


7.14.4. 外部アプリケーショントークン

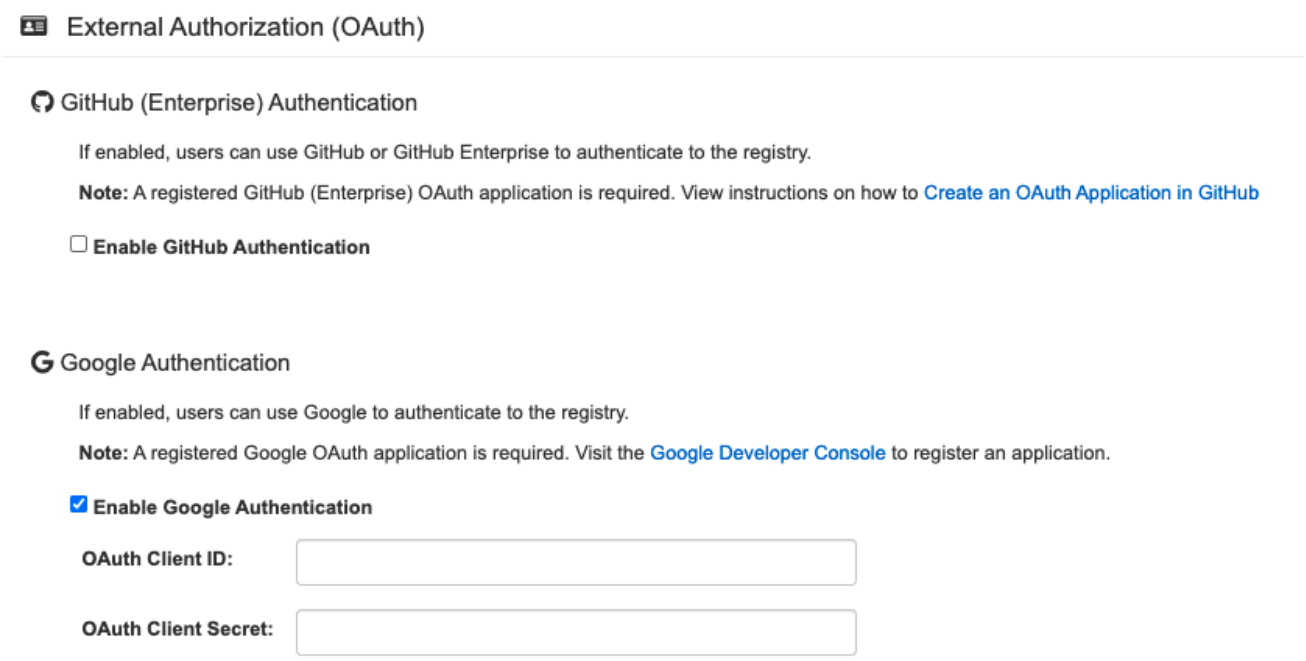


7.15. 外部認証 (OAUTH) の設定

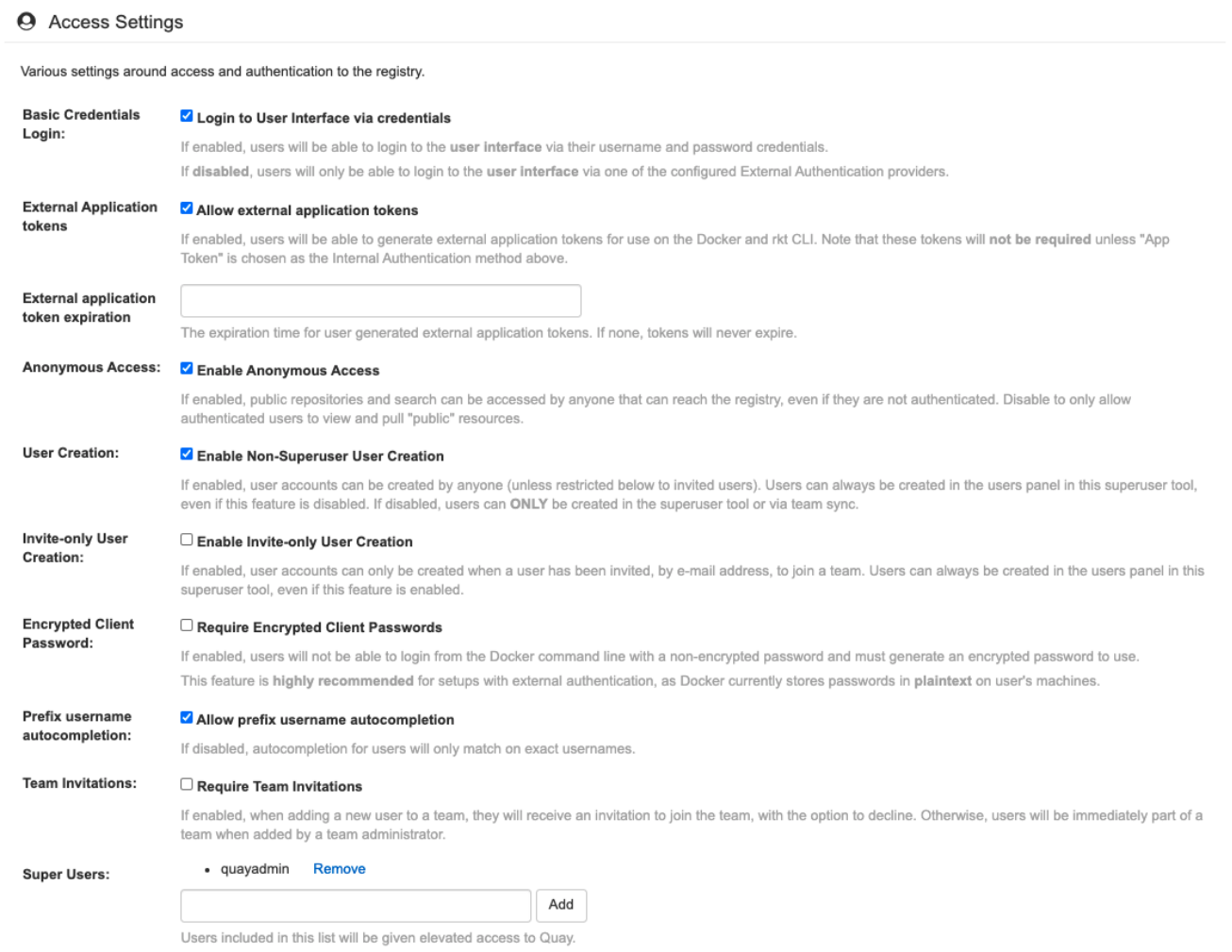
7.15.1. GitHub (Enterprise) 認証



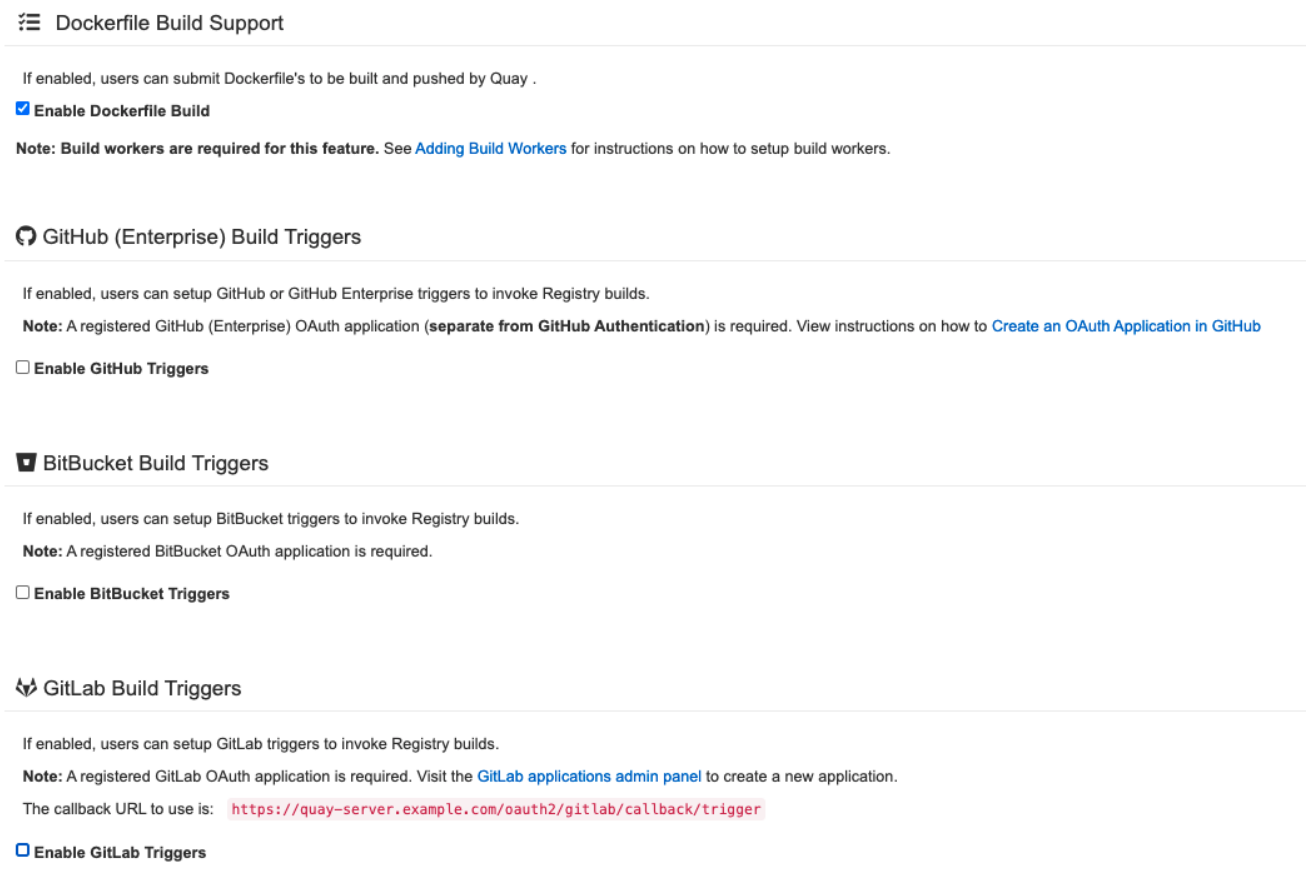
7.15.2. Google 認証



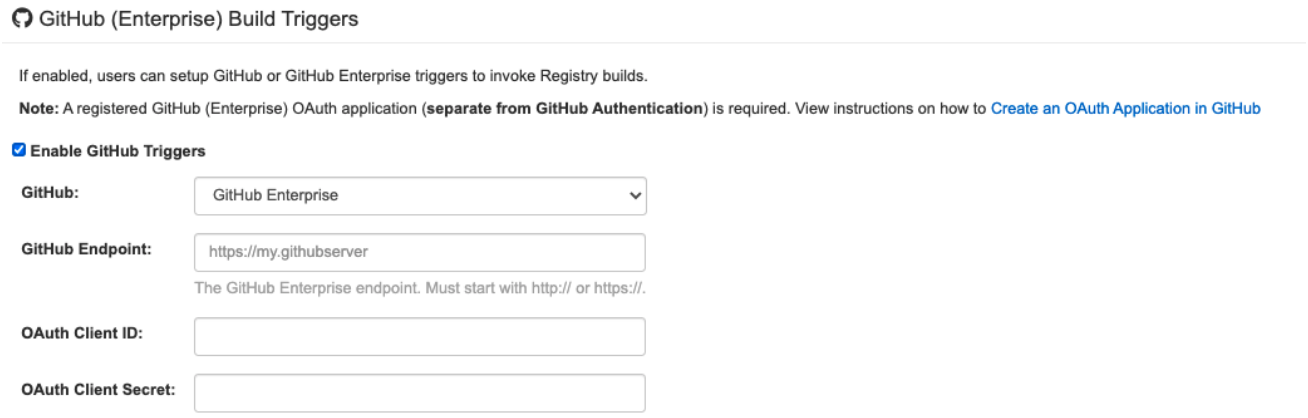
7.16. アクセス設定



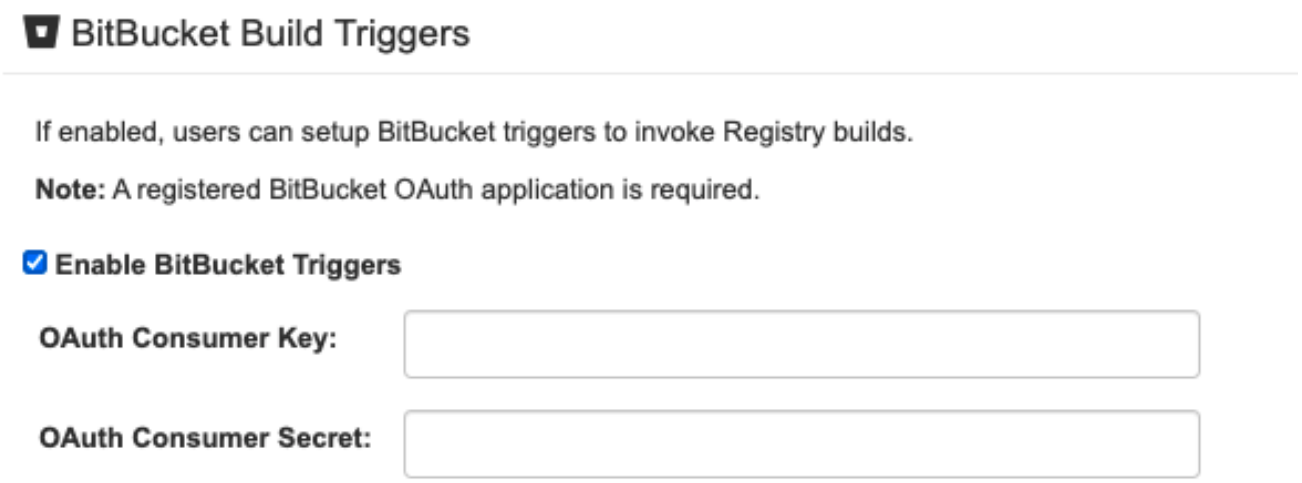
7.17. DOCKERFILE ビルドのサポート



7.17.1. GitHub (Enterprise) ビルドトリガー



7.17.2. Bitbucket ビルドトリガー



7.17.3. GitLab ビルドトリガー