



Red Hat Quay 3.7

概念実証 (実稼働以外) 向けの Red Hat Quay の デプロイ

Red Hat Quay のデプロイ

Red Hat Quay 3.7 概念実証 (実稼働以外) 向けの Red Hat Quay のデプロイ

Red Hat Quay のデプロイ

法律上の通知

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

Red Hat Quay を使い始める

目次

はじめに	3
第1章 概要	4
1.1. アーキテクチャー	4
第2章 RED HAT QUAY の使用開始	6
2.1. 前提条件	6
2.2. RED HAT ENTERPRISE LINUX サーバーの設定	6
2.3. データベースの設定	9
2.4. REDIS の設定	9
2.5. RED HAT QUAY の設定	9
2.6. RED HAT QUAY のデプロイ	11
2.7. RED HAT QUAY の使用	12
第3章 RED HAT QUAY の高度なデプロイメント	14
3.1. SSL を使用した RED HAT QUAY への接続の保護	14
3.2. RED HAT QUAY スーパーユーザー	19
3.3. リポジトリのミラーリング	21
3.4. CLAIR V4 のデプロイ	26
3.5. コンテナの起動	30
3.6. FIPS の準備状態およびコンプライアンス	33
第4章 次のステップ	35

はじめに

Red Hat Quay は、コンテナイメージをビルドし、保護し、これを提供するためのエンタープライズ品質のレジストリーです。この手順では、概念実証 (実稼働以外) 向けの Red Hat Quay のデプロイ方法について説明します。

第1章 概要

Red Hat Quay の機能は次のとおりです。

- 高可用性
- Geo レプリケーション
- リポジトリのミラーリング
- Docker v2、スキーマ 2 (multiarch) のサポート
- 継続的インテグレーション
- Clair によるセキュリティースキャン
- カスタムログローテーション
- ダウンタイムなしのガベージコレクション
- 24 時間 365 日のサポート

Red Hat Quay は以下のサポートを提供します。

- 複数の認証およびアクセス方法
- 複数のストレージバックエンド
- Quay、Clair、およびストレージバックエンドのカスタム証明書
- アプリケーションレジストリー
- 異なるコンテナイメージタイプ

1.1. アーキテクチャー

Red Hat Quay は、内部と外部の両方で数多くのコアコンポーネントで設定されます。

1.1.1. 内部コンポーネント

- **Quay (コンテナレジストリー)**: Pod の複数のコンポーネントで設定される **Quay** コンテナをサービスとして実行します。
- **Clair**: コンテナイメージで脆弱性の有無をスキャンし、修正を提案します。

1.1.2. 外部コンポーネント

- **データベース**: Red Hat Quay で、プライマリーメタデータストレージとして使用されます。これはイメージストレージ用ではないことに注意してください。
- **Redis (key-value ストア)**: ライブビルダーログおよび Red Hat Quay チュートリアルを保存します。
- **クラウドストレージ**: サポートされるデプロイメントでは、以下のストレージタイプのいずれかを使用する必要があります。

- **パブリッククラウドストレージ:** パブリッククラウド環境では、Amazon Web Services の Amazon S3 や Google Cloud の Google Cloud Storage などのクラウドプロバイダーのオブジェクトストレージを使用する必要があります。
- **プライベートクラウドストレージ:** プライベートクラウドでは、Ceph RADOS や OpenStack Swift などの S3 または Swift 準拠のオブジェクトストアが必要です。



警告

実稼働環境の設定にローカルにマウントされたディレクトリーのストレージエンジンを使用しないでください。マウントされた NFS ボリュームはサポートされません。ローカルストレージは Red Hat Quay のテスト専用のインストールに使用されることが意図されています。

第2章 RED HAT QUAY の使用開始

Red Hat Quay レジストリーは、以下の仕様を使用して、単一のマシン (物理または仮想) で実稼働以外の目的でデプロイできます。

2.1. 前提条件

- **Red Hat Enterprise Linux (RHEL) 8:** [ダウンロードページ](#) から、最新の Red Hat Enterprise Linux 8 サーバーメディアを取得し、[Red Hat Enterprise Linux 8 の製品ドキュメント](#) のインストール手順に従います。
- **Valid Red Hat Subscription:** 有効な Red Hat Enterprise Linux 8 サーバーのサブスクリプションを設定します。
- **CPU:** 2 つ以上の仮想 CPU
- **RAM:** 4GB 以上
- **ディスク領域:** 必要なディスク領域は、レジストリーのストレージのニーズによって異なります。テストシステムには、約 30 GB のディスク領域があれば十分です (以下は内訳です)。
 - オペレーティングシステム (Red Hat Enterprise Linux Server) 用に少なくとも 10GB のディスク領域
 - 10GB 以上: docker ストレージ用のディスク容量 (3 つのコンテナを実行するために必要)
 - Quay ローカルストレージ用に少なくとも 10 GB のディスク領域。CEPH またはその他のローカルストレージではさらにメモリーが必要となる可能性があることに注意してください。

サイジングについての詳細は [Quay 3.x Sizing Guidelines](#) を参照してください。



注記

Red Hat Enterprise Linux (RHEL) 8 は、Red Hat Quay 3.7 の高可用性で本番環境の品質のデプロイメントに強く推奨されます。RHEL 7 は Red Hat Quay 3.7 でテストされておらず、将来のリリースで非推奨になる予定です。

2.1.1. podman の使用

本書では、コンテナを作成し、デプロイするために podman を使用します。システムに podman がインストールされていない場合は、同等の docker コマンドを使用できる必要があります。podman および関連技術の詳細は、[Red Hat Enterprise Linux 8 のコンテナの構築、実行、および管理](#) を参照してください。



注記

Podman は、Red Hat Quay 3.7 の高可用性で本番環境の品質のデプロイメントに強く推奨されます。Docker は Red Hat Quay 3.7 でテストされておらず、将来のリリースで非推奨になる予定です。

2.2. RED HAT ENTERPRISE LINUX サーバーの設定

2.2.1. RHEL サーバーのインストールおよび登録

1. 最新の RHEL 8 サーバーをインストールします。最小インストール (シェルアクセスのみ) を実行するか、Server plus GUI (デスクトップが必要な場合) を実行できます。
2. [How to register and subscribe a system...](#) で説明されているように RHEL サーバーシステムを登録し、これをサブスクライブします。
3. 以下のコマンドを使用して、システムを登録し、利用可能なサブスクリプションを一覧表示します。利用可能な RHEL サーバーのサブスクリプションを選択し、プール ID に割り当て、最新のソフトウェアにアップグレードします。

```
# subscription-manager register --username=<user_name> --password=<password>
# subscription-manager refresh
# subscription-manager list --available
# subscription-manager attach --pool=<pool_id>
# yum update -y
```

2.2.2. podman のインストール

- podman を、システムにインストールされていない場合はインストールします。

```
$ sudo yum install -y podman
```

- または、コンテナソフトウェアパッケージの完全なセットをプルする **container-tools** モジュールをインストールできます。

```
$ sudo yum module install -y container-tools
```

2.2.3. レジストリー認証

- [Red Hat コンテナレジストリーの認証](#) で説明されているように、**Quay** コンテナをプルできるように、**registry.redhat.io** に対して認証をセットアップします。これは、イメージが **quay.io** でホストされる以前の Red Hat Quay リリースとは異なることに注意してください。以下のコマンドを使用してレジストリーにログインできます。

```
$ sudo podman login registry.redhat.io
Username: <username>
Password: <password>
```

2.2.4. ファイアウォールの設定

- システムでファイアウォールを実行している場合は、Red Hat Quay へのアクセスが許可されるルールを追加する必要がある場合があります。必要なコマンドは、マップされたポートによって異なります。以下に例を示します。

```
$ firewall-cmd --permanent --add-port=80/tcp
$ firewall-cmd --permanent --add-port=443/tcp
$ firewall-cmd --permanent --add-port=5432/tcp
$ firewall-cmd --permanent --add-port=5433/tcp
$ firewall-cmd --permanent --add-port=6379/tcp
$ firewall-cmd --reload
```

2.2.5. IP アドレスおよび命名サービス

Red Hat Quay でコンポーネントコンテナを設定し、それらが相互に通信できるようにする方法は多数あります。

- **コンテナの IP アドレスの使用:** `podman inspect` でコンテナの IP アドレスを判別し、接続文字列を指定する際に設定ツールでこれらの値を使用できます。以下に例を示します。

```
$ sudo podman inspect -f "{{.NetworkSettings.IPAddress}}" postgresql-quay
```

この方法は、コンテナの IP アドレスが再起動後に変更されるためにホストの再起動の影響を受けます。

- **命名サービスの使用:** デプロイメントをコンテナの再起動後も存続させたい場合は、つまり IP アドレスが変更されることとなりますが、ネーミングサービスを実装できます。たとえば、`dnsname` プラグインは、コンテナが名前でも相互に解決できるように使用します。
- **ホストネットワークの使用:** `podman run` コマンドを `--net=host` オプションと共に使用してから、アドレスを設定に指定する際にホストでコンテナポートを使用できます。2つのコンテナが同じポートを使用したい場合は、ポートの競合の影響を受けます。したがって、このオプションは推奨されません。
- **ポートマッピングの設定:** ポートマッピングを使用してホストでポートを公開し、これらのポートをホスト IP アドレスまたはホスト名と組み合わせて使用できます。

本書では、ポートマッピングを使用し、ホストシステムの静的 IP アドレスを使用することを前提としています。このデプロイメントでは、システムの IP アドレス `192.168.1.112` で `quay-server.example.com` を使用し、`/etc/hosts` ファイルでこの情報を設定します。

```
$ cat /etc/hosts
...
192.168.1.112 quay-server.example.com
```

Component	ポートマッピング	アドレス
Quay	-p 80:8080 -p 443:8443	http://quay-server.example.com
Postgres for Quay	-p 5432:5432	quay-server.example.com:5432
Redis	-p 6379:6379	quay-server.example.com:6379
Postgres for Clair V4	-p 5433:5432	quay-server.example.com:5433
Clair V4	-p 8081:8080	http://quay-server.example.com:8081

2.3. データベースの設定

Quay にはメタデータを格納するためのデータベースが必要です。Postgres は本書全体で使用され、高可用性設定での使用が推奨されています。また、以下で説明されている設定と同様の方法で MySQL を使用できます。

2.3.1. Postgres のセットアップ

この概念実証のシナリオでは、ローカルファイルシステムのディレクトリーを使用してデータベースデータを永続化します。

1. ここでは \$QUAY 変数で示されるインストールフォルダーで、データベースデータのディレクトリーを作成し、パーミッションを適切に設定します。

```
$ mkdir -p $QUAY/postgres-quay
$ setfacl -m u:26:-wx $QUAY/postgres-quay
```

2. データベースデータのボリューム定義と共にユーザー名、パスワード、データベース名、ポートを指定して、**podman run** を使用して **Postgres** コンテナを起動します。

```
$ sudo podman run -d --rm --name postgresql-quay \
-e POSTGRES_USER=quayuser \
-e POSTGRES_PASSWORD=quaypass \
-e POSTGRES_DATABASE=quay \
-e POSTGRES_ADMIN_PASSWORD=adminpass \
-p 5432:5432 \
-v $QUAY/postgres-quay:/var/lib/pgsql/data:Z \
registry.redhat.io/rhel8/postgresql-10:1
```

3. Quay で必要な Postgres **pg_trgm** モジュールがインストールされていることを確認します。

```
$ sudo podman exec -it postgresql-quay /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS pg_trgm" | psql -d quay -U postgres'
```

2.4. REDIS の設定

Redis は、ライブビルダーログおよび Red Hat Quay チュートリアル用に Quay によって使用される key-value ストアです。

2.4.1. Redis のセットアップ

- ポートとパスワードを指定して、**podman run** を使用して **Redis** を起動します。

```
$ sudo podman run -d --rm --name redis \
-p 6379:6379 \
-e REDIS_PASSWORD=strongpassword \
registry.redhat.io/rhel8/redis-5:1
```

2.5. RED HAT QUAY の設定

Red Hat Quay サービスを実行する前に、レジストリー設定、データベース、および Redis 接続パラメーターを含むすべてのコンポーネントの詳細を示す設定ファイルを生成する必要があります。

1. 設定ファイルを生成するには、文字列 **secret** などのパスワードを指定して、**config** モードで **Quay** コンテナを実行します。

```
$ sudo podman run --rm -it --name quay_config -p 80:8080 -p 443:8443 registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

2. ブラウザーを使用して、**http://quay-server.example.com** で設定ツールのユーザーインターフェイスにアクセスします。本書では、**quay-server.example.com** ホスト名を **/etc/hosts** ファイルに設定していることに留意してください。
3. ユーザー名 **quayconfig** およびパスワード **secret** (または上記の **podman run** コマンドで指定した値) を使用してログインします。

2.5.1. Red Hat Quay のセットアップ

設定エディターで、以下の情報を入力します。

- 基本設定
- サーバー設定
- データベース
- Redis

2.5.1.1. 基本設定

基本的な設定設定で、レジストリーのタイトルおよびレジストリーの短いタイトルフィールドに入力します。デフォルト値が設定されていればそのデフォルト値を使用できます。

2.5.1.2. サーバー設定

レジストリーにアクセスできるネットワーク上の場所について HTTP ホストおよびポートを指定します。本書の手順に従っている場合は、**quay-server.example.com** を入力します。

2.5.1.3. データベース

データベースセクションで、Red Hat Quay がメタデータを保存するために使用するデータベースの接続の詳細を指定します。本書の手順に従って概念実証用のシステムをデプロイする場合は、以下の値を入力します。

- **Database Type:** Postgres
- **Database Server:** quay-server.example.com:5432
- **Username:** quayuser
- **Password:** quaypass
- **Database Name:** quay

2.5.1.4. Redis

Redis のキー/値ストアは、リアルタイムイベントとビルドログを保存するために使用されます。本書の手順に従って概念実証用のシステムをデプロイする場合は、以下の値を入力します。

- **Redis Hostname:** quay-server.example.com
- **Redis port:** 6379 (デフォルト)
- **Redis password:** strongpassword

2.5.2. 設定の検証およびダウンロード

すべての必須フィールドが設定されたら、**Validate Configuration Changes** ボタンを選択して、設定を検証します。エラーが報告される場合はすべての必須フィールドが有効となり、Red Hat Quay がデータベースおよび Redis サーバーに接続できるまで設定の編集を続けます。

設定が有効である場合は、**configuration** ファイルをダウンロードします。設定エディターを実行している **Quay** コンテナを停止します。

2.6. RED HAT QUAY のデプロイ

2.6.1. 前提条件

- Quay データベースおよび Redis サーバーが実行中である。
- 有効な設定バンドルを生成している。
- 設定エディターの実行に使用した **Quay** コンテナを停止している。

2.6.2. 設定フォルダーの準備

- 設定バンドルをデプロイし、Quay がこれを使用できるようにします。

```
$ mkdir $QUAY/config
$ cp ~/Downloads/quay-config.tar.gz $QUAY/config
$ cd $QUAY/config
$ tar xvf quay-config.tar.gz
```

2.6.3. イメージデータ用のローカルストレージの準備

- この概念実証用のデプロイメントでは、ローカルファイルシステムを使用してレジストリーイメージを保存します。

```
$ mkdir $QUAY/storage
$ setfacl -m u:1001:-wx $QUAY/storage
```

2.6.4. Red Hat Quay レジストリーのデプロイ

- **podman run** を使用して **Quay** コンテナを起動します。設定データ用の適切なボリュームを指定し、イメージデータのローカルストレージを指定します。

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.7.13
```

2.7. RED HAT QUAY の使用

以下の手順を実行すると、ユーザーインターフェイスを使用して、新しい組織およびリポジトリを作成し、既存のリポジトリを検索および参照できます。手順 3 の後に、コマンドラインインターフェイスを使用してレジストリーと対話し、イメージのプルおよびプッシュを実行できます。

1. ブラウザーを使用して、**http://quay-server.example.com** で Red Hat Quay レジストリーのユーザーインターフェイスにアクセスします (**quay-server.example.com** を **/etc/hosts** ファイルのホスト名として設定していることを前提とします)。
2. **Create Account** をクリックし、ユーザーを追加します (例: **quayadmin** とパスワード **password**)。
3. コマンドラインで、レジストリーにログインします。

```
$ sudo podman login --tls-verify=false quay-server.example.com
Username: quayadmin
Password: password
Login Succeeded!
```

2.7.1. イメージのプッシュおよびプル

1. Red Hat Quay レジストリーからイメージのプッシュおよびプルをテストするには、まず外部レジストリーからサンプルイメージをプルします。

```
$ sudo podman pull busybox
Trying to pull docker.io/library/busybox...
Getting image source signatures
Copying blob 4c892f00285e done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2. **podman images** コマンドを使用して、ローカルコピーを表示します。

```
$ sudo podman images
REPOSITORY          TAG      IMAGE ID      CREATED      SIZE
docker.io/library/busybox  latest  22667f53682a  14 hours ago  1.45 MB
...
```

3. このイメージにタグを付け、これを Red Hat Quay レジストリーにプッシュできるようにします。

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

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

```
$ sudo podman push --tls-verify=false quay-server.example.com/quayadmin/busybox:test
Getting image source signatures
Copying blob 6b245f040973 done
```

```
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
```

5. コマンドラインからイメージへのアクセスをテストするには、まずイメージのローカルコピーを削除します。

```
$ sudo podman rmi quay-server.example.com/quayadmin/busybox:test
Untagged: quay-server.example.com/quayadmin/busybox:test
```

6. 今度は Red Hat Quay レジストリーからイメージを再度プルします。

```
$ sudo podman pull --tls-verify=false quay-server.example.com/quayadmin/busybox:test
Trying to pull quay-server.example.com/quayadmin/busybox:test...
Getting image source signatures
Copying blob 6ef22a7134ba [-----] 0.0b / 0.0b
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

第3章 RED HAT QUAY の高度なデプロイメント

3.1. SSL を使用した RED HAT QUAY への接続の保護

3.1.1. SSL の使用について

[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。

以下の例では、`/etc/hosts` ファイルにエントリーを追加するなど、DNS または別の命名メカニズムを使用してサーバーホスト名 `quay-server.example.com` を設定していることを前提としています。

```
$ cat /etc/hosts
...
192.168.1.112 quay-server.example.com
```

3.1.2. 認証局を作成して証明書に署名

この手順の最後に、証明書ファイルと、`ssl.cert` および `ssl.key` という名前のプライマリーキーファイルがあります。

3.1.2.1. 認証局の作成

1. ルート CA キーを生成します。

```
$ openssl genrsa -out rootCA.key 2048
```

2. ルート CA 証明書を生成します。

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

3. サーバーのホスト名など、証明書の要求に組み込まれる情報を入力します。以下に例を示します。

```
Country Name (2 letter code) [XX]:IE
State or Province Name (full name) []:GALWAY
Locality Name (eg, city) [Default City]:GALWAY
Organization Name (eg, company) [Default Company Ltd]:QUAY
Organizational Unit Name (eg, section) []:DOCS
Common Name (eg, your name or your server's hostname) []:quay-server.example.com
```

3.1.2.2. 証明書に署名

1. サーバーキーを生成します。

```
$ openssl genrsa -out ssl.key 2048
```

2. 署名要求を生成します。

```
$ openssl req -new -key ssl.key -out ssl.csr
```

3. サーバーのホスト名など、証明書の要求に組み込まれる情報を入力します。以下に例を示します。

```
Country Name (2 letter code) [XX]:IE
State or Province Name (full name) []:GALWAY
Locality Name (eg, city) [Default City]:GALWAY
Organization Name (eg, company) [Default Company Ltd]:QUAY
Organizational Unit Name (eg, section) []:DOCS
Common Name (eg, your name or your server's hostname) []:quay-server.example.com
```

4. 以下のようにサーバーのホスト名を指定して、設定ファイルの **openssl.cnf** を作成します。

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 = quay-server.example.com
IP.1 = 192.168.1.112
```

5. 設定ファイルを使用して、証明書 **ssl.cert** を生成します。

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

3.1.3. UI を使用した SSL の設定

このセクションでは、Quay UI を使用して SSL を設定します。コマンドラインインターフェイスを使用して SSL を設定するには、以下のセクションを参照してください。

1. **Quay** コンテナを設定モードで起動します。

```
$ sudo podman run --rm -it --name quay_config -p 80:8080 -p 443:8443
registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

2. Server Configuration セクションで、TLS に **Red Hat Quay handle TLS** を選択します。先に作成した証明書ファイルとプライベートキーファイルをアップロードし、証明書の作成時に Server Hostname が使用された値と一致することを確認します。更新された設定を検証およびダウンロードします。
3. **Quay** コンテナを停止し、レジストリーを再起動します。

```
$ sudo podman rm -f quay
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13
```

3.1.4. コマンドラインを使用した SSL の設定

別の方法として、コマンドラインインターフェイスを使用できます。

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** コンテナを停止し、レジストリーを再起動します。

```
$ sudo podman rm -f quay
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.7.13
```

3.1.5. コマンドラインを使用した SSL 設定のテスト

- **podman login** コマンドを使用して、SSL が有効になっている Quay レジストリーへのログインを試みます。

```
$ sudo podman login quay-server.example.com
Username: quayadmin
Password:
```

```
Error: error authenticating creds for "quay-server.example.com": error pinging docker registry
quay-server.example.com: Get "https://quay-server.example.com/v2/": x509: certificate
signed by unknown authority
```

- Podman は自己署名証明書を信頼しません。回避策として、**--tls-verify** オプションを使用します。

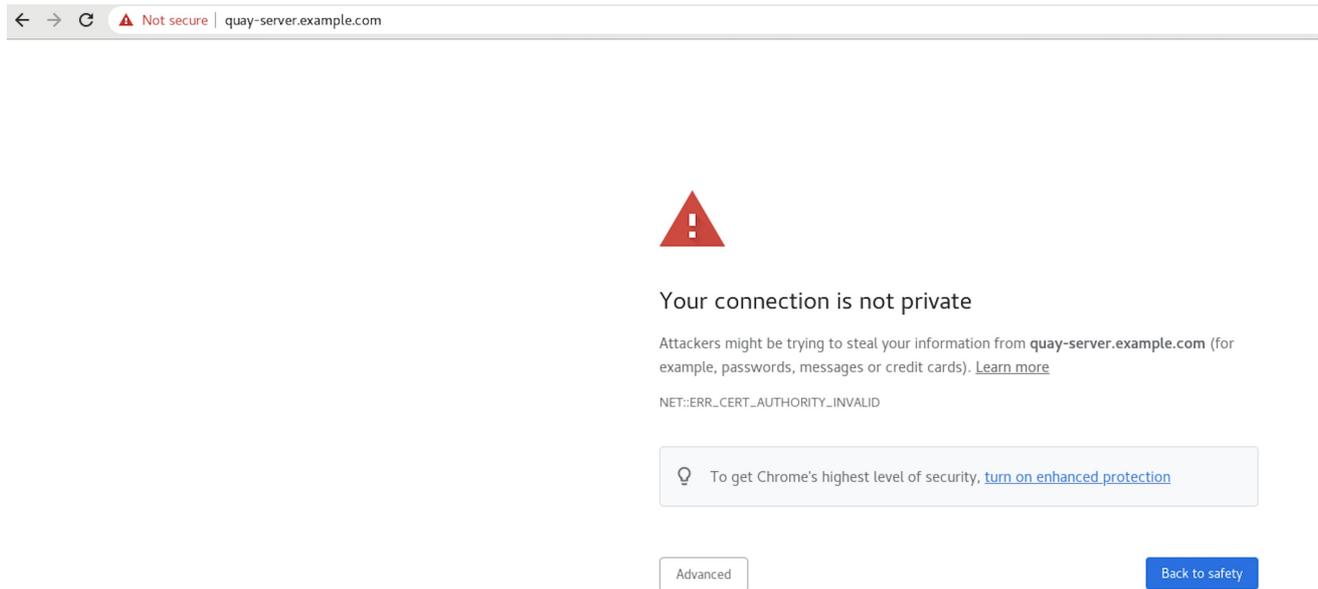
```
$ sudo podman login --tls-verify=false quay-server.example.com
Username: quayadmin
Password:
```

```
Login Succeeded!
```

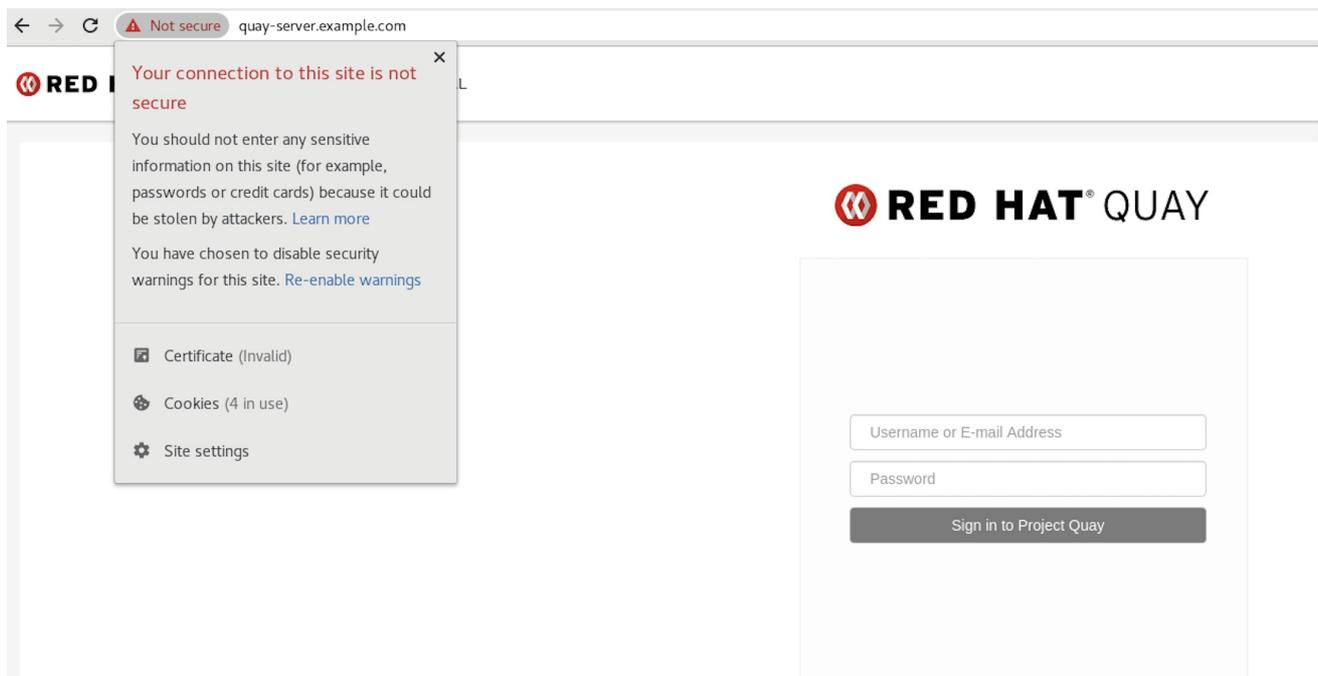
ルート認証局 (CA) を信頼するように Podman を設定する方法は、後続のセクションで説明します。

3.1.6. ブラウザーを使用した SSL 設定のテスト

Quay レジストリーへのアクセスを試みると (この場合は <https://quay-server.example.com>)、ブラウザーは潜在的なリスクについて警告します。



画面にログインすると、ブラウザーは接続が安全ではないことを通知します。



ルート認証局 (CA) を信頼するようにシステムを設定する方法は、後続のセクションで説明します。

3.1.7. 認証局を信頼するように podman を設定する

Podman は、`/etc/containers/certs.d/` および `/etc/docker/certs.d/` の 2 つのパスを使用して CA ファイルを見つけます。

- ルート CA ファイルをこれらの場所のいずれかにコピーし、サーバーのホスト名によって判別されるパスを使用して、`ca.crt` ファイルに名前を付けます。

```
$ sudo cp rootCA.pem /etc/containers/certs.d/quay-server.example.com/ca.crt
```

- または、Docker を使用している場合は、ルート CA ファイルを同等の Docker ディレクトリーにコピーします。

```
$ sudo cp rootCA.pem /etc/docker/certs.d/quay-server.example.com/ca.crt
```

レジストリーにログインする際に、**--tls-verify=false** オプションを使用する必要がなくなります。

```
$ sudo podman login quay-server.example.com
```

```
Username: quayadmin
```

```
Password:
```

```
Login Succeeded!
```

3.1.8. 認証局を信頼するようにシステムを設定

1. ルート CA ファイルを統合されたシステム全体のトラストストアにコピーします。

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

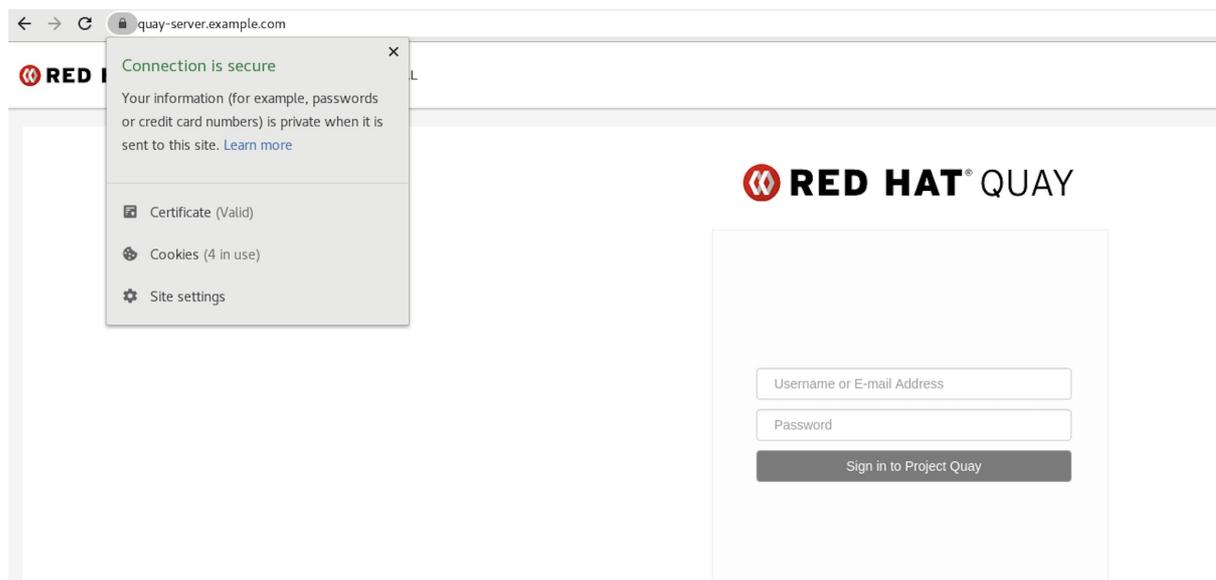
2. システム全体のトラストストア設定を更新します。

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

3. **trust list** コマンドを使用して、Quay サーバーが設定されていることを確認できます。

```
$ trust list | grep quay
label: quay-server.example.com
```

<https://quay-server.example.com> でレジストリーを参照すると、接続が安全であることを示すロックアイコンが表示されます。



4. システム全体の信頼からルート CA を削除するには、ファイルを削除し、設定を更新します。

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

```
$ sudo update-ca-trust extract
$ trust list | grep quay
$
```

詳細は、RHEL 8 のドキュメントの [共有システム証明書の使用](#) について参照してください。

3.2. RED HAT QUAY スーパーユーザー

superuser は、以下を実行することができる拡張された特権を持つ Quay ユーザーアカウントです。

- ユーザーの管理
- 組織の管理
- サービスキーの管理
- 変更ログの閲覧
- 使用状況ログのクエリー
- グローバルに表示されるユーザーメッセージの作成

3.2.1. UI を使用したスーパーユーザーの Quay への追加

本セクションでは、Quay UI を使用してスーパーユーザーを追加する方法を説明します。コマンドラインインターフェイスを使用してスーパーユーザーを追加するには、以下のセクションを参照してください。

1. **Quay** コンテナを接続モードで起動し、既存の設定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 \
-p 443:8443 \
-v $QUAY/config:/conf/stack:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

2. UI の **Access Settings** セクションで、**Super Users** フィールドにユーザーの名前 (この例では **quayadmin**) を入力し、**Add** をクリックします。
3. **configuration** ファイルを検証し、ダウンロードしてから、設定モードで実行されている **Quay** コンテナを終了します。**config.yaml** ファイルを設定ディレクトリーに展開し、**Quay** コンテナをレジストリーモードで再起動します。

```
$ sudo podman rm -f quay
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13
```

3.2.2. スーパーユーザーを追加するための config.yaml ファイルの編集

config.yaml ファイルを直接編集してスーパーユーザーを追加することもできます。スーパーユーザーアカウントの一覧は、**SUPER_USERS** フィールドの配列として保存されます。

- コンテナレジストリーが実行中の場合はこれを停止し、**SUPER_USERS** 配列を **config.yaml** ファイルに追加します。

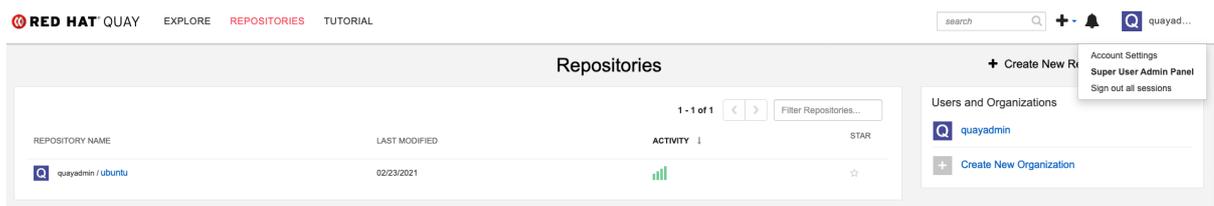
```
SERVER_HOSTNAME: quay-server.example.com
SETUP_COMPLETE: true
SUPER_USERS:
  - quayadmin
...
```

3.2.3. スーパーユーザー管理パネルへのアクセス

1. Quay レジストリーを再起動します。

```
$ sudo podman rm -f quay
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13
```

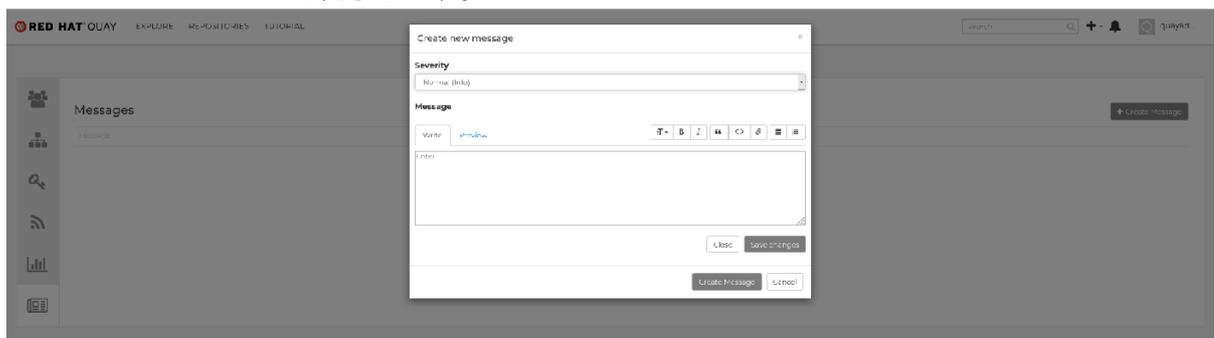
2. Super User Admin パネルにアクセスするには、UI の右上にある現在のユーザーの名前またはアバターをクリックします。ユーザーがスーパーユーザーとして追加されている場合は、Super User Admin Panel というドロップダウンリストに追加のアイテムが表示されます。



3.2.3.1. グローバルに表示されるユーザーメッセージの作成

Superuser Admin Panel を使用すると、組織の **Normal**、**Warning**、または **Error** メッセージを作成できます。

1. UI の右上でユーザー名をクリックします。 **Super User Admin Panel** を選択します。
2. Red Hat Quay Management ページの、左側のペインで **Globally visible user messages** をクリックします。
3. **Create Message** をクリックし、**Normal**、**Warning**、および **Error** メッセージタイプを含むドロップダウンメニューを表示します。



4. メッセージの入力は、**Click to set message** を選択してから、**Create Message** をクリックして実行します。

メッセージの削除には、**Options** をクリックし、続いて **Delete Message** をクリックします。

3.3. リポジトリのミラーリング

3.3.1. リポジトリのミラーリング

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

リポジトリ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。

- 外部のレジストリーからミラーリングするリポジトリを選択する。
- 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。
- 同期する特定のコンテナイメージリポジトリ名とタグを特定する。
- リポジトリが同期される間隔を設定する。
- 同期の現在の状態を確認する。

ミラーリング機能を使用するには、以下を行う必要があります。

- Red Hat Quay 設定でリポジトリのミラーリングを有効にします。
- リポジトリミラーリングワーカーを実行する。
- ミラーリングされたリポジトリを作成します。

すべてのリポジトリミラーリングの設定は、コンフィグレーションツールの UI または Red Hat Quay API を使用して行うことができます。

3.3.2. 設定のミラーリング UI

1. 設定モードで **Quay** コンテナを起動し、Enable Repository Mirroring チェックボックスを選択します。HTTPS 通信を必要とし、ミラーリング時に証明書を検証する必要がある場合は、HTTPS を選択し、証明書の検証のチェックボックスを選択します。

 Repository Mirroring

If enabled, scheduled mirroring of repositories from remote registries will be available.

Enable Repository Mirroring

 A repository mirror service must be running to use this feature. Documentation on setting up and running this service can be found at [Running Repository Mirroring Service](#).

Require HTTPS and verify certificates of Quay registry during mirror.

2. **configuration** を検証し、ダウンロードしてから、更新された設定ファイルを使用してレジストリーモードで Quay を再起動します。

3.3.3. ミラーリングワーカー

次の手順を使用して、リポジトリミラーリングワーカーを開始します。

手順

- `/root/ca.crt` 証明書を使用して TLS 通信を設定していない場合は、次のコマンドを入力し、`repomirror` オプションを指定して、**Quay** Pod を開始します。

```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
{productrepo}/{quayimage}:{productminv} repomirror
```

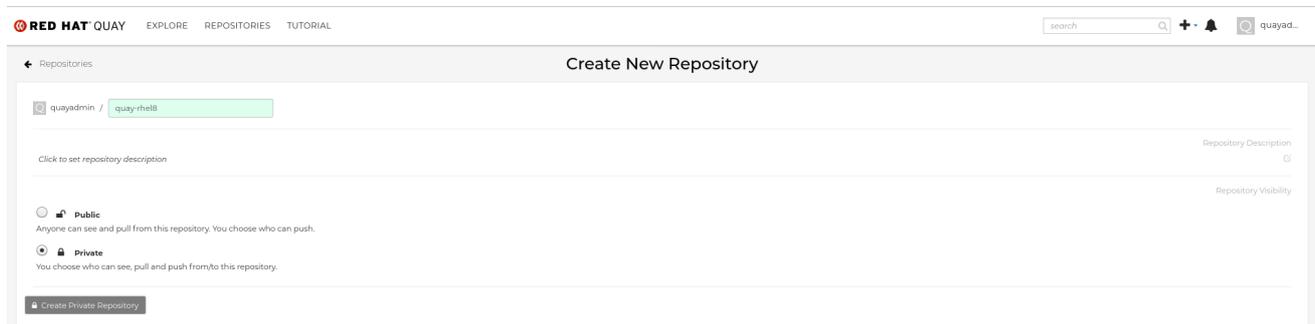
- `/root/ca.crt` 証明書を使用して TLS 通信を設定している場合は、次のコマンドを入力して、リポジトリミラーリングワーカーを開始します。

```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
-v /root/ca.crt:/etc/pki/ca-trust/source/anchors/ca.crt:Z \
{productrepo}/{quayimage}:{productminv} repomirror
```

3.3.4. ミラーリングされたリポジトリの作成

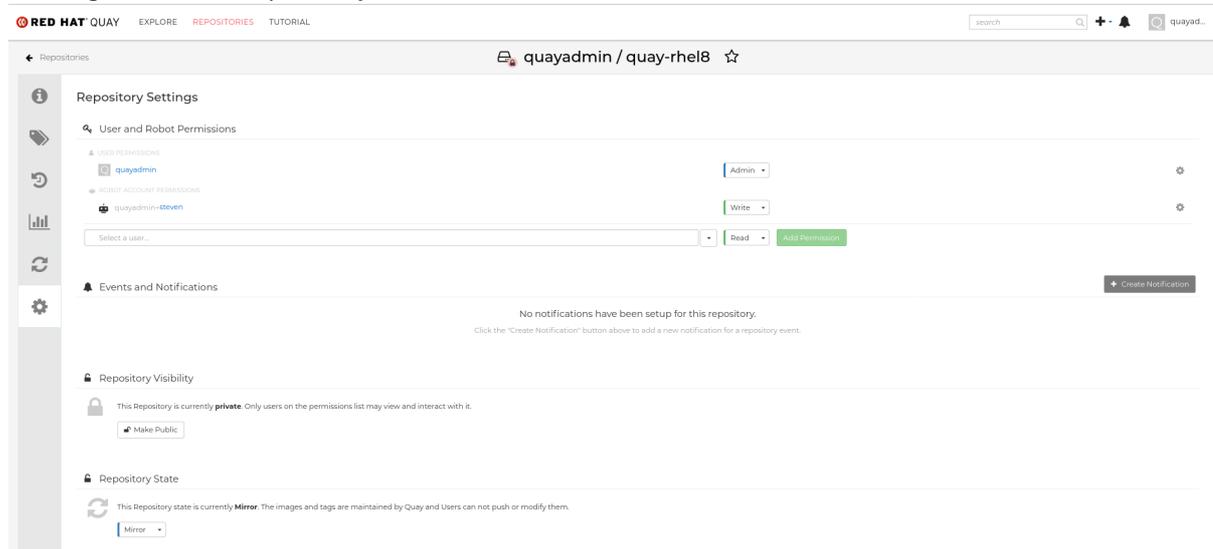
以下のセクションで説明する手順は、Red Hat Quay クラスターの設定でリポジトリのミラーリングが有効にされており、リポジトリミラーリングワーカーをデプロイしていることを前提としています。

外部コンテナレジストリーからリポジトリをミラーリングする場合は、新しいプライベートリポジトリを作成します。通常、ターゲットリポジトリと同じ名前が使用されます (例: `quay-rhel8`)。



3.3.4.1. リポジトリのミラーリングの設定

1. Settings タブで、Repository State を **Mirror** に設定します。



2. Mirror タブで、タグ、スケジューリング、およびアクセス情報と共に外部レジストリーに接続するための情報を入力します。

3. 必要に応じて、以下のフィールドに詳細を入力します。

- **Registry Location:** ミラーリングする外部リポジトリー (例: **registry.redhat.io/quay/quay-rhel8**)。
- **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

Quay がリモートリポジトリーのタグの一覧を取得するには、以下のいずれかの要件を満たす必要があります。

- latest タグのあるイメージがリモートリポジトリー OR に存在している必要があります
 - パターンの一致のない少なくとも1つの明示的なタグが、指定するタグの一覧に存在する必要があります。
- **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。
 - **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:** 新しい robot アカウントを作成するか、既存の robot アカウントを選択してミラーリングを実行します。
 - **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アクセスするためのユーザー名。

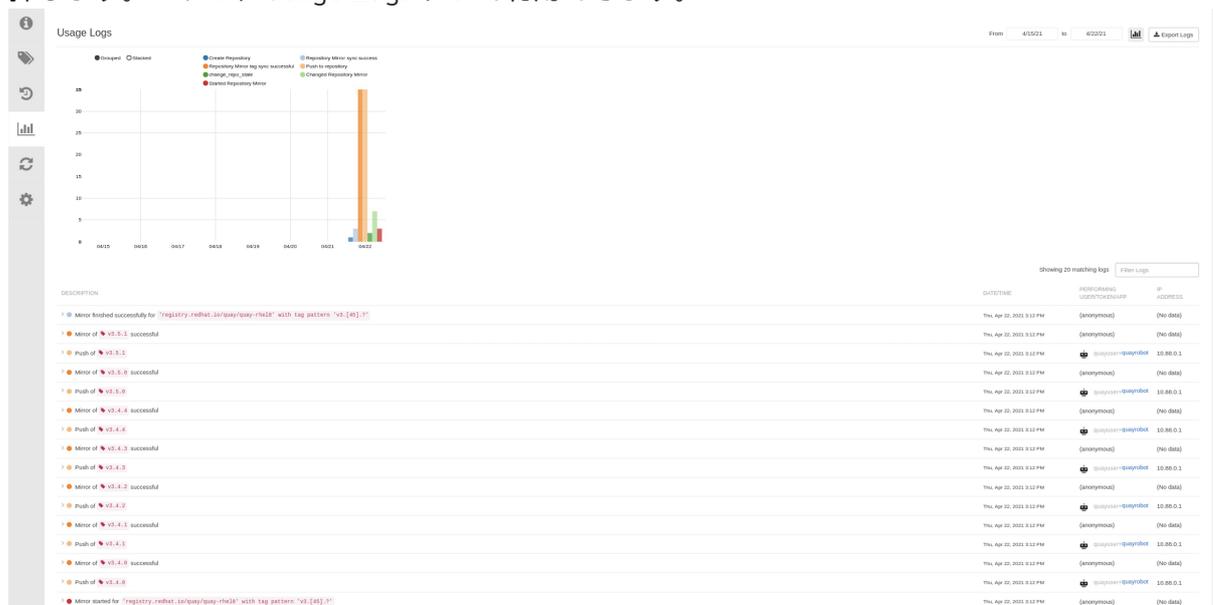
- **Password:** ユーザー名に関連付けられたパスワード。パスワードにはエスケープ文字 (\) を必要とする文字を含めることができないことに注意してください。

3.3.4.2. 詳細設定

- Advanced Settings セクションで、必要な場合は TLS およびプロキシを設定します。
- **Verify TLS:** ターゲットリモートレジストリーと通信する際に、HTTPS が必要であり、証明書を検証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。
- **HTTP Proxy:** リモートサイトへのアクセスに必要な HTTP プロキシサーバーを特定します (必要な場合)。
- **HTTPS Proxy:** リモートサイトへのアクセスに必要な HTTPS プロキシサーバーを特定します (必要な場合)。
- **No Proxy:** プロキシを必要としない場所の一覧。

3.3.4.3. 今すぐ同期する

- 即時にミラーリング操作を実行するには、リポジトリーの Mirroring タブで Sync Now ボタンを押します。ログは、Usage Logs タブで利用できます。

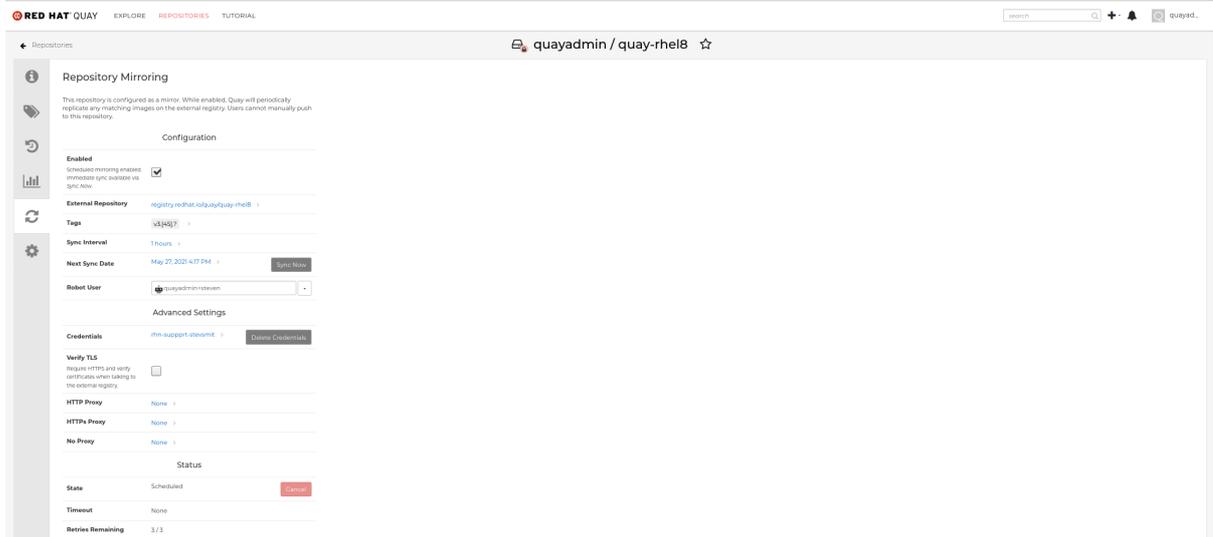


ミラーリングが完了すると、イメージは Tags タブに表示されます。

The screenshot shows the 'Repository Tags' interface for the 'quayuser / quay-rhel8' repository. It displays a list of tags with columns for TAG, LAST MODIFIED, SIZE, EXPIRES, and MANIFEST. The tags listed are v3.5.1, v3.5.0, v3.4.4, v3.4.3, v3.4.2, v3.4.1, and v3.4.0.

TAG	LAST MODIFIED	SIZE	EXPIRES	MANIFEST
<input type="checkbox"/> v3.5.1	a minute ago	N/A	Never	sha256: 3448f7a273
<input type="checkbox"/> v3.5.0	a minute ago	N/A	Never	sha256: 951916a60c
<input type="checkbox"/> v3.4.4	a minute ago	N/A	Never	sha256: ea1c33a29e
<input type="checkbox"/> v3.4.3	a minute ago	N/A	Never	sha256: 36221191f2
<input type="checkbox"/> v3.4.2	a minute ago	N/A	Never	sha256: 768b10e607
<input type="checkbox"/> v3.4.1	a minute ago	N/A	Never	sha256: 49f5c5268e
<input type="checkbox"/> v3.4.0	a minute ago	N/A	Never	sha256: 5e1c488093

以下は、完了した Repository Mirroring 画面の例です。



3.3.5. タグパターンのミラーリング

上述のように、少なくとも1つのタグが明示的に入力されている（つまりタグパターンではない）、またはタグ latest がレポトリポジトリーに存在している必要があります。（タグ latest は、タグリストで指定されていないと同期されません。）これは Quay がリモートリポジトリーのタグリストを取得し、ミラーリングするために指定したリストと比較するために必要です。

3.3.5.1. パターン構文

パターン	説明
*	すべての文字に一致します。
?	任意の単一文字に一致します。
[seq]	seq の任意の文字と一致します。
[!seq]	seq がない文字と一致します。

3.3.5.2. タグのパターン例

パターン例	マッチの例
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

3.4. CLAIR V4 のデプロイ

Clair はイメージのコンテンツを解析し、コンテンツに影響する脆弱性を報告するアプリケーションです。これは静的分析によって実行され、ランタイム時には実行されません。Clair の分析は 3 つの部分に分かれています。

- **Indexing:** インデックス作成はマニフェストの Clair への送信から開始します。受信時に、Clair はレイヤーを取得し、その内容をスキャンし、IndexReport という中間表現を返します。マニフェストは、Clair のコンテナイメージの表示です。Clair は OCI マニフェストとレイヤーがコンテンツ対応であることを活かして、重複作業を削減します。マニフェストがインデックス化されると、IndexReport は後で取得できるように永続化されます。
- **Matching:** マッチングは IndexReport を取り、レポートが表すマニフェストに影響を与える脆弱性を相関させます。Clair は継続的に新しいセキュリティーデータを取り込み、matcher への要求は IndexReport の最新の脆弱性分析を提供します。
- **Notifications:** Clair は通知サービスを実装します。新しい脆弱性が発見されると、通知サービスは、これらの脆弱性がインデックスされたマニフェストに影響するかどうかを判断します。その後、通知機能は設定に応じてアクションを実行します。

3.4.1. Clair 用の個別データベースのデプロイ

Clair には Postgres データベースが必要です。Quay が Postgres を使用している場合に Quay と Clair 間で共通データベースを共有できますが、この例では個別の Clair 固有のデータベースがデプロイされます。

この概念実証のシナリオでは、ローカルファイルシステムのディレクトリーを使用してデータベースデータを永続化します。

1. ここでは \$QUAY 変数で示されるインストールフォルダーで、Clair データベースデータのディレクトリーを作成し、パーミッションを適切に設定します。

```
$ mkdir -p $QUAY/postgres-clairv4
$ setfacl -m u:26:-wx $QUAY/postgres-clairv4
```

2. Podman を使用し、データベースデータのボリューム定義と共にユーザー名、パスワード、データベース名、ポートを指定して Postgres コンテナを実行します。標準の Postgres ポートである **5432** は Quay のデプロイメントですでに使用されているため、別のポート (この例では **5433**) を公開します。

```
$ sudo podman run -d --rm --name postgresql-clairv4 \
-e POSTGRES_USER=clairuser \
-e POSTGRES_PASSWORD=clairpass \
-e POSTGRES_DATABASE=clair \
-e POSTGRES_ADMIN_PASSWORD=adminpass \
-p 5433:5432 \
-v $QUAY/postgres-clairv4:/var/lib/pgsql/data:Z \
registry.redhat.io/rhel8/postgresql-10:1
```

3. Clair で必要な Postgres **uuid-oss** モジュールがインストールされていることを確認します。

```
$ sudo podman exec -it postgresql-clairv4 /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS \"uuid-osp\"\" | psql -d clair -U postgres'
```

3.4.2. Clair の Quay 設定

1. **Quay** コンテナが実行中である場合はこれを停止し、コンテナを設定モードで再起動し、既存の設定をボリュームとして読み込みます。

```
$ sudo podman run --rm -it --name quay_config \
  -p 80:8080 -p 443:8443 \
  -v $QUAY/config:/conf/stack:Z \
  registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

2. 設定ツールにログインし、UI の Security Scanner セクションでスキャンを有効にします。 **quay-server** システムでまだ使用されていないポート (例: **8081**) を使用して、Clair の HTTP エンドポイントを設定します。 **Generate PSK** ボタンを使用して Clair の PSK (事前共有鍵) を作成します。以下はその例です。

- **Security Scanner Endpoint:** `http://quay-server.example.com:8081`

- **Security Scanner PSK:** `MTU5YzA4Y2ZkNzJoMQ==`
スキャナーデータを設定するための UI が以下のイメージに表示されます。

セキュリティスキャナー UI

🔒 Security Scanner

If enabled, all images pushed to Quay will be scanned via the external security scanning service, with vulnerability information available in the UI and API, as well as async notification support.

Enable Security Scanning

i A scanner compliant with the Quay Security Scanning API must be running to use this feature. Documentation on running Clair can be found at [Running Clair Security Scanner](#).

Security Scanner Endpoint:

The HTTP URL at which the security scanner is running.

Security Scanner PSK:

Clair Pre-Shared Key. Make sure to include this value in your Clair config.

3. **configuration** ファイルを検証し、ダウンロードしてから、設定エディターを実行している **Quay** コンテナを停止します。設定バンドルを以前と同様に `$QUAY/config` ディレクトリーに展開します。

```
$ cp ~/Downloads/quay-config.tar.gz $QUAY/config
$ cd $QUAY/config
$ tar xvf quay-config.tar.gz
```

Red Hat Quay 設定ファイルが更新され、セキュリティスキャナーの以下のフィールドが含まれるようになりました。

```
...
FEATURE_SECURITY_NOTIFICATIONS: true
FEATURE_SECURITY_SCANNER: true
...
SECURITY_SCANNER_INDEXING_INTERVAL: 30
SECURITY_SCANNER_V4_ENDPOINT: http://quay-server.example.com:8081
```

```
SECURITY_SCANNER_V4_PSK: MTU5YzA4Y2ZkNzJoMQ==
SERVER_HOSTNAME: quay-server.example.com
...
```

3.4.3. Clair の設定

Clair 設定についての詳細

は、<https://github.com/quay/clair/blob/main/Documentation/reference/config.md> を参照してください。

- `/etc/` ディレクトリーで `config.yaml` ファイルを作成します (例: `/etc/clairv4/config/config.yaml`)。以下の例を使用します。これは、概念実証用のデプロイメントで使用する最小の設定を示しています。

```
http_listen_addr: :8081
introspection_addr: :8089
log_level: debug
indexer:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  max_conn_pool: 100
  run: ""
  migrations: true
  indexer_addr: clair-indexer
notifier:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  delivery_interval: 1m
  poll_interval: 5m
  migrations: true
auth:
  psk:
    key: "MTU5YzA4Y2ZkNzJoMQ=="
    iss: ["quay"]
# tracing and metrics
trace:
  name: "jaeger"
  probability: 1
  jaeger:
    agent_endpoint: "localhost:6831"
    service_name: "clair"
metrics:
  name: "prometheus"
```

- `http_listen_addr` は、Quay 設定ツールで指定した Clair HTTP エンドポイントのポート (この場合は `8081`) に設定します。
- 認証には Quay の設定ツールで生成した Clair の事前共有鍵 (PSK) が使用され、`iss` フィールドで指定した発行者は `quay` に設定されます。

3.4.4. Clair の実行

1. **podman run** コマンドを使用して Clair コンテナを実行し、設定ツールで指定した HTTP エンドポイントポート (ここでは **8081**) を公開します。

```
$ sudo podman run -d --rm --name clairv4 \
  -p 8081:8081 -p 8089:8089 \
  -e CLAIR_CONF=/clair/config.yaml -e CLAIR_MODE=combo \
  -v /etc/clairv4/config:/clair:Z \
  registry.redhat.io/quay/clair-rhel8:v3.7.13
```

2. 次に、スキャナー設定が含まれる更新された設定ファイルを使用して **Quay** コンテナを再起動します。

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.7.13
```

3.4.5. Clair セキュリティースキャンの使用

1. コマンドラインで、レジストリーにログインします。

```
$ sudo podman login --tls-verify=false quay-server.example.com
Username: quayadmin
Password:
Login Succeeded!
```

2. サンプルイメージをプルし、これにタグを付け、レジストリーにプッシュします。

```
$ sudo podman pull ubuntu:20.04
$ sudo podman tag docker.io/library/ubuntu:20.04 quay-
server.example.com/quayadmin/ubuntu:20.04
$ sudo podman push --tls-verify=false quay-server.example.com/quayadmin/ubuntu:20.04
```

以下のイメージが示すように、セキュリティースキャンの結果が Quay UI に表示されます。

スキャンの要約

The screenshot shows the Quay UI interface for the 'quayadmin / ubuntu' repository. The 'Repository Tags' section displays a table with the following data:

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
20.04	a minute ago	2 Medium	28.6 MB	Never	SHA256:b2c5c6eebc86

スキャンの詳細

RED HAT QUAY EXPLORE REPOSITORIES TUTORIAL

search + quayad...

quayadmin/ubuntu b2c5c6eebc06

Quay Security Scanner has detected 37 vulnerabilities.

- 2 Medium-level vulnerabilities.
- 27 Low-level vulnerabilities.
- 8 Negligible-level vulnerabilities.

Vulnerabilities

CVE	SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN LAYER
CVE-2018-20839 on Ubuntu 20.04 (focal) - medium	Medium	libsystemd0	245.4-4ubuntu3.4	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2018-20839 on Ubuntu 20.04 (focal) - medium	Medium	libudev1	245.4-4ubuntu3.4	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2019-18276 on Ubuntu 20.04 (focal) - low	Low	bash	5.0-6ubuntu1.1	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2017-18018 on Ubuntu 20.04 (focal) - low	Low	coreutils	8.30-3ubuntu2	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2016-2781 on Ubuntu 20.04 (focal) - low	Low	coreutils	8.30-3ubuntu2	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2019-13050 on Ubuntu 20.04 (focal) - low	Low	gpgv	2.2.19-3ubuntu2	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2019-25013 on Ubuntu 20.04 (focal) - low	Low	libc-bin	2.31-0ubuntu9.1	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...
CVE-2020-27618 on Ubuntu 20.04 (focal) - low	Low	libc-bin	2.31-0ubuntu9.1	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d...

3.4.6. 脆弱性情報データベース (NVD: National Vulnerability Database) の CVE 評価

Clair v4.2 では、エンリッチメントデータを Quay UI で表示できるようになりました。さらに、Clair v4.2 は、検出された脆弱性について National Vulnerability Database から CVSS スコアを追加します。

この変更により、脆弱性の CVSS スコアがディストリビューションのスコアの 2 レベル以内である場合は、デフォルトで Quay UI present が、ディストリビューションのスコアになります。以下に例を示します。

DESCRIPTION

The SUSE coreutils-118n_patch for GNU coreutils allows context-dependent attackers to cause a denial of service (segmentation fault and crash) via a long string to the uniq command, which triggers a stack-based buffer overflow in the alloca function.

CVE-2015-4041 Unknown coreutils 8.30-3 0.0 ADD roots.tar / # buildkit

SEVERITY NOTE

Note that this vulnerability was originally given a CVSSv3 score of 7.8 by NVD but was subsequently reclassified as Unknown by Unknown

VECTORS

Attack Vector	Attack Complexity	Privileges Required	User Interaction	Scope	Confidentiality Impact	Integrity Impact	Availability Impact
Local	Low	None	None	Unchanged	High	High	High

DESCRIPTION

The keycompare_mb function in sort.c in sort in GNU Coreutils through 8.23 on 64-bit platforms performs a size calculation without considering the number of bytes occupied by multibyte characters, which allows attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via long UTF-8 strings.

これは以前のインターフェイスとは異なり、以下の情報のみを表示します。

CVE-2015-4041 Unknown coreutils 8.30-3 0.0 ADD roots.tar / # buildkit

DESCRIPTION

The keycompare_mb function in sort.c in sort in GNU Coreutils through 8.23 on 64-bit platforms performs a size calculation without considering the number of bytes occupied by multibyte characters, which allows attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via long UTF-8 strings.

3.5. コンテナの起動

--restart オプションは podman で完全にサポートされていないため、Podman を使用した systemd へのコンテナの移植で説明されているように、podman を systemd サービスとして設定できます。

3.5.1. Podman での systemd ユニットファイルの使用

デフォルトで、Podman は既存のコンテナまたは Pod のユニットファイルを生成します。podman generate systemd --new を使用して、移植可能な別の systemd ユニットファイルを生成できます。--new フラグでは、コンテナの作成、起動、削除を行うユニットファイルを生成するように Podman

に指示します。

1. 以下のように、実行中の Red Hat Quay レジストリーから systemd ユニットファイルを作成します。

```
$ sudo podman generate systemd --new --files --name redis
$ sudo podman generate systemd --new --files --name postgresql-quay
$ sudo podman generate systemd --new --files --name quay
$ sudo podman generate systemd --new --files --name postgresql-clairv4
$ sudo podman generate systemd --new --files --name clairv4
```

2. **/usr/lib/systemd/system** にユニットファイルをコピーして root ユーザーとしてインストールします。

```
$ sudo cp -Z container-redis.service /usr/lib/systemd/system
$ sudo cp -Z container-postgresql-quay.service /usr/lib/systemd/system
$ sudo cp -Z container-quay.service /usr/lib/systemd/system
$ sudo cp -Z container-postgresql-clairv4.service /usr/lib/systemd/system
$ sudo cp -Z container-clairv4.service /usr/lib/systemd/system
```

3. systemd マネージャーの設定を再読み込みするには、次のコマンドを実行します。

```
$ sudo systemctl daemon-reload
```

4. サービスを有効にし、システムの起動時に起動します。

```
$ sudo systemctl enable --now container-redis.service
$ sudo systemctl enable --now container-postgresql-quay.service
$ sudo systemctl enable --now container-quay.service
$ sudo systemctl enable --now container-postgresql-clairv4.service
$ sudo systemctl enable --now container-clairv4.service
```

3.5.2. サービスの起動、停止、およびステータスのチェック

1. Quay コンポーネントのステータスを確認します。

```
$ sudo systemctl status container-redis.service
$ sudo systemctl status container-postgresql-quay.service
$ sudo systemctl status container-quay.service
$ sudo systemctl status container-postgresql-clairv4.service
$ sudo systemctl status container-clairv4.service
```

2. Quay コンポーネントサービスを停止するには、以下を実行します。

```
$ sudo systemctl stop container-redis.service
$ sudo systemctl stop container-postgresql-quay.service
$ sudo systemctl stop container-quay.service
$ sudo systemctl stop container-postgresql-clairv4.service
$ sudo systemctl stop container-clairv4.service
```

3. Quay コンポーネントサービスを起動するには、以下を実行します。

```
$ sudo systemctl start container-redis.service
```

```
$ sudo systemctl start container-postgresql-quay.service
$ sudo systemctl start container-quay.service
$ sudo systemctl start container-postgresql-clairv4.service
$ sudo systemctl start container-clairv4.service
```

3.5.3. 再起動後の再起動のテスト

サービスを設定して有効にしたら、システムを再起動します。システムを再起動したら、**podman ps** を使用して Quay コンポーネントのすべてのコンテナが再起動されていることを確認します。

```
$ sudo podman ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
4e87c7889246 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 19 seconds ago Up 18
seconds ago 0.0.0.0:5432->5432/tcp postgresql-quay
b8fbac1920d4 registry.redhat.io/rhel8/redis-5:1 run-redis 19 seconds ago Up 18 seconds
ago 0.0.0.0:6379->6379/tcp redis
d959d5bf7a24 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 18 seconds ago Up 18
seconds ago 0.0.0.0:5433->5432/tcp postgresql-clairv4
e75ff8651dbd registry.redhat.io/quay/clair-rhel8:v3.4.0 18 seconds ago Up 17 seconds
ago 0.0.0.0:8081->8080/tcp clairv4
```

この場合、**Quay** コンテナ自体は起動できませんでした。これは、セキュリティースキャンが Quay で有効にされている場合に、起動時に Clair への接続を試みるためです。ただし、Clair は初期化を完了せず、接続を受け入れることができないため、結果として Quay はすぐに終了します。この問題を解決するには、以下のセクションにあるように、Quay サービスを Clair サービスの依存関係を持つように設定する必要があります。

3.5.4. Clair の Quay の依存関係の設定

Quay の **systemd** サービスファイルで、**After=container-clairv4.service** を設定して、**[Unit]** セクションに Clair サービスの依存関係を設定します。Clair コンテナの初期化に時間を指定するには **[Service]** セクションに遅延を追加します (**RestartSec=30** など)。以下は、Clair の依存関係を設定した後に変更された Quay ファイルの例です。

/usr/lib/systemd/system/container-quay.service

```
# container-quay.service
# autogenerated by Podman 2.0.5
# Tue Feb 16 17:02:26 GMT 2021

[Unit]
Description=Podman container-quay.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=container-clairv4.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
RestartSec=30
ExecStartPre=/bin/rm -f %t/container-quay.pid %t/container-quay.ctr-id
ExecStart=/usr/bin/podman run --common-pidfile %t/container-quay.pid --cidfile %t/container-quay.ctr-id --cgroups=no-common -d --rm -p 8080:8080 --name=quay -v
```

```

/home/user1/quay/config:/conf/stack:Z -v /home/user1/quay/storage:/datastorage:Z
registry.redhat.io/quay/quay-rhel8:v3.4.0
ExecStop=/usr/bin/podman stop --ignore --cidfile %t/container-quay.ctr-id -t 10
ExecStopPost=/usr/bin/podman rm --ignore -f --cidfile %t/container-quay.ctr-id
PIDFile=%t/container-quay.pid
KillMode=none
Type=forking

[Install]
WantedBy=multi-user.target default.target

```

Quay サービス設定を更新したら、サーバーを再起動して **podman ps** をすぐに実行します。

```

$ sudo podman ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
4e87c7889246 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 29 seconds ago Up 28
seconds ago 0.0.0.0:5432->5432/tcp postgresql-quay
b8fbac1920d4 registry.redhat.io/rhel8/redis-5:1 run-redis 29 seconds ago Up 28 seconds
ago 0.0.0.0:6379->6379/tcp redis
d959d5bf7a24 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 28 seconds ago Up 28
seconds ago 0.0.0.0:5433->5432/tcp postgresql-clairv4
e75ff8651dbd registry.redhat.io/quay/clair-rhel8:v3.4.0 28 seconds ago Up 27 seconds
ago 0.0.0.0:8081->8080/tcp clairv4

```

最初は **Quay** コンテナは利用できませんが、**RestartSec** の遅延の期限が切れると、起動するはずで

```

$ sudo podman ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
4e87c7889246 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 35 seconds ago Up 34
seconds ago 0.0.0.0:5432->5432/tcp postgresql-quay
ab9f0e6ad7c3 registry.redhat.io/quay/quay-rhel8:v3.4.0 registry 3 seconds ago Up 2 seconds
ago 0.0.0.0:8080->8080/tcp quay
b8fbac1920d4 registry.redhat.io/rhel8/redis-5:1 run-redis 35 seconds ago Up 34 seconds
ago 0.0.0.0:6379->6379/tcp redis
d959d5bf7a24 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 34 seconds ago Up 34
seconds ago 0.0.0.0:5433->5432/tcp postgresql-clairv4
e75ff8651dbd registry.redhat.io/quay/clair-rhel8:v3.4.0 34 seconds ago Up 33 seconds
ago 0.0.0.0:8081->8080/tcp clairv4

```

Quay コンテナの **CREATED** フィールドには、サービス定義に設定されるように作成時間に 30 秒の差異が示されます。

quay-server.example.com で Red Hat Quay レジストリーにログインし、すべてが正常に再起動されていることを確認します。

3.6. FIPS の準備状態およびコンプライアンス

FIPS (NIST (National Institute of Standards and Technology) が開発した Federal Information Processing Standard) は、特に銀行、医療、公共部門などの厳しく規制された分野で、機密データを保護および暗号化するための絶対的基準と見なされています。Red Hat Enterprise Linux および Red Hat

OpenShift Container Platform は、システムが **openssl** などの特定の FIPS 検証済み暗号モジュールの使用のみを許可する FIPS モードを提供することで、この標準をサポートします。これにより、FIPS への準拠が保証されます。

Red Hat Quay は、FIPS 対応の RHEL および Red Hat OpenShift Container Platform バージョン 3.5 の実行をサポートしています。

第4章 次のステップ

本書では、概念実証用の Red Hat Quay を設定し、デプロイする方法を説明します。実稼働環境へのデプロイに関する詳細は、Red Hat Quay のデプロイ - 高可用性 (HA) を参照してください。

Red Hat Quay の使用ガイドでは、以下の方法について説明しています。

- ユーザーおよびリポジトリの追加
- タグの使用
- ビルドワーカーを使用した Dockerfile の自動ビルド
- GitLab ビルドトリガー
- リポジトリイベントの通知の追加

Red Hat Quay の管理ガイドでは、以下の方法について説明しています。

- SSL および TLS の使用
- Clair によるセキュリティスキャンの有効化
- リポジトリミラーリングの使用
- LDAP 認証の設定
- ストレージの Georeplication の使用