Red Hat Quay 3.3

Red Hat Quay の管理

Red Hat Quay の管理

Last Updated: 2022-09-16

Red Hat Quay 3.3 Red Hat Quay の管理

Red Hat Quay の管理

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

概要

Red Hat Quay の管理

目次

序文

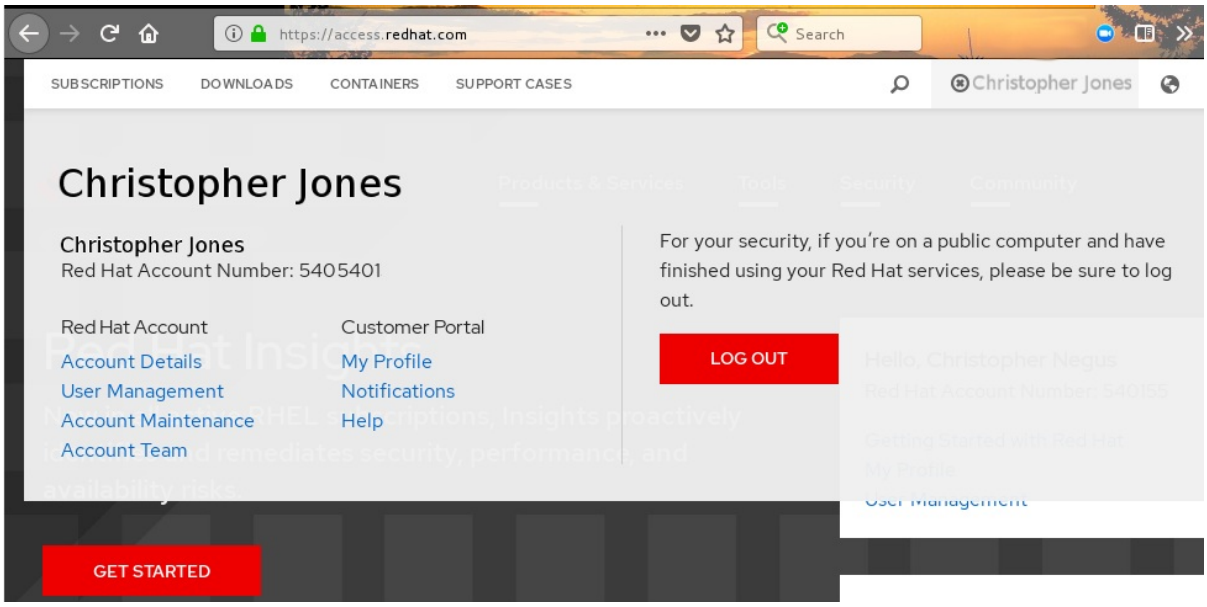
Red Hat Quay レジストリーをデプロイしたら、そのデプロイメントをさらに設定し、管理する方法は多数あります。ここで取り上げるトピックには以下が含まれます。

- 新規 Red Hat Quay リリースについてのアラートを送信するための通知の設定
- SSL および TLS 証明書による接続のセキュリティー保護
- アクションログのストレージの Elasticsearch へのダイレクト
- Clair でのイメージセキュリティースキャンの設定
- コンテナーセキュリティー Operator での Pod イメージのスキャン
- Quay Bridge Operator を使用した Red Hat Quay の OpenShift への統合
- リポジトリーミラーリングを使用したイメージのミラーリング
- BitTorrent サービスとの Quay イメージの共有
- LDAP を使用したユーザーの認証
- Prometheus および Grafana メトリクスの Quay の有効化
- Geo-replication のセットアップ
- Quay のトラブルシューティング

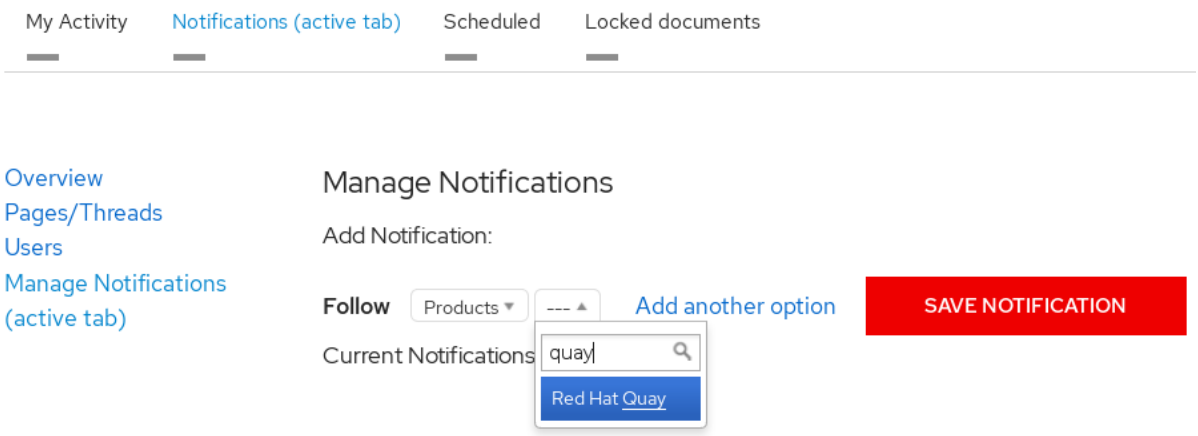
第1章 RED HAT QUAY リリース通知の取得

最新の Red Hat Quay リリースや、Red Hat Quay に関連するその他の変更に対応するには、Red Hat カスタマーポータル で更新通知にサインアップできます。通知へのサインアップ後に、Red Hat Quay の新規バージョン、ドキュメントの更新、またはその他の Red Hat Quay に関する最新情報の通知を受信できます。

1. Red Hat カスタマーポータルアカウントの認証情報を使用して Red Hat カスタマーポータル にログインします。
2. ユーザー名 (右上隅) を選択すると、Red Hat Account とカスタマーポータルの選択項目が表示されます。



3. Notifications を選択します。プロファイルアクティビティーページが表示されます。
4. Notifications タブを選択します。
5. Manage Notifications を選択します。
6. Follow を選択してから、ドロップダウンメニューから Products を選択します。
7. 製品の横にあるドロップダウンボックスで、Red Hat Quay を検索して選択します。



8. SAVE NOTIFICATION ボタンを選択します。今後は、新しいリリースなどの Red Hat Quay 製品に変更が加えられた場合に通知が送信されます。

第2章 SSL を使用した RED HAT QUAY への接続の保護

本書では、単一ノード、または 高可用性 デプロイメントに Red Hat Quay をデプロイしていることを前提としています。

自己署名証明書 で Red Hat Quay を設定するには、認証局 (CA) を作成し、必要なキーおよび証明書ファイルを生成する必要があります。次に、Red Hat Quay Config Tool またはコマンドラインを使用してこれらのファイルを入力します。

2.1. CA を作成して証明書に署名する

1. ルート CA を作成します。

```
$ openssl genrsa -out rootCA.key 2048
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
```

その結果、カレントディレクトリーに rootCA.key と rootCA.pem のファイルが作成されます。

2. **証明書およびプライベートキーの作成**: Red Hat Quay が TLS を処理する場合は、設定時に指定する証明書およびプライベートキーを作成する必要があります。これらのファイルは署名認証局 (certificate signing authority) から取得できます。ここでは、作成した自己署名認証局を使用してこれらのファイルを作成する方法を説明します。
この例では、device.crt ファイルおよび device.key ファイルを作成します。これらのファイルは Red Hat Quay にアップロードされ、名前 (ssl.cert と ssl.key) の変更が行われます。

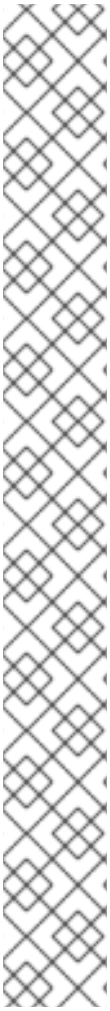
   OpenShift は長い完全修飾ドメイン名を作成するため、Red Hat Quay アプリケーションへの特定のルートを使用する代わりに、ワイルドカードを使用して大規模なドメインを特定できるようにすることを検討してください。例えば、サーバーのホスト名を聞かれたら、*.apps.openshift.example.com のように入力します。

```
Common Name (eg, your name or your server's hostname) []:*apps.openshift.example.com
```

```
$ openssl genrsa -out device.key 2048
$ openssl req -new -key device.key -out device.csr
```

そして、先ほど作成したルート CA で証明書に署名します。

```
$ openssl x509 -req -in device.csr -CA rootCA.pem \
      -CAkey rootCA.key -CAcreateserial -out device.crt -days 500 -sha256
```



注記

先ほどの*.key と*.crt ファイルを生成する代わりに、**openssl.cnf** ファイルを作成することができます。これにより、証明書要求を生成するコマンドのプロンプトに応答するだけで取得できる情報よりも多くの情報を生成される証明書に追加できます。以下の例の **openssl.cnf** ファイルでは、**DNS.1** と **IP.1** を、Red Hat Quay サーバーのホスト名および IP アドレスに置き換えます。

openssl.cnf

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = reg.example.com
IP.1 = 12.345.678.9
```

そして、次のようにして鍵を生成することができます。

```
$ openssl x509 -req -in ssl.csr -CA rootCA.pem \
   -CAkey rootCA.key -CAcreateserial -out ssl.cert \
   -days 356 -extensions v3_req -extfile openssl.cnf
```

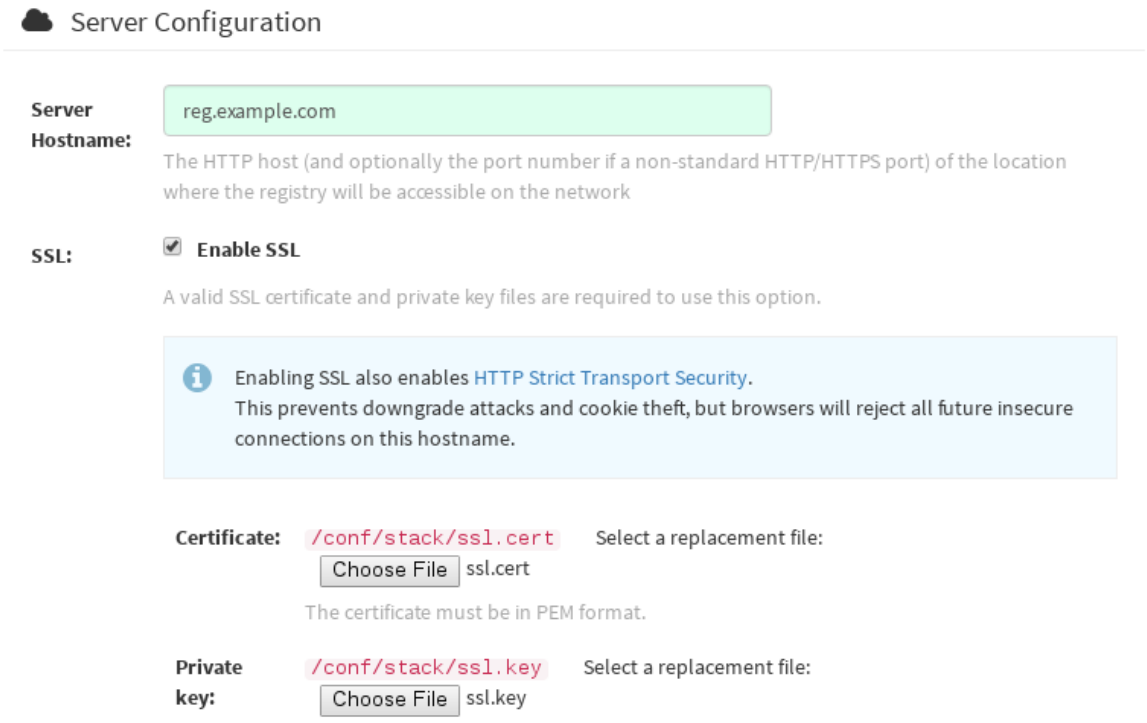
2.2. RED HAT QUAY が新しい証明書を使用するように設定します。

次のステップは、Red Hat Quay 画面またはターミナルで実行できます。

2.2.1. Red Hat Quay Setup 画面からの SSL の設定

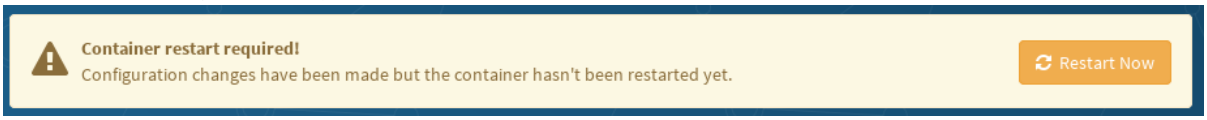
それぞれのデプロイメントガイドで説明されているように、config モードで quay コンテナーを起動します。server Configuration セクションで、以下のように SSL を有効にします。

1. **Server Hostname** を適切な値に設定し、**Enable SSL** ボックスにチェックを付けてから **ssl.key** ファイルと **ssl.cert** ファイル（この例では名前は **device.key** と **device.crt**）をアップロードします。



2. 設定を保存します。Red Hat Quay は SSL 証明書を自動的に検証します。

3. コンテナーを再起動します。



2.2.2. コマンドラインでの設定

Web インターフェイスを使用しないため、Red Hat Quay に組み込まれた設定チェックメカニズムは使用できません。可能な場合は Web インターフェイスを使用することが推奨されます。OpenShift 以外のインストールでは、以下のようにコマンドラインインターフェイスから SSL を設定できます。

1. **ssl.key および ssl. cert** を、指定した **config** ディレクトリーにコピーします。この例では、Red Hat Quay の設定ディレクトリーは、reg.example.com という名前のホスト上の /mnt/quay/config という名前のディレクトリーにあります。

注記

証明書/鍵ファイルの名前は ssl.key と ssl.cert でなければなりません。

```
$ ls
ssl.cert  ssl.key
$ scp ssl.* root@reg.example.com:/mnt/quay/config/
[root@reg.example.com ~]$ ls /mnt/quay/config/
config.yaml  ssl.cert  ssl.key
```

2. config.yaml の **PREFERRED_URL_SCHEME:** パラメーターを **http** から **https** に変更します。

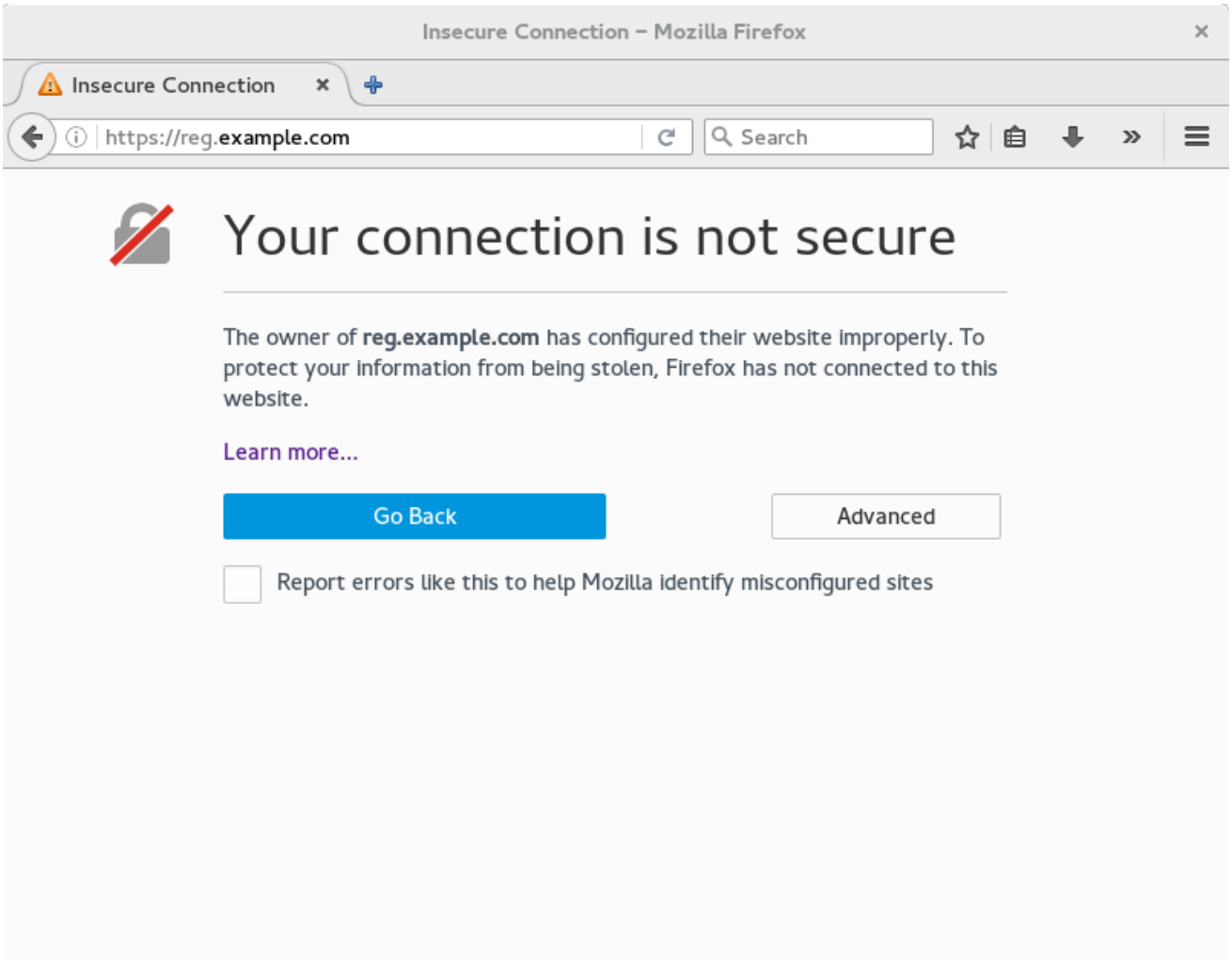
```
PREFERRED_URL_SCHEME: https
```

3. Red Hat Quay コンテナーを再起動します。

```
$ docker ps
CONTAINER ID  IMAGE                 COMMAND              CREATED       STATUS
PORTS                 NAMES
eaf45a4aa12d  ...redhat.com/rhscl/redis  "/usr/bin/redis-serve" 22 hours ago  Up 22 hours
0.0.0.0:6379->6379/tcp  dreamy...
cbe7b0fa39d8  quay.io/redhat/quay       "/sbin/my_init"         22 hours ago  Up one hour
80/tcp,443/tcp,443/tcp  ferv...
705fe7311940  mysql:5.7                 "/entrypoint.sh mysql"  23 hours ago  Up 22 hours
0.0.0.0:3306->3306/tcp  mysql
$ docker restart cbe7b0fa39d8
```

2.2.3. 安全な接続のテスト

ブラウザー



から URL にアクセスして設定を確認します。「Your Connection is not secure」は、CA が信頼されていないが、SSL が適切に機能していることを意味します。https://reg.example.com/ **これらのメッセージを回避するには、信頼される認証局から証明書を取得する必要があります。**

2.3. 認証局を信頼するように DOCKER を設定する

Docker では、カスタム証明書をホスト名のプライベートレジストリーと同じ名前のディレクトリーにある /etc/docker/certs.d/ にインストールする必要があります。また、証明書を ca.crt という名前にする必要もあります。以下に、これを実行する手順を示します。

1. **rootCA ファイルをコピーします。**

```
$ cp tmp/rootCA.pem /etc/docker/certs.d/reg.example.com/ca.crt
```

2. **rootCA.pem ファイルをコピーすると、docker login が正常に認証され、リポジトリーへのプッシュが成功するはずです。**

```
$ sudo docker push reg.example.com/kbrwn/hello
The push refers to a repository [reg.example.com/kbrwn/hello]
5f70bf18a086: Layer already exists
e493e9cb9dac: Pushed
1770dbc4af14: Pushed
a7bb4eb71da7: Pushed
9fad7adcbd46: Pushed
2cec07a74a9f: Pushed
f342e0a3e445: Pushed
b12f995330bb: Pushed
2016366cdd69: Pushed
a930437ab3a5: Pushed
15eb0f73cd14: Pushed
latest: digest:
sha256:c24be6d92b0a4e2bb8a8cc7c9bd044278d6abdf31534729b1660a485b1cd315c size:
7864
```

第3章 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 設定ディレクトリーの下に extra_ca_certs/ という名前の新規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。

3.1. TLS 証明書の RED HAT QUAY への追加

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. 証明書ディレクトリーを作成し、証明書をそこにコピーします。

```
$ 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. docker ps で quay コンテナーの CONTAINER ID を取得します。

```
$ docker ps
CONTAINER ID        IMAGE                         COMMAND                  CREATED
STATUS              PORTS
5a3e82c4a75f        <registry>/<repo>/quay:v3.3.4 "/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。

```
$ docker 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。

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

3.2. KUBERNETES へのデプロイ時に証明書を追加

Kubernetes にデプロイされると、Red Hat Quay は設定アセットを保存するためのボリュームとしてシークレットにマウントします。ただし、現時点でこれを実行すると、スーパーユーザーパネルの証明書アップロード機能に障害が発生します。

このエラーを回避するには、Red Hat Quay のデプロイ 後 に base64 でエンコードされた証明書をシークレットに追加できます。以下に、これを実行する方法を示します。

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、置き換え Pod を

新しい証明書データで自動的にスケジュールします。

第4章 ELASTICSEARCH 用のアクションログストレージの設定

デフォルトで、過去 3 カ月間の使用ログは Red Hat Quay データベースに保存され、Web UI から組織およびリポジトリーレベルで公開されます。ログエントリーを表示するには、適切な管理者権限が必要です。多数のログに記録された操作を使用するデプロイメントの場合、使用ログは Red Hat Quay データベースバックエンドではなく Elasticsearch に保存できるようになりました。これを実行するには、独自の Elasticsearch スタックを提供する必要があります。これはカスタマイズ可能なコンポーネントとして Red Hat Quay に含まれていないためです。

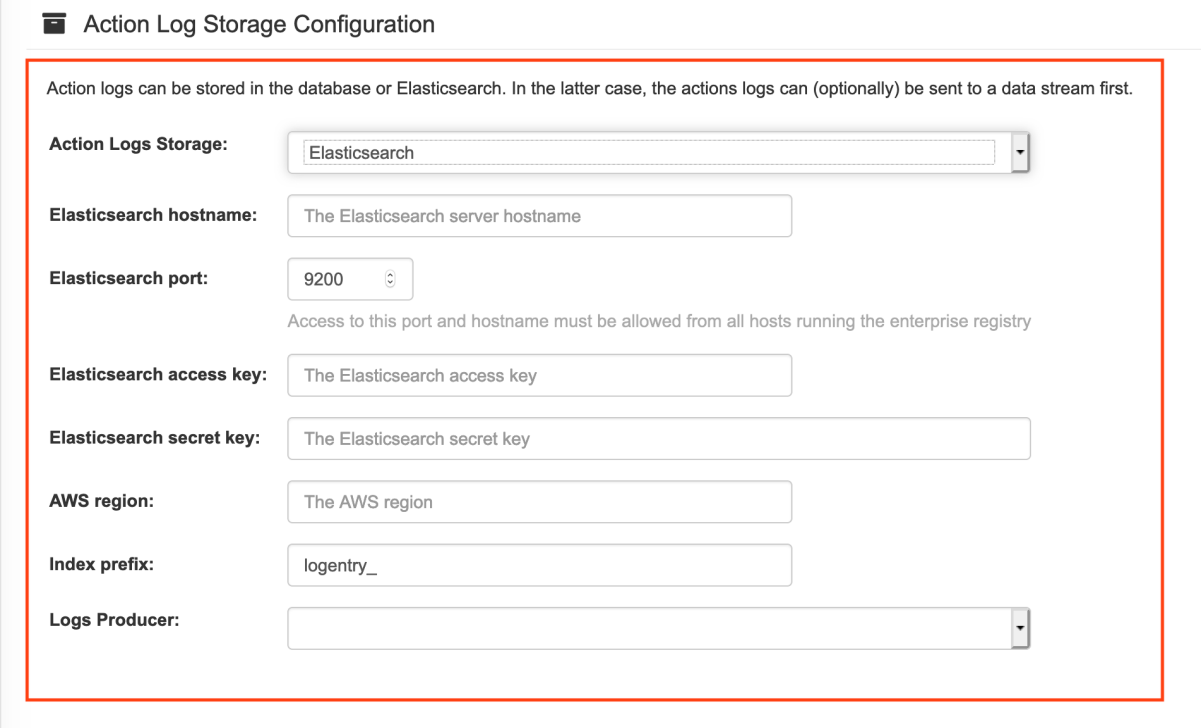
Elasticsearch ロギングは、Red Hat Quay デプロイメント時または Red Hat Quay Config Tool を使用してデプロイメント後に有効にできます。生成される設定は config.yaml ファイルに保存されます。設定後は、リポジトリーや組織用に Web UI を使用して使用ログのアクセスが引き続き提供されます。

以下は、デフォルトの Red Hat Quay データベースから Elasticsearch を使用するようにアクションログストレージを設定する方法です。

1. Elasticsearch アカウントを取得します。

2. Red Hat Quay Config Tool を開きます (Red Hat Quay のデプロイメント時またはその後のいずれか)。

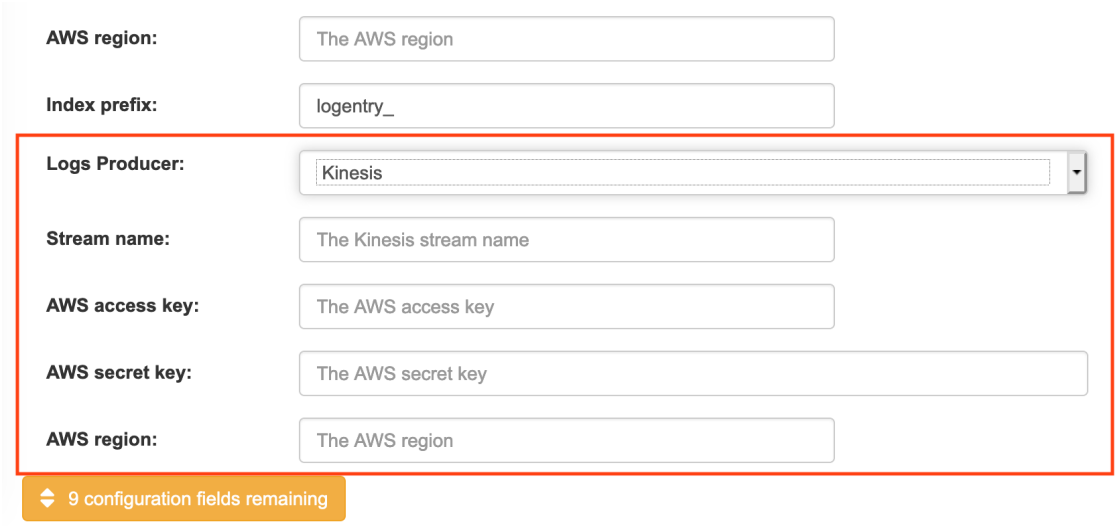
3. Action Log Storage Configuration 設定までスクロールし、Database ではなく Elasticsearch を選択します。以下の図は、表示される Elasticsearch 設定を示しています。



4. **Elasticsearch インスタンスについて以下の情報を入力します。**

- **Elasticsearch hostname: Elasticsearch サービスを提供するシステムのホスト名または IP アドレス。**
- **Elasticsearch port: 入力したホストで Elasticsearch サービスを提供するポート番号。ポートは、Red Hat Quay レジストリーを実行しているすべてのシステムからアクセスできる必要があることに注意してください。デフォルトは TCP ポート 9200 です。**
- **Elasticsearch access key: Elastic 検索サービスへのアクセスに必要なアクセスキー (必要な場合)。**
- **Elasticsearch secret key: Elastic 検索サービスへのアクセスに必要なシークレットキー (必要な場合)。**
- **AWS region: AWS で実行している場合は、AWS リージョンを設定します (それ以外の場合は、空白のままにします)。**
- **Index prefix: ログエントリーに割り当てる接頭辞を選択します。**

- Log Producer: Elasticsearch (デフォルト) または Kinesis のいずれかを選択し、AWS の中間 Kinesis ストリームにログをダイレクトします。ログを Kinesis から Elasticsearch (例: Logstash) に送信するために独自のパイプラインを設定する必要があります。以下の図は、Kinesis についての入力に必要な追加のフィールドを示しています。



5. Elasticsearch を Logs Producer として選択した場合、追加の設定は必要ありません。Kinesis を選択する場合は、以下を入力します。

- Stream name: Kinesis ストリームの名前。
- AWS access key: Kinesis ストリームへのアクセスを取得するために必要な AWS アクセスキーの名前 (必要な場合)。
- AWS secret key: Kinesis ストリームへのアクセスを取得するために必要な AWS シークレットキーの名前 (必要な場合)。
- AWS region: AWS リージョン。

6. 完了したら、設定を保存します。Config Tool は設定を確認します。Elasticsearch または Kinesis サービスへの接続に問題がある場合には、エラーが表示され、編集を続行できます。それ以外の場合、クラスターが新規設定で再起動された後に、ロギングは Elasticsearch 設定にダイレクトされます。

第5章 CLAIR による RED HAT QUAY のセキュリティースキャン

Red Hat Quay では、Clair などのスキャンエンジンを使用して、既知の脆弱性についてのコンテナーイメージのスキャンをサポートします。本書では、Red Hat Quay で Clair を設定する方法について説明しています。

5.1. RED HAT QUAY 設定ツールでの CLAIR のセットアップ

Red Hat Quay で Clair を有効にすることには、以下が必要になります。

- Red Hat Quay 設定ツールを起動します。特定の環境について設定ツールを起動する方法については、デプロイメントタイプ (OpenShift、Basic、または HA) ごとに Red Hat Quay デプロイメントガイドを参照してください。
- セキュリティースキャンを有効にし、設定ツールでプライベートキーおよび PEM ファイルを生成します。
- キーおよび PEM ファイルを Clair 設定ファイルに組み込みます。
- Clair コンテナーを起動します。

OpenShift で Red Hat Quay を実行するか、またはホスト上で直接実行しているかによって手順は異なります。

5.1.1. Red Hat Quay OpenShift デプロイメントでの Clair の有効化

OpenShift で Red Hat Quay に Clair をセットアップするには、Add Clair image scanning to Red Hat Quay を参照してください。

5.1.2. Clair の Red Hat Quay Basic または HA デプロイメントでの有効化

コンテナーがホストシステムで直接実行されている Red Hat Quay デプロイメントで Clair をセットアップするには、以下を実行します。

1. Red Hat Quay 設定ツールの再起動: config モードで quay コンテナーを再度実行し、ブラ

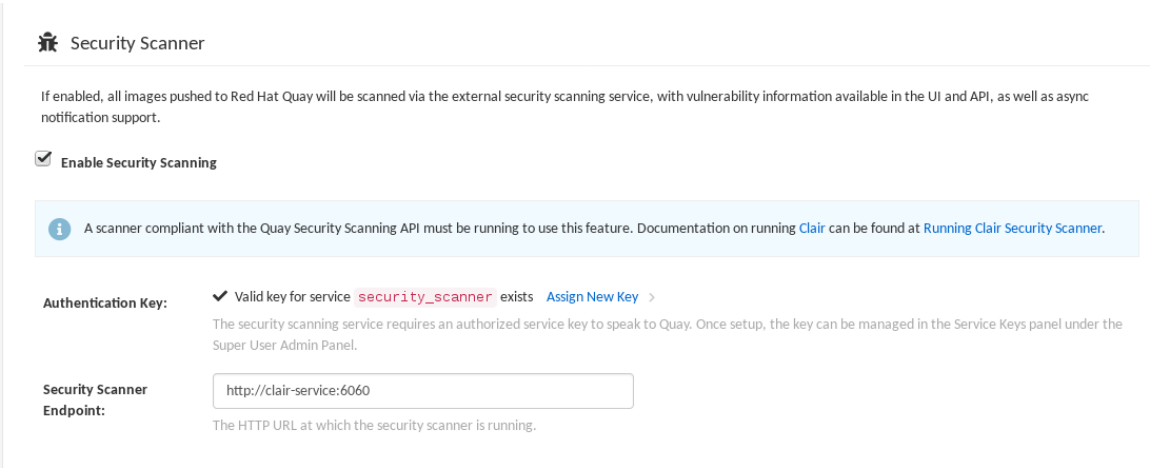
ウザーで設定 UI を開き、Modify an existing configuration を選択します。プロンプトが出されたら、デプロイメント用に最初に作成された quay-config.tar.gz ファイルをアップロードします。

2. セキュリティースキャンを有効にします。セキュリティースキャナーのセクションにスクロールし、Enable Security Scanning チェックボックスを選択します。表示されるフィールドで認証キーを作成し、セキュリティースキャナーエンドポイントを入力する必要があります。以下に、これを実行する方法を示します。

- キーの生成: Create Key をクリックし、ポップアップウインドウから Clair プライベートキーの名前とオプションの有効期限を入力します (空白の場合、キーが有効期限切れになることはありません)。次に Generate Key を選択します。

- Clair キーおよび PEM ファイルのコピー: キー ID を (notepad などに) 保存し、Download Private Key を選択して、プライベートキー PEM ファイル (security_scanner.pem) のコピーをダウンロードします (キーを紛失した場合は、キーを新たに生成する必要があります)。Clair コンテナーを後で起動する際に、キーと PEM ファイルが必要になります。

終了したらポップアップを閉じます。以下は、完了したセキュリティースキャナー設定の例です。



3. 設定を保存します。Save Configuration Changes をクリックしてから Download Configuration を選択し、これをローカルシステムに保存します。

4. 設定をデプロイします。スキャンを有効にする変更や設定に加えたその他の変更を取得するには、quay-config.tar.gz を展開し、生成されるファイルを config ディレクトリーにコピー

します。例を以下に示します。

```
$ tar xvf quay-config.tar.gz
config.yaml  ssl.cert  ssl.key
$ cp config.yaml ssl* /mnt/quay/config
```

次に、以下のセクションで説明されているように、Clair コンテナーと関連するデータベースを起動します。

第6章 CLAIR セキュリティースキャンのセットアップ

Red Hat Quay 設定 UI から必要なキーおよび pem ファイルを作成したら、Clair コンテナーおよび関連するデータベースを起動することができます。これが完了したら、Red Hat Quay クラスターを再起動してこれらの変更を反映します。

Clair コンテナーおよび関連するデータベースを実行する手順は、OpenShift でこれらを実行する場合とホスト上でこれらのコンテナーを直接実行する場合とは異なります。

6.1. RED HAT QUAY OPENSHIFT デプロイメントでの CLAIR の実行

Red Hat Quay クラスターを含む OpenShift 環境で Clair イメージスキャンコンテナーと関連するデータベースを実行するには、Add Clair image scanning to Red Hat Quay を参照してください。

6.2. CLAIR の RED HAT QUAY BASIC または HA デプロイメントでの実行

OpenShift 以外の環境で (ホストで直接) Clair とその関連データベースを実行するには、以下を行う必要があります。

- データベースの起動
- Clair の設定および起動

6.2.1. Postgres および Clair の取得

Clair を実行するには、データベースが必要です。実稼働環境のデプロイメントでは、MySQL はサポートされません。実稼働環境には、PostgreSQL またはその他のサポートされるデータベースを使用することが推奨されます。

- Red Hat Quay を実行しているマシン以外のマシンで実行する。
- (可能な場合) 自動レプリケーションとフェイルオーバーを使用する。

テストの目的で、単一の PostgreSQL インスタンスをローカルで起動することができます。

1. **Postgres をローカルで起動するには、以下を実行します。**

```
# docker run --name postgres -p 5432:5432 -d postgres
# sleep 5
# docker run --rm --link postgres:postgres postgres \
   sh -c 'echo "create database clairtest" | psql -h \
   "$POSTGRES_PORT_5432_TCP_ADDR" -p  \
   "$POSTGRES_PORT_5432_TCP_PORT" -U postgres'
```

このテストデータベースの設定文字列は以下のようになります。

```
postgresql://postgres@{DOCKER HOST GOES HERE}:5432/clairtest?sslmode=disable
```

2. **セキュリティー対応の Clair イメージをプルします。**

```
docker pull quay.io/redhat/clair-jwt:v3.3.4
```

3. **Clair の設定ディレクトリーを作成します。**

```
# mkdir clair-config
# cd clair-config
```

6.2.2. Clair の設定

Clair は単一インスタンスとして、または高可用性モードで実行できます。Clair の複数のインスタンスを実行することが推奨されます (可能な場合は、自動修復機能を使用して自動スケーリンググループで実行する)。

1. **以下に示されるように 2 つの Clair 設定ファイルのいずれかから、Clair 設定ディレクトリー (/clair/config) で使用される config.yaml ファイルを作成します。**

2. **高可用性インストールを行う場合は、Authentication for high-availability scanners の手順に従ってキー ID およびプライベートキー (PEM) を作成します。**

3. **プライベートキー (PEM) をファイルに保存します (例: $HOME/config/security_scanner.pem)。**

4.

key_id (CLAIR_SERVICE_KEY_ID) の値を生成したキー ID に置き換え、private_key_path の値を PEM ファイルの場所に置き換えます (例: /config/security_scanner.pem)。

たとえば、これらの 2 つの値は以下のように表示される場合があります。

```
key_id: { 4fb9063a7cac00b567ee921065ed16fed7227afd806b4d67cc82de67d8c781b1 }
private_key_path: /clair/config/security_scanner.pem
```

5. **必要に応じて、設定ファイルの他の値を変更します。**

6.2.2.1. Clair 設定: 高可用性

```
clair:
  database:
    type: pgsql
    options:
      # A PostgreSQL Connection string pointing to the Clair Postgres database.
      # Documentation on the format can be found at: http://www.postgresql.org/docs/9.4/static/libpq-
connect.html
      source: { POSTGRES_CONNECTION_STRING }
      cachesize: 16384
  api:
    # The port at which Clair will report its health status. For example, if Clair is running at
    # https://clair.mycompany.com, the health will be reported at
    # http://clair.mycompany.com:6061/health.
    healthport: 6061

    port: 6062
    timeout: 900s

    # paginationkey can be any random set of characters. *Must be the same across all Clair
instances*.
    paginationkey: "XxoPtCUzrUv4JV5dS+yQ+MdW7yLEJnRMwigVY/bpgtQ="

  updater:
    # interval defines how often Clair will check for updates from its upstream vulnerability databases.
    interval: 6h
  notifier:
    attempts: 3
    renotifyinterval: 1h
    http:
      # QUAY_ENDPOINT defines the endpoint at which Quay is running.
      # For example: https://myregistry.mycompany.com
      endpoint: { QUAY_ENDPOINT }/secscan/notify
      proxy: http://localhost:6063

jwtproxy:
  signer_proxy:
    enabled: true
    listen_addr: :6063
```

```
  ca_key_file: /certificates/mitm.key # Generated internally, do not change.
  ca_crt_file: /certificates/mitm.crt # Generated internally, do not change.
  signer:
    issuer: security_scanner
    expiration_time: 5m
    max_skew: 1m
    nonce_length: 32
    private_key:
      type: preshared
      options:
        # The ID of the service key generated for Clair. The ID is returned when setting up
        # the key in [Quay Setup](security-scanning.md)
        key_id: { CLAIR_SERVICE_KEY_ID }
        private_key_path: /clair/config/security_scanner.pem

 verifier_proxies:
 - enabled: true
   # The port at which Clair will listen.
   listen_addr: :6060

   # If Clair is to be served via TLS, uncomment these lines. See the "Running Clair under TLS"
   # section below for more information.
   # key_file: /clair/config/clair.key
   # crt_file: /clair/config/clair.crt

   verifier:
     # CLAIR_ENDPOINT is the endpoint at which this Clair will be accessible. Note that the port
     # specified here must match the listen_addr port a few lines above this.
     # Example: https://myclair.mycompany.com:6060
     audience: { CLAIR_ENDPOINT }

     upstream: http://localhost:6062
     key_server:
       type: keyregistry
       options:
         # QUAY_ENDPOINT defines the endpoint at which Quay is running.
         # Example: https://myregistry.mycompany.com
         registry: { QUAY_ENDPOINT }/keys/
```

6.2.2.2. Clair 設定: 単一インスタンス

```
clair:
  database:
    type: pgsql
    options:
      # A PostgreSQL Connection string pointing to the Clair Postgres database.
      # Documentation on the format can be found at: http://www.postgresql.org/docs/9.4/static/libpq-
connect.html
      source: { POSTGRES_CONNECTION_STRING }
      cachesize: 16384
  api:
    # The port at which Clair will report its health status. For example, if Clair is running at
    # https://clair.mycompany.com, the health will be reported at
    # http://clair.mycompany.com:6061/health.
    healthport: 6061
```

```
    port: 6062
    timeout: 900s

    # paginationkey can be any random set of characters. *Must be the same across all Clair
instances*.
    paginationkey:

  updater:
    # interval defines how often Clair will check for updates from its upstream vulnerability databases.
    interval: 6h
  notifier:
    attempts: 3
    renotifyinterval: 1h
    http:
      # QUAY_ENDPOINT defines the endpoint at which Quay is running.
      # For example: https://myregistry.mycompany.com
      endpoint: { QUAY_ENDPOINT }/secscan/notify
      proxy: http://localhost:6063

jwtproxy:
  signer_proxy:
    enabled: true
    listen_addr: :6063
    ca_key_file: /certificates/mitm.key # Generated internally, do not change.
    ca_crt_file: /certificates/mitm.crt # Generated internally, do not change.
    signer:
      issuer: security_scanner
      expiration_time: 5m
      max_skew: 1m
      nonce_length: 32
      private_key:
        type: autogenerated
        options:
          rotate_every: 12h
          key_folder: /clair/config/
          key_server:
            type: keyregistry
            options:
              # QUAY_ENDPOINT defines the endpoint at which Quay is running.
              # For example: https://myregistry.mycompany.com
              registry: { QUAY_ENDPOINT }/keys/


  verifier_proxies:
  - enabled: true
    # The port at which Clair will listen.
    listen_addr: :6060

    # If Clair is to be served via TLS, uncomment these lines. See the "Running Clair under TLS"
    # section below for more information.
    # key_file: /clair/config/clair.key
    # crt_file: /clair/config/clair.crt

    verifier:
      # CLAIR_ENDPOINT is the endpoint at which this Clair will be accessible. Note that the port
```

```
# specified here must match the listen_addr port a few lines above this.
# Example: https://myclair.mycompany.com:6060
audience: { CLAIR_ENDPOINT }

upstream: http://localhost:6062
key_server:
  type: keyregistry
  options:
    # QUAY_ENDPOINT defines the endpoint at which Quay is running.
    # Example: https://myregistry.mycompany.com
    registry: { QUAY_ENDPOINT }/keys/
```

6.2.3. TLS 用の Clair の設定

Clair を TLS で実行するように設定するには、追加の手順をいくつか実行する必要があります。

6.2.3.1. パブリック CA からの証明書の使用

パブリック認証局からの証明書の場合は、以下の手順に従います。

1. Clair のアクセスに使用される DNS 名の TLS 証明書とキーペアを生成します。
2. これらのファイルを clair.crt および clair.key として Clair 設定ディレトリーに配置します。
3. Clair config.yaml の verifier_proxies で、 key_file および crt_file 行のコメントを解除します。

証明書がパブリック CA を使用すると、Clair を実行する準備が整います。独自の認証局を使用している場合は、以下のように Clair をこれを信頼するように設定します。

6.2.3.2. 自己署名 SSL の信頼の設定

自己署名証明書を信頼する ように Docker をセットアップするプロセスと同様に、Clair も証明書を信頼するように設定される必要があります。Docker の設定に使用するものと同じ CA 証明書バンドルを使用して、以下の手順を実行します。

1. Quay レジストリーのセットアップに使用されるものと同じ CA 証明書バンドルの名前を ca.crt に変更します。

2. 以下の例のように、**ca.crt** ファイルが **/etc/pki/ca-trust/source/anchors/** の下にある **Clair** コンテナー内にマウントされていることを確認します。

```
# docker run --restart=always -p 6060:6060 -p 6061:6061 \
  -v /path/to/clair/config/directory:/clair/config \
  -v /path/to/quay/cert/ca.crt:/etc/pki/ca-trust/source/anchors/ca.crt  \
  quay.io/redhat/clair-jwt:v3.3.4
```

これで、**Clair** は **TLS** 証明書のソースを信頼し、それらを使用して **Clair** と **Quay** 間の通信を保護することができます。

6.2.4. Clair データソースの使用

コンテナーイメージをスキャンする前に、**Clair** はコンテナーがビルドされたオペレーティングシステムを判別しようとします。これは、そのイメージ内の特定のファイル名を検索して行います (表 **1** を参照)。**Clair** がオペレーティングシステムを認識したら、特定のセキュリティーデータベースを使用して脆弱性の有無を確認します (表 **2** を参照)。

表6.1 オペレーティングシステムを特定するコンテナーファイル

オペレーティングシステム	OS タイプを特定するファイル
Redhat/CentOS/Oracle	etc/oracle-release etc/centos-release etc/redhat-release etc/system-release
Alpine	etc/alpine-release
Debian/Ubuntu:	etc/os-release usr/lib/os-release etc/apt/sources.list
Ubuntu	etc/lsb-release

Clair がコンテナーをスキャンするために使用するデータソースが表 **2** に表示されています。

注記

各データソースの場所へのアクセスをホワイトリスト化することで Clair が一覧表示されたすべてのデータソースにアクセスできることを確認する必要があります。コードによって動的にビルドされるために完全に完了していない可能性のある一部の URL の最後にワイルドカード文字 (*) を追加する必要がある場合があります。

表6.2 Clair データソースおよび収集されるデータ

データソース	収集されるデータ	ホワイトリストリンク	フォーマット	ライセンス
Debian Security Bug Tracker	Debian 6、7、8、不安定な namespace	https://security-tracker.debian.org/tracker/data/json https://security-tracker.debian.org/tracker	dpkg	Debian
Ubuntu CVE Tracker	Ubuntu 12.04、12.10、13.04、14.04、14.10、15.04、15.10、16.04 namespace	https://git.launchpad.net/ubuntu-cve-tracker http://people.ubuntu.com/~ubuntu-security/cve/%s	dpkg	GPLv2
Red Hat Security Data	CentOS 5、6、7 namespace	https://www.redhat.com/security/data/oval/	rpm	CVRF
Oracle Linux セキュリティーデータ	Oracle Linux 5、6、7 の namespace	https://linux.oracle.com/oval/	rpm	CVRF
Alpine SecDB	Alpine 3.3、3.4、3.5 namespace	https://github.com/alpinelinux/alpine-secdb https://cve.mitre.org/cgi-bin/cvename.cgi?name=	apk	MIT

データソース	収集されるデータ	ホワイトリストリンク	フォーマット	ライセンス
NIST NVD	汎用的な脆弱性メタデータ	https://nvd.nist.gov/feeds/xml/cve/2.0/nvdcve-2.0-%s.xml.gz https://nvd.nist.gov/feeds/xml/cve/2.0/nvdcve-2.0-%s.meta	該当なし	パブリックドメイン
Amazon Linux Security Advisories	Amazon Linux 2018.03, 2 namespaces	amazonaws.com mirror list amazon.com mirror list	rpm	MIT-0

6.2.5. Clair の実行

以下のコマンドを実行して **Clair** を実行します。

```
# docker run --restart=always -p 6060:6060 -p 6061:6061 \
    -v /path/to/clair/config/directory:/clair/config \
    quay.io/redhat/clair-jwt:v3.3.4
```

成功すると、以下のような出力が表示されます。

```
2016-05-04 20:01:05,658 CRIT Supervisor running as root (no user in config file)
2016-05-04 20:01:05,662 INFO supervisord started with pid 1
2016-05-04 20:01:06,664 INFO spawned: 'jwtproxy' with pid 8
2016-05-04 20:01:06,666 INFO spawned: 'clair' with pid 9
2016-05-04 20:01:06,669 INFO spawned: 'generate_mitm_ca' with pid 10
time="2016-05-04T20:01:06Z" level=info msg="No claims verifiers specified, upstream should be
configured to verify authorization"
time="2016-05-04T20:01:06Z" level=info msg="Starting reverse proxy (Listening on ':6060')"
2016-05-04 20:01:06.715037 I | pgsql: running database migrations
time="2016-05-04T20:01:06Z" level=error msg="Failed to create forward proxy: open
/certificates/mitm.crt: no such file or directory"
goose: no migrations to run. current version: 20151222113213
2016-05-04 20:01:06.730291 I | pgsql: database migration ran successfully
2016-05-04 20:01:06.730657 I | notifier: notifier service is disabled
2016-05-04 20:01:06.731110 I | api: starting main API on port 6062.
2016-05-04 20:01:06.736558 I | api: starting health API on port 6061.
2016-05-04 20:01:06.736649 I | updater: updater service is disabled.
2016-05-04 20:01:06,740 INFO exited: jwtproxy (exit status 0; not expected)
```

```
2016-05-04 20:01:08,004 INFO spawned: 'jwtproxy' with pid 1278
2016-05-04 20:01:08,004 INFO success: clair entered RUNNING state, process has stayed up for >
than 1 seconds (startsecs)
2016-05-04 20:01:08,004 INFO success: generate_mitm_ca entered RUNNING state, process has
stayed up for > than 1 seconds (startsecs)
time="2016-05-04T20:01:08Z" level=info msg="No claims verifiers specified, upstream should be
configured to verify authorization"
time="2016-05-04T20:01:08Z" level=info msg="Starting reverse proxy (Listening on ':6060')"
time="2016-05-04T20:01:08Z" level=info msg="Starting forward proxy (Listening on ':6063')"
2016-05-04 20:01:08,541 INFO exited: generate_mitm_ca (exit status 0; expected)
2016-05-04 20:01:09,543 INFO success: jwtproxy entered RUNNING state, process has stayed up for
> than 1 seconds (startsecs)
```

Clair が実行されていることを確認するには、以下のコマンドを実行します。

```
curl -X GET -I http://path/to/clair/here:6061/health
```

200 OK コードが返される場合、Clair は実行されています。

```
HTTP/1.1 200 OK
Server: clair
Date: Wed, 04 May 2016 20:02:16 GMT
Content-Length: 0
Content-Type: text/plain; charset=utf-8
```

Clair と関連するデータベースが実行された後に、変更を有効にするために quay アプリケーションを再起動する必要がある場合があります。

第7章 CLAIR V4 セキュリティースキャンの使用

Clair v4 は、Red Hat Quay で利用可能な次世代 Clair イメージスキャンです。Clair v4 は現在 テクノロジープレビュー としてリリースされています。つまり、実稼働環境での使用はサポートされていません。ただし、Clair イメージスキャン開発の方向を表す Clair v4 をテストすることが推奨されます。

Red Hat Quay リリースに対応するため、現在の Clair v4 リリースイメージは clair:v3.3.4 になります。

7.1. CLAIR V4 について

技術的には、Clair v4 は、Red Hat Quay と共に使用できる、Linux オペレーティング システムのセットに関連付けられたコンテナー イメージの脆弱性スキャンを実行するためのマイクロサービスのセットです。Clair v4 のマイクロサービスは、設計上、企業環境に合わせてコンポーネントを個別に拡張できる拡張性の高い設定で実行することに適しています。

Clair v4 の試用には、これを combo モードで実行することが推奨されます (clair-combo.yaml を参照)。このモードは、説明されているように、すべてのマイクロサービスを 1 つのプロセスとしてまとめます。

Clair v4 でサポートされるセキュリティーデータベースはすべてオンになっています。これらのデータベースには、以下が含まれます。

- Alpine SecDB データベース
- AWS UpdateInfo
- Debian Oval データベース
- Oracle Oval データベース
- RHEL Oval データベース

- SUSE Oval データベース
- Ubuntu Oval データベース

Clair が各種のデータベースでセキュリティーマッピングを行う方法についての詳細は、 ClairCore Severity Mapping を参照してください。

警告

Clair v4 はテクノロジープレビューであるため、100% 正確なレポートを期待することはできません。脆弱性の結果の表示は v2 とは異なる可能性があることに留意してください。今後 Clair v4 はさらに多くの結果を生成します。

次に、既存の Red Hat Quay + Clair v2 環境と共に Clair v4 を使用する手順を説明します。

7.2. CLAIR V4 の設定

Clair v4 の試用のために、実行中の Clair v2 インスタンスで Red Hat Quay クラスターを立ち上げます。次に、以下の手順を使用してこれと共に Clair v4 を実行します。以下は、AWS クラウドで OpenShift v4.2 以降のクラスターでこれを実行する方法です。

1. 現在のプロジェクトを Red Hat Quay が実行されているプロジェクトの名前に設定します。以下は例になります。

```
$ oc project quay-enterprise
```

2. Clair v4 用の Postgres デプロイファイル (例: clairv4-postgres.yaml) を以下のように作成します。

clairv4-postgres.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: clairv4-postgres
  namespace: quay-enterprise
  labels:
    quay-component: clairv4-postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-component: clairv4-postgres
  template:
    metadata:
      labels:
        quay-component: clairv4-postgres
    spec:
      volumes:
        - name: postgres-data
          persistentVolumeClaim:
            claimName: clairv4-postgres
      containers:
        - name: postgres
          image: postgres:11.5
          imagePullPolicy: "IfNotPresent"
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: "postgres"
            - name: POSTGRES_DB
              value: "clair"
            - name: POSTGRES_PASSWORD
              value: "postgres"
            - name: PGDATA
              value: "/etc/postgres/data"
          volumeMounts:
            - name: postgres-data
              mountPath: "/etc/postgres"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: clairv4-postgres
  labels:
    quay-component: clairv4-postgres
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "5Gi"
    volumeName: "clairv4-postgres"
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: clairv4-postgres
  labels:
    quay-component: clairv4-postgres
spec:
  type: ClusterIP
  ports:
    - port: 5432
      protocol: TCP
      name: postgres
      targetPort: 5432
  selector:
    quay-component: clairv4-postgres
```

3. 以下のように postgres データベースをデプロイします。

```
$ oc create -f ./clairv4-postgres.yaml
```

4. Clair v4 に使用する Clair の config.yaml ファイルを作成します。例を以下に示します。

config.yaml

```
introspection_addr: :8089
http_listen_addr: :8080
log_level: debug
indexer:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
password=postgres sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
password=postgres sslmode=disable
  max_conn_pool: 100
  run: ""
  migrations: true
  indexer_addr: clair-indexer
# tracing and metrics
trace:
  name: "jaeger"
  probability: 1
  jaeger:
    agent_endpoint: "localhost:6831"
```

```
    service_name: "clair"
metrics:
  name: "prometheus"
```

5. **Clair の config.yaml からシークレットを作成します。**

```
$ oc create secret generic clairv4-config-secret --from-file=./config.yaml
```

6. **Clair v4 デプロイメントファイル (例: clair-combo.yaml) を作成し、必要に応じてこれを変更します。**

clair-combo.yaml

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    quay-component: clair-combo
  name: clair-combo
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-component: clair-combo
  template:
    metadata:
      labels:
        quay-component: clair-combo
    spec:
      containers:
        - image: quay.io/redhat/clair:v3.3.4 (1)
          imagePullPolicy: IfNotPresent
          name: clair-combo
          env:
            - name: CLAIR_CONF
              value: /clair/config.yaml
            - name: CLAIR_MODE
              value: combo
          ports:
            - containerPort: 8080
              name: clair-http
              protocol: TCP
            - containerPort: 8089
              name: clair-intro
```

```
            protocol: TCP
          volumeMounts:
            - mountPath: /clair/
              name: config
      imagePullSecrets:
        - name: redhat-pull-secret
      restartPolicy: Always
      volumes:
        - name: config
          secret:
            secretName: clairv4-config-secret
---
apiVersion: v1
kind: Service
metadata:
  name: clairv4 2
  labels:
    quay-component: clair-combo
spec:
  ports:
    - name: clair-http
      port: 80
      protocol: TCP
      targetPort: 8080
    - name: clair-introspection
      port: 8089
      protocol: TCP
      targetPort: 8089
  selector:
    quay-component: clair-combo
  type: ClusterIP
```

1

イメージを最新の clair イメージ名とバージョンに変更します。

2

Service が clairv4 に設定されると、Clair v4 のスキャナーエンドポイントは、後に http://clairv4 として SECURITY_SCANNER_V4_ENDPOINT の Red Hat Quay config.yaml に入力されます。

7. 以下のように Clair v4 デプロイメントを作成します。

```
$ oc create -f ./clair-combo.yaml
```

8.

Red Hat Quay 配置の config.yaml ファイルを変更して、最後に以下のエントリーを追加します。

```
...
FEATURE_SECURITY_SCANNER: true
SECURITY_SCANNER_V4_ENDPOINT: http://clairv4 (1)
SECURITY_SCANNER_V4_NAMESPACE_WHITELIST: (2)
  - "clairv4-org"
  - "foo-org"
```

(1) Clair v4 サービスエンドポイントを特定します。

(2) Clair v4 スキャンを使用する Red Hat Quay クラスターの namespace (組織およびユーザー) に clair4-org および foo-org を置き換えます。

9. 変更された config.yaml を、そのファイルを含むシークレットに再デプロイします (例: quay-enterprise-config-secret)。

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

10. 新しい config.yaml を有効にするには、Red Hat Quay の Pod を再起動する必要があります。quay-app Pod を削除するだけで、更新された設定を持つ Pod がデプロイされます。

この時点で、namespace ホワイトリストで特定される組織内のイメージが Clair v4 によってスキャンされます。

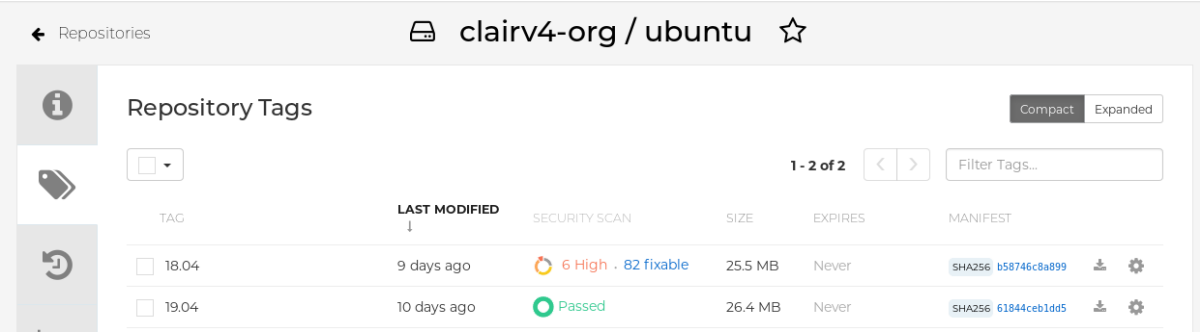
7.3. CLAIR V4 の使用

Clair v4 によって収集される脆弱性情報を表示するユーザーインターフェイスは、基本的に Clair v2 で使用するものと同じです。

1. Red Hat Quay クラスターにログインし、Clair v4 スキャンを設定している組織を選択します。

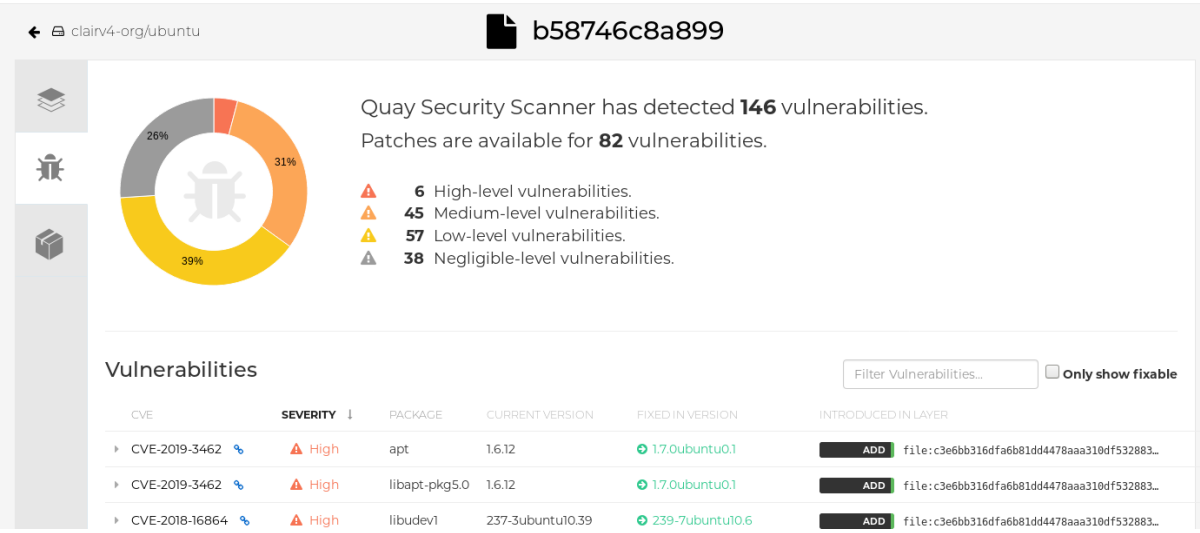
2. イメージを保持する組織からリポジトリーを選択し、左側のナビゲーションから Tags を選

択します。以下の図は、スキャンされた 2 つのイメージがあるリポジトリーの例を示しています。



3.

脆弱性が見つかった場合は、イメージの Security Scan 列の下で選択し、すべての脆弱性または修正可能な脆弱性のいずれかを確認します。以下の図は、検出されるすべての脆弱性の情報を示しています。



第8章 コンテナーセキュリティー OPERATOR での POD イメージのスキャン

Container Security Operator (CSO) を使用すると、既知の脆弱性について OpenShift(4.2 以降) および他の Kubernetes プラットフォームで実行されるアクティブな Pod に関連付けられたコンテナーイメージをスキャンできます。CSO:

- すべての namespace または指定された namespace の Pod に関連付けられたコンテナーを監視します。
- イメージのレジストリーがイメージスキャンをサポートしている場合 (例: Clair スキャンを含む Quay レジストリー)、脆弱性の情報についてコンテナーの出所となったコンテナーレジストリーをクエリーします。
- Kubernetes API の ImageManifestVuln オブジェクトを使用して脆弱性を公開します。

この手順を使用すると、CSO は marketplace-operators namespace にインストールされ、OpenShift クラスターのすべての namespace で利用可能になります。

注記

CSO を Kubernetes にインストールする手順を表示するには、Container Security OperatorHub.io ページから Install ボタンを選択します。

8.1. OPENSHIFT での CSO の実行

OpenShift で CSO の使用を開始するには、以下を実行します。

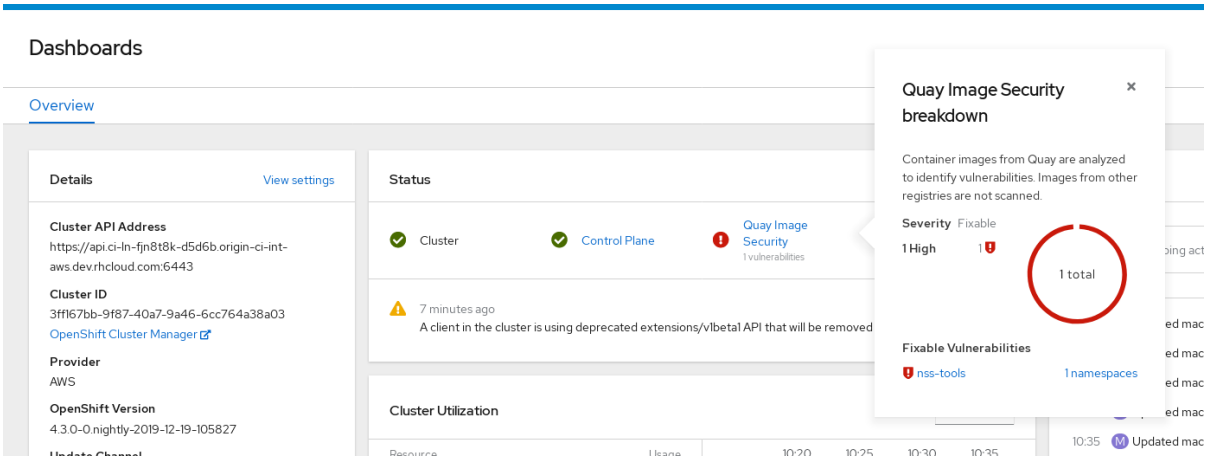
1. Operators → OperatorHub(Security を選択します) に移動し、利用可能な Container Security Operator を表示します。
2. Container Security Operator を選択し、Install を選択して Create Operator Subscription ページに移動します。
3. 設定 (デフォルトではすべての namespace および自動承認ストラテジー) をチェックし、Subscribe を選択します。Container Security は、Installed Operators 画面に数分後に表

示されます。

4. オプションで、カスタム証明書を CSO に追加できます。以下の例では、現在のディレクトリーに quay.crt という名前の証明書を作成します。その後、次のコマンドを実行して、CSO に証明書を追加します (新しい証明書を有効にするために、Operator Pod を再起動します)。

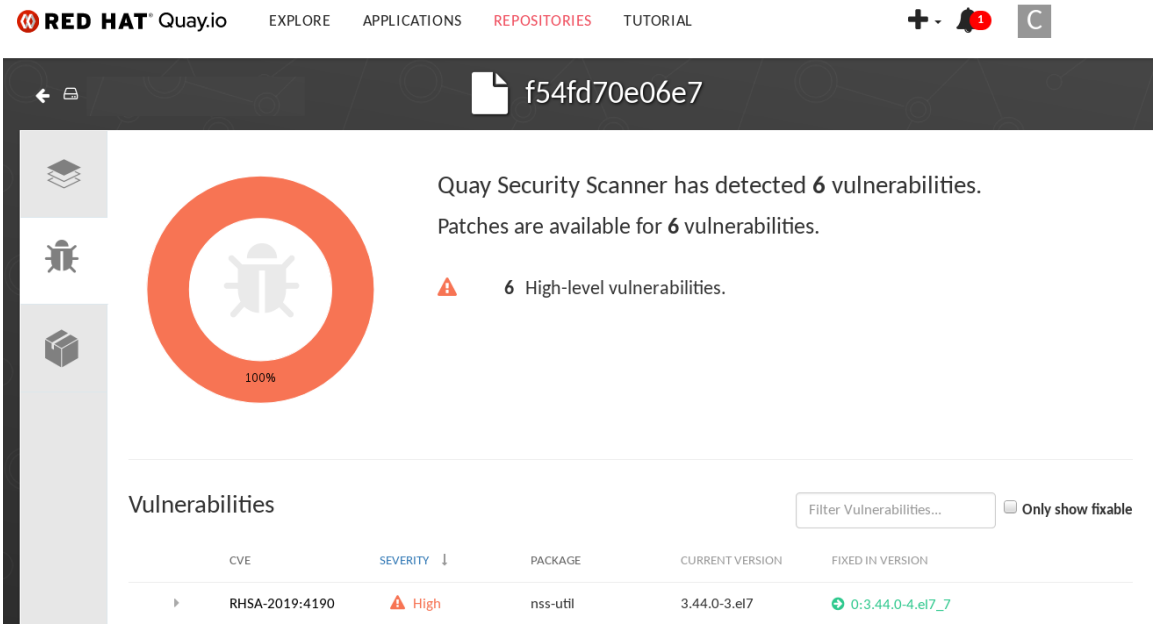
```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n
openshift-operators
```

5. OpenShift Dashboard を開きます (Home → Dashboards)。Image Security へのリンクが status セクションに表示され、これまでに見つかった脆弱性の数の一覧が表示されます。以下の図のように、リンクを選択してセキュリティーの内訳を表示します。

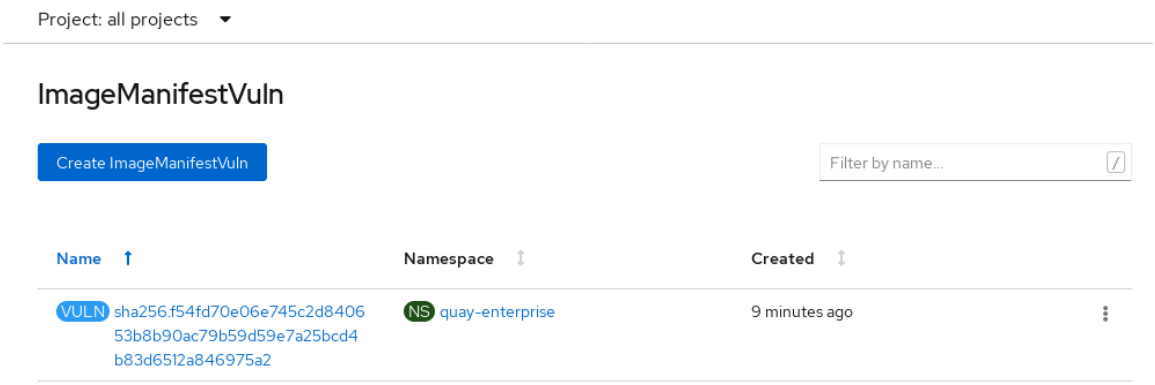


6. この時点で、検出された脆弱性をフォローするために以下の 2 つのいずれかの操作を実行できます。

- 脆弱性へのリンクを選択します。コンテナーを取得したコンテナーレジストリー、Red Hat Quay または他のレジストリーにアクセスし、脆弱性についての情報を確認できます。以下の図は、Quay.io レジストリーから検出された脆弱性の例を示しています。



- namespaces リンクを選択し、ImageManifestVuln 画面に移動します。ここでは、選択されたイメージの名前、およびイメージが実行されているすべての namespace を確認できます。以下の図は、特定の脆弱なイメージが 2 つの namespace で実行されていることを示しています。



この時点では、脆弱性のあるイメージや、イメージの脆弱性を解決するために必要なこと、およびイメージが実行されたすべての namespace を確認できます。以下を実行することができます。

- 脆弱性を修正する必要のあるイメージを実行しているユーザーに警告します。

- (イメージが置かれている Pod を起動したデプロイメントまたは他のオブジェクトを削除して) イメージの実行を停止します。

Pod を削除すると、Dashboard で脆弱性のある状態がリセットされるまで数分かかる場合があります。

8.2. CLI でのイメージ脆弱性のクエリー

コマンドラインからセキュリティーに関する情報をクエリーできます。検出された脆弱性を照会するには、次のように入力します。

```
$ oc get vuln --all-namespaces
NAMESPACE     NAME              AGE
default       sha256.ca90...    6m56s
skynet        sha256.ca90...    9m37s
```

特定の脆弱性の詳細を表示するには、脆弱性の 1 つを特定し、その名前空間および describe オプションを指定します。以下の例は、イメージに脆弱性のある RPM パッケージが含まれるアクティブなコンテナーを示しています。

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
Name:         sha256.ac50e3752...
Namespace:    quay-enterprise
...
Spec:
  Features:
    Name:            nss-util
    Namespace Name:  centos:7
    Version:         3.44.0-3.el7
    Versionformat:   rpm
    Vulnerabilities:
      Description: Network Security Services (NSS) is a set of libraries...
```

第9章 BRIDGE OPERATOR で RED HAT QUAY を OPENSHIFT に統合

Quay Bridge Operator を使用して、OpenShift の統合コンテナーレジストリーを Red Hat Quay レジストリーに置き換えることができます。これを実行することにより、統合 OpenShift レジストリーは、強化された RBAC(ロールベースアクセス制御) 機能を備えた可用性の高い、エンタープライズレベルの Red Hat Quay レジストリーになります。

Bridge Operator の主な目的は、統合 OpenShift レジストリーの機能を新規の Red Hat Quay レジストリーに複製することです。この Operator で有効にされている機能には以下が含まれます。

- OpenShift namespace を Red Hat Quay の組織として同期します。
  - 各デフォルト namespace サービスアカウント用のロボットアカウントの作成
  - 作成された各ロボットアカウントのシークレットの作成 (各ロボットシークレットをマウント可能なイメージプルシークレットとしてサービスアカウントに関連付ける)
  - OpenShift ImageStream の Quay リポジトリーとしての同期
- Red Hat Quay への出力に ImageStream を使用した新規ビルドの自動再作成
- ビルド完了後の ImageStream タグの自動インポート

Quay Bridge Operator でこの手順を使用すると、Red Hat Quay と OpenShift クラスター間の双方向通信が有効になります。

警告

Quay Bridge Operator から同じ Red Hat Quay インスタンスをポイントする OpenShift Container Platform クラスターを複数指定することはできません。複数指定した場合は、2 つのクラスターに同じ名前の namespace を作成できなくなります。

9.1. QUAY BRIDGE OPERATOR の実行

9.1.1. 前提条件

Bridge Operator をセットアップする前に、以下を有効にしておく必要があります。

- スーパーユーザーパーミッションを持つ既存の Red Hat Quay 環境
- クラスター管理者パーミッションのある Red Hat OpenShift Container Platform 環境 (4.2 以降のバージョンを推奨)
- OpenShift コマンドラインツール (oc コマンド)

9.1.2. OpenShift Red Hat Quay のセットアップおよび設定

Red Hat Quay および OpenShift 設定の両方が必要です。

9.1.2.1. Red Hat Quay のセットアップ

専用の Red Hat Quay 組織を作成し、その組織内で作成する新規アプリケーションから、OpenShift の Quay Bridge Operator で使用する OAuth トークンを生成します。

1. スーパーユーザーアクセスのあるユーザーとして Red Hat Quay にログインし、外部アプリケーションを設定する組織を選択します。

2. 左側のナビゲーションにある **Applications** を選択します。

3. **Create New Application** を選択し、新規アプリケーションの名前を入力します (例: **openshift**)。

4. 新しいアプリケーションが表示されたら、これを選択します。

5. 左側のナビゲーションで **Generate Token** を選択し、新規 **OAuth2** トークンを作成します。

6. すべてのチェックボックスを選択し、統合に必要なアクセスを付与します。

7. 割り当てられたパーミッションを確認してから **Authorize Application** を選択してこれを確定します。

8. 次のセクションで使用する生成されたアクセストークンをコピーして保存します。

9.1.2.2. OpenShift セットアップ

Quay Bridge Operator 用に **OpenShift** を設定するには、以下を含む複数の手順が必要になります。

- **OpenShift シークレットの作成: Quay** の初期段階で作成された **OAuth** トークンを使用して、**OpenShift** シークレットを作成します。

- **MutatingWebhookConfiguration** サポートの追加: **Red Hat Quay** の **OpenShift** への統合をサポートするには、新しいビルド要求をインターセプトして、**OpenShift** の統合レジストリーではなく **Red Hat Quay** をターゲットに出力を変更できるようにする必要があります。

OpenShift の通常のビルドプロセスの一部として実行される **API** 要求の動的インターセプションのサポートは、**MutatingWebhookConfiguration** によって容易になります。**MutatingWebhookConfiguration** は、特定の **API** 要求が受信される際に、**OpenShift** のプロジェクト内で実行中の **API** を呼び出すことを可能にします。

Kubernetes では、クラスターの認証局を使用する証明書を使用して SSL で Webhook エンドポイントのセキュリティーを保護する必要があります。OpenShift はクラスターによって署名される証明書の生成をサポートします。

1. OpenShift oc コマンドラインツールを使用して、クラスター管理者として OpenShift にログインします。

2. openshift-operators などの、使用する OpenShift namespace を選択するか、または新規 namespace を作成します。

3. OpenShift シークレットを作成し、<access_token> を先に Red Hat Quay から取得したアクセストークンに置き換えます。例えば、これはあなたの<access_token>を使って quay-integration というシークレットを token というキーで作成します。

```
$ oc create secret generic quay-integration --from-literal=token=<access_token>
```

その結果、新たに作成された秘密鍵と証明書が、指定された秘密の中に配置されます。シークレットは Operator の Deployment に宣言されるように Operator 内の適切な場所にマウントされます。

4. Operator の Webhook エンドポイントのサービスを作成します。

quay-webhook.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: quay-bridge-operator
  name: quay-bridge-operator
  namespace: openshift-operators
spec:
  ports:
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8443
  selector:
    name: quay-bridge-operator
  sessionAffinity: None
  type: ClusterIP
```

5. 次のように Webhook サービスを作成します。

```
$ oc create -f quay-webhook.yaml
```

6. webhook-create-signed-cert.sh スクリプトをダウンロードして、Kubernetes 認証局で署名された証明書を生成するために使用できるようにします。

7. 以下のコマンドを実行して、証明書を要求します。

```
$ ./webhook-create-signed-cert.sh --namespace openshift-operators \
  --secret quay-bridge-operator-webhook-certs \
  --service quay-bridge-operator
```

8. 以下のコマンドを実行して CA を取得し、その結果を 1 行にまとめて MutatingWebhookConfiguration リソースに入力できるようにします。

```
$ oc get configmap -n kube-system \
  extension-apiserver-authentication \
  -o=jsonpath='{.data.client-ca-file}' | base64 | tr -d '\n'
```

9. 以下の MutatingWebhookConfiguration の YAML で、${CA_BUNDLE}変数を置き換えてください。

quay-mutating-webhook.yaml

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
metadata:
  name: quay-bridge-operator
webhooks:
  - name: quayintegration.redhatcop.redhat.io
    clientConfig:
      service:
        namespace: openshift-operators
        name: quay-bridge-operator
        path: "/admissionwebhook"
      caBundle: "${CA_BUNDLE}" (1)
    rules:
```

```
  - operations:  [ "CREATE" ]
    apiGroups: [ "build.openshift.io" ]
    apiVersions: ["v1" ]
    resources: [ "builds" ]
  failurePolicy: Fail
```

1

${CA_BUNDLE} を直前の手順の出力に置き換えます。${CA_BUNDLE} の置き換えとしてコピーし、貼り付ける 1 行は長い行で表示されます。

10. 以下のように MutatingWebhookConfiguration を作成します。

```
$ oc create -f quay-mutating-webhook.yaml
```

オペレータが動作するまでは、MutatingWebhookConfiguration が呼び出す Web サーバーが利用できず、オブジェクトを etcd に永続化するために適切な is レスポンスが必要となるため、ビルドに対する新しいリクエストは失敗します。

11. OpenShift コンソールに移動し、以下のように Quay Bridge Operator をインストールします。

- OperatorHub を選択し、Quay Bridge Operator を検索します。
- Install を選択します。
- Installation Mode (all namespaces)、 Update Channel、および Approval Strategy (Automatic or Manual) を選択します。
- Subscribe を選択します。

12. QuayIntegration というカスタムリソース (CR) を作成します。以下は例になります。

quay-integration.yaml

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayIntegration
metadata:
  name: example-quayintegration
spec:
  clusterID: openshift 1
  credentialsSecretName: openshift-operators/quay-integration 2
  quayHostname: https://<QUAY_URL> 3
  whitelistNamespaces: 4
   - default
  insecureRegistry: false 5
```

1

clusterID の値はエコシステム全体で一意である必要があります。この値はオプションであり、デフォルトは openshift に設定されます。

2

credentialsSecretName の場合、openshift-operators/quay-integration を namespace の名前および先に作成したトークンが含まれるシークレットに置き換えます。

3

QUAY_URL を Red Hat Quay インスタンスのホスト名に置き換えます。

4

whitelistNamespaces は任意です。これが使用されない場合、Bridge Operator は openshift の接頭辞が付けられたプロジェクトを除き、すべての namespace を Red Hat Quay に同期します。この例では、ホワイトリストに入れられた namespace (デフォルト) で Red Hat Quay 組織が関連付けられます。ここで任意の namespace を使用します。

5

Quay が自己署名証明書を使用している場合、プロパティー insecureRegistry: true を設定します。

その結果、Red Hat Quay 内の組織は、OpenShift の関連 namespace 用に作成されます。

13. 以下のように **QuayIntegration** を作成します。

```
$ oc create -f quay-integration.yaml
```

この時点で **Quay** の統合リソースが作成され、**OpenShift** クラスターと **Red Hat Quay** インスタンスがリンクされます。

作成済みのホワイトリストに入れられた **namespace** には **Red Hat Quay** 組織が含まれるはずです。**oc new-app** などのコマンドを使用して、その **namespace** で新規アプリケーションを作成する場合、内部レジストリーを使用する代わりに、作成された新しい **Red Hat Quay** リポジトリーが表示されます。

第10章 RED HAT QUAY でのリポジトリーのミラーリング

Red Hat Quay リポジトリーミラーリングを使用すると、外部コンテナーレジストリー (またはローカルレジストリー) からローカルの Red Hat Quay クラスターにイメージをミラーリングできます。リポジトリーミラーリングを使用すると、リポジトリー名とタグに基づいてイメージを Red Hat Quay に同期できます。

10.1. リポジトリーミラーリングの概要

リポジトリーミラーリングが有効にされた Red Hat Quay クラスターから、以下を実行できます。

- 外部レジストリーからミラーリングするリポジトリーを選択する。
- 外部レジストリーにアクセスするための認証情報を追加する。
- リポジトリーが同期される間隔を設定する。
- 同期する特定のコンテナーイメージリポジトリー名とタグを特定する。
- 同期の現在の状態を確認する。

リポジトリーのミラーリングでは、複数の異なるレジストリー間で、特定のコンテンツのサブセットを選択したデータセンター、クラスターまたはリージョンに対してミラーリングします。一方、Georeplication は、ローカライズされたストレージからコンテナーイメージを提供するために単一のグローバルに分散される Red Hat Quay を提供します。以下に、コンテンツを共有する 2 つの方法はの違いについて示します。

表10.1 Red Hat Quay リポジトリーミラーリング vs.ジオレプリーション

機能と性能	Geo レプリケーション	リポジトリーのミラーリング
設計されている機能	共有される、グローバルレジストリー	個別の異なるレジストリー
レプリケーションまたはミラーリングが完了しないとどうなるか？	リモートコピーが使用される (スピードが遅くなる)。	イメージは提供されない。

両方のリージョンのすべてのストレージバックエンドへのアクセスが必要か？	Yes (すべての Red Hat Quay ノード)	No (個別のストレージ)
ユーザーは両方のサイトから同じリポジトリーにイメージをプッシュできるか？	Yes	No
すべてのレジストリーのコンテンツと設定はすべてのリージョンで同一か (共有データベース)?	Yes	No
ユーザーはミラーリングする個別の namespace またはリポジトリーを選択できるか？	No (デフォルト)	Yes
ユーザーはフィルターを同期ルールに適用できるか？	No	Yes

以下は、Red Hat Quay リポジトリーミラーリングを使用する際のヒントです。

- **リポジトリーのミラーリングでは、リポジトリー全体をミラーリングしたり、同期するイメージを選択的に制限したりすることができます。同期するイメージは、タグのコンマ区切りの一覧、タグの範囲、または正規表現やグロブ (glob) でタグを特定する他の方法で制限できます。**

- **ミラーリングされたリポジトリーとして設定されると、そのリポジトリーに他のイメージを手動で追加することはできません。**

- **ミラーリングされたリポジトリーは設定するリポジトリーとタグをベースとするため、そのリポジトリー/タグのペアで表されるコンテンツのみを保持します。つまり、リポジトリーの一部のイメージがこれ以上一致しないようにタグを変更すると、これらのイメージは削除されます。**

- **指定されたロボットのみがイメージをミラーリングされたリポジトリーにプッシュでき、これはリポジトリーに設定されたロールベースのアクセス制御パーミッションよりも優先されます。**

- **ミラーリングされたリポジトリーを使用すると、ユーザーはリポジトリーからイメージをプルできますが (読み取りパーミッションが付与される)、イメージをリポジトリーにプッシュす**

ることはできません。

- ミラーリングされたリポジトリーの設定の変更は、作成するミラーリングされたリポジトリーについての **Repositories** ページの **Mirrors** タブから行われます。

- イメージは設定される間隔で同期されますが、随時必要に応じて同期することもできます。

10.2. 前提条件

リポジトリーミラーリングを使用するには、**Red Hat Quay** 設定画面からリポジトリーミラーリングを有効にし、リポジトリーミラーリングワーカーを起動する必要があります。このサービスの起動方法については、**Red Hat Quay** デプロイメントガイドを参照してください。

- Deploy Red Hat Quay - Basic

- Deploy Red Hat Quay - High Availability

- Deploy Red Hat Quay on OpenShift

以下のセクションで説明する手順は、リポジトリーミラーリングサービスが実行されており、**Red Hat Quay** クラスターでリポジトリーミラーリングが有効にされていることを前提としています。

10.3. ミラーリングされたリポジトリーの作成

外部コンテナーレジストリーから外部リポジトリーをミラーリングするには、以下を実行します。

1. **Red Hat Quay** レジストリーにログインします。

2. ミラーリングされたリポジトリーのイメージをプルするためのロボットアカウントを作成します。

   - 右上のドロップダウンメニューから **Account Settings** を選択します。

- 左側の列の **Robot Accounts** ボタンを選択します。
- **Create Robot Account** を選択します。
- ロボットアカウントの名前および説明を追加し、**Create robot account** を選択します。
- 追加しているミラーリングされたリポジトリーは存在していないため、**Close** を選択します。
- 一覧から **ROBOT ACCOUNT NAME** を選択します。
- プロンプトが出されたら、ミラーリングするリポジトリーの外部レジストリーにアクセスするためにロボットが必要とする認証情報を追加し、**Credentials** ウィンドウを閉じます。

3. **REPOSITORIES** を選択します。
4. **Create New Repository** を選択し、これに名前を付けます。
5. リポジトリー名を入力し、**Public** または **Private** を選択して、**Create Repository** を選択します。
6. **Settings** ボタンを選択し、リポジトリーの状態を **MIRROR** に変更します。
7. 新規ポジトリーを開き、左側の列の **Mirroring** ボタンを選択します。
8. フィールドに入力し、新規リポジトリーでミラーリングするリポジトリーを特定します。

- **Registry URL:** ミラーリングするコンテナーレジストリーの場所。

- **User or Organization:** 通常は、ミラーリングするコンテンツに関連付けられているアカウント名。例えば、イメージ **registry.example.com/jsmith/myimage:latest** の場合、**jsmith** がここに入力されます。

- **Repository Name:** イメージのセットの名前を識別する名前。例えば、イメージ **registry.example.com/jsmith/myimage:latest** の場合、**myimage** がここに入力されます。

- **Sync Interval:** デフォルトで **24** 時間ごとの同期に設定されます。これは時間または日に基づいて変更できます。

- **Robot User:** ミラーリングを実行するために先に作成したロボットアカウントを選択します。

- **Username:** ミラーリングするリポジトリーを保持する外部レジストリーにログインするためのユーザー名。

- **Password:** ユーザー名に関連付けられたパスワード。パスワードにはエスケープ文字 (\) を必要とする文字を含めることができないことに注意してください。

- **Start Date:** ミラーリングが開始する日付。現在の日時がデフォルトで使用されます。

- **Verify TLS:** 外部レジストリーの信頼性を確認する必要がある場合にこのボックスにチェックを付けます。たとえば、自己署名証明書を使用してテストするために **Red Hat Quay** を設定する場合や、証明書を使用しない場合は、このボックスのチェックを外します。

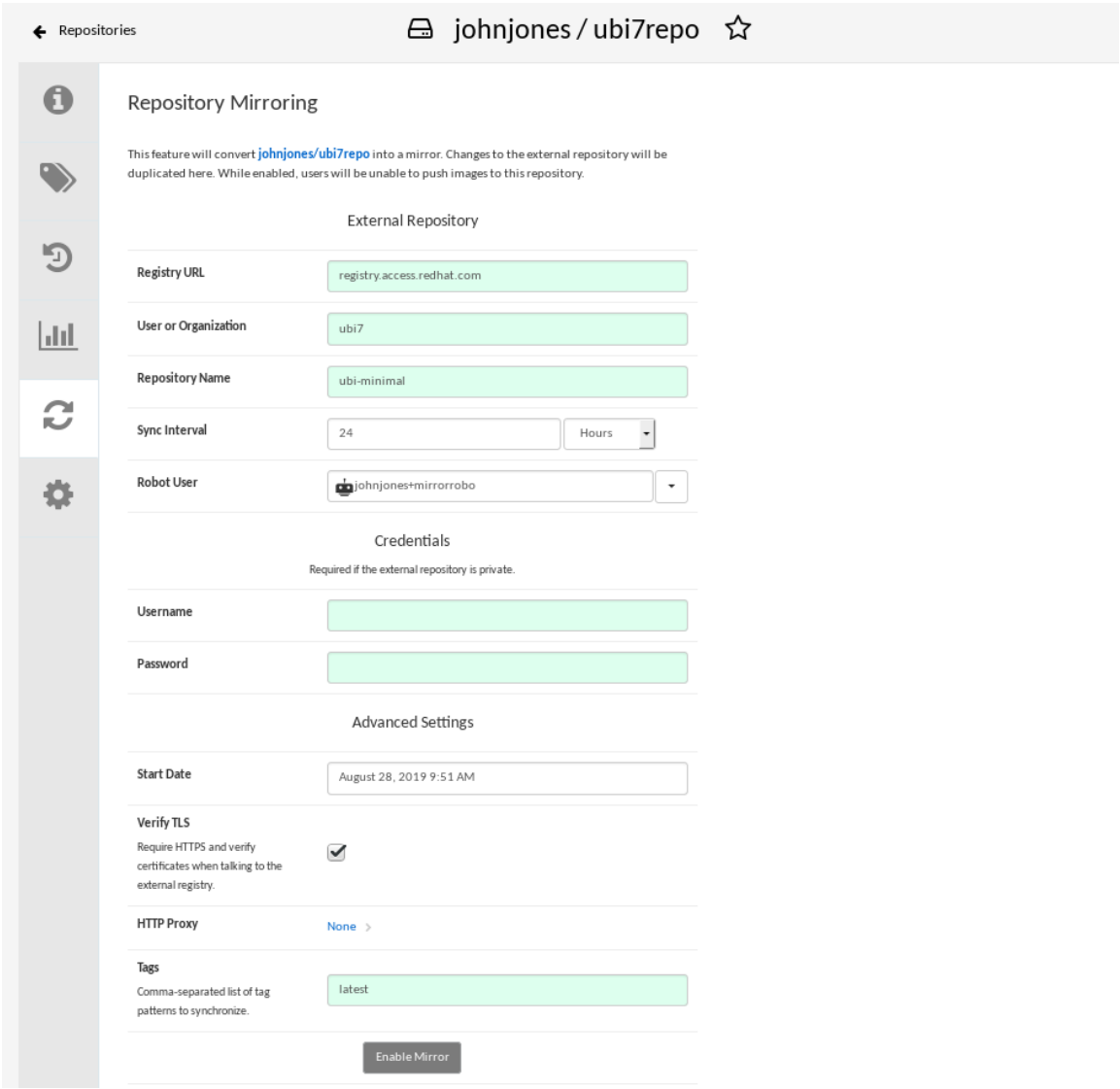
- **HTTP Proxy:** リモートサイトへのアクセスに必要なプロキシーサーバーを特定します (必要な場合)。

- **Tags:** このフィールドは必須です。個別のタグまたはタグパターンのコンマ区切りの一覧を入力できます。(詳細は、タグパターンのセクションを参照してください)。

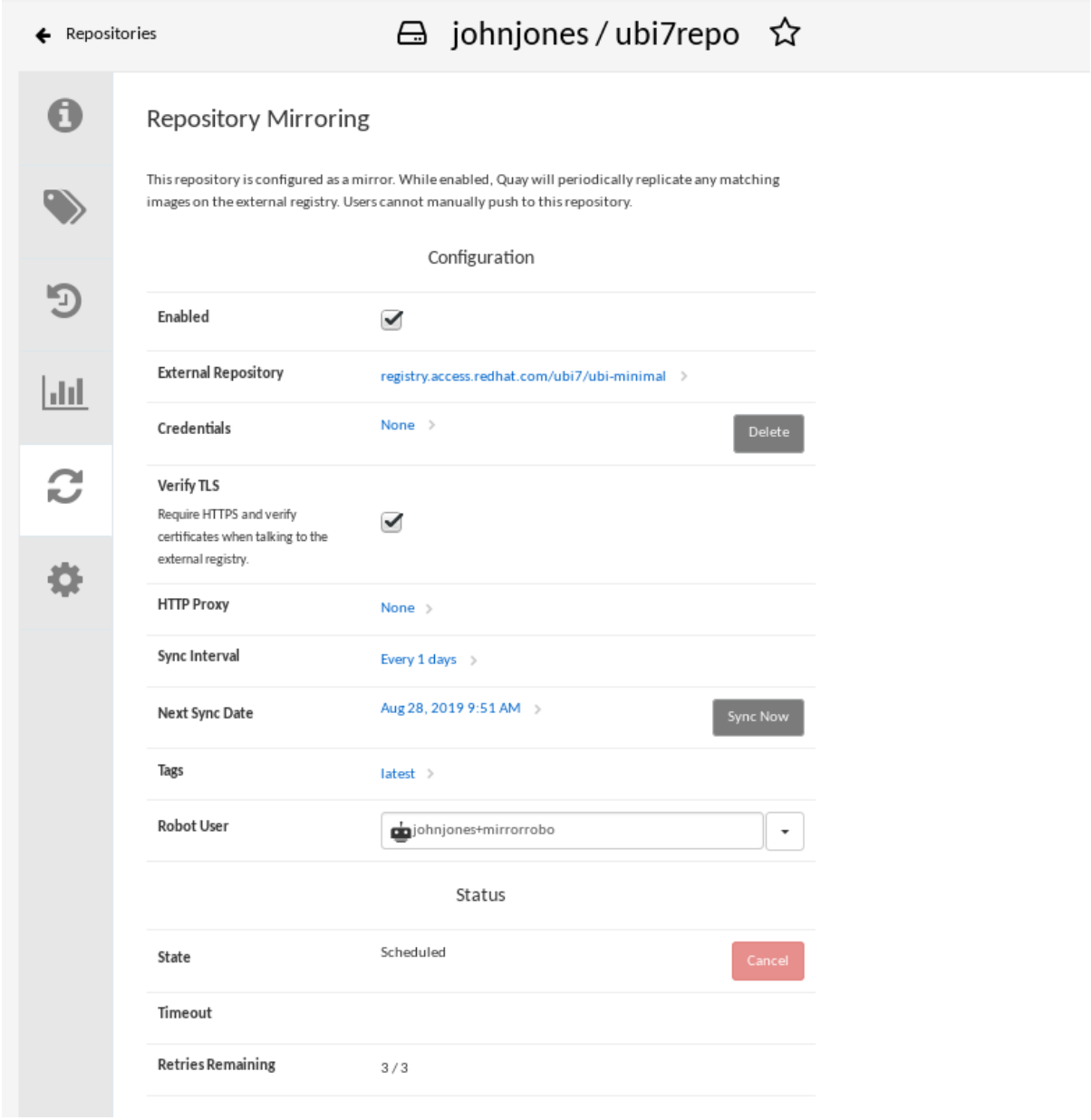
注記

少なくとも 1 つのタグを明示的に入力する (タグパターンではない) か、または、タグの **latest** がリモートリポジトリーに存在する必要があります。これは、**Quay** でミラーリングに使用する指定の一覧との比較用にリモートリポジトリーでタグ一覧を取得するために必要です。

以下は、完了した **Repository Mirroring** 画面の例です。



9. **Enable Mirror** ボタンを選択します。以下は、作成されるリポジトリーミラーリングページです。



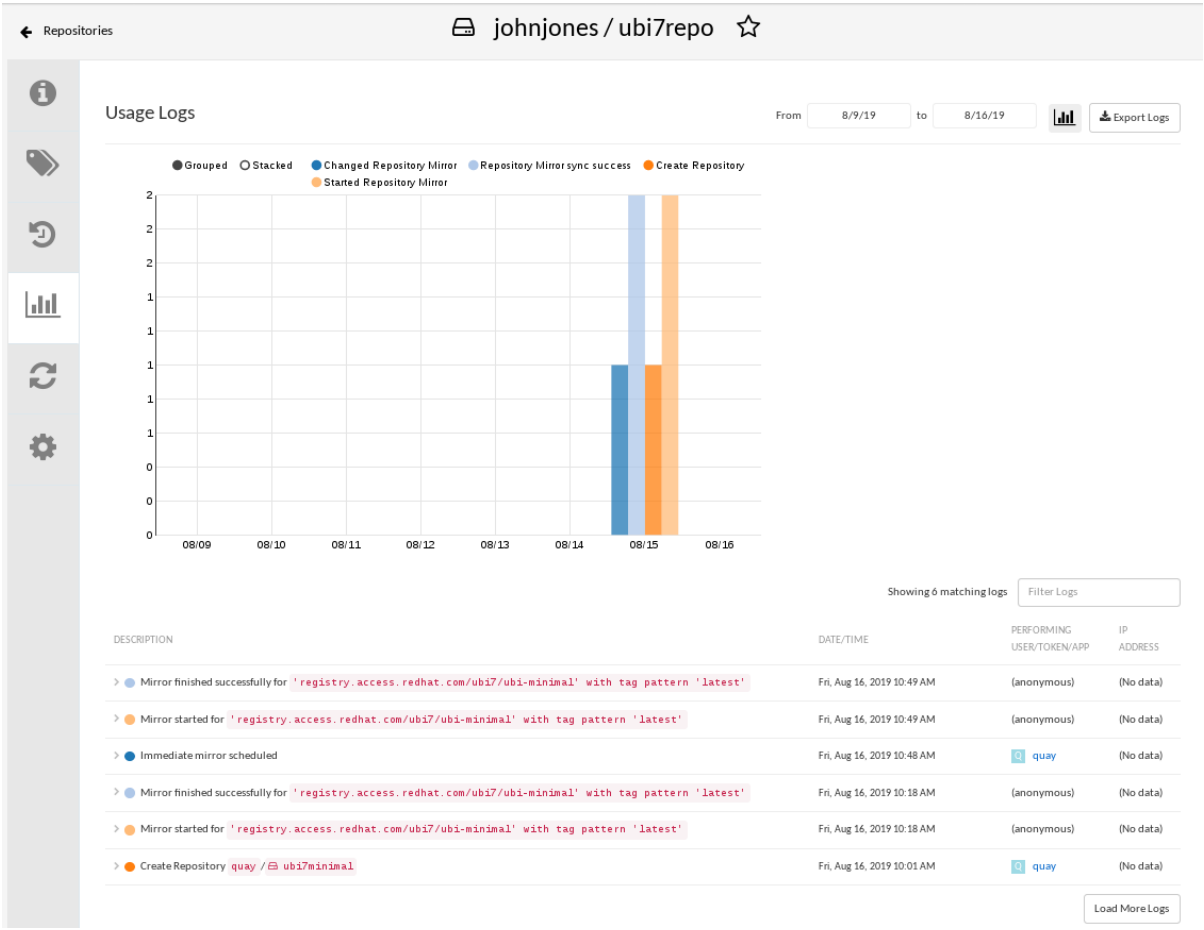
後でこのページに戻り、これらの設定を変更することができます。

10.4. ミラーリングされたリポジトリーの使用

ミラーリングされたリポジトリーの作成後は、そのリポジトリーを使用する複数の方法を実行できます。**Repositories** ページからミラーリングされたリポジトリーを選択し、以下のいずれかを行います。

- リポジトリーの有効化/無効化: 左列の **Mirroring** ボタンを選択してから **Enabled** チェックボックスを切り替え、一時的にリポジトリーを有効または無効にします。

ミラーログの確認: ミラーリングされたリポジトリーが適切に機能していることを確認するために、ミラーログを確認できます。これを実行するには、左側の列の **Usage Logs** ボタンを選択します。以下に例を示します。



- 今すぐミラー同期を実行する: リポジトリーのイメージをすぐに同期するには、**Sync Now** ボタンを選択します。

- 認証情報の変更: ユーザー名およびパスワードを変更するには、**Credentials** 行で **DELETE** を選択します。次に、**None** を選択し、プロンプトが出されたら外部レジストリーにログインするために必要なユーザー名およびパスワードを追加します。

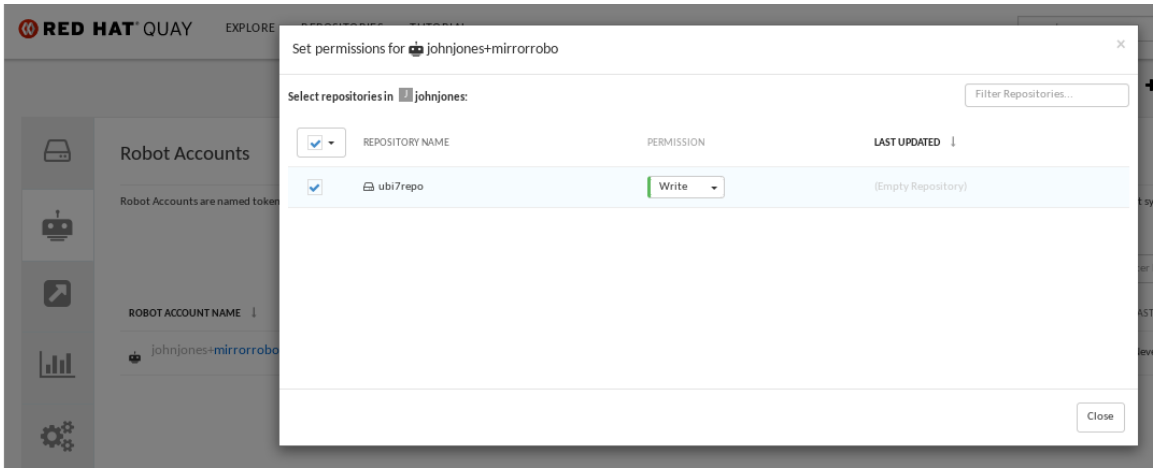
- ミラーリングの取り消し: ミラーリングを停止する (現在利用できるイメージを維持しながら新しいイメージが同期されないようにする) には、**CANCEL** ボタンを選択します。

- ロボットパーミッションの設定: **Red Hat Quay** ロボットアカウントには、外部リポジトリーにアクセスするために認証情報を保持するトークンが指定されます。ロボットに認証情報

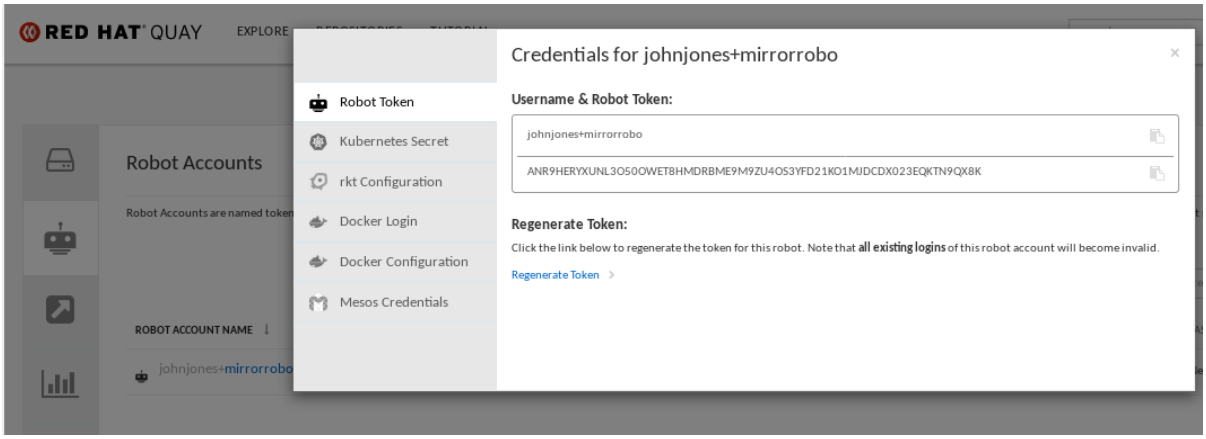
を割り当てることで、そのロボットを同じ外部レジストリーにアクセスする必要のある複数のミラーリングされたリポジトリーで使用することができます。

Account Settings に移動し、左側の列の Robot Accounts アイコンを選択して、既存のロボットをリポジトリーに割り当てることができます。ロボットアカウントには、REPOSITORIES 列にあるリンクを選択します。ポップアップウィンドウから、以下を行うことができます。

- そのロボットに割り当てられているリポジトリーを確認します。
- この図に示されている PERMISSION フィールドから対象のロボットに読み取り、書き込み、または管理者権限を割り当てます。



- ロボット認証情報の変更: ロボットは、Kubernetes シークレット、Docker ログイン情報、および Mesos バンドルなどの認証情報を保持できます。ロボットの認証情報を変更するには、Robot Accounts ウインドウのロボットのアカウント行にある Options 歯車を選択し、View Credentials を選択します。ロボットがアクセスする必要のある外部リポジトリーの適切な認証情報を追加します。



- **一般的な設定の確認および変更: ミラーリングされたリポジトリーページの左側の列から Settings ボタン (歯車アイコン) を選択します。生成されるページでは、ミラーリングされたリポジトリーに関連付けられた設定を変更することができます。とくに、ユーザーおよびロボットのパーミッションを変更して、リポジトリーから/への読み取りまたは書き込みを実行できるユーザーやロボットを指定できます。**

10.5. タグパターン

前述のように、少なくとも 1 つのタグを明示的に入力する (タグパターンではない) か、または、タグの latest がリモートリポジトリーに存在する必要があります。(タグ latest はタグの一覧に指定されていない限り同期されません)。これは、Quay でミラーリングに使用する指定の一覧との比較用にリモートリポジトリーでタグ一覧を取得するために必要です。

パターン構文

パターン	説明
*	すべての文字に一致します。
?	任意の 1 文字に一致します。
[seq]	**seq** の任意の文字と一致します。
[!sEQ]	**seq** にない文字と一致します。

タグのパターン例

パターン例	一致例
v3*	v32、v3.1、v3.2、v3.2-4beta、v3.3
v3.*	v3.1、v3.2、v3.2-4beta
v3.?	v3.1、v3.2、v3.3
v3.[12]	v3.1、v3.2
v3.[12]*	v3.1、v3.2、v3.2-4beta

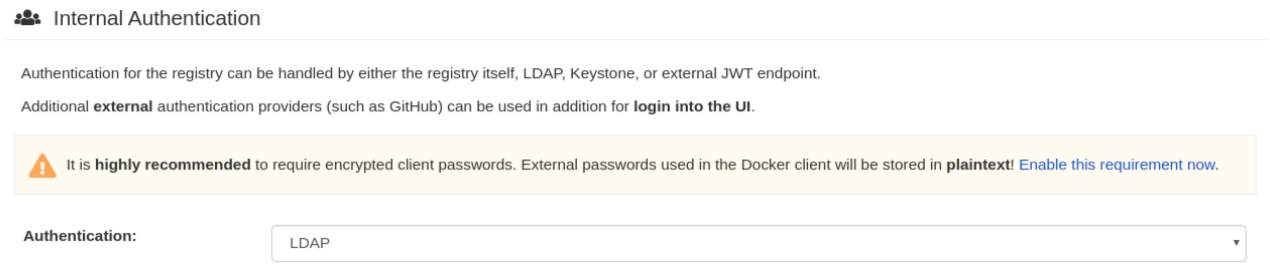
v3.[!1]*	v3.2、v3.2-4beta、v3.3

第11章 RED HAT QUAY の LDAP 認証設定

LDAP (Lightweight Directory Access Protocol) は、インターネットプロトコル (IP) ネットワークで分散ディレクトリー情報サービスにアクセスし、これを維持するために使用するオープンな、ベンダーに依存しない業界標準のアプリケーションプロトコルです。Red Hat Quay は、アイデンティティープロバイダーとしての LDAP の使用をサポートします。

11.1. LDAP の設定

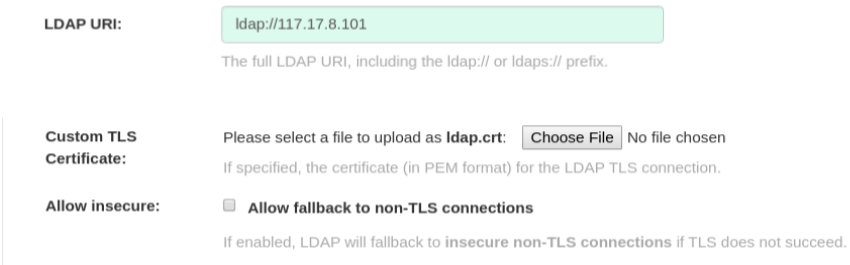
設定ツールで、Authentication セクションを見つけ、ドロップダウンメニューから LDAP を選択します。必要に応じて LDAP 設定フィールドを更新します。



- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

```
AUTHENTICATION_TYPE: LDAP
```

11.1.1. 完全な LDAP URI



- ldap:// または ldaps:// 接頭辞が含まれる完全な LDAP URI。

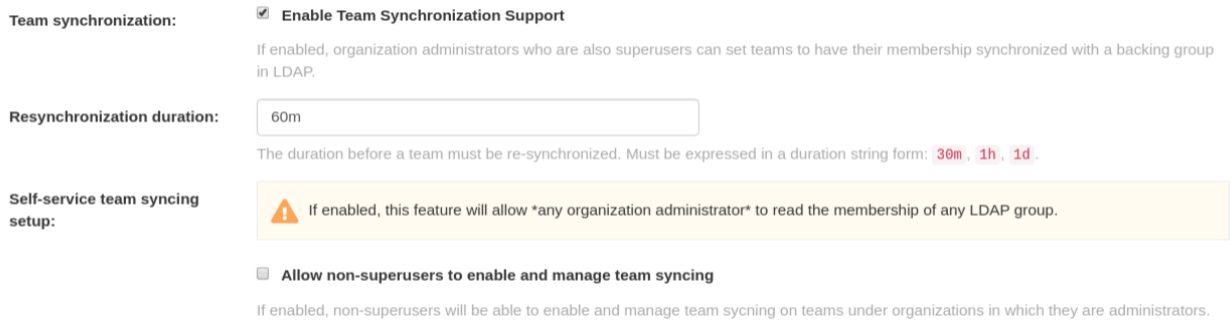
- ldaps:// で始まる URI は、TLS 設定に指定される SSL 証明書を使用します。

- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

```
LDAP_URI: ldaps://ldap.example.org
```

11.1.2. チームシンクロナイゼーション

- 有効にすると、スーパーユーザーでもある組織管理者は、チームをメンバーシップが LDAP のバッキンググループと同期するように設定できます。



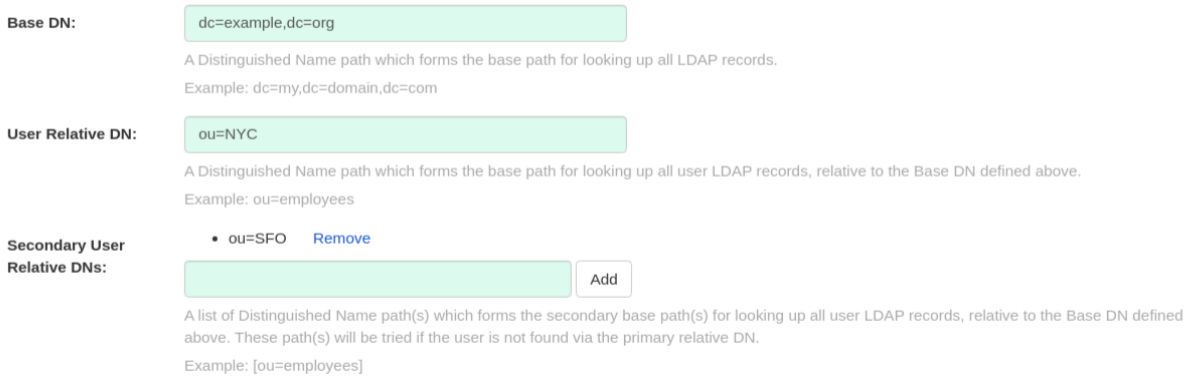
- 再同期の期間は、チームを再同期する必要のある期間です。期間の文字列の形式 30m、1h、1d で表記する必要があります。

- オプションで、スーパーユーザー以外のユーザーは、自らが管理者である組織でチームの同期を有効にし、管理することができます。

- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

```
FEATURE_TEAM_SYNCING: true
TEAM_RESYNC_STALE_TIME: 60m
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: true
```

11.1.3. ベースおよび相対的な区別された名前



- 識別名パスはすべての LDAP レコードを検索するためのベースパスを設定します。例: dc=my,dc=domain,dc=com

- 識別名パスのオプションの一覧は、上記で定義されたベース DN と関連して、すべてのユーザー LDAP レコードを検索するためのセカンダリーベースパスを設定します。ユーザーがプライマリーの相対 DN で見つからない場合に、これらのパスが試行されます。

- ユーザーの相対 DN は BaseDN に相対します。例: ou=NYC not ou=NYC,dc=example,dc=org

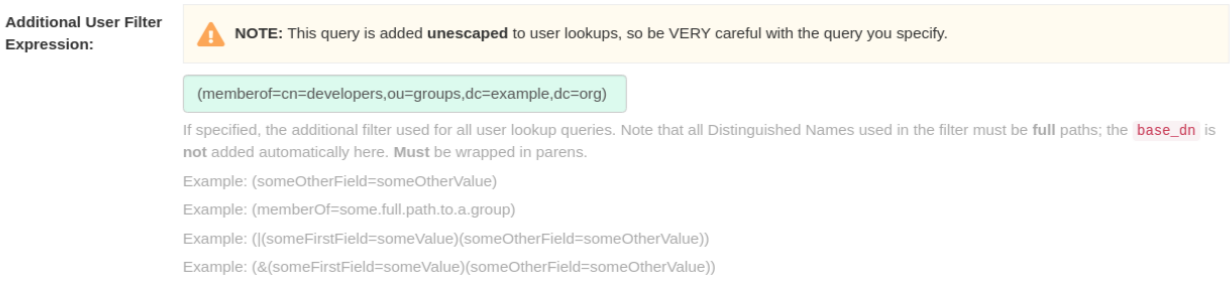
- ユーザーオブジェクトが置かれている組織単位が複数ある場合に、複数のセカンダリーユーザー相対 DN を入力します。組織単位に入力して Add ボタンをクリックし、複数の RDN (相対 DN) を追加します。例: ou=Users,ou=NYC and ou=Users,ou=SFO

- ユーザー相対 DN では、サブツリーの範囲を指定して検索します。たとえば、組織に Users OU 下に組織単位 NYC および SFO がある場合 (ou=SFO,ou=Users および ou=NYC,ou=Users)、Red Hat Quay は、ユーザー相対 DN がユーザーに設定されている場合 (ou=Users)、 NYC および SFO 組織単位のどちらのユーザーも認証できます。

- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

```
LDAP_BASE_DN:
- dc=example
- dc=com
LDAP_USER_RDN:
- ou=users
LDAP_SECONDARY_USER_RDNS:
- ou=bots
- ou=external
```

11.1.4. ユーザーフィルターの追加



- 指定されている場合は、すべてのユーザー検索クエリーに使用される追加のフィルターになります。フィルターで使用されるすべての識別名は 完全 パスである必要があることに注意してください。ベース DN は自動的に追加されません。括弧でラップする 必要があります。例: (&(someFirstField=someValue)(someOtherField=someOtherValue))

- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

```
LDAP_USER_FILTER: (memberof=cn=developers,ou=groups,dc=example,dc=com)
```

11.1.5. 管理者 DN

-

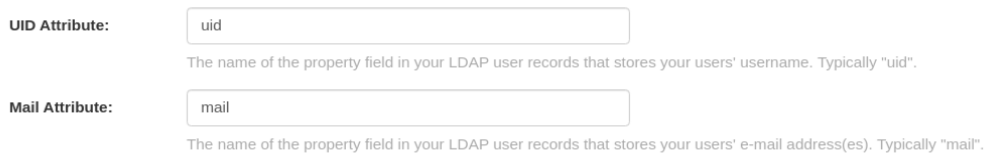
管理者アカウントの識別名およびパスワード。このアカウントは、すべてのユーザーアカウントのレコードにログインし、表示できる必要があります。例: uid=admin,ou=employees,dc=my,dc=domain,dc=com

- パスワードは config.yaml 内の プレーンテキスト に保管されるため、専用のアカウントを設定するか、またはパスワードハッシュを使用することが推奨されます。

- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

```
LDAP_ADMIN_DN: cn=admin,dc=example,dc=com
LDAP_ADMIN_PASSWD: changeme
```

11.1.6. UID およびメール属性



- UID 属性は、username として使用する LDAP ユーザーレコードのプロパティーフィールドの名前です。通常は uid です。

- Mail 属性は、ユーザーのメールアドレスを保存する LDAP ユーザーレコードのプロパティーフィールドの名前です。通常は mail です。

- これらはいずれもログイン時に使用できます。

- ログインしたユーザー名は User Relative DN(ユーザー相対 DN) に存在する必要があります。

- sAMAccountName は、Microsoft Active Directory 設定に対する UID 属性です。

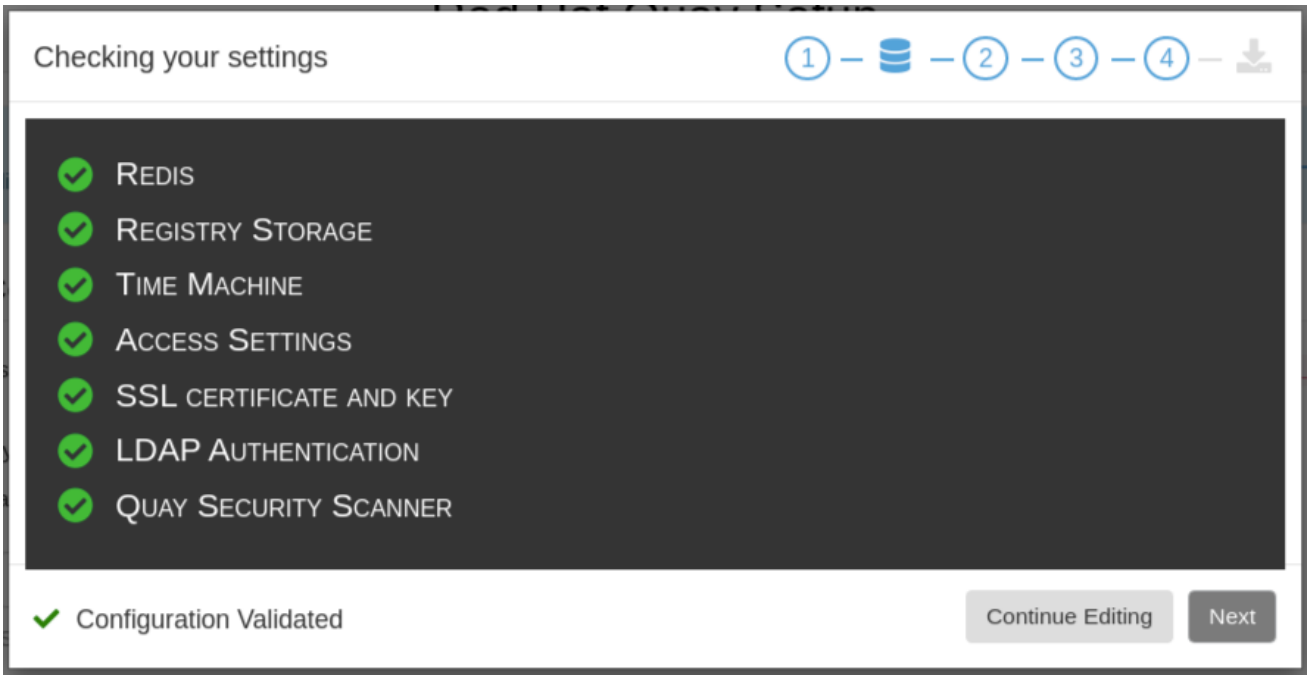
- 以下は、config.yaml ファイルの結果としてのエントリーの例です。

-

```
LDAP_UID_ATTR: uid
LDAP_EMAIL_ATTR: mail
```

11.1.7. 検証

設定が完了したら、Save Configuration Changes ボタンをクリックして設定を検証します。



すべての検証を正常に実行したから続行する必要があります。または、Continue Editing ボタンを選択して追加の設定を実行することができます。

11.2. 一般的な問題

無効な認証情報

管理者 DN または管理者 DN パスワードの値が正しくありません。

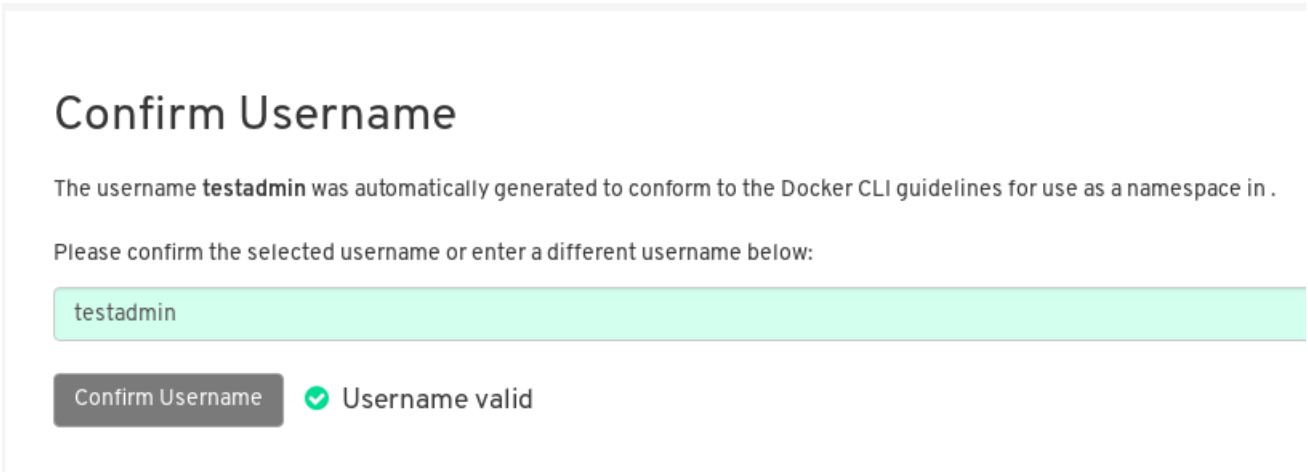
スーパーユーザー %USERNAME% の検証が失敗する: ユーザーが見つかりません。ユーザーはリモート認証システムに存在しないか、または LDAP 認証の設定が正しくありません。

Red Hat Quay は、Administrator DN フィールドに指定された Username/Password を使用して

LDAP サーバーに接続できますが、User Relative DN Path の UID Attribute または Mail Attribute フィールドで現在ログインしているユーザーを見つけることはできません。現在ログインしているユーザーが User Relative DN Path に存在しないか、または管理者 DN ユーザーにこの LDAP パスの検索/読み取り権限がありません。

11.3. LDAP ユーザーをスーパーユーザーとして設定する

LDAP が設定されたら、有効な LDAP のユーザー名およびパスワードを使用して Red Hat Quay インスタンスにログインすることができます。以下の図のように、Red Hat Quay のユーザー名を確認することを求めるプロンプトが出されます。



LDAP ユーザーにスーパーユーザー権限を割り当てるには、config.yaml ファイルをユーザー名で変更します。例:

```
SUPER_USERS:
- testadmin
```

更新された config.yaml ファイルで Red Hat Quay コンテナーを再起動します。次回ログイン時に、ユーザーにはスーパーユーザー権限が付与されます。

第12章 RED HAT QUAY の PROMETHEUS および GRAFANA メトリクス

Red Hat Quay は、各インスタンスで Prometheus および Grafana 互換エンドポイントをエクスポートし、モニターおよびアラートを容易にします。

12.1. PROMETHEUS エンドポイントの公開

Red Hat Quay インスタンスの Prometheus および Grafana 互換エンドポイントは、ポート 9092 にあります。Prometheus および Grafana を Quay リポジトリー数をモニターするように設定する方法についての詳細は、Monitoring Quay with Prometheus and Grafana を参照してください。

12.1.1. メトリクスを使用するための Prometheus の設定

Prometheus では、クラスターで実行されているすべての Red Hat Quay インスタンスにアクセスできるようにする必要があります。通常の設定では、これは単一の named DNS エントリーの Red Hat Quay インスタンスの一覧 (これは Prometheus に渡されます) を表示して実行します。

12.1.2. Kubernetes での DNS 設定

単純な Kubernetes サービス は、Prometheus の DNS エントリーを提供するように設定できます。Kubernetes での Prometheus の実行に関する詳細は、Prometheus and Kubernetes および Monitoring Kubernetes with Prometheus を参照してください。

12.1.3. 手動クラスターの DNS 設定

SkyDNS は、 Kubernetes を使用していない場合にこの DNS レコードを管理するための単純なソリューションです。SkyDNS は etcd クラスター上で実行できます。クラスターの各 Red Hat Quay インスタンスのエントリーは etcd ストアで追加および削除できます。SkyDNS はここでエントリーを定期的に読み取り、DNS レコードの Quay インスタンスの一覧を適宜更新します。

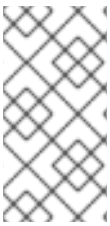
第13章 RED HAT QUAY でのストレージの GEOREPLICATION

Georeplication は、ローカライズされたストレージからコンテナーイメージを提供するために単一のグローバルに分散される Red Hat Quay を許可します。

Georeplication が設定されていると、コンテナーイメージのプッシュはその Red Hat Quay インスタンスの推奨ストレージエンジンに書き込まれます。最初のプッシュ後に、イメージデータは他のストレージエンジンにバックグラウンドでレプリケートされます。レプリケーションの場所の一覧は設定可能です。イメージプルは、プルのパフォーマンスを最大化するために、利用可能な最も近いストレージエンジンを常に使用します。

13.1. 前提条件

Georeplication では、各地理的に高可用性ストレージエンジン (S3、GCS、RADOS、Swift) が必要です。さらに、レプリケーションの要件により、各リージョンが すべての ストレージエンジンにアクセスできる必要があります。



注記

現時点では、ローカルディスクストレージは Georeplication と互換性がありません。

13.2. CONFIG TOOL へのアクセス

Red Hat Quay Config Tool を開き、Georeplication のストレージを設定します。

13.3. ストレージレプリケーションの有効化

1. Registry Storage というセクションまでスクロールダウンします。
2. Enable Storage Replication をクリックします。
3. データをレプリケートするストレージエンジンをそれぞれ追加します。使用するストレージエンジンをすべて一覧表示する必要があります。
4. すべてのストレージエンジンに対してすべてのイメージの完全なレプリケーションが必要な

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。namespace 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. これを実行した後に、Save Configuration Changes をクリックします。設定の変更は、Red Hat Quay の次回の再起動時に有効になります。

6. ストレージを追加し、Georeplication について Replicate to storage engine by default を有効にした後に、すべてのストレージで既存のイメージデータを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。

13.4. ストレージの優先情報を使用した RED HAT QUAY の実行

1. Red Hat Quay を実行しているすべてのマシンに config.yaml をコピーします。

2. 各リージョンの各マシンについて、マシンが実行されているリージョンの優先ストレージエンジンを使用して QUAY_DISTRIBUTED_STORAGE_PREFERENCE 環境変数を追加します。

たとえば、マシンが /mnt/quay/config で利用可能なホスト上の config ディレクトリーを使用してヨーロッパで実行されている場合は、以下を実行します。

```
# docker login quay.io
Username: yourquayuser
Password: *****
# docker run -d -p 443:8443 -p 8080:8080 -v /mnt/quay/config:/conf/stack:Z \
   -e QUAY_DISTRIBUTED_STORAGE_PREFERENCE=europestorage \
   quay.io/redhat/quay:v3.3.4
```

注記

指定された環境変数の値は、config パネルで定義されている Location ID の名前に一致する必要があります。

3. **すべての Red Hat Quay コンテナーを再起動します。**

第14章 RED HAT QUAY のトラブルシューティング

一般的な障害モードおよびリカバリーのベストプラクティス。

- HTTP ステータスコード 429 が出される
- 実行権限があっても 403 が出される
- Dockerfile でのベースイメージのプルが 403 を出して失敗する
- ビルドトリガーを追加できない
- ビルドログが読み込まれない
- Cannot locate specified Dockerfile が出される * いずれのレジストリーエンドポイントにも到達できない
- EC2 コンテナーサービスを使用してプライベートリポジトリーにアクセスできない
- Docker が i/o タイムアウトを返す
- Docker ログインが異常なエラーを出して失敗する
- プルが異常なエラーを出して失敗する
- プッシュしたばかりだが、タイムスタンプが正しくない
- Marathon/Mesos でのプライベート Quay.io イメージのプルに失敗する

第15章 RED HAT QUAY のスキーマ

注記

とくにマークされていない限り、すべてのフィールドはオプションになります。

- AUTHENTICATION_TYPE [string] 必須: 認証情報の認証に使用する認証エンジン。
  - enum: Database、LDAP、JWT、Keystone、OIDC。
  - 例: Database
- BUILDLOGS_REDIS [object] 必須: ビルドログキャッシュの Redis の接続情報。
  - HOST [string] 必須: Redis にアクセス可能なホスト名。
    - 例: my.redis.cluster
  - PASSWORD [string]: Redis インスタンスに接続するためのパスワード。
    - 例: mypassword
  - PORT [number]: Redis にアクセスできるポート。
    - 例: 1234
- DB_URI [string] 必須: データベースへのアクセスに使用する URI(認証情報を含む)。
  - 参照: https://www.postgresql.org/docs/9.3/static/libpq-connect.html#AEN39495

- 例: mysql+pymysql://username:password@dns.of.database/quay

- DEFAULT_TAG_EXPIRATION [string] 必須: 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。デフォルトは 2w に設定されます。

  - パターン: ^[0-9]+(w|m|d|h|s)$

- DISTRIBUTED_STORAGE_CONFIG [object] 必須: Red Hat Quay で使用するストレージエンジンの設定。それぞれのキーはストレージエンジンに一意の ID で、値はそのエンジンのタイプと設定のタプルになります。

  - 例: {"local_storage": ["LocalStorage", {"storage_path": "some/path/"}]}

- DISTRIBUTED_STORAGE_PREFERENCE [array] 必須: 使用する優先されるストレージ (DISTRIBUTED_STORAGE_CONFIG の ID 別)。エンジンを優先するとは、優先されるエンジンがプルの実行時に最初にプルされ、イメージがこれにプッシュされるという意味です。

  - Min Items: None

    - 例: [u's3_us_east', u's3_us_west']

    - array item [string]

  - preferred_url_scheme [string] 必須: Red Hat Quay をヒットする際に使用する URL スキーム。Red Hat Quay が たしかに SSL の背後にある場合、これは https である はずです。

    - enum: http, https

    - 例: https

- SERVER_HOSTNAME [string] 必須: スキームなしに Red Hat Quay にアクセスできる URL。

- 例: quay.io

- TAG_EXPIRATION_OPTIONS [array] 必須: ユーザーが namespace でタグの有効期限について選択できるオプション (有効にされている場合)。
  - Min Items: None
  - array item [string]
  - パターン: ^[0-9]+(w|m|d|h|s)$

- USER_EVENTS_REDIS [object] 必須: ユーザーイベント処理についての Redis の接続情報。
  - HOST [string] 必須: Redis にアクセス可能なホスト名。
    - 例: my.redis.cluster
  - PASSWORD [string]: Redis インスタンスに接続するためのパスワード。
    - 例: mypassword
  - PORT [number]: Redis にアクセスできるポート。
    - 例: 1234

- ACTION_LOG_ARCHIVE_LOCATION [string]: アクションログのアーカイブが有効にされている場合の、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' [string]: アクションログのアーカイブが有効にされている場合の、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

- APP_SPECIFIC_TOKEN_EXPIRATION [string, null]: 外部アプリケーショントークンの有効期限。デフォルトは None になります。

  - パターン: ^[0-9]+(w|m|d|h|s)$

- ALLOW_PULLS_WITHOUT_STRICT_LOGGING [boolean]: true の場合、プル監査ログエントリーを書き込めないプルは正常に実行されます。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。デフォルトは false に設定されます。

  - 例: True

- AVATAR_KIND [string]: 表示する avatar のタイプ。インライン (ローカル) または Gravatar (gravatar)。

  - enum: local, gravatar

- BITBUCKET_TRIGGER_CONFIG ['object', 'null']: ビルドトリガーに BitBucket を使用するための設定。

  - consumer_key [string] 必須: この Red Hat 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)。

    - 例: 0e8dbe15c4c7630b6780

  - CONSUMER_SECRET [string] 必須: この Red Hat 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)

- 例: e4a58ddd3d7408b7aec109e85564a0d153d3e846

- BROWSER_API_CALLS_XHR_ONLY [boolean]: 有効にされている場合、XHR による呼び出しとしてマークが付けられた API のみがブラウザーから許可されます。デフォルトは True に設定されます。
  - 例: False

- CONTACT_INFO [array]: 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。連絡先情報が 1 つだけ指定される場合、連絡先フッターが直接リンクされます。
  - Min Items: 1
  - Unique Items: True
    - array item 0 [string]: 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
  - array item 1 [string]: 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
  - array item 2 [string]: 電話番号に電話するためのリンクを追加します。

        - パターン: ^tel:(.)+$
        - 例: tel:+1-888-930-3475
    - array item 3 [string]: 定義された URL へのリンクを追加します。
        - パターン: ^http(s)?://(.)+$
        - 例: https://twitter.com/quayio

- BLACKLIST_V2_SPEC [string]: 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。デフォルトは <1.6.0 に設定されます。
    - 参照: http://pythonhosted.org/semantic_version/reference.html#semantic_version.Spec
    - 例: <1.8.0

- DB_CONNECTION_ARGS [object]: 指定されている場合、タイムアウトや SSL などのデータベースの接続引数。
    - threadlocals [boolean] 必須: 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。
    - autorollback [boolean] 必須: 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。
    - ssl [object]: SSL 接続の設定
        - ca [string] 必須: SSL 接続に使用する CA 接続への絶対コンテナーパス。

- 例: conf/stack/ssl-ca-cert.pem

- DEFAULT_NAMESPACE_MAXIMUM_BUILD_COUNT [number, null]: None では無い場合、namespace でキューに入れることができるビルドのデフォルトの最大数。
  - 例: 20
- DIRECT_OAUTH_CLIENTID_WHITELIST [array]: 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。
  - Min Items: None
  - Unique Items: True
  - 参照: https://coreos.com/quay-enterprise/docs/latest/direct-oauth.html
    - array item [string]
- DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS [array]: 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)。
  - Min Items: None
  - 例: s3_us_east, s3_us_west
    - array item [string]
- EXTERNAL_TLS_TERMINATION [boolean]: TLS がサポートされているが、Red Hat Quay の前の層で終了する場合は true になります。

- 例: True

- ENABLE_HEALTH_DEBUG_SECRET [string, null]: 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。

  - 例: somesecrethere

- EXPIRED_APP_SPECIFIC_TOKEN_GC [string, null]: 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。デフォルトは 1d に設定されます。

  - パターン: ^[0-9]+(w|m|d|h|s)$

- FEATURE_ACI_CONVERSION [boolean]: ACI への変換を有効にするかどうか。デフォルトは false に設定されます。

  - 例: False

- FEATURE_ACTION_LOG_ROTATION [boolean]: ストレージに対する古いアクションログのローテーションを行うかどうか。デフォルトは false に設定されます。

  - 例: False

- FEATURE_ADVERTISE_V2 [boolean]: v2/ エンドポイントが表示されるかどうか。デフォルトは True に設定されます。

  - 例: True

- FEATURE_ANONYMOUS_ACCESS [boolean]: 匿名ユーザーがパブリックリポジトリーを参照し、プルすることを許可するかどうか。デフォルトは True に設定されます。

- 例: True

- FEATURE_APP_REGISTRY [boolean]: アプリケーションリポジトリーのサポートを有効にするかどうか。デフォルトは false に設定されます。

  - 例: False

- FEATURE_APP_SPECIFIC_TOKENS [boolean]: 有効にされる場合、ユーザーは Docker CLI で使用するトークンを作成できます。デフォルトは True に設定されます。

  - 例: False

- FEATURE_BITBUCKET_BUILD [boolean]: Bitbucket ビルドトリガーをサポートするかどうか。デフォルトは false に設定されます。

  - 例: False

- FEATURE_BUILD_SUPPORT [boolean]: Dockerfile ビルドをサポートするかどうか。デフォルトは True に設定されます。

  - 例: True

- FEATURE_CHANGE_TAG_EXPIRARTION [boolean]: ユーザーおよび組織が namespace でタグのタグ有効期限を変更することを許可するかどうか。デフォルトは True に設定されます。

  - 例: False

- FEATURE_DIRECT_LOGIN [boolean]: ユーザーが UI に直接ログインできるかどうか。デフォルトは True に設定されます。

  - 例: True

- **FEATURE_GITHUB_BUILD [boolean]: GitHub ビルドトリガーをサポートするかどうか。デフォルトは false に設定されます。**
  - **例: False**
- **FEATURE_GITHUB_LOGIN [boolean]: GitHub ログインがサポートされるかどうか。デフォルトは false に設定されます。**
  - **例: False**
- **FEATURE_GITLAB_BUILD[boolean]: GitLab ビルドトリガーをサポートするかどうか。デフォルトは false に設定されます。**
  - **例: False**
- **FEATURE_GOOGLE_LOGIN [boolean]: Google ログインがサポートされるかどうか。デフォルトは false に設定されます。**
  - **例: False**
- **FEATURE_INVITE_ONLY_USER_CREATION [boolean]: 作成されるユーザーが他のユーザーによって招待されるようにするかどうか。デフォルトは false に設定されます。**
  - **例: False**
- **FEATURE_LIBRARY_SUPPORT [boolean]: Docker への/からのプルおよびプッシュの実行時に namespace のないリポジトリーを許可するかどうか。デフォルトは True に設定されます。**
  - **例: True**
- **FEATURE_MAILING [boolean]: メールを有効にするかどうか。デフォルトは True に設定されます。**

- 例: True

- FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP [boolean]: 有効にされる場合、スーパーユーザー以外のユーザーがバッキング LDAP または Keystone に対してチームの同期を設定できます。デフォルトは False に設定されます。

  - 例: True

- FEATURE_PARTIAL_USER_AUTOCOMPLETE [boolean]: true に設定される場合、部分的なユーザーに自動補完が適用されます。デフォルトは True に設定されます。

  - 例: True

- FEATURE_PERMANENT_SESSIONS [boolean]: セッションが永続するかどうか。デフォルトは True に設定されます。

  - 例: True

- FEATURE_PROXY_STORAGE [boolean]: レジストリー 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。デフォルトは false に設定されます。

  - 例: False

- FEATURE_PUBLIC_CATALOG [boolean]: true に設定される場合、_catalog エンドポイントはパブリックリポジトリーを返します。それ以外の場合は、プライベートリポジトリーのみが返されます。デフォルトは false に設定されます。

  - 例: False

- FEATURE_READER_BUILD_LOGS [boolean]: 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。デフォルトは False に設定されます。

- 例: False

- FEATURE_RECAPTCHA [boolean]: Recaptcha がユーザーログインおよびリカバリーに必要かどうか。デフォルトは False に設定されます。

  - 例: False

  - 参照: https://www.google.com/recaptcha/intro/

- FEATURE_REQUIRE_ENCRYPTED_BASIC_AUTH [boolean]: (暗号化されたトークンに対して) 暗号化されていないパスワードが基本認証に使用できるかどうか。デフォルトは False に設定されます。

  - 例: False

- FEATURE_REQUIRE_TEAM_INVITE [boolean]: ユーザーをチームに追加する際に招待を必要とするかどうか。デフォルトは True に設定されます。

  - 例: True

- FEATURE_SECURITY_NOTIFICATIONS [boolean]: セキュリティースキャナーが有効にされている場合、セキュリティー通知をオン/オフにするかどうか。デフォルトは False に設定されます。

  - 例: False

- FEATURE_SECURITY_SCANNER [boolean]: セキュリティースキャナーをオン/オフにするかどうか。デフォルトは False に設定されます。

  - 参照: https://access.redhat.com/documentation/ja-jp/red_hat_quay/3.3/html-single/manage_red_hat_quay/#clair-initial-setup

- 例: False

- FEATURE_STORAGE_REPLICATION [boolean]: 複数のストレージ間で自動的にレプリケートするかどうか。デフォルトは False に設定されます。

  - 例: False

- FEATURE_SUPER_USERS [boolean]: スーパーユーザーがサポートされているかどうか。デフォルトは True に設定されます。

  - 例: True

- FEATURE_TEAM_SYNCING [boolean]: 認証エンジン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 (LDAP または Keystone)。

  - 例: True

- FEATURE_USER_CREATION [boolean] : ユーザーが (スーパーユーザー以外のユーザーによって) 作成可能かどうか。デフォルトは True に設定されます。

  - 例: True

- FEATURE_USER_LOG_ACCESS [boolean]: true に設定される場合、ユーザーは namespace の監査ログにアクセスできます。デフォルトは False に設定されます。

  - 例: True

- FEATURE_USER_METADATA [boolean]: ユーザーメタデータを収集し、サポートするかどうか。デフォルトは False に設定されます。

  - 例: False

- FEATURE_USER_RENAME [boolean]: true に設定される場合、ユーザーは独自の namespace の名前を変更できます。デフォルトは False に設定されます。
  - 例: True
- GITHUB_LOGIN_CONFIG [object, 'null']: 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。
  - 参照: https://coreos.com/quay-enterprise/docs/latest/github-auth.html
  - allowed_organizations [array]: 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。
    - Min Items: None
    - Unique Items: True
      - array item [string]
  - API_ENDPOINT [string]: 使用する GitHub (Enterprise) API のエンドポイント。github.com について上書きする必要があります。
    - 例: https://api.github.com/
  - CLIENT_ID [string] 必須: 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 と共有にすることはできません。
    - 参照: https://coreos.com/quay-enterprise/docs/latest/github-app.html
    - 例: 0e8dbe15c4c7630b6780
  -

CLIENT_SECRET [string] 必須: 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。

- 参照: https://coreos.com/quay-enterprise/docs/latest/github-app.html
- 例: e4a58ddd3d7408b7aec109e85564a0d153d3e846

GITHUB_ENDPOINT [string] 必須: ヒットする GitHub (Enterprise) のエンドポイント。

- 例: https://github.com/

ORG_RESTRICT [boolean]: 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。

例: True

- GITHUB_TRIGGER_CONFIG [object, null]: ビルドトリガーに GitHub (Enterprise) を使用するための設定。

参照: https://coreos.com/quay-enterprise/docs/latest/github-build.html

API_ENDPOINT [string]: 使用する GitHub (Enterprise) API のエンドポイント。github.com について上書きする必要があります。

- 例: https://api.github.com/

CLIENT_ID [string] 必須: 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。

- 参照: https://coreos.com/quay-enterprise/docs/latest/github-app.html

- 例: 0e8dbe15c4c7630b6780

- CLIENT_SECRET [string] 必須: 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。

- 参照: https://coreos.com/quay-enterprise/docs/latest/github-app.html

- 例: e4a58ddd3d7408b7aec109e85564a0d153d3e846

- GITHUB_ENDPOINT [string] 必須: ヒットする GitHub (Enterprise) のエンドポイント。

- 例: https://github.com/

- GITLAB_TRIGGER_CONFIG [object]: 外部認証に Gitlab (Enterprise) を使用するための設定。

- CLIENT_ID [string] 必須: 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 [string] 必須: 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。

- 例: e4a58ddd3d7408b7aec109e85564a0d153d3e846

- gitlab_endpoint [string] 必須: Gitlab(Enterprise) が実行されているエンドポイント

- 例: https://gitlab.com

- GOOGLE_LOGIN_CONFIG [object, null]: 外部認証に Google を使用するための設定。
  - CLIENT_ID [string] 必須: 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 [string] 必須: 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。
    - 例: e4a58ddd3d7408b7aec109e85564a0d153d3e846
- HEALTH_CHECKER [string]: 設定済みのヘルスチェック。
  - 例: ('RDSAwareHealthCheck', {'access_key': 'foo', 'secret_key': 'bar'})
- LOG_ARCHIVE_LOCATION [string]: ビルドが有効にされている場合の、アーカイブされたビルドログを配置するストレージエンジン。
  - 例: s3_us_east
- LOG_ARCHIVE_PATH [string]: ビルドが有効にされている場合、アーカイブされたビルドログを配置するストレージのパス。
  - 例: archives/buildlogs
- MAIL_DEFAULT_SENDER [string, null]: 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。指定されていない場合は、デフォルトで support@quay.io に設定されます。
  - 例: support@myco.com

- MAIL_PASSWORD [string, null]: メールの送信時に使用する SMTP パスワード。
  - 例: mypassword
- MAIL_PORT [number]: 使用する SMTP ポート。指定されていない場合は、デフォルトで 587 に設定されます。
  - 例: 588
- MAIL_SERVER [string]: 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.somedomain.com
- MAIL_USERNAME [string, 'null']: メールの送信時に使用する SMTP ユーザー名。
  - 例: myuser
- MAIL_USE_TLS [boolean]: 指定されている場合、メールの送信に TLS を使用するかどうか。
  - 例: True
- MAXIMUM_LAYER_SIZE [string]: イメージ層の最大許容サイズ。デフォルトは 20G に設定されます。
  - パターン: ^[0-9]+(G|M)$
  - 例: 100G
- PUBLIC_NAMESPACES [array]: namespace がパブリック namespace 一覧に定義されて

いる場合、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。通常、これは企業顧客がよく知られている namespace のセットを設定する際に使用します。

- Min Items: None
- Unique Items: True
  - array item [string]

- PROMETHEUS_NAMESPACE [string]: 公開されているすべての Prometheus メトリクスの適用される接頭辞。デフォルトは quay に設定されます。
  - 例: myregistry
- RECAPTCHA_SITE_KEY [string]: recaptcha が有効にされている場合は、Recaptcha サービスのサイトキー。
- RECAPTCHA_SECRET_KEY [string]: '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。
- REGISTRY_TITLE [string]: 指定される場合はレジストリーの長いタイト。デフォルトは Quay Enterprise に設定されます。
  - 例: Corp Container Service
- REGISTRY_TITLE_SHORT [string]: 指定されている場合はレジストリーの短いタイトル。デフォルトは Quay Enterprise に設定されます。
  - 例: CCS
- SECURITY_SCANNER_ENDPOINT [string]: 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_INDEXING_INTERVAL [number]: 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)。デフォルトは 30 に設定されます。
  - 例: 30

- SESSION_COOKIE_SECURE [boolean]: secure なプロパティーをセッション cookie に設定するかどうか。デフォルトは False に設定されます。SSL を使用するすべてのインストールでは、True にすることが推奨されます。
  - 例: True
  - 参考: https://en.wikipedia.org/wiki/Secure_cookies

- SSL_PROTOCOLS [array]: 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。一覧から SSL プロトコルを削除すると、Red Hat Quay の起動中にプロトコルが無効になります。
  - SSL_PROTOCOLS: ['TLSv1','TLSv1.1','TLSv1.2']

- SUPER_USERS [array]: スーパーユーザー権限が付与されるユーザーの Red Hat Quay ユーザー名。
  - Min Items: None
  - Unique Items: True
    - array item [string]

- TEAM_RESYNC_STALE_TIME [string]: 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、再同期する頻度 (デフォルト: 30m)。
  - パターン: ^[0-9]+(w|m|d|h|s)$
  - 例: 2h
- USERFILES_LOCATION [string]: 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 [string]: 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 [string]: ユーザーアカウントを回復するためのトークンが有効な時間の長さ。デフォルトは 30m に設定されます。
  - 例: 10m
  - パターン: ^[0-9]+(w|m|d|h|s)$
- V2_PAGINATION_SIZE [number]: V2 レジストリー API のページごとに返される結果の数。
  - 例: 100

追加リソース