Red Hat Quay 3.5

Quay Operator の使用による OpenShift への Red Hat Quay のデプロイ

Quay Operator の使用による OpenShift への Red Hat Quay のデプロイ

Last Updated: 2021-10-04

Red Hat Quay 3.5 Quay Operator の使用による OpenShift への Red Hat Quay のデプロイ

Quay Operator の使用による OpenShift への Red Hat Quay のデプロイ

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 Operator の使用による OpenShift クラスターへの Red Hat Quay のデプロイ

目次

序文 4

第1章 OPENSHIFT での RED HAT QUAY の前提条件 5

第2章 QUAY OPERATOR のインストール 6
2.1. 以前のバージョンとの相違点 6
2.2. QUAY OPERATOR のインストール前に 6
2.2.1. ストレージソリューションの決定 6
2.2.2. スタンドアロン Object Gateway について 7
2.2.3. スタンドアロン Object Gateway の作成 7
2.3. OPERATORHUB からの OPERATOR のインストール 9

第3章 高レベルの概念 10
3.1. QUAYREGISTRY API 10
3.2. QUAY コンポーネント 10
3.3. 管理コンポーネントの使用 11
3.4. 依存関係向けの管理対象外コンポーネントの使用 11
3.4.1. 既存の Postgres データベースの使用 12
3.4.2. 管理対象外ストレージ 12
3.4.3. Horizontal Pod Autoscaler の無効化 13
3.5. 設定バンドルシークレット 14
3.6. QUAYREGISTRY ステータス 14
3.6.1. レジストリーエンドポイント 14
3.6.2. 設定エディターのエンドポイント 14
3.6.3. 設定エディターの認証情報シークレット 14
3.6.4. 現行バージョン 14
3.6.5. 条件 14

第4章 QUAY OPERATOR を使用した QUAY のデプロイ 15
4.1. QUAY レジストリーの作成 15
4.1.1. OpenShift Console 15
4.1.2. コマンドライン 15
4.2. インフラストラクチャーノードでの QUAY のデプロイ 16
4.2.1. インフラストラクチャーに使用するノードのラベルおよびテイント 16
4.2.2. ノードセレクターおよび容認を使用したプロジェクトの作成 17
4.2.3. Quay Operator の namespace へのインストール 17
4.2.4. レジストリーの作成 17

第5章 QUAY OPERATOR を使用した QUAY のアップグレード 19
5.1. OPERATOR LIFECYCLE MANAGER 19
5.2. QUAY OPERATOR のアップグレードによる QUAY のアップグレード 19
5.2.1. Quay のアップグレード 19
5.2.2. Operator の更新チャネルの変更 19
5.2.3. 保留中の Operator アップグレードの手動による承認 20
5.3. QUAYREGISTRY のアップグレード 20
5.4. QUAY 3.5 の新機能の有効化 21
5.4.1. コンソールでのモニタリングおよびアラート 21
5.4.2. OCI および Helm サポート 21
5.5. QUAYECOSYSTEM のアップグレード 21
5.5.1. QuayEcosystem アップグレードを元に戻す 22
5.5.2. アップグレードでサポートされる QuayEcosystem 設定 22

第6章 QUAY OPERATOR の機能 24

6.1. HELM OCI サポートおよび RED HAT QUAY 24
6.1.1. Helm および OCI の前提条件 24
6.1.2. Quay での Helm チャートの使用 25
6.1.3. OCI および Helm の設定 26
6.1.4. Operator を使用した OCI および Helm の設定 27
6.2. コンソールでのモニタリングおよびアラート 28
6.2.1. ダッシュボード 28
6.2.2. メトリクス 30
6.2.3. アラート 32
6.3. FIPS の READINESS およびコンプライアンス 32

第7章 高度な概念 34
7.1. QUAY デプロイメントのカスタマイズ 34
7.1.1. Quay アプリケーション設定 34
7.1.2. レジストリーへの外部アクセスのカスタマイズ 34
7.1.2.1. カスタムのホスト名および TLS の使用 34
7.1.2.2. OpenShift で提供される TLS 証明書の使用 35
7.1.3. Route コンポーネントの無効化 36
7.1.4. 管理ストレージのサイズ変更 36
7.1.4.1. Noobaa PVC のサイズ変更 36
7.1.4.2. 別のストレージプールの追加 37
7.1.5. デフォルトの Operator イメージのカスタマイズ 37
7.1.5.1. 環境変数 37
7.1.5.2. 実行中の Operator へのオーバーライドの適用 38
7.1.6. AWS S3 CloudFront 38
関連資料 39

序文

Red Hat Quay は、エンタープライズレベルの品質の高いコンテナーレジストリー製品です。Red Hat Quay を使用してコンテナーイメージをビルドし、保存してから、企業全体にデプロイできるようにします。

Red Hat Quay Operator は、Red Hat Quay クラスターをデプロイし、管理する簡単な方法を提供します。これは OpenShift への Red Hat Quay のデプロイで優先される手順であり、この手順については本書で説明します。

今回のバージョンの Red Hat Quay Operator は完全に再作成されており、以前のバージョンと大きく異なることに注意してください。このドキュメントは注意深く確認してください。

第1章 OPENSHIFT での RED HAT QUAY の前提条件

以下に、OpenShift デプロイメントで Red Hat Quay Operator を開始する前に知っておく必要があるいくつかの点を示します。

- **OpenShift クラスター**: Red Hat Quay Operator をデプロイする OpenShift 4.5 以降のクラスターへの特権付きアカウントが必要です。そのアカウントでは、namespace をクラスタースコープで作成できる必要があります。
- **リソース要件**: 各 Red Hat Quay アプリケーション Pod には、以下のリソース要件があります。
  - 8Gi のメモリー
  - 2000 ミリコアの CPU

Red Hat Quay Operator は、管理する Red Hat Quay デプロイメントごとに少なくとも 1 つのアプリケーション Pod を作成します。OpenShift クラスターにこれらの要件に必要なコンピュートリソースがあることを確認します。

- **オブジェクトストレージ**: デフォルトで、Red Hat Quay Operator は **ObjectBucketClaim** Kubernetes API を使用してオブジェクトストレージをプロビジョニングします。この API を消費すると、ベンダー固有の実装から Operator を分離することができます。OpenShift Container Storage は、この例で使用される NooBaa コンポーネントを介してこの API を提供します。それ以外の場合、Red Hat Quay は以下のサポートされるクラウドストレージオプションのいずれかを使用するように手動で設定できます。
  - Amazon S3（Red Hat Quay の S3 バケットポリシーの設定に関する詳細は、「S3 IAM バケットポリシー」を参照してください）
  - Azure Blob Storage
  - Google Cloud Storage
  - Ceph Object Gateway (RADOS)
  - OpenStack Swift
  - CloudFront + S3

第2章 QUAY OPERATOR のインストール

2.1. 以前のバージョンとの相違点

Red Hat Quay 3.4.0 の時点で、Operator は改善された追加設定なしのエクスペリエンスおよび追加の Day 2 操作のサポートを提供できるように完全に再作成されています。その結果、新規 Operator の使用はより容易になり、より使用しやすく事前設定されるようになりました。Operator の以前のバージョンとの主な相違点は以下のとおりです。

- **QuayEcosystem** カスタムリソースは **QuayRegistry** カスタムリソースに置き換えられる
- デフォルトのインストールオプションは、実稼働環境で使用できるすべての管理された依存関係 (データベース、キャッシュ、オブジェクトストレージなど) と共に完全にサポートされる Quay 環境を生成する (一部のコンポーネントには可用性がない場合があります)
- 一貫性を確保するために Quay アプリケーションと設定ツールで共有される Quay の設定向けの新しい堅牢な検証ライブラリー
- オブジェクトストレージを、**ObjectBucketClaim** Kubernetes API を使用して Operator で管理できるようになる（Red Hat OpenShift Data Foundations を使用すると、OpenSHift でこの API のサポートされる実装を提供できる)。
- テストおよび開発シナリオ用にデプロイされた Pod によって使用されるコンテナーイメージのカスタマイズ

2.2. QUAY OPERATOR のインストール前に

2.2.1. ストレージソリューションの決定

Operator が Quay のオブジェクトストレージを管理する必要がある場合、クラスターは **ObjectBucketClaim** API 経由でオブジェクトストレージを提供できる必要があります。Red Hat OpenShift Data Foundations (ODF) Operator を使用する場合、2 つのサポートされるオプションを使用できます。

- ローカルの Kubernetes **PersistentVolume** ストレージでサポートされる Multi-Cloud Object Gateway のスタンドアロンインスタンス
  - 高可用性がない
  - Quay サブスクリプションに含まれる
  - ODF に別のサブスクリプションは必要ない
- スケールアウト Object Service と Ceph を使用する ODF の実稼働デプロイメント
  - 高可用性がある
  - ODF に別のサブスクリプションが必要

スタンドアロンのインスタンスオプションを使用するには、以下の読み取りを続行します。ODF の実稼働デプロイメントについては、公式ドキュメントを参照してください。

ObjectBucketClaim API 経由でオブジェクトストレージがすでに利用可能な場合や、外部 S3 互換オブジェクトストレージサービス（クラウドプロバイダーなど）を使用してオブジェクトストレージを利用可能にしている場合は、Operator のインストール に進みます。

2.2.2. スタンドアロン Object Gateway について

Red Hat Quay サブスクリプションの一環として、Red Hat OpenShift Data Foundations Operator (以前は OpenShift Container Storage Operator として知られる) の **Multi-Cloud Object Gateway** (MCG) コンポーネントを使用することができます。このゲートウェイコンポーネントを使用すると、Kubernetes **PersistentVolume** ベースのブロックストレージがサポートする Quay への S3 互換のオブジェクトストレージインターフェースを指定できます。この使用は、Operator によって管理される Quay デプロイメントや、以下に示す MCG インスタンスの仕様に限定されます。

Red Hat Quay はローカルファイルシステムのストレージをサポートしないため、ユーザーは代わりに Kubernetes **PersistentVolume** ストレージと組み合わせてゲートウェイを利用し、サポートされるデプロイメントを提供できます。**PersistentVolume** はオブジェクトストレージのバッキングストアとしてゲートウェイインスタンスに直接マウントされ、ブロックベースの **StorageClass** がサポートされます。

PersistentVolume の性質上、これはスケールアウトできる高可用性ソリューションではなく、Red Hat OpenShift Data Foundations (ODF) などのスケールアウトストレージシステムを置き換えることはできません。ゲートウェイの単一インスタンスのみが実行されています。再スケジュール、更新、または予定外のダウンタイムが原因でゲートウェイを実行している Pod が利用できなくなると、接続された Quay インスタンスのパフォーマンスが一時的に低下します。

2.2.3. スタンドアロン Object Gateway の作成

ODF（以前の名前は OpenShift Container Storage）Operator をインストールし、単一のインスタンスの Multi-Cloud Gateway サービスを設定するには、以下の手順に従います。

1. OpenShift コンソールを開き、Operators → OperatorHub を選択してから OpenShift Container Storage Operator を選択します。

2. 「Install」を選択します。 すべてのデフォルトオプションを受け入れ、「Install」を再度選択します。

3. 約 1 分以内に、Operator はインストールを開始し、 namespace **openshift-storage** を作成します。**Status** 列に **Succeeded** のマークが付けられると、完了を確認できます。

4. NooBaa オブジェクトストレージを作成します。以下の YAML を **noobaa.yml** という名前のファイルに保存します。

```
apiVersion: noobaa.io/v1alpha1
kind: NooBaa
metadata:
  name: noobaa
  namespace: openshift-storage
spec:
 dbResources:
   requests:
     cpu: '0.1'
     memory: 1Gi
 dbType: postgres
 coreResources:
   requests:
     cpu: '0.1'
     memory: 1Gi
```

これにより、**Multi-cloud Object Gateway** の単一のインスタンスデプロイメントが作成されます。

5. 以下のコマンドを使用して設定を適用します。

```
$ oc create -n openshift-storage -f noobaa.yaml
noobaa.noobaa.io/noobaa created
```

6. 数分後に、MCG インスタンスがプロビジョニングを終了していることを確認できるはずです（**PHASE** 列が **Ready** に設定されます）。

```
$ oc get -n openshift-storage noobaas noobaa -w
NAME     MGMT-ENDPOINTS               S3-ENDPOINTS               IMAGE
PHASE   AGE
noobaa   [https://10.0.32.3:30318]   [https://10.0.32.3:31958]   registry.redhat.io/ocs4/mcg-
core-
rhel8@sha256:56624aa7dd4ca178c1887343c7445a9425a841600b1309f6deace37ce6b8678d
Ready   3d18h
```

7. 次に、ゲートウェイのバッキングストアを設定します。以下の YAML を **noobaa-pv-backing-store.yaml** という名前のファイルに保存します。

noobaa-pv-backing-store.yaml

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: noobaa-pv-backing-store
  namespace: openshift-storage
spec:
  pvPool:
    numVolumes: 1
    resources:
      requests:
        storage: 50Gi (1)
    storageClass: STORAGE-CLASS-NAME (2)
  type: pv-pool
```

(1) オブジェクトストレージサービスの全体的な容量。必要に応じて調整します。

(2) 要求される **PersistentVolumes** に使用する **StorageClass**。クラスターのデフォルトを使用するようにこのプロパティーを削除します。

8. 以下のコマンドを使用して設定を適用します。

```
$ oc create -f noobaa-pv-backing-store.yaml
backingstore.noobaa.io/noobaa-pv-backing-store created
```

これにより、ゲートウェイのバッキングストア設定が作成されます。Quay のすべてのイメージは、上記の設定によって作成される **PersistentVolume** のゲートウェイを経由してオブジェクトとして保存されます。

9. 最後に、以下のコマンドを実行して **PersistentVolume** バッキングストアを Operator によって発行されるすべての **ObjectBucketClaims** のデフォルトにします。

```
$ oc patch bucketclass noobaa-default-bucket-class --patch '{"spec":{"placementPolicy":
{"tiers":[{"backingStores":["noobaa-pv-backing-store"]}]}}}' --type merge -n openshift-storage
```

これにより、Red Hat Quay の **Multi-Cloud Object Gateway** インスタンスの設定が完了します。この設定は、Red Hat OpenShift Data Foundations がインストールされているクラスターで並行して実行できないことに注意してください。

2.3. OPERATORHUB からの OPERATOR のインストール

1. OpenShift コンソールを使用して、Operators → OperatorHub を選択してから Quay Operator を選択します。コミュニティーバージョンが複数ある場合は、必ず Red Hat 認定 Operator を使用するようにし、コミュニティーバージョンは使用しないでください。

2. 「Install」を選択します。 Operator Subscription ページが表示されます。

3. 以下を選択して「Subscribe」を選択します。

   - インストールモード: Operator をクラスター全体で利用可能にするか、または単一の namespace 内のみで利用できるかに応じて、「All namespaces」または「A specific namespace」のいずれかを選択します。

   - Update Channel: 更新チャネルの選択します（1 つのみ選択可能)。

   - Approval Strategy: 自動の更新または手動の更新を承認するよう選択します。

4. 「Install」を選択します。

5. 1 分後に、「Installed Operators」ページで Operator が正常にインストールされたことを確認できます。

第3章 高レベルの概念

3.1. QUAYREGISTRY API

Quay Operator は、クラスターで **Quay** コンテナーレジストリーを宣言的に管理するために、**QuayRegistry** カスタムリソース API を提供します。OpenShift UI またはコマンドラインツールを使用してこの API と対話します。

- **QuayRegistry** を作成すると、Operator は Quay をクラスターで実行するために必要なすべてのリソースをデプロイし、設定します。
- **QuayRegistry** を編集すると、Operator は変更を調整し、オブジェクトが必要な設定に一致するようにオブジェクトの作成/更新/削除を実行します。
- **QuayRegistry** を削除すると、以前に作成されたすべてのリソースのガベージコレクションが実行され、**Quay** コンテナーレジストリーは利用できなくなります。

QuayRegistry API はかなり単純であり、そのフィールドは以下のセクションで説明されています。

3.2. QUAY コンポーネント

Quay は強力なコンテナーレジストリープラットフォームであるため、十分な数の依存関係が必要となります。これらには、データベース、オブジェクトストレージ、Redis などが含まれます。Quay Operator は、Quay と Kubernetes 上のそれらの依存関係の事前に設定されたデプロイメントを管理します。これらの依存関係は **コンポーネント** として処理され、**QuayRegistry** API で設定されます。

QuayRegistry カスタムリソースでは、**spec.components** フィールドでコンポーネントを設定します。各コンポーネントには、コンポーネントの名前である **kind** およびコンポーネントライフサイクルが Operator によって処理されるかどうかを示すブール値の **managed** の 2 つのフィールドが含まれます。デフォルトでは (このフィールドを省略する場合)、すべてのコンポーネントが管理され、調整時に表示するために自動的に入力されます。

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

QuayRegistry カスタムリソースが指定しない場合、Operator は以下の管理されたコンポーネントについてデフォルトを使用します。

- **postgres** はレジストリーのメタデータを保存します。Software Collections から Postgres 10 のあるバージョンを使用します。
- **redis** は Quay ビルダーの調整および一部のロギングを処理します。
- **objectstorage** はイメージレイヤー Blob を保存します。Noobaa/RHOCS によって提供される **ObjectBucketClaim** Kubernetes API を活用します。
- **clair** はイメージの脆弱性スキャンを提供します。
- **horizontalpodautoscaler** は、メモリー/CPU の消費に応じて Quay Pod の数を調整します。
- **mirror** は、リポジトリーミラーワーカーを設定します (オプションのリポジトリーミラーリングをサポートします)。
- **route** は、OpenShift の外部から Quay レジストリーへの外部エントリーポイントを提供します。
- **モニタリング機能** には、Grafana ダッシュボード、個別のメトリクスへのアクセス、Quay Pod が頻繁に再起動されていることを通知するアラートなどが含まれます。

3.3. 管理コンポーネントの使用

Operator は Red Hat Quay が管理コンポーネントを使用するために必要な設定およびインストール作業を処理しますが、いくつかの点を考慮する必要があります。

- Postgres イメージで提供されるツールまたは独自のバックアップインフラストラクチャーのいずれかを使用して、データベースのバックアップを定期的に実行する必要があります。現時点で、Operator は Postgres データベースがバックアップされていることを確認しません。
- バックアップから Postgres データベースを復元するには、Postgres ツールおよび関連する手順を使用する必要があります。データベースの復元が実行中の場合には、Quay **Pod** を実行できないことに注意してください。
- データベースのディスク領域は、50 GiB が Operator によって自動的に割り当てられます。この数は、ほとんどの小規模/中規模の Red Hat Quay インストールで利用可能なストレージの容量を表しますが、実際のユースケースには十分ではない可能性があります。現時点で、データベースボリュームのサイズ変更は Operator によって処理されません。
- オブジェクトストレージのディスク容量は、50 GiB が Operator によって自動的に割り当てられます。この数は、ほとんどの小規模/中規模の Red Hat Quay インストールで利用可能なストレージの容量を表しますが、実際のユースケースには十分ではない可能性があります。現時点で、RHOCS ボリュームのサイズ変更は Operator によって処理されません。詳細は、管理ストレージのサイズ変更についてのセクションを参照してください。
- Operator は、OpenShift **Route** をレジストリーへのデフォルトのエントリーポイントとしてデプロイします。別のエントリーポイントを選択する場合 (**Ingress** または直接の **Service** アクセスなど)、その設定は手動で行う必要があります。

これらの考慮事項のいずれかがお使いの環境で反映することができない場合は、以下のセクションで説明されているように、Operator に管理対象外のリソースまたはオーバーライドを指定することが推奨されます。

3.4. 依存関係向けの管理対象外コンポーネントの使用

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

注記

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

3.4.1. 既存の Postgres データベースの使用

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

   config.yaml:

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

2. 設定ファイルを使用してシークレットを作成します。

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

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

   quayregistry.yaml

   ```
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
     name: test
   spec:
     configBundleSecret: config-bundle-secret
     components:
       - kind: postgres
         managed: false
   ```

4. QuayRegistry を作成します。

   ```
   $ oc create -f quayregistry.yaml
   ```

デプロイされた Quay アプリケーションが外部データベースを使用するようになりました。

3.4.2. 管理対象外ストレージ

以下の例では、NooBaa ストレージを使用していますが、Azure、S3 などの他のイメージストレージオプションに適用できます。

1. Storage → Object Bucket Claims のコンソールで NooBaa Object Bucket Claim (オブジェクトバケット要求) を作成します。

2. アクセスキー、バケット名、エンドポイント (ホスト名)、およびシークレットキーを含む Object Bucket Claim データの詳細を取得します。

3. Object Bucket Claim (オブジェクトバケット要求) の情報を使用して **config.yaml** 設定ファイルを作成します。

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

4. 設定ファイルを使用してシークレットを作成します。

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

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

quayregistry.yaml

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: test
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: storage
      managed: false
```

6. QuayRegistry を作成します。

```
oc create -f quayregistry.yaml
```

デプロイされた Quay アプリケーションで、作成されたストレージが使用されるようになりました。

3.4.3. Horizontal Pod Autoscaler の無効化

自動スケーリングを無効にするか、または独自の **HorizontalPodAutoscaler** を作成する場合、コンポーネントを単純に **QuayRegistry** インスタンスで管理対象外として指定します。

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

```
spec:
  components:
    - kind: horizontalpodautoscaler
      managed: false
```

3.5. 設定バンドルシークレット

spec.configBundleSecret フィールドは、**QuayRegistry** と同じ namespace の **Secret** **metadata.name** への参照です。この **Secret** には、 **config.yaml** のキー/値ペアが含まれる必要があります。この **config.yaml** ファイルは Quay 設定 YAML ファイルです。このフィールドは任意で、指定されていない場合には Operator によって自動入力されます。これは指定されている場合、後に管理コンポーネントの他のフィールドにマージされる設定フィールドのベースセットとして機能し、Quay アプリケーション Pod にマウントされる最終出力の **Secret** を生成します。

3.6. QUAYREGISTRY ステータス

特定の Quay デプロイメントのライフサイクルの可観測性については、対応する **QuayRegistry** オブジェクトの **status** セクションでレポートされます。Operator はこのセクションを継続的に更新するため、これは、Quay またはその管理対象の依存関係の問題または状態の変更を確認するために最初に参照する場所となります。

3.6.1. レジストリーエンドポイント

Quay を使用する準備ができると、**status.registryEndpoint** フィールドにレジストリーの一般に利用可能なホスト名が設定されます。

3.6.2. 設定エディターのエンドポイント

status.configEditorEndpoint を使用して Quay の UI ベースの設定エディターにアクセスします。

3.6.3. 設定エディターの認証情報シークレット

設定エディター UI のユーザー名/パスワードは、**status.configEditorCredentialsSecret** によって参照される **QuayRegistry** として同じ namespace の **Secret** に保存されます。

3.6.4. 現行バージョン

実行中の Quay の現行バージョンは **status.currentVersion** で報告されます。

3.6.5. 条件

特定の条件は **status.conditions** に報告されます。

第4章 QUAY OPERATOR を使用した QUAY のデプロイ

4.1. QUAY レジストリーの作成

デフォルト設定は、Operator に対して Quay の依存関係（データベース、Redis、オブジェクトストレージなど）を管理するよう指示します。

4.1.1. OpenShift Console

1. Operators → Installed Operators を選択してから Quay Operator を選択し、Operator の詳細ビューに移動します。
2. 「Provided APIs」の下で「Quay Registry」タイルの「Create Instance」をクリックします。
3. オプションで、**QuayRegistry** の「Name」を変更します。これにより、レジストリーのホスト名が影響を受けます。その他のフィールドはすべてデフォルトで設定されます。
4. 「Create」をクリックし、Quay Operator によってデプロイされる **QuayRegistry** を送信します。
5. **QuayRegistry** 一覧ビューにリダイレクトされるはずです。作成した **QuayRegistry** をクリックし、詳細ビューを表示します。
6. 「Registry Endpoint」の値が設定されたら、その値をクリックして UI で新規の Quay レジストリーにアクセスします。「Create Account」を選択して、ユーザーを作成し、サインインできるようになりました。

4.1.2. コマンドライン

CLI を使用して同じ結果を得ることができます。

1. **quay.yaml** というファイルに、以下の **QuayRegistry** カスタムリソースを作成します。

   quay.yaml:

   ```
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
     name: my-registry
   ```

2. **QuayRegistry** を使用している namespace に作成します。

   ```
   $ oc create -n <your-namespace> -f quay.yaml
   ```

3. **status.registryEndpoint** にデータが設定されるまで待機します。

   ```
   $ oc get -n <your-namespace> quayregistry my-registry -o jsonpath="
   {.status.registryEndpoint}" -w
   ```

4. **status.registryEndpoint** に値が設定されたら、Web ブラウザーを使用してこれに移動し、UI で新規の Quay レジストリーにアクセスします。「Create Account」を選択して、ユーザーを作成し、サインインできるようになりました。

4.2. インフラストラクチャーノードでの QUAY のデプロイ

デフォルトで、Operator を使用してレジストリーをデプロイする際に Quay 関連の Pod は任意のワーカーノードに配置されます。OpenShift Container Platform ドキュメントでは、マシンセットを使用してノードがインフラストラクチャーコンポーネントのみをホストするように設定する方法が記載されています (https://docs.openshift.com/container-platform/4.7/machine_management/creating-infrastructure-machinesets.html を参照してください)。

OCP MachineSet リソースを使用して infra ノードをデプロイしていない場合のために、本セクションでは、インフラストラクチャーの目的でノードに手動でラベルを付け、テイントを付ける方法を説明します。

手動またはマシンセットを使用してインフラストラクチャーノードを設定したら、ノードセレクターおよび容認を使用してこれらのノードへの Quay Pod の配置を制御できます。

4.2.1. インフラストラクチャーに使用するノードのラベルおよびテイント

この例で使用されるクラスターには、3 つのマスターノードと 6 つのワーカーノードがあります。

```
$ oc get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
user1-jcnp6-master-0.c.quay-devel.internal        Ready    master   3h30m   v1.20.0+ba45583
user1-jcnp6-master-1.c.quay-devel.internal        Ready    master   3h30m   v1.20.0+ba45583
user1-jcnp6-master-2.c.quay-devel.internal        Ready    master   3h30m   v1.20.0+ba45583
user1-jcnp6-worker-b-65plj.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-b-jr7hc.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-c-jrq4v.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal   Ready    worker   3h22m   v1.20.0+ba45583
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
```

インフラストラクチャーに使用する最終的な 3 つのワーカーノードにラベルを付けます。

```
$ oc label node --overwrite user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal node-
role.kubernetes.io/infra=
$ oc label node --overwrite user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal node-
role.kubernetes.io/infra=
$ oc label node --overwrite user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal node-
role.kubernetes.io/infra=
```

クラスターのノードを一覧表示すると、最後の 3 つのワーカーノードには **infra** というロールが追加されます。

```
$ oc get nodes
NAME                                          STATUS   ROLES          AGE     VERSION
user1-jcnp6-master-0.c.quay-devel.internal        Ready    master         4h14m   v1.20.0+ba45583
user1-jcnp6-master-1.c.quay-devel.internal        Ready    master         4h15m   v1.20.0+ba45583
user1-jcnp6-master-2.c.quay-devel.internal        Ready    master         4h14m   v1.20.0+ba45583
user1-jcnp6-worker-b-65plj.c.quay-devel.internal   Ready    worker         4h6m    v1.20.0+ba45583
user1-jcnp6-worker-b-jr7hc.c.quay-devel.internal   Ready    worker         4h5m    v1.20.0+ba45583
user1-jcnp6-worker-c-jrq4v.c.quay-devel.internal   Ready    worker         4h5m    v1.20.0+ba45583
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal   Ready    infra,worker   4h6m    v1.20.0+ba45583
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal   Ready    infra,worker   4h6m    v1.20.0+ba45583
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal   Ready    infra,worker   4h6m    v1.20.0+ba45583
```

infra ノードがワーカーとして割り当てられると、ユーザーのワークロードが誤って infra ノードに割り当てられる可能性があります。これを回避するには、テイントを、制御する必要のある Pod の infra ノードに適用し、容認を追加できます。

```
$ oc adm taint nodes user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal node-
role.kubernetes.io/infra:NoSchedule
$ oc adm taint nodes user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal node-
role.kubernetes.io/infra:NoSchedule
$ oc adm taint nodes user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal node-
role.kubernetes.io/infra:NoSchedule
```

4.2.2. ノードセレクターおよび容認を使用したプロジェクトの作成

Quay Operator を使用して Quay をすでにデプロイしている場合、インストールされた Operator およびデプロイメント用に作成した特定の namespace を削除します。

以下の例のようにノードセレクターおよび容認を指定して Project リソースを作成します。

quay-registry.yaml

```
kind: Project
apiVersion: project.openshift.io/v1
metadata:
  name: quay-registry
  annotations:
    openshift.io/node-selector: 'node-role.kubernetes.io/infra='
    scheduler.alpha.kubernetes.io/defaultTolerations: >-
      [{"operator": "Exists", "effect": "NoSchedule", "key":
      "node-role.kubernetes.io/infra"}
      ]
```

oc apply コマンドを使用してプロジェクトを作成します。

```
$ oc apply -f quay-registry.yaml
project.project.openshift.io/quay-registry created
```

quay-registry namespace に作成される後続のリソースは、専用のインフラストラクチャーノードでスケジュールされます。

4.2.3. Quay Operator の namespace へのインストール

Quay Operator をインストールする場合、適切なプロジェクト namespace（この場合は **quay-registry**）を明示的に指定します。これにより、Operator Pod 自体が 3 つのインフラストラクチャーノードのいずれかに到達します。

```
$ oc get pods -n quay-registry -o wide
NAME                                    READY   STATUS    RESTARTS   AGE   IP            NODE

quay-operator.v3.4.1-6f6597d8d8-bd4dp   1/1     Running   0          30s   10.131.0.16   user1-jcnp6-
worker-d-h5tv2.c.quay-devel.internal
```

4.2.4. レジストリーの作成

前述のようにレジストリーを作成してから、デプロイメントが準備されるのを待機します。Quay Pod を一覧表示する場合、それらがインフラストラクチャー用としてラベルを付けた 3 つのノードにのみスケジュールされていることを確認できます。

```
$ oc get pods -n quay-registry -o wide
NAME                                        READY   STATUS      RESTARTS   AGE     IP            NODE

example-registry-clair-app-789d6d984d-gpbwd            1/1     Running     1          5m57s   10.130.2.80
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
example-registry-clair-postgres-7c8697f5-zkzht         1/1     Running     0          4m53s   10.129.2.19
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-app-56dd755b6d-glbf7             1/1     Running     1          5m57s   10.129.2.17
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-config-editor-7bf9bccc7b-dpc6d   1/1     Running     0          5m57s
10.131.0.23   user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
example-registry-quay-database-8dc7cfd69-dr2cc         1/1     Running     0          5m43s   10.129.2.18
  user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-mirror-78df886bcc-v75p9          1/1     Running     0          5m16s   10.131.0.24
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
example-registry-quay-postgres-init-8s8g9              0/1     Completed   0          5m54s   10.130.2.79
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
example-registry-quay-redis-5688ddcdb6-ndp4t           1/1     Running     0          5m56s   10.130.2.78
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
quay-operator.v3.4.1-6f6597d8d8-bd4dp                  1/1     Running     0          22m     10.131.0.16
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
```

第5章 QUAY OPERATOR を使用した QUAY のアップグレード

Quay Operator は **同期されたバージョン管理** スキームに従います。つまり、Operator のそれぞれのバージョンは Quay とその管理するコンポーネントに関連付けられます。**QuayRegistry** カスタムリソースには、デプロイする Quay のバージョンを設定するフィールドはありません。Operator は単一バージョンのすべてのコンポーネントをデプロイする方法のみを認識します。このスキームは、すべてのコンポーネントが適切に機能するように、また Kubernetes 上の複数バージョンの Quay のライフサイクルを管理する方法に関する Operator の複雑さを軽減するために選択されています。

5.1. OPERATOR LIFECYCLE MANAGER

Quay Operator は Operator Lifecycle Manager (OLM) を使用してインストールし、アップグレードする必要があります。デフォルトの **approvalStrategy: Automatic** で **Subscription** を作成する場合、OLM は新規バージョンが利用可能になると常に Quay Operator を自動的にアップグレードします。

警告

Quay Operator が Operator Lifecycle Manager でインストールされる場合、自動または手動アップグレードをサポートするように設定できます。このオプションは、インストール時に Quay Operator の Operator Hub ページに表示されます。また、これは **approvalStrategy** フィールドで Quay Operator **Subscription** オブジェクトを参照して見つけることもできます。**Automatic** を選択すると、新規 Operator バージョンがリリースされるたびに Quay Operator が自動的にアップグレードされます。この動作が望ましくない場合には、**Manual** 承認ストラテジーを選択する必要があります。

5.2. QUAY OPERATOR のアップグレードによる QUAY のアップグレード

インストールされた Operator を OpenShift にアップグレードする一般的な方法は、インストールされた Operator のアップグレードについて参照してください。

5.2.1. Quay のアップグレード

Red Hat Quay 側から見ると、3.4 → 3.5 のように 1 つのマイナーバージョンから次のマイナーバージョンに更新するには、Quay Operator の更新チャネルをアクティブに変更する必要があります。

3.4.2 → 3.4.3 などの **z** ストリームのアップグレードの場合、更新は、ユーザーが最初のインストール時選択した major-minor (メジャーマイナー) チャネルでリリースされます。**z** ストリームのアップグレードを実行する手順は、上記のように **approvalStrategy** によって異なります。承認ストラテジーが **Automatic** に設定されている場合、Operator は最新の **z** ストリームに自動的にアップグレードするため、ダウンタイムがほとんどなく発生することなく、新しい **z** ストリームへの Quay の自動のローリング更新が行われます。それ以外の場合は、インストールを開始する前に更新を手動で承認する必要があります。

5.2.2. Operator の更新チャネルの変更

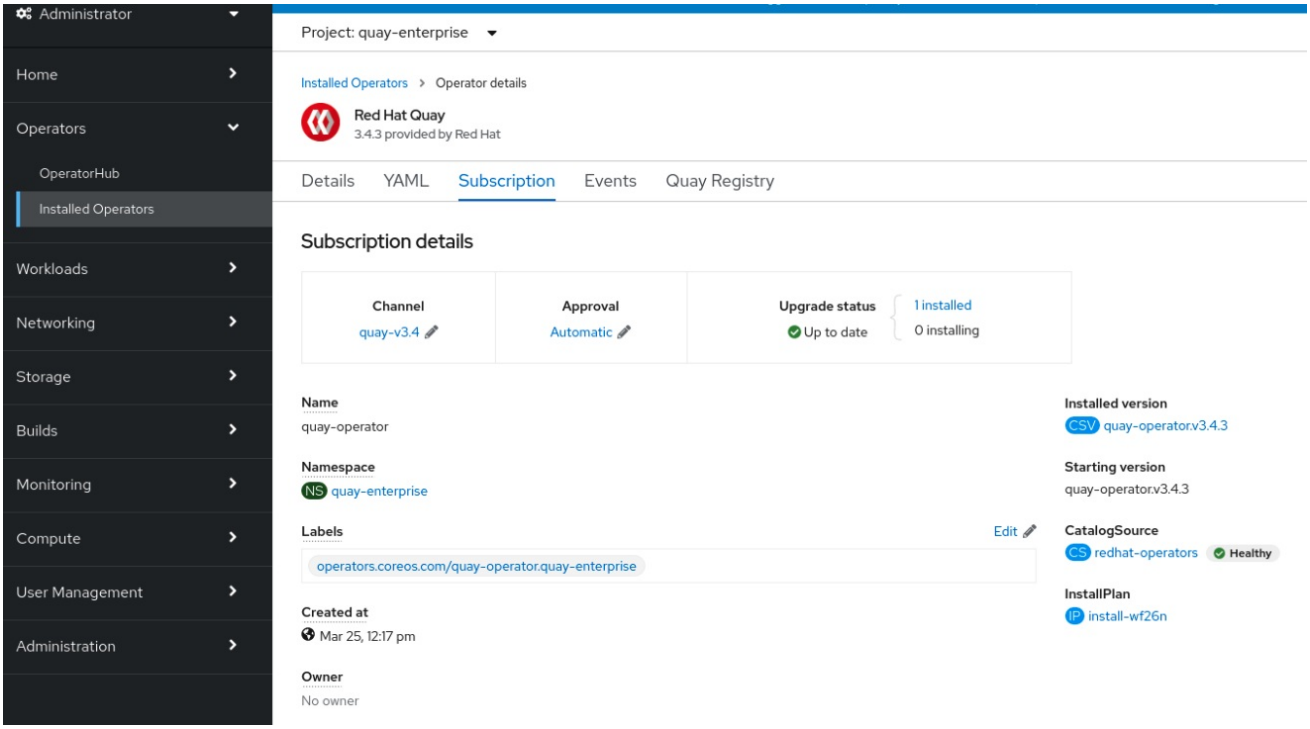
インストールされた Operator のサブスクリプションは、Operator の更新を追跡し、受信するために使用される更新チャネルを指定します。Operator Operator をアップグレードして新規チャネルからの更新の追跡および受信を開始するには、インストールされた Quay Operator の **Subscription** タブで更新

チャネルを変更します。**Automatic** 承認ストラテジーのサブスクリプションの場合、アップグレードは自動的に開始し、インストールされた Operator を一覧表示したページでモニターできます。

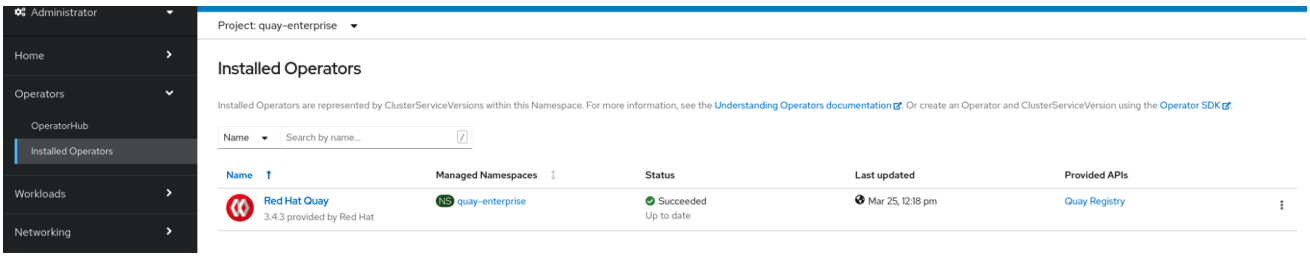
5.2.3. 保留中の Operator アップグレードの手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。Quay Operator に保留中のアップグレードがある場合、このステータスは Installed Operators の一覧に表示されます。Quay Operator の **Subscription** タブで、インストール計画をプレビューし、アップグレードに利用可能なリソースとして一覧表示されるリソースを確認できます。問題がなければ、**Approve** をクリックし、インストールされた Operator を一覧表示しているページに戻り、アップグレードの進捗をモニターします。

以下のイメージには、更新 **Channel**、**Approval** ストラテジー、**Upgrade status** および **InstallPlan** を含む、UI の **Subscription** タブが表示されています。



Installed Operator の一覧は、現在の Quay インストールの概要を示します。



5.3. QUAYREGISTRY のアップグレード

Quay Operator が起動すると、これは監視できるように設定された namespace で検索可能な **QuayRegistries** をすぐに検索します。これが検索されると、以下のロジックが使用されます。

- **status.currentVersion** が設定されていない場合は、調整が通常の方法で行われます。

- **status.currentVersion** が Operator バージョンと等しい場合、調整が通常の方法で行われます。
- **status.currentVersion** が Operator バージョンと等しくない場合、これがアップグレード可能かどうかを確認します。可能な場合は、アップグレードタスクを実行し、完了後に **status.currentVersion** を Operator のバージョンに設定します。アップグレードできない場合は、エラーを返し、**QuayRegistry** とそのデプロイされた Kubernetes オブジェクトのみを残します。

5.4. QUAY 3.5 の新機能の有効化

5.4.1. コンソールでのモニタリングおよびアラート

OpenShift コンソールでの Quay 3.5 のモニタリングのサポートを使用するには、Operator がすべての namespace でインストールされている必要があります。Operator を特定の namespace にインストールしている場合、Operator 自体を削除し、アップグレードが実行されたら、これをすべての namespace に対して再インストールします。

5.4.2. OCI および Helm サポート

Helm および OCI アーティファクトのサポートが、Red Hat Quay 3.5 でデフォルトで有効にされるようになりました。機能を明示的に有効にする必要がある場合（機能がデフォルトで有効にされていないバージョンからアップグレードする場合など) は、以下のプロパティーを使用して OCI アーティファクトの使用を有効にするために、Quay デプロイメントを再設定する必要があります。

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

5.5. QUAYECOSYSTEM のアップグレード

制限された設定のセットについて、**QuayEcosystem** API を使用した Operator の以前のバージョンからのアップグレードがサポートされます。移行が予期されない方法で発生することを防ぐために、移行時に特殊なラベルが **QuayEcosystem** に適用される必要があります。新規 **QuayRegistry** は Operator が管理できるように作成されますが、古い **QuayEcosystem** は、ロールバックし、問題が生じた場合に Quay に依然としてアクセスできるよう、手動で削除されるまでそのままになります。既存の **QuayEcosystem** を新規の **QuayRegistry** に移行するには、以下の手順を実行します。

1. **"quay-operator/migrate": "true"** を **QuayEcosystem**の **metadata.labels** に追加します。

```
$ oc edit quayecosystem <quayecosystemname>
```

```
metadata:
  labels:
    quay-operator/migrate: "true"
```

2. **QuayRegistry** が **QuayEcosystem** と同じ **metadata.name** で作成されるまで待機します。**QuayEcosystem** にはラベル **"quay-operator/migration-complete": "true"** でマークが付けられます。
3. 新規 **QuayRegistry** の **status.registryEndpoint** が設定された後に、Quay にアクセスし、すべてのデータと設定が正常に移行されたことを確認します。

4. すべてが正常に機能することを確認した後に、**QuayEcosystem** を削除でき、Kubernetes ガベージコレクションはすべての古いリソースをクリーンアップします。

5.5.1. QuayEcosystem アップグレードを元に戻す

QuayEcosystem から **QuayRegistry** への自動アップグレードに問題が生じた場合は、以下の手順を実行して **QuayEcosystem** を使用できるように元に戻します。

- UI または **kubectl** のいずれかを使用して **QuayRegistry** を削除します。

```
$ kubectl delete -n <namespace> quayregistry <quayecosystem-name>
```

- 外部アクセスが **Route** を使用して提供される場合、UI または **kubectl** を使用して **Route** を元の **Service** を参照し直すように変更します。

注記

QuayEcosystem が Postgresデータベースを管理している場合、アップグレードプロセスはデータを、アップグレードされた Operator によって管理される新規 Postgres データベースに移行します。古いデータベースは変更または削除されませんが、移行が完了すると Quay はこのデータベースを使用しなくなります。データの移行中に問題が発生した場合には、アップグレードプロセスは終了し、データベースを管理対象外コンポーネントとして継続して使用することが推奨されます。

5.5.2. アップグレードでサポートされる QuayEcosystem 設定

Quay Operator はエラーをログでレポートし、**QuayEcosystem** コンポーネントの移行が失敗するか、またはサポートされない場合には **status.conditions** でレポートされます。管理対象外のすべてのコンポーネントの移行は、Kubernetes リソースを導入する必要がなく、必要なすべての値が Quay の **config.yaml** にすでに指定されているため、正常に実行されるはずです。

データベース

一時データベースはサポートされません (**volumeSize** フィールドを設定する必要があります)。

Redis

特別な設定は必要ありません。

外部アクセス

passthrough **Route** アクセスのみが自動移行でサポートされます。他の方法の場合、手動の移行が必要です。

- カスタムホスト名なしの **LoadBalancer**: **QuayEcosystem** にラベル **"quay-operator/migration-complete": "true"** のマークが付けられた後に、**metadata.ownerReferences** フィールドを既存の **Service** から削除した **後に**、**QuayEcosystem** を削除して Kubernetes の **Service** のガベージコレクションおよびロードバランサーの削除を防ぎます。新規 **Service** は **metadata.name** 形式の **<QuayEcosystem-name>-quay-app** で作成されます。既存 **Service** の **spec.selector** を新規 **Service** の **spec.selector** に一致させるように編集し、古いロードバランサーのエンドポイントへのトラフィックが新規 Pod に送られるようにします。これで古い **Service** を管理できます (Quay Operator はこれを管理しません)。

- カスタムホスト名を持つ **LoadBalancer/NodePort/Ingress**: タイプ **LoadBalancer** の新規

Service は **metadata.name** 形式の **<QuayEcosystem-name>-quay-app** で作成されます。DNS 設定を、新規 **Service** で指定される **status.loadBalancer** エンドポイントを参照するように変更します。

Clair

特別な設定は必要ありません。

オブジェクトストレージ

QuayEcosystem には管理オブジェクトストレージコンポーネントがないため、オブジェクトストレージには常に管理外のマークが付けられます。ローカルストレージはサポートされません。

リポジトリーのミラーリング

特別な設定は必要ありません。

第6章 QUAY OPERATOR の機能

6.1. HELM OCI サポートおよび RED HAT QUAY

Red Hat Quay などのコンテナーレジストリーは、当初は Docker イメージ形式でコンテナーイメージをサポートするように設計されています。Docker 以外で追加のランタイムの使用をプロモートするために、コンテナーランタイムとイメージ形式に関連する標準化を提供するために Open Container Initiative (OCI) が作成されました。ほとんどのコンテナーレジストリーは、Docker イメージマニフェスト V2、Schema 2 形式をベースとして OCI 標準化をサポートします。

コンテナーイメージのほかにも、個別のアプリケーションだけでなく、Kubernetes プラットフォームを全体としてサポートする各種のアーティファクトが新たに出現しました。これらは、アプリケーションのデプロイメントを支援するセキュリティーおよびガバナンスの Open Policy Agent (OPA) ポリシーから Helm チャートおよび Operator に及びます。

Red Hat Quay は、コンテナーイメージを格納するだけでなく、コンテナーの管理を支援するツールのエコシステム全体をサポートするプライベートコンテナーレジストリーです。Red Hat Quay 3.5 のリリースでは、OCI ベースのアーティファクト（具体的には Helm チャート) の使用のサポートが、テクニカルプレビュー (TP) ではなく一般公開 (GA) 機能として利用可能になりました。

OpenShift Operator を使用して Red Hat Quay 3.5 をデプロイすると、Helm および OCI アーティファクトのサポートがデフォルトで有効にされるようになりました。機能を明示的に有効にする必要がある場合（機能が無効にされている場合や、デフォルトで有効にされていないバージョンからアップグレードした場合など）は、「 OCI および Helm サポートの有効化 」のセクションを参照してください。

6.1.1. Helm および OCI の前提条件

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

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

- **実験的な機能:** Helm および OCI レジストリーと対話するコマンドの多くは、 **helm chart** サブコマンドを利用します。本書の作成時点では、Helm での OCI サポートは「experimental (実験的な)」機能とマークされており、明示的に有効にする必要があります。これは、環境変数 **HELM_EXPERIMENTAL_OCI=1** を設定して実行されます。

- **Helm クライアントのインストール:** 必要なバージョンを https://github.com/helm/helm/releases からダウンロードします (例: https://get.helm.sh/helm-v3.5.3-linux-amd64.tar.gz)。これを展開し、helm バイナリーをその必要な宛先に移動します。

```
$ tar -zxvf helm-v3.5.3-linux-amd64.tar.gz
$ mv linux-amd64/helm /usr/local/bin/helm
```

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

6.1.2. Quay での Helm チャートの使用

Helm は、Cloud Native Computing Foundation (CNCF) から進展したプロジェクトとして、アプリケーションのパッケージ化およびデプロイを単純化する、Kubernetes の事実上のパッケージマネージャーです。Helm は、アプリケーションを表す Kubernetes リソースが含まれる Chart というパッケージ形式を使用します。Chart (チャート) は、リポジトリーでの一般的なディストリビューションや消費用に利用できます。Helm リポジトリーは、index.yaml メタデータファイルと、オプションでパッケージ化されたチャートのセットを提供する HTTP サーバーです。Helm バージョン 3 以降、従来のリポジトリーの代わりとして OCI レジストリーでチャートを提供するためのサポートが利用できるようになりました。Quay を Helm チャートのレジストリーとして使用する方法を説明するために、Helm リポジトリーの既存チャートを使用してチャート開発者とユーザー向けに OCI レジストリーとの対話を示します。

以下の例では、以下の手順に従って、サンプルの etherpad チャートが Red Community of Practice (CoP) リポジトリーからダウンロードされ、ローカルの Red Hat Quay リポジトリーにプッシュされます。

- 適切なリポジトリーを追加します。
- リポジトリーを最新のメタデータで更新します
- チャートをダウンロードして展開し、 **etherpad** というローカルディレクトリーを作成します。

以下は例になります。

```
$ helm repo add redhat-cop https://redhat-cop.github.io/helm-charts
$ helm repo update
$ helm pull redhat-cop/etherpad --version=0.0.4 --untar
```

チャートにタグ付けするには、**helm chart save** コマンドを使用する必要があります。これは、イメージにタグ付けするために **podman タグ** を使用することに対応します。

```
$ helm chart save ./etherpad example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  6850d9b21dd4b87cf20ad49f2e2c7def9655c52ea573e1ddb9d1464eeb6a46a6
size:    3.5 KiB
name:    etherpad
version: 0.0.4
0.0.4: saved
```

helm chart list コマンドを使用して、チャートのローカルインスタンスを表示します。

```
helm chart list

REF                                                     NAME     VERSION DIGEST SIZE   CREATED
example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4 etherpad 0.0.4
ce0233f 3.5 KiB 23 seconds
```

チャートをプッシュする前に、**helm registry login** コマンドを使用してリポジトリーにログインします。

```
$ helm registry login example-registry-quay-quay-enterprise.apps.user1.example.com
Username: quayadmin
```

```
Password:
Login succeeded
```

helm chart push コマンドを使用してチャートをローカル Quay リポジトリーにプッシュします。

```
$ helm chart push example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

The push refers to repository [example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad]
ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  ce0233fd014992b8e27cc648cdabbebd4dd6850aca8fb8e50f7eef6f2f49833d
size:    3.5 KiB
name:    etherpad
version: 0.0.4
0.0.4: pushed to remote (1 layer, 3.5 KiB total)
```

プッシュが機能することをテストするには、ローカルコピーを削除してから、チャートをリポジトリーからプルします。

```
$ helm chart rm example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
$ rm -rf etherpad
$ helm chart pull example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

0.0.4: Pulling from example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad
ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  6850d9b21dd4b87cf20ad49f2e2c7def9655c52ea573e1ddb9d1464eeb6a46a6
size:    3.5 KiB
name:    etherpad
version: 0.0.4
Status: Downloaded newer chart for example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4
```

helm chart export コマンドを使用してチャートファイルを展開します。

```
$ helm chart export example-registry-quay-quay-
enterprise.apps.user1.example.com/helm/etherpad:0.0.4

ref:     example-registry-quay-quay-enterprise.apps.user1.example.com/helm/etherpad:0.0.4
digest:  ce0233fd014992b8e27cc648cdabbebd4dd6850aca8fb8e50f7eef6f2f49833d
size:    3.5 KiB
name:    etherpad
version: 0.0.4
Exported chart to etherpad/
```

6.1.3. OCI および Helm の設定

Helm および OCI アーティファクトのサポートが、Red Hat Quay 3.5 でデフォルトで有効にされるようになりました。機能を明示的に有効にする必要がある場合（機能が無効にされている場合や、デフォルトで有効にされていないバージョンからアップグレードした場合など) は、OCI アーティファクトの使用を有効にするために 2 つのプロパティーを Quay 設定に追加する必要があります。

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

表6.1 OCI および Helm の設定

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

6.1.4. Operator を使用した OCI および Helm の設定

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

quay-config-bundle.yaml

```
apiVersion: v1
stringData:
  config.yaml: |
    FEATURE_GENERAL_OCI_SUPPORT: true
    FEATURE_HELM_OCI_SUPPORT: true
kind: Secret
metadata:
  name: quay-config-bundle
  namespace: quay-enterprise
type: Opaque
```

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

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

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

quay-registry.yaml

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

指定された設定でレジストリーを作成します。

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

6.2. コンソールでのモニタリングおよびアラート

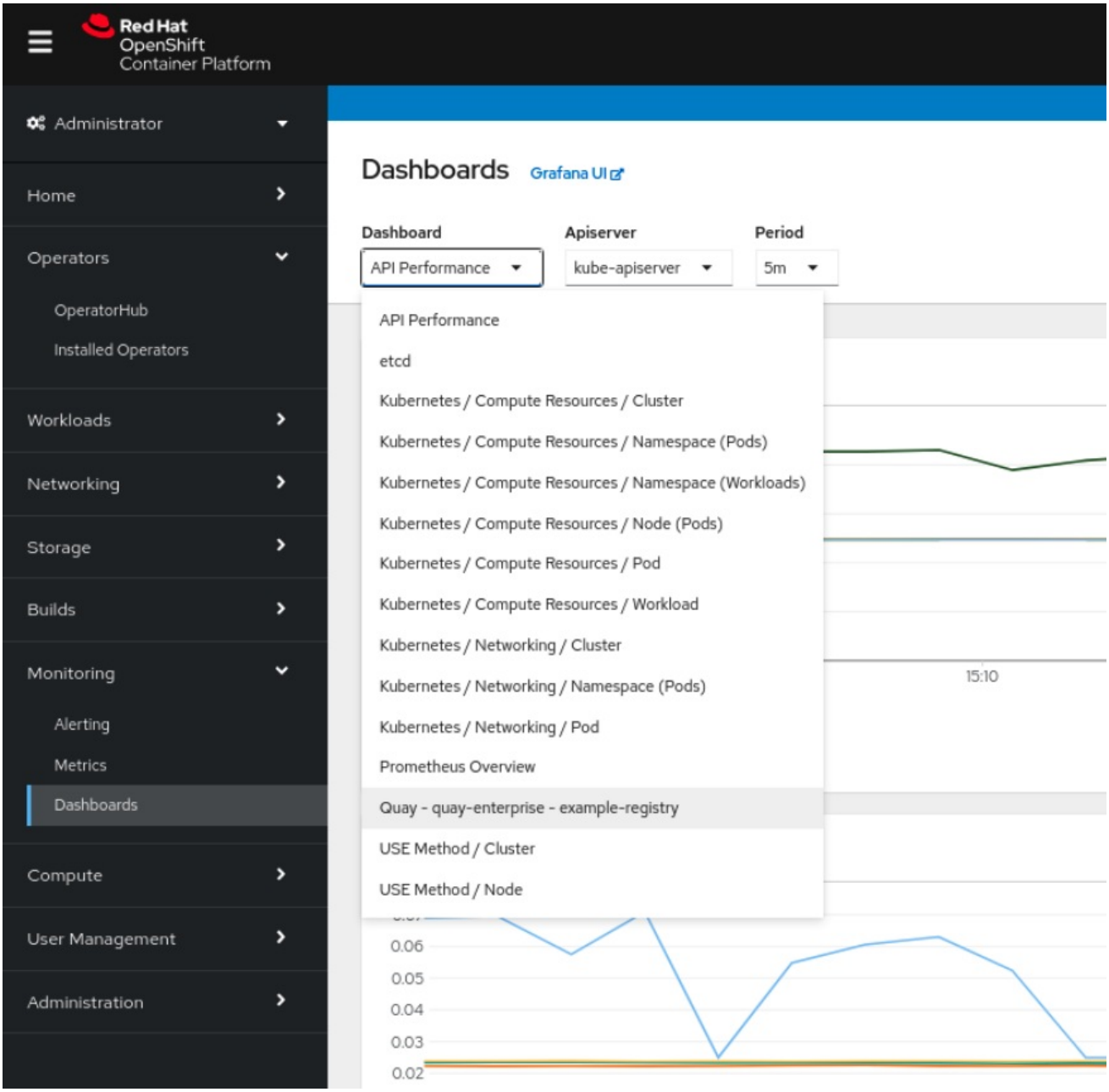
Red Hat Quay 3.5 は、OpenShift コンソールから Operator を使用してデプロイされた Quay インスタンスのモニターのサポートを提供します。新規のモニタリング機能には、Grafana ダッシュボード、個別のメトリクスへのアクセス、Quay Pod を頻繁に再起動するために通知するアラートなどが含まれます。

注記

モニタリング機能を有効にするには、Operator を「all namespace」モードでインストールする必要があります。

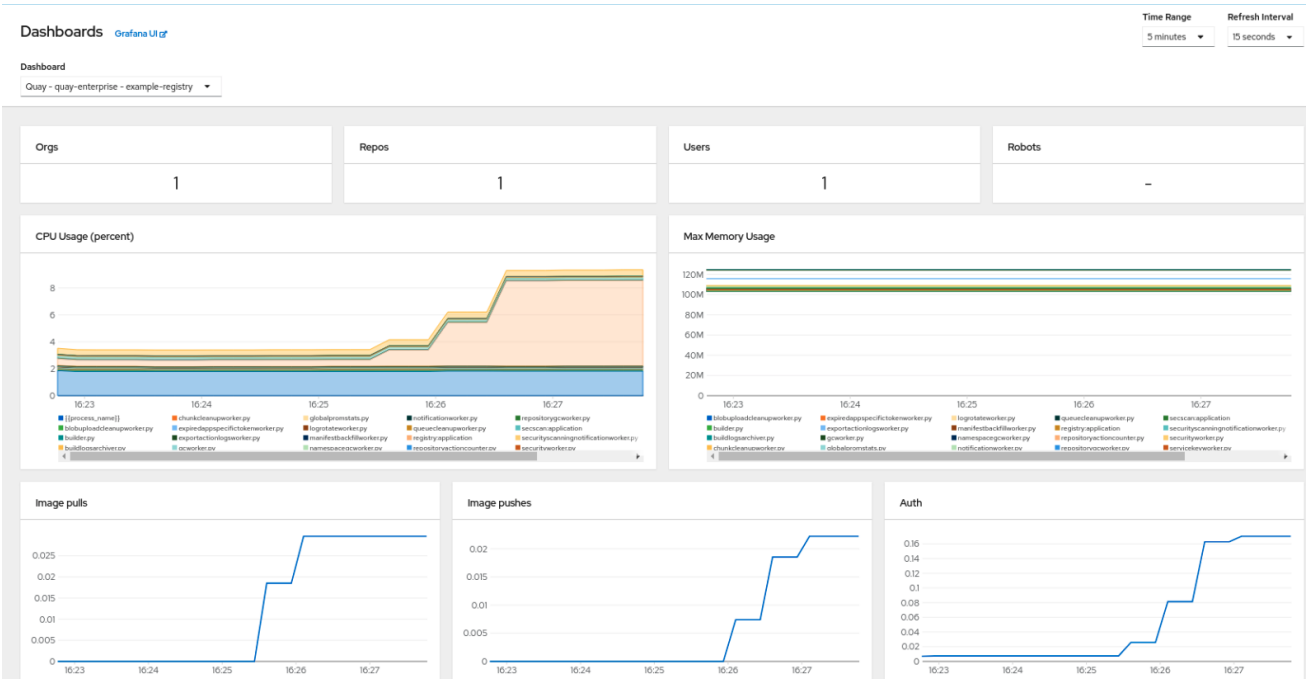
6.2.1. ダッシュボード

OpenShift コンソールで、Monitoring → Dashboards に移動し、必要な Quay レジストリーインスタンスのダッシュボードを検索します。



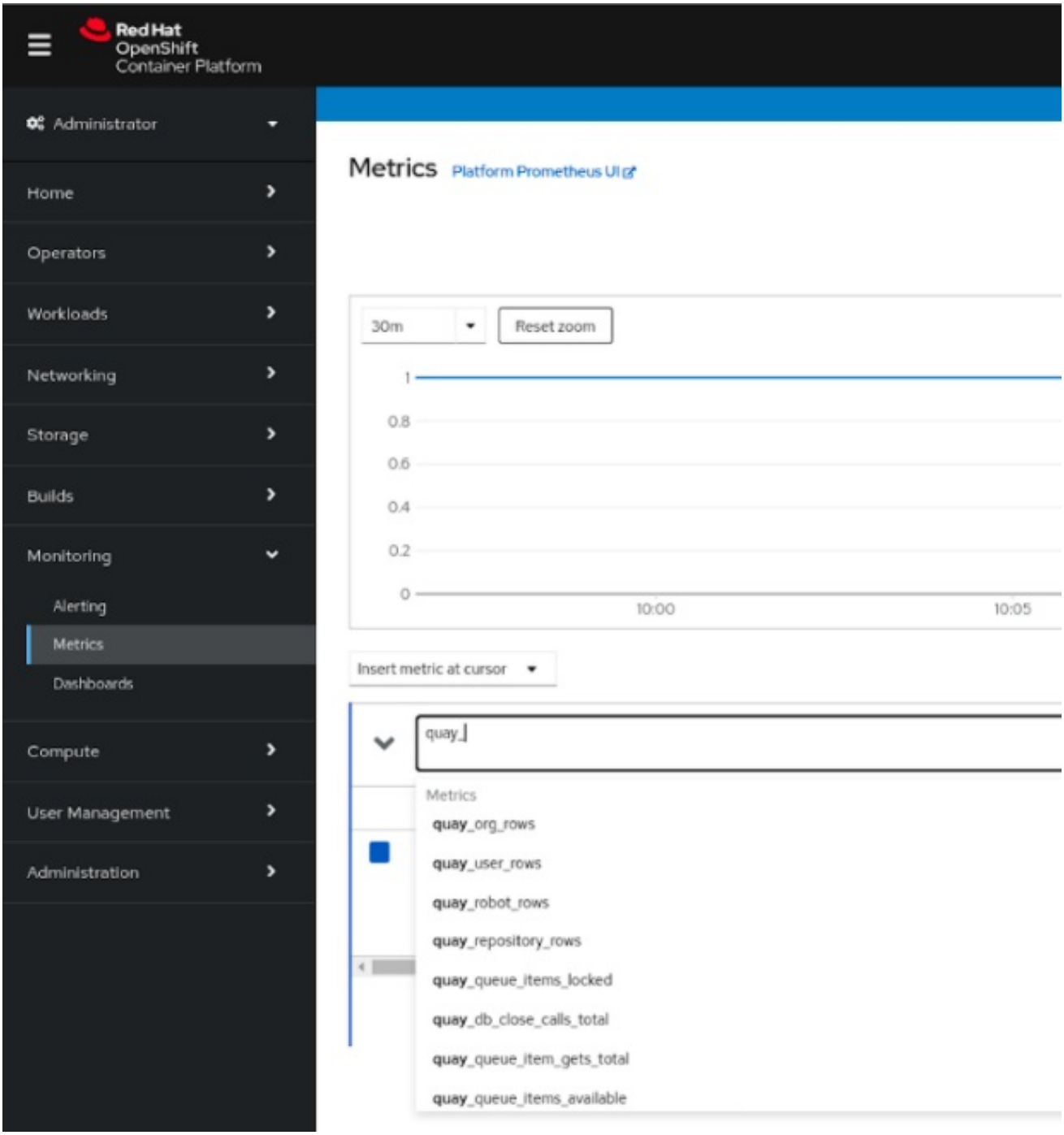
ダッシュボードには、以下を含むさまざまな統計が表示されます。

- Organization (組織)、Repository (リポジトリー)、User (ユーザー)、および Robot (ロボット) アカウントの数
- CPU 使用率および最大メモリー使用量
- イメージプルおよびプッシュのレート、および認証要求
- API 要求レート
- 待機時間

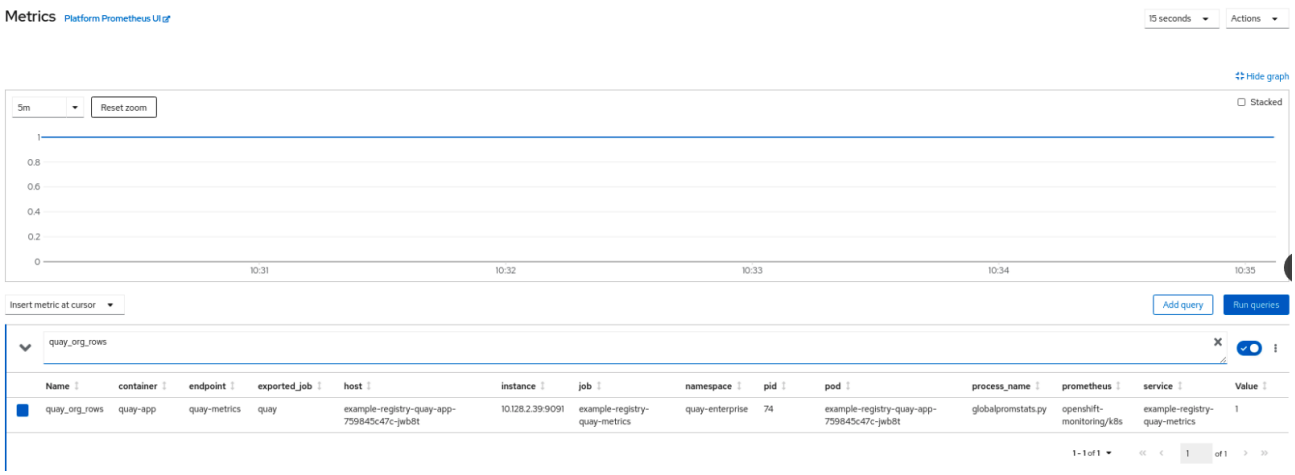


6.2.2. メトリクス

UI で Monitoring → Metrics にアクセスすることで、Quay ダッシュボードの背後にある基礎となるメトリクスを表示できます。Expression フィールドにテキストの **quay_** を入力し、利用可能なメトリクスの一覧を表示します。



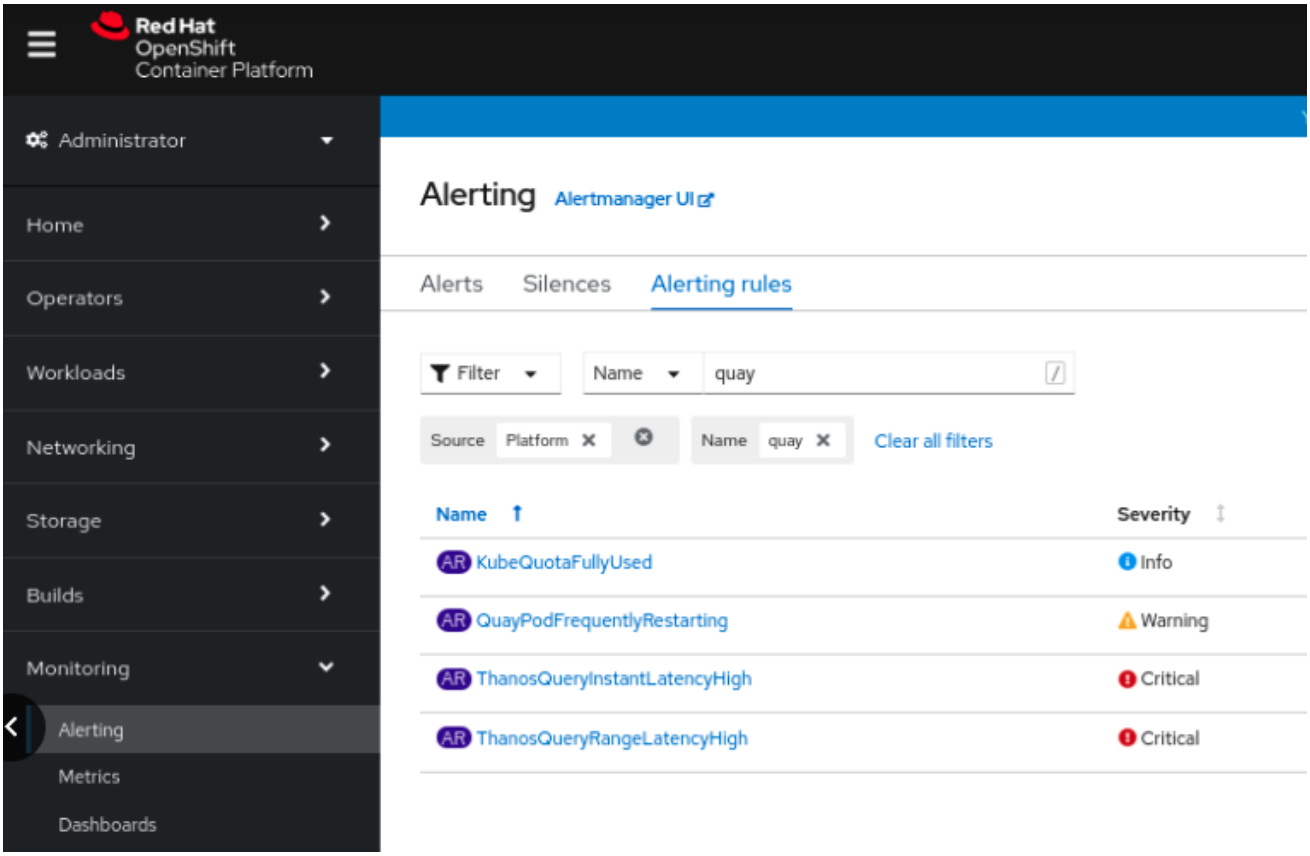
quay_org_rows など、サンプルメトリクスを選択します。



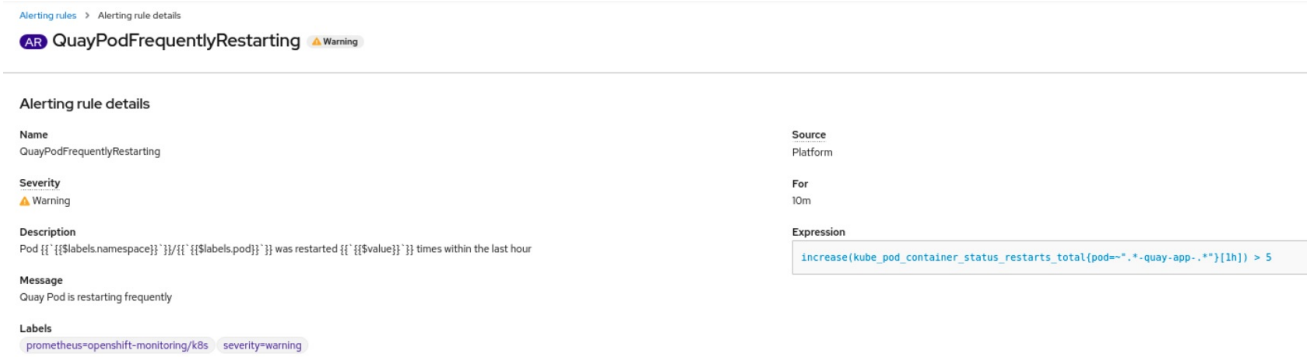
このメトリクスには、レジストリー内の組織の数が表示され、これはダッシュボードに直接表示されます。

6.2.3. アラート

Quay Pod が頻繁に再起動する場合はアラートが発生します。アラートは、コンソールI UI の Monitoring → Alerting から Alerting ルールタブにアクセスし、Quay 固有のアラートを検索して設定できます。



QuayPodFrequentlyRestarting ルールの詳細を選択し、アラートを設定します。



==== Clair 設定の取得

6.3. FIPS の READINESS およびコンプライアンス

FIPS(Federal Information Processing Standard: National Institute of Standards and Technology, NIST) は、特に銀行、ヘルスcare、パブリックセクターなどの非常に規制データの保護と暗号化のためのゴールド標準と見なされます。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 準拠が確保されます。

Red Hat Quay は、バージョン 3.5 以降の FIPS モードでの RHEL および OCP での実行をサポートします。さらに、Red Hat Quay 自体は、検証される暗号化ライブラリーのみ、または NIST が検証されるプロセスにある暗号ライブラリーのみにコミットします。Red Hat Quay 3.5 では、RHEL 8.3 暗号ライブラリーをベースとした保留中の FIPS 140-2 検証があります。検証が確定されると、Red Hat Quay は公式に FIPS に準拠します。

第7章 高度な概念

7.1. QUAY デプロイメントのカスタマイズ

Quay Operator は、Quay とその依存関係のデプロイに事前に設定されたストラテジーを使用しますが、Quay デプロイメントをカスタマイズできる場合もあります。

7.1.1. Quay アプリケーション設定

デプロイ後、設定エディター UI を使用するか、Quay 設定バンドルを含む **Secret** を変更して、Quay アプリケーション自体を通常の方法で設定できます。Operator は **spec.configBundleSecret** フィールドで名前が指定される **Secret** を使用しますが、このリソースで変更の有無についての監視は行いません。設定の変更を新規の **Secret** リソースに加え、**spec.configBundleSecret** フィールドを変更を反映するように更新することが推奨されます。新規の設定に問題がある場合は、 **spec.configBundleSecret** の値をより古い **Secret** に簡単に戻すことができます。

7.1.2. レジストリーへの外部アクセスのカスタマイズ

OpenShift で実行している場合、**Routes** API が利用可能になり、管理コンポーネントとして自動的に使用されます。**QuayRegistry** の作成後に、外部アクセスポイントは **QuayRegistry** のステータスブロックで確認できます。

```
status:
  registryEndpoint: some-quay.my-namespace.apps.mycluster.com
```

ネイティブ Kubernetes で実行される場合、Operator はレジストリー用に **type: ClusterIP** の Service を作成します。次に、外部アクセスを行います (**Ingress** の場合と同様)。

```
$ kubectl get services -n <namespace>
NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP         PORT(S)             AGE
some-quay               ClusterIP   172.30.143.199   <none>              443/TCP,9091/TCP    23h
```

7.1.2.1. カスタムのホスト名および TLS の使用

デフォルトで、**Route** はデフォルトの生成されたホスト名を使用して作成され、証明書/キーペアが TLS 用に生成されます。カスタムホスト名を使用して Red Hat Quay にアクセスし、独自の TLS 証明書/キーペアを使用する場合は、以下の手順を実行します。

FEATURE_BUILD_SUPPORT: true の場合、証明書/キーのペアが **BUILDMAN_HOSTNAME** についても有効であることを確認します。

上記のホスト名について指定の証明書/キーのペアが無効な場合、Quay Operator は提供される証明書/キーのペアを拒否し、Red Hat Quay で使用される証明書/キーのペアを生成します。

以下の内容を含む **Secret** を作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: my-config-bundle
data:
```

```
  config.yaml: <must include SERVER_HOSTNAME field with your custom hostname>
  ssl.cert: <your TLS certificate>
  ssl.key: <your TLS key>
```

次に、作成された **Secret** を参照する QuayRegistry を作成します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: some-quay
spec:
  configBundleSecret: my-config-bundle
```

7.1.2.2. OpenShift で提供される TLS 証明書の使用

TLS は Quay アプリケーションコンテナーで終了している状態であることが推奨されます。そのため、OpenShift で提供される TLS を使用するには、「reencrypt」タイプで **Route** を作成する必要があります。これは OpenShift で提供される TLS をエッジで使用し、Quay Operator が生成する TLS をクラスター内で使用します。これは、**route** コンポーネントに unmanaged (対象外) のマークを付け、Operator が生成する CA 証明書を使用して TLS を再暗号化する独自の **Route** を作成して実行します。

SERVER_HOSTNAME フィールドに **<route-name>-<namespace>.apps.<cluster-domain>** の値が含まれる **config.yaml** キーを使用して **Secret** を作成します (このホスト名の **Route** は後の手順で作成されます)。

```
apiVersion: v1
kind: Secret
metadata:
  name: my-config-bundle
data:
  config.yaml: <must include SERVER_HOSTNAME field with your custom hostname>
```

上記の **Secret** を参照する **QuayRegistry** を **route** コンポーネントを対象外にした状態で作成します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: some-quay
spec:
  configBundleSecret: my-config-bundle
  components:
  - kind: route
    managed: false
```

QuayRegistry が Quay Operator によって完全に調整されるのを待機します。次に、Quay アプリケーション Pod にマウントされている **Secret** を検索し、**tls.cert** 値をコピーして、生成された TLS 証明書を取得します。

TLS 再暗号化および上記でコピーした宛先 CA 証明書で **Route** を作成します。

```
apiVersion: v1
kind: Route
metadata:
  name: registry
```

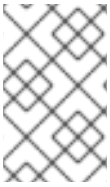
```
  namespace: <namespace>
spec:
  to:
    kind: Service
    name: <quay-service-name>
  tls:
    termination: reencrypt
    destinationCACertificate:
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

作成された **Route** を使用して Quay レジストリーにアクセスできます。

7.1.3. Route コンポーネントの無効化

Operator が **Route** を作成しないようにするには、**QuayRegistry** でコンポーネントに管理対象外のマークを付けます。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: some-quay
spec:
  components:
    - kind: route
      managed: false
```



注記

デフォルトの **Route** を無効にすると、Quay インスタンスにアクセスするために **Route**、**Service**、または **Ingress** を作成でき、使用するすべての DNS が Quay 設定の **SERVER_HOSTNAME** に一致する必要が生じマます。

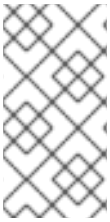
7.1.4. 管理ストレージのサイズ変更

Quay Operator は、**NooBaa** オブジェクト (50 Gib) の作成時に RHOCS によって提供されるデフォルトを使用してデフォルトのオブジェクトストレージを作成します。このストレージを拡張する方法は 2 つあります。既存の PVC のサイズを変更するか、または新規ストレージプールに PVC を追加することができます。

7.1.4.1. Noobaa PVC のサイズ変更

1. OpenShift コンソールにログインし、**Storage → Persistent Volume Claims** を選択します。
2. **noobaa-default-backing-store-noobaa-pvc-*** などの名前の **PersistentVolumeClaim** を選択します。
3. Action メニューから **Expand PVC** を選択します。
4. Persistent Volume Claim（永続ボリューム要求、PVC）の新規サイズを入力し、**Expand** を選択します。

数分後に (PVC のサイズによって異なる)、拡張されたサイズは PVC の **Capacity** フィールドに反映されるはずです。



注記

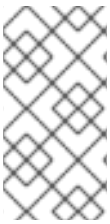
CSI ボリュームの拡張は、テクノロジープレビュー機能としてのみ利用できます。詳細は、https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/storage/expanding-persistent-volumes を参照してください。

7.1.4.2. 別のストレージプールの追加

1. OpenShift コンソールにログインし、**Networking → Routes** を選択します。**openshift-storage** プロジェクトが選択されていることを確認します。
2. **noobaa-mgmt** ルートの **Location** フィールドをクリックします。
3. Noobaa 管理コンソールにログインします。
4. メインダッシュボードの **Storage Resources** の下で、**Add Storage Resources** を選択します。
5. **Deploy Kubernetes Pool** を選択します。
6. 新しいプール名を入力します。**Next** をクリックします。
7. Pod 数を選択してプールを管理し、ノードごとのサイズを設定します。**Next** をクリックします。
8. **Delete** をクリックします。

数分後に、追加ストレージプールが Noobaa リソースに追加され、Red Hat Quay で利用できるようになります。

7.1.5. デフォルトの Operator イメージのカスタマイズ



注記

このメカニズムの使用は実稼働環境用の Quay 環境ではサポートされません。これは開発およびテストの目的でのみ使用することが強く推奨されます。Quay Operator でデフォルト以外のイメージを使用する場合、デプロイメントが適切に機能する保証はありません。

特定の状況では、Operator で使用されるデフォルトイメージを上書きすることが役に立つ場合があります。これは、Quay Operator の **ClusterServiceVersion** に 1 つ以上の環境変数を設定して実行できます。

7.1.5.1. 環境変数

以下の環境変数は、コンポーネントイメージを上書きするために Operator で使用されます。

環境変数	コンポーネント

RELATED_IMAGE_COMPONENT_QUAY	**base**
RELATED_IMAGE_COMPONENT_CLAIR	**clair**
RELATED_IMAGE_COMPONENT_POSTGRES	**postgres** および **clair** データベース
RELATED_IMAGE_COMPONENT_REDIS	**redis**

注記

上書きイメージは、タグ (:latest) ではなくマニフェスト (@sha256:) 参照される **必要** があります。

7.1.5.2. 実行中の Operator へのオーバーライドの適用

Quay Operator が Operator Lifecycle Manager (OLM) でクラスターにインストールされている場合、管理コンポーネントのコンテナーイメージは、クラスター内で実行中の Operator の OLM 表現である **ClusterServiceVersion** オブジェクトを変更して簡単に上書きできます。Kubernetes UI または **kubectl**/**oc** のいずれかを使用して、Quay Operator の **ClusterServiceVersion** を検索します。

```
$ oc get clusterserviceversions -n <your-namespace>
```

UI、**oc edit**、またはその他の方法を使用して Quay **ClusterServiceVersion** を変更し、上記の環境変数を追加して上書きイメージを参照します。

JSONPath: **spec.install.spec.deployments[0].spec.template.spec.containers[0].env**

```
- name: RELATED_IMAGE_COMPONENT_QUAY
  value:
quay.io/projectquay/quay@sha256:c35f5af964431673f4ff5c9e90bdf45f19e38b8742b5903d41c10cc7f63
39a6d
- name: RELATED_IMAGE_COMPONENT_CLAIR
  value:
quay.io/projectquay/clair@sha256:70c99feceb4c0973540d22e740659cd8d616775d3ad1c1698ddf71d
0221f3ce6
- name: RELATED_IMAGE_COMPONENT_POSTGRES
  value: centos/postgresql-10-
centos7@sha256:de1560cb35e5ec643e7b3a772ebaac8e3a7a2a8e8271d9e91ff023539b4dfb33
- name: RELATED_IMAGE_COMPONENT_REDIS
  value: centos/redis-32-
centos7@sha256:06dbb609484330ec6be6090109f1fa16e936afcf975d1cbc5fff3e6c7cae7542
```

これは Operator レベルで実行されるため、すべての QuayRegistry はこれらの同じオーバーライドを使用してデプロイされることに注意してください。

7.1.6. AWS S3 CloudFront

バックエンドレジストリーストレージに AWS S3 CloudFront を使用する場合は、以下の例のようにプライベートキーを指定します。

```
$ oc create secret generic --from-file config.yaml=./config_awss3cloudfront.yaml --from-file default-cloudfront-signing-key.pem=./default-cloudfront-signing-key.pem test-config-bundle
```

関連資料

- Red Hat Quay Operator についての詳細は、アップストリームの quay-operator プロジェクトを参照してください。