Red Hat Quay 3.7

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: 2023-08-23

Red Hat Quay 3.7 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

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

はじめに

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、OpenShift クラスターで Red Hat Quay をデプロイし、管理する簡単な方法を提供します。

Red Hat Quay 3.4.0 では、オペレーターは完全に書き直されており、より多くの Day 2 オペレーションをサポートするだけでなく、改善されたアウトオブボックスエクスペリエンスを提供しています。その結果、新しい Operator はよりシンプルな操作性になり、その特注的な指向性がより反映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次のとおりです。

- **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の 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

第1章 RED HAT QUAY OPERATOR の概要

本書では、Red Hat Quay Operator を使用して OpenShift で Red Hat Quay を設定、デプロイ、管理、およびアップグレードする手順を説明します。

以下を行う方法を示します。

- Red Hat Quay Operator をインストールする
- オブジェクトストレージ (managed または unmanaged のいずれか) の設定
- 必要に応じて、他の管理外のコンポーネント (データベース、Redis、ルート、TLS など) の設定
- Operator を使用した OpenShift への Red Hat Quay レジストリーのデプロイ
- Operator でサポートされる高度な機能の使用
- Operator のアップグレードによるレジストリーのアップグレード

1.1. 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、フィールドについては以下のセクションで説明されています。

1.2. QUAY OPERATOR コンポーネント

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

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

```
spec:
  components:
    - kind: quay
      managed: true
```

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

1.3. 管理コンポーネントの使用

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

- **quay:** 環境変数やレプリカの数など、Quay デプロイメントのオーバーライドを保持します。このコンポーネントは Red Hat Quay 3.7 の新機能であり、アンマネージドに設定することはできません。
- **postgres:** レジストリーメタデータを保存するには、Software Collections から Postgres 10 のバージョンを使用します。
- **clair:** イメージの脆弱性スキャンを提供します。
- **redis:** Quay ビルダーの調整および一部の内部ロギングを処理します。
- **horizontalpodautoscaler:** メモリー/CPU の消費に応じて Quay Pod 数を調整します。
- **ObjectStorage:** イメージレイヤー Blob を格納するには、Noobaa/RHOCS によって提供される **ObjectBucketClaim** Kubernetes API を使用します。
- **route:** OpenShift の外部から Quay レジストリーへの外部エントリーポイントを提供します。
- **mirror:** リポジトリーミラーワーカーを設定します (オプションのリポジトリーミラーリングをサポートするため)。
- **monitoring:** Grafana ダッシュボード、個別のメトリクスへのアクセス、Quay Pod が頻繁に再起動されていることを通知するアラートなどが含まれます。
- **tls:** Red Hat Quay または OpenShift が TLS を処理するかどうかを設定します。
- **clairpostgres:** 管理された Clair データベースを設定します

Operator は Red Hat Quay が管理コンポーネントを使用するために必要な設定およびインストール作業を処理します。Quay Operator によって実行される事前に設定されたデプロイメントがお使いの環境に

適さない場合、以下のセクションで説明されているように Operator に **unmanaged** のリソース (オーバーライド) を指定できます。

1.4. 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は、まず Quay 設定バンドル (**config.yaml**) 内でそれらを設定し、**QuayRegistry** でバンドルを参照します (Kubernetes **Secret**)。これは、非管理対象のコンポーネントを示します。

注記

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

1.5. 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** の **metadata.name** への参照です。この **Secret** には **config.yaml** のキー/値の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、指定されていないと Operator によって自動入力されます。これは指定されていると、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 **Secret** を形成します。

1.6. OPENSHIFT での RED HAT QUAY の前提条件

OpenShift での Red Hat Quay Operator のデプロイメントを開始する前に、以下を考慮する必要があります。

1.6.1. 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。そのアカウントには、namespace をクラスタースコープで作成できる必要があります。

1.6.2. 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。

1.6.3. オブジェクトストレージ

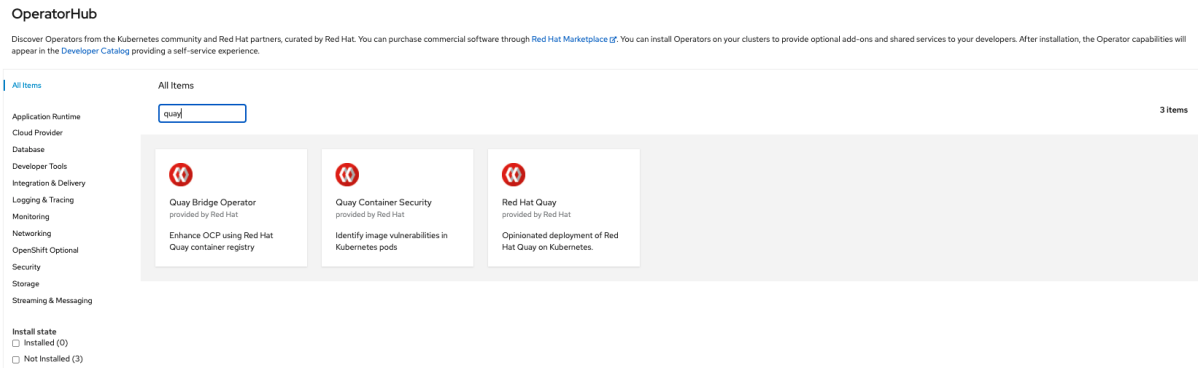
デフォルトで、Red Hat Quay Operator は **ObjectBucketClaim** Kubernetes API を使用してオブジェクトストレージをプロビジョニングします。この API を消費すると、ベンダー固有の実装から Operator を分離します。Red Hat OpenShift Data Foundation は、この例で使用される NooBaa コンポーネントを介してこの API を提供します。

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。

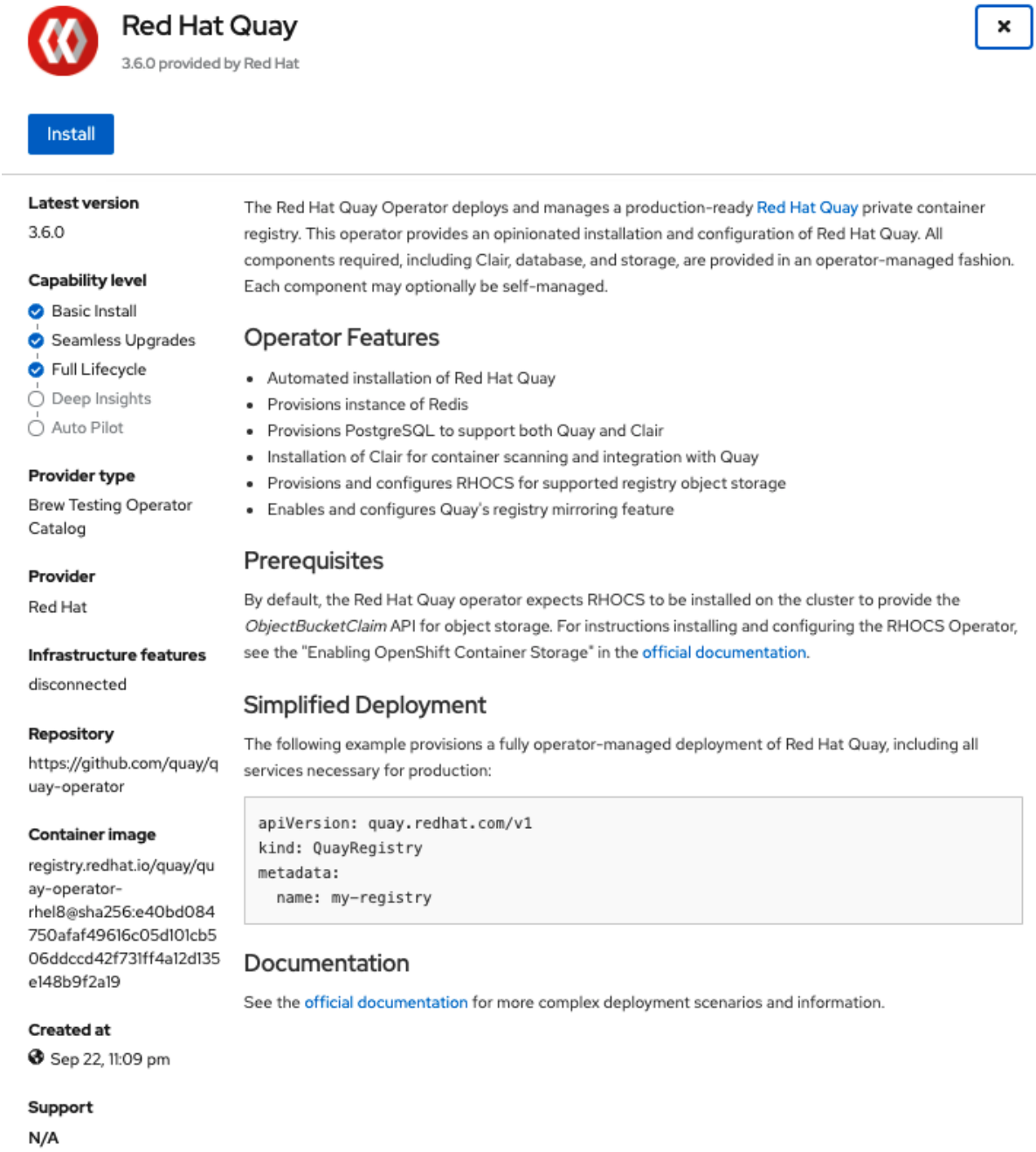
- Amazon S3 (Red Hat Quay 用の S3 バケットポリシーの設定については S3 IAM Bucket Policy を参照)
- Azure Blob Storage
- Google Cloud Storage
- Ceph Object Gateway(RADOS)
- OpenStack Swift
- CloudFront + S3

第2章 OPERATORHUB からの QUAY OPERATOR のインストール

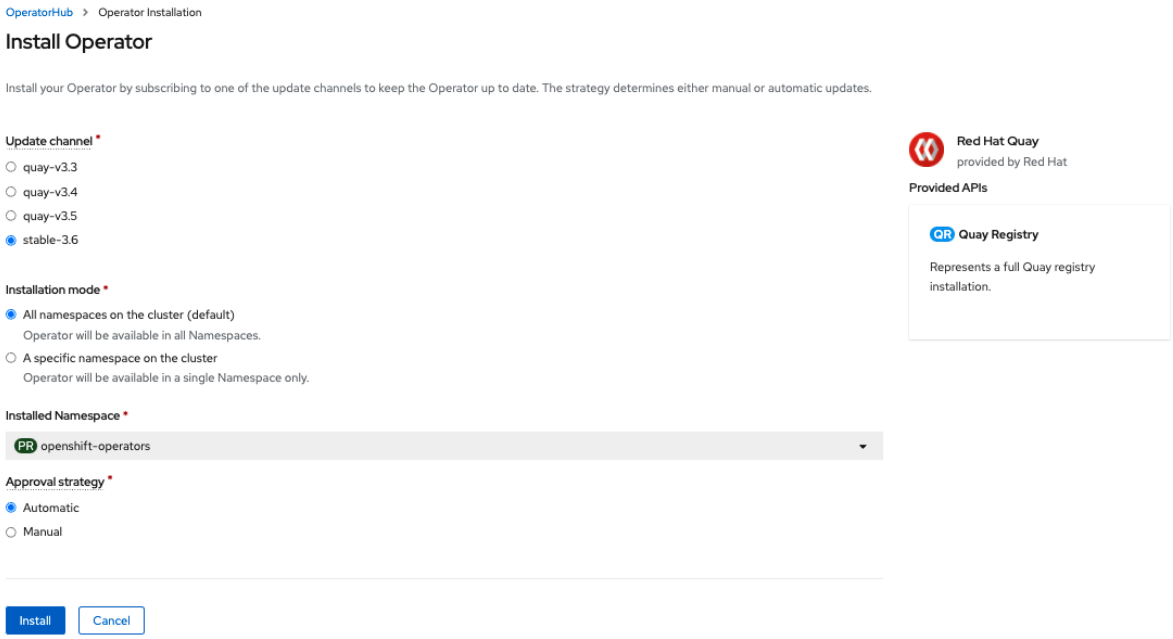
1. OpenShift コンソールを使用して、Operators → OperatorHub を選択してから Red Hat Quay Operator を選択します。コミュニティーバージョンが複数ある場合は、必ず Red Hat 認定 Operator を使用し、コミュニティーバージョンは使用しないでください。



2. Installation ページでは、機能と前提条件の概要を説明します。



3. インストールを選択します。Operator Installation ページが表示されます。



4. インストールのカスタマイズには、以下の選択肢を使用できます。

   - **Update Channel:** 更新チャネルを選択します。たとえば、最新リリースの場合は **stable-3.7** を選択します。

   - **Installation Mode:** Operator をクラスター全体で使用できるようにする場合は、**All namespaces on the cluster** を選択します。これを単一の namespace 内にのみデプロイする必要がある場合は、**A specific namespace on the cluster** を選択します。クラスター全体で Operator をインストールすることが推奨されます。単一 namespace を選択する場合、モニタリングコンポーネントはデフォルトで利用できなくなります。

   - **Approval Strategy:** 自動更新または手動更新のいずれかを承認します。自動更新ストラテジーが推奨されます。

5. インストールを選択します。

6. しばらくすると、Operator が Installed Operators ページに正常にインストールされていることを確認できます。

第3章 デプロイメント前の QUAY の設定

Operator は OpenShift へのデプロイ時にすべての Red Hat Quay コンポーネントを管理できます。これはデフォルト設定です。または、1 つ以上のコンポーネントを外部から管理でき、セットアップに対する制御を強化し、Operator が残りのコンポーネントを管理できるようにできます。

管理外のコンポーネントを設定するための標準的なパターンは以下のとおりです。

1. 適切な設定で **config.yaml** 設定ファイルを作成します。
2. 設定ファイルを使用してシークレットを作成します。

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

3. QuayRegistry YAML ファイル **quayregistry.yaml** を作成し、管理対象外コンポーネントを特定し、作成された Secret も参照します。以下に例を示します。

quayregistry.yaml

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

4. YAML ファイルを使用してレジストリーをデプロイします。

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

3.1. 自動化のための RED HAT QUAY の事前設定

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

3.1.1. API による最初のユーザー作成の許可

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

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

3.1.2. API 一般アクセスの有効化

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

3.1.3. スーパーユーザーの追加

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

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

3.1.4. ユーザー作成の制限

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

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

3.1.5. 新機能の有効化

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

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

3.1.6. 自動化の推奨設定

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

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

3.2. オブジェクトストレージの設定

Operator がストレージを管理したり、独自に管理することを許可しているかに関係なく、Red Hat Quay をインストールする前にオブジェクトストレージを設定する必要があります。

Operator でストレージの管理を担当する必要がある場合は、NooBaa/RHOCS Operator をインストールし、設定する方法について Managed storage セクションを参照してください。

別のストレージソリューションを使用している場合は、Operator の設定時に **objectstorage** を **Unmanaged** (管理外) として設定します。以下のセクションを参照してください。既存のストレージの設定の詳細は、アンマネージドストレージ を参照してください。

3.2.1. アンマネージドストレージ

アンマネージドストレージの設定例については、本セクションで紹介しています。オブジェクトストレージの設定についての詳細は、Red Hat Quay 設定ガイドを参照してください。

3.2.1.1. AWS S3 ストレージ

```
DISTRIBUTED_STORAGE_CONFIG:
  s3Storage:
    - S3Storage
    - host: s3.us-east-2.amazonaws.com
      s3_access_key: ABCDEFGHIJKLMN
      s3_secret_key: OL3ABCDEFGHIJKLMN
      s3_bucket: quay_bucket
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - s3Storage
```

3.2.1.2. Google Cloud storage

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

3.2.1.3. Azure ストレージ

```
DISTRIBUTED_STORAGE_CONFIG:
  azureStorage:
    - AzureStorage
    - azure_account_name: azure_account_name_here
      azure_container: azure_container_here
      storage_path: /datastorage/registry
```

```
    azure_account_key: azure_account_key_here
    sas_token: some/path/
    endpoint_url: https://[account-name].blob.core.usgovcloudapi.net (1)
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - azureStorage
```

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

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

3.2.1.4. Ceph / RadosGW ストレージ / 日立 HCP ストレージ

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

3.2.1.5. Swift ストレージ:

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

3.2.1.6. NooBaa アンマネージドストレージ

次の手順を使用して、NooBaa を管理対象外のストレージ設定としてデプロイします。

手順

1. **Storage → Object Bucket Claims** に移動して、{product-title} コンソールで NooBaa Object Bucket Claim を作成します。

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

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

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

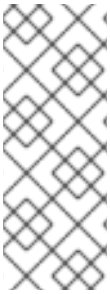
Object Bucket Claim の設定の詳細は、オブジェクトバケットクレーム を参照してください。

3.2.2. マネージドストレージ

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は、クラスターが **ObjectBucketClaim** API 経由でオブジェクトストレージを提供できるようにする必要があります。Red Hat OpenShift Data Foundations (ODF) Operator を使用する場合は、サポートされるオプションを 2 つ使用できます。

- 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。



注記

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。

3.2.2.1. 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イスを指定できます。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。

3.2.2.1.1. 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 Data Foundation Operator を選択します。
2. インストールを選択します。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。

> When the installation of the ODF Operator is complete, you are prompted to create a storage system. Do not follow this instruction. Instead, create NooBaa object storage as outlined the following steps.

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。

3.3. データベースの設定

3.3.1. 既存の Postgres データベースの使用

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

config.yaml:

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

2. 設定ファイルを使用してシークレットを作成します。

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

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

quayregistry.yaml

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

4. 以下のセクションで説明されているようにレジストリーをデプロイします。

3.3.2. データベースの設定

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

3.3.2.1. データベース URI

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

以下の表は **DB_URI** 設定フィールドについて説明しています。

表3.1 データベース URI

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

3.3.2.2. データベース接続引数

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

以下の表は、データベース接続引数について説明しています。

表3.2 データベース接続引数

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

3.3.2.2.1. PostgreSQL SSL 接続引数

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

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

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

表3.3 SSL オプション

Mode	説明
disable	設定は SSL 以外の接続のみを試みます。
allow	設定は、SSL 以外の接続を最初に試行します。障害が発生したときに、SSL 接続を試行します。
prefer (デフォルト)	設定は最初に SSL 接続を試みます。障害が発生したときに、SSL 以外の接続を試みます。
require	設定は SSL 接続のみを試みます。ルート CA ファイルが存在する場合は、verify-ca が指定されているのと同じ方法で証明書を検証します。
verify-ca	設定は SSL 接続のみを試行し、サーバー証明書が信頼できる認証局 (CA) によって発行されたことを確認します。
verify-full	SSL 接続のみを試行し、信頼された CA によりサーバー証明書が発行され、要求されたサーバーのホスト名が証明書と一致することを確認します。

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

3.3.2.2.2. MySQL SSL 接続引数

以下の例は、MySQL SSL 設定の例を示しています。

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

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

3.3.3. マネージド PostgreSQL の使用

推奨事項:

- 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 **Pods** 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。

3.4. TLS およびルートの設定

OpenShift Container Platform のエッジターミネーションルートのサポートが新しいマネージドコンポーネント **tls** を介して追加されました。これにより、**route** コンポーネントが TLS から分離され、ユーザーは両方を個別に設定できるようになります。**EXTERNAL_TLS_TERMINATION: true** は事前に設定された設定です。マネージド **tls** は、デフォルトのクラスターワイルドカード証明書が使用されることを意味します。アンマネージド **tls** は、ユーザーが指定した証明書/キーのペアが **Route** に挿入されることを意味します。

ssl.cert および **ssl.key** は、個別の永続的なシークレットに移動しました。これにより、調整のたびに証明書とキーのペアが再生成されないようになります。これらは **edge** ルートとしてフォーマットされ、Quay コンテナーの同じディレクトリーにマウントされます。

TLS およびルートを設定する際には、複数の調整が可能ですが、以下のルールが適用されます。

- TLS が **managed** されている場合は、ルートも **managed** する必要があります。
- TLS が **unmanaged** の場合は、設定ツールを使用するか、設定バンドルに直接証明書を指定する必要があります。

次の表に、有効なオプションを示します。

表3.4 TLS およびルートの有効な設定オプション

オプション	ルート	TLS	提供される証明書	結果
独自のロードバランサーが TLS を処理する	マネージド	マネージド	いいえ	デフォルトのワイルドカード証明書を使用したエッジルート
Red Hat Quay が TLS を処理する	マネージド	アンマネージド	はい	Pod 内にマウントされる証明書を含むパススルールート
Red Hat Quay が TLS を処理する	アンマネージド	アンマネージド	はい	証明書は quay Pod 内に設定されますが、ルートは手動で作成する必要があります。

注記

Red Hat Quay 3.7 は、TLS が Operator で管理される場合にビルダーをサポートしません。

3.4.1. TLS 証明書、キーペアを使用して設定バンドルシークレットの作成

独自の TLS 証明書およびキーを追加するには、以下のように設定バンドルシークレットに追加します。

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
```

3.5. 他のコンポーネントの設定

3.5.1. 外部 Redis の使用

外部の Redis データベースを使用する場合は、**QuayRegistry** インスタンスでコンポーネントをアンマネージドに設定します。

1. 必要な redis フィールドで設定ファイル **config.yaml** を作成します。

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

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

2. 設定ファイルを使用してシークレットを作成します。

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

3. redis 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。

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

4. レジストリーをデプロイします。

3.5.1.1. Redis 設定フィールド

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

3.5.1.1.1. ビルドログ

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

表3.5 ビルドログの設定

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

3.5.1.1.2. ユーザーイベント

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

表3.6 ユーザーイベント設定

フィールド	タイプ	説明
USER_EVENTS_REDIS (必須)	オブジェクト	ユーザーイベント処理の Redis 接続の詳細
.host (必須)	文字列	Redis にアクセスできるホスト名。 **例:** **quay-server.example.com**

フィールド	タイプ	説明
.port (必須)	数値	Redis にアクセスできるポート。 **例:** **6379**
.password	文字列	Redis にアクセスできるポート **例:** **strongpassword**
ssl	ブール値	Redis と Quay 間の TLS 通信を有効にするかどうか。デフォルトは false です。

3.5.1.1.3. redis の設定例

以下の YAML は、Redis を使用した設定例を示しています。

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

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

注記

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

3.5.2. Horizontal Pod Autoscaler の無効化

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

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

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

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

```
  name: example-registry
  namespace: quay-enterprise
spec:
  components:
    - kind: horizontalpodautoscaler
      managed: false
```

3.5.3. 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、以下を実行します。

1. **QuayRegistry** でコンポーネントを管理対象外としてマークします。

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

2. **config.yaml** ファイルを編集して Quay が設定で TLS を処理するように指定します。

config.yaml

```
...
EXTERNAL_TLS_TERMINATION: false
...
SERVER_HOSTNAME: example-registry-quay-quay-enterprise.apps.user1.example.com
...
PREFERRED_URL_SCHEME: https
...
```

管理外のルートを正しく設定しないと、以下のようなエラーが表示されます。

```
{
  {
    "kind":"QuayRegistry",
    "namespace":"quay-enterprise",
    "name":"example-registry",
    "uid":"d5879ba5-cc92-406c-ba62-8b19cf56d4aa",
    "apiVersion":"quay.redhat.com/v1",
    "resourceVersion":"2418527"
  },
  "reason":"ConfigInvalid",
  "message":"required component `route` marked as unmanaged, but `configBundleSecret` is
missing necessary fields"
}
```

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** を作成し、使用するすべての DNS が Quay 設定の **SERVER_HOSTNAME** に一致する必要があることを意味します。

3.5.4. 管理外のモニタリング

Quay Operator を単一 namespace にインストールする場合、モニタリングコンポーネントは 'unmanaged' に自動的に設定されます。このシナリオでモニタリングを有効にするには、「Operator が単一 namespace にインストールされている場合のモニタリングの有効化」 セクションを参照してください。

モニタリングを明示的に無効にするには、以下を実行します。

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

3.5.5. 非管理ミラーリング

ミラーリングを明示的に無効にするには、以下を実行します。

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

第4章 QUAY OPERATOR を使用した QUAY のデプロイ

Operator はコマンドラインまたは OpenShift コンソールからデプロイできますが、基本的な手順は同じです。

4.1. コマンドラインからの RED HAT QUAY のデプロイ

1. namespace (例: **quay-enterprise**) を作成します。
2. デプロイメントのあらゆる側面を事前に設定する必要がある場合は、設定バンドルのシークレットを作成します。
3. **quayregistry.yaml** という名前のファイルに **QuayRegistry** カスタムリソースを作成します。
   a. 最小限のデプロイメントでは、すべてのデフォルトを使用します。

   quayregistry.yaml:

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

   b. 一部のコンポーネントをアンマネージドにする必要がある場合、この情報を **spec** フィールドに追加します。たとえば、最小限のデプロイメントは以下のようになります。

   quayregistry.yaml:

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

   c. 設定バンドル (例: **init-config-bundle-secret**) を作成している場合は、これを **quayregistry.yaml** ファイルで参照します。

   quayregistry.yaml:

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

```
  namespace: quay-enterprise
spec:
  configBundleSecret: init-config-bundle-secret
```

d. プロキシーを設定している場合は、Quay、Clair、およびミラーリングのオーバーライドを使用して情報を追加できます。

quayregistry.yaml:

```
  kind: QuayRegistry
  metadata:
    name: quay37
  spec:
    configBundleSecret: config-bundle-secret
    components:
      - kind: objectstorage
        managed: false
      - kind: route
        managed: true
      - kind: mirror
        managed: true
        overrides:
          env:
            - name: DEBUGLOG
              value: "true"
            - name: HTTP_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: HTTPS_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
      - kind: tls
        managed: false
      - kind: clair
        managed: true
        overrides:
          env:
            - name: HTTP_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: HTTPS_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
      - kind: quay
        managed: true
        overrides:
          env:
            - name: DEBUGLOG
              value: "true"
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
            - name: HTTP_PROXY
```

```
      value: quayproxy.qe.devcluster.openshift.com:3128
    - name: HTTPS_PROXY
      value: quayproxy.qe.devcluster.openshift.com:3128
```

4. 指定された namespace に **QuayRegistry** を作成します。

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

5. デプロイメントの進捗を追跡する方法については、デプロイメントプロセスの監視およびデバッグ セクションを参照してください。

6. **status.registryEndpoint** が設定されるまで待機します。

```
$ oc get quayregistry -n quay-enterprise example-registry -o jsonpath="
{.status.registryEndpoint}" -w
```

4.1.1. コマンドラインで作成されたコンポーネントの表示

oc get pods コマンドを使用して、デプロイされたコンポーネントを表示します。

```
$ oc get pods -n quay-enterprise

NAME                                             READY   STATUS      RESTARTS   AGE
example-registry-clair-app-5ffc9f77d6-jwr9s      1/1     Running     0          3m42s
example-registry-clair-app-5ffc9f77d6-wgp7d      1/1     Running     0          3m41s
example-registry-clair-postgres-54956d6d9c-rgs8l 1/1     Running     0          3m5s
example-registry-quay-app-79c6b86c7b-8qnr2       1/1     Running     4          3m42s
example-registry-quay-app-79c6b86c7b-xk85f       1/1     Running     4          3m41s
example-registry-quay-app-upgrade-5kl5r          0/1     Completed   4          3m50s
example-registry-quay-config-editor-597b47c995-svqrl 1/1 Running     0          3m42s
example-registry-quay-database-b466fc4d7-tfrnx   1/1     Running     2          3m42s
example-registry-quay-mirror-6d9bd78756-6lj6p    1/1     Running     0          2m58s
example-registry-quay-mirror-6d9bd78756-bv6gq    1/1     Running     0          2m58s
example-registry-quay-postgres-init-dzbmx        0/1     Completed   0          3m43s
example-registry-quay-redis-8bd67b647-skgqx      1/1     Running     0          3m42s
```

4.1.2. Horizontal Pod Autoscaling (HPA)

デフォルトのデプロイメントでは、以下の実行中の Pod が表示されます。

- Quay アプリケーション自体の 2 つの Pod (**example-registry-quay-app-*'**)
- Quay ロギング用の 1 つの Redis Pod (**example-registry-quay-redis-***)
- メタデータストレージ用に Quay が使用する PostgreSQL の 1 つのデータベース Pod (**example-registry-quay-database-***)
- Quay 設定エディターの 1 つの Pod (**example-registry-quay-config-editor-***)
- 2 つの Quay ミラーリング Pod (**example-registry-quay-mirror-***)
- Clair アプリケーションの 2 つの Pod (**example-registry-clair-app-***)
- Clair 用の 1 つの PostgreSQL Pod (**example-registry-clair-postgres-***)

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

```
$ oc get hpa -n quay-enterprise
NAME                        REFERENCE                           TARGETS          MINPODS   MAXPODS
REPLICAS   AGE
example-registry-clair-app     Deployment/example-registry-clair-app     16%/90%, 0%/90%   2
10        2          13d
example-registry-quay-app      Deployment/example-registry-quay-app      31%/90%, 1%/90%   2
20        2          13d
example-registry-quay-mirror   Deployment/example-registry-quay-mirror   27%/90%, 0%/90%   2
20        2          13d
```

4.1.3. API を使用した Red Hat Quay のデプロイ

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

前提条件

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

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

4.1.3.1. API を使用した最初のユーザーの作成

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

注記

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

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

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

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

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

データベースにユーザーが存在している場合は、エラーが返されます。

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

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

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

4.1.4. デプロイメントプロセスの監視およびデバッグ

ユーザーは、デプロイメントフェーズ中に問題のトラブルシューティングを行えるようになりました。**QuayRegistry** オブジェクトのステータスは、デプロイメント時にコンポーネントの正常性をモニターするのに役立ちます。これにより、発生する可能性のある問題のデバッグに役立ちます。

```
$ oc get quayregistry -n quay-enterprise -o yaml
```

デプロイメント直後に、QuayRegistry オブジェクトに基本設定が表示されます。

```
apiVersion: v1
items:
- apiVersion: quay.redhat.com/v1
  kind: QuayRegistry
  metadata:
    creationTimestamp: "2021-09-14T10:51:22Z"
    generation: 3
    name: example-registry
    namespace: quay-enterprise
    resourceVersion: "50147"
    selfLink: /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
    uid: e3fc82ba-e716-4646-bb0f-63c26d05e00e
  spec:
    components:
    - kind: postgres
      managed: true
    - kind: clair
      managed: true
    - kind: redis
      managed: true
    - kind: horizontalpodautoscaler
      managed: true
    - kind: objectstorage
      managed: true
    - kind: route
      managed: true
    - kind: mirror
      managed: true
    - kind: monitoring
      managed: true
    - kind: tls
      managed: true
    configBundleSecret: example-registry-config-bundle-kt55s
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

oc get pods コマンドを使用して、デプロイされたコンポーネントの現在の状態を表示します。

```
$ oc get pods -n quay-enterprise

NAME                                               READY   STATUS              RESTARTS   AGE
example-registry-clair-app-86554c6b49-ds7bl        0/1     ContainerCreating   0          2s
example-registry-clair-app-86554c6b49-hxp5s        0/1     Running             1          17s
example-registry-clair-postgres-68d8857899-lbc5n   0/1     ContainerCreating   0          17s
example-registry-quay-app-upgrade-h2v7h            0/1     ContainerCreating   0          9s
example-registry-quay-config-editor-5f646cbcb7-lbnc2   0/1     ContainerCreating   0          17s
example-registry-quay-database-66f495c9bc-wqsjf    0/1     ContainerCreating   0          17s
example-registry-quay-mirror-854c88457b-d845g      0/1     Init:0/1            0          2s
example-registry-quay-mirror-854c88457b-fghxv      0/1     Init:0/1            0          17s
example-registry-quay-postgres-init-bktdt          0/1     Terminating         0          17s
example-registry-quay-redis-f9b9d44bf-4htpz        0/1     ContainerCreating   0          17s
```

デプロイメントが進行中の間、QuayRegistry オブジェクトに現在のステータスが表示されます。この場合、データベースの移行が行われ、その他のコンポーネントはこれが完了するまで待機します。

```
  status:
    conditions:
    - lastTransitionTime: "2021-09-14T10:52:04Z"
      lastUpdateTime: "2021-09-14T10:52:04Z"
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: "False"
      type: RollOutBlocked
    - lastTransitionTime: "2021-09-14T10:52:05Z"
      lastUpdateTime: "2021-09-14T10:52:05Z"
      message: running database migrations
      reason: MigrationsInProgress
      status: "False"
      type: Available
    configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-btbkcg8dc9
    configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.quayteam.org
    lastUpdated: 2021-09-14 10:52:05.371425635 +0000 UTC
    unhealthyComponents:
      clair:
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-clair-postgres: Deployment does not have minimum
availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
        type: Available
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-clair-app: Deployment does not have minimum
availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
        type: Available
      mirror:
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-quay-mirror: Deployment does not have minimum
```

```
availability.'
      reason: MinimumReplicasUnavailable
      status: "False"
      type: Available
```

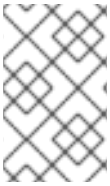
デプロイメントプロセスが正常に終了すると、QuayRegistry オブジェクトのステータスには正常でないコンポーネントが表示されません。

```
  status:
    conditions:
    - lastTransitionTime: "2021-09-14T10:52:36Z"
      lastUpdateTime: "2021-09-14T10:52:36Z"
      message: all registry component healthchecks passing
      reason: HealthChecksPassing
      status: "True"
      type: Available
    - lastTransitionTime: "2021-09-14T10:52:46Z"
      lastUpdateTime: "2021-09-14T10:52:46Z"
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: "False"
      type: RolloutBlocked
    configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-hg7gg7h57m
    configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.quayteam.org
    currentVersion: {producty}
    lastUpdated: 2021-09-14 10:52:46.104181633 +0000 UTC
    registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
    unhealthyComponents: {}
```

4.2. OPENSHIFT コンソールからの RED HAT QUAY のデプロイ

1. namespace (例: **quay-enterprise**) を作成します。
2. 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。
3. Provided APIs の下で Quay Registry タイルの Create Instance をクリックします。
4. 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入力されています。
5. 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。
6. **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。
7. 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 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。

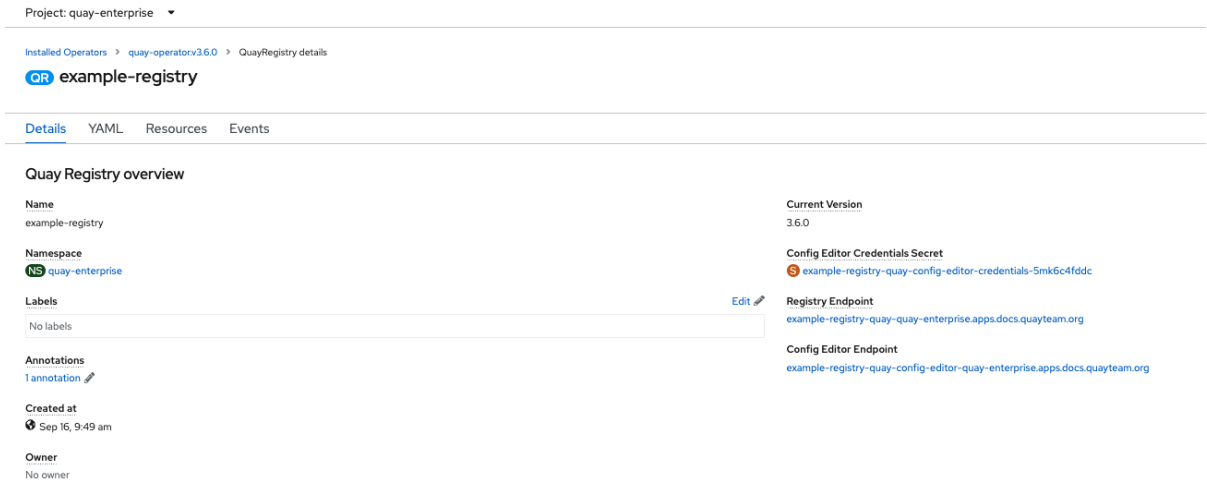
4.2.1. Quay UI を使用した最初のユーザーの作成



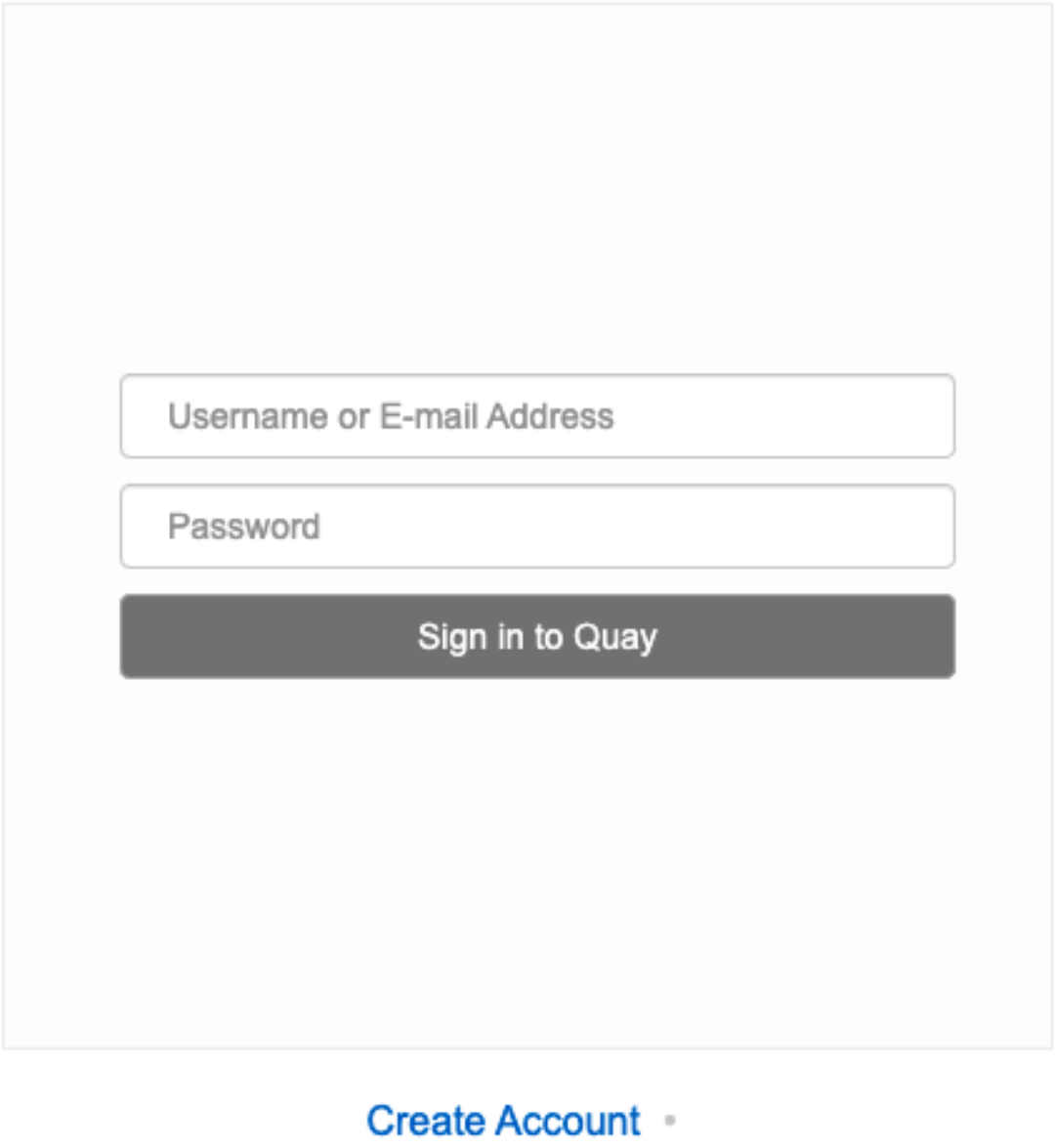
注記

この手順では、**FEATURE_USER_CREATION** 設定オプションが **false** に設定されていることを前提としています。**false** の場合は、UI での **Create Account** 機能が無効になり、API を使用して最初のユーザーを作成する必要があります。

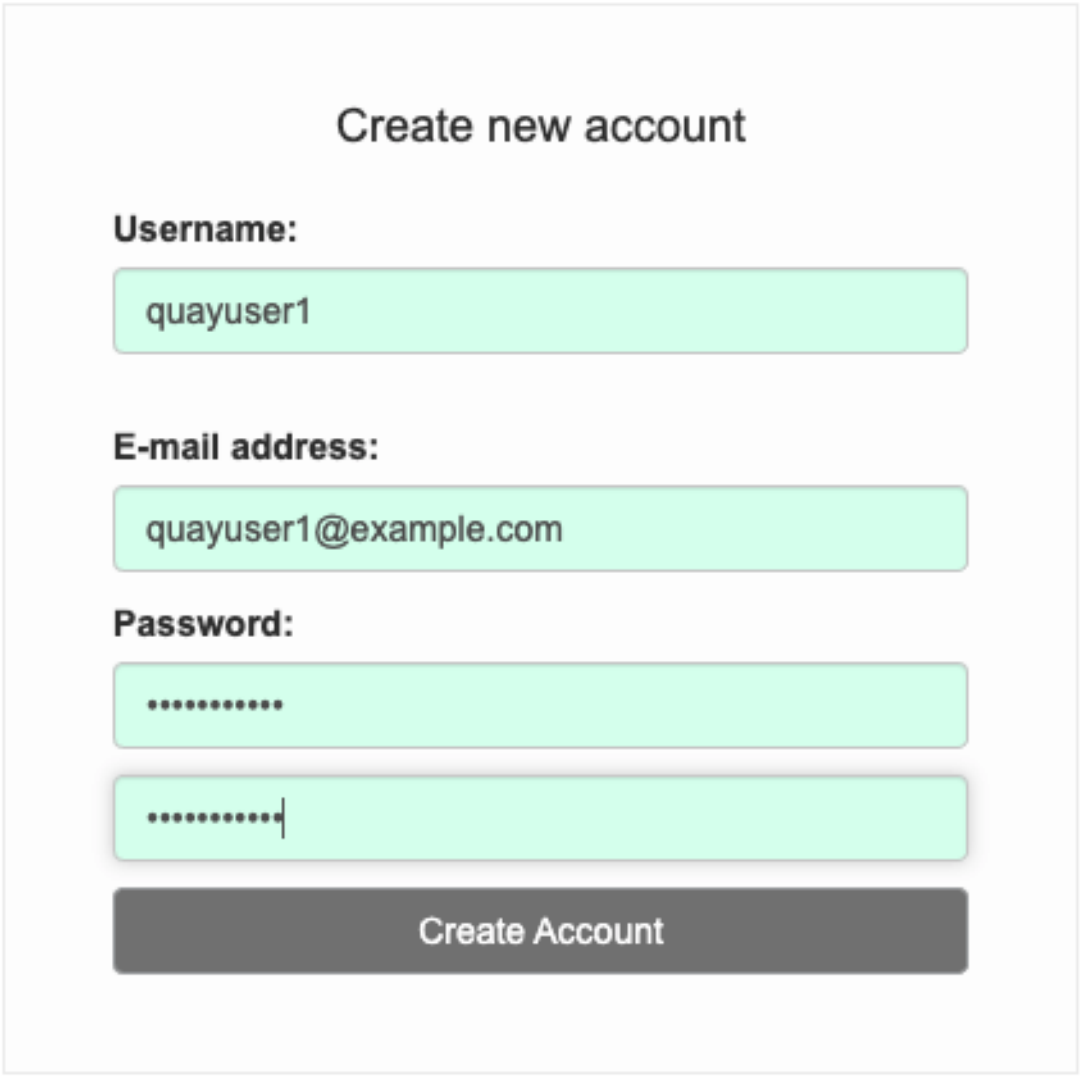
1. OpenShift コンソールで、適切な namespace / プロジェクトを使用して Operators → Installed Operators に移動します。
2. 新規インストールされた QuayRegistry をクリックし、詳細を表示します。



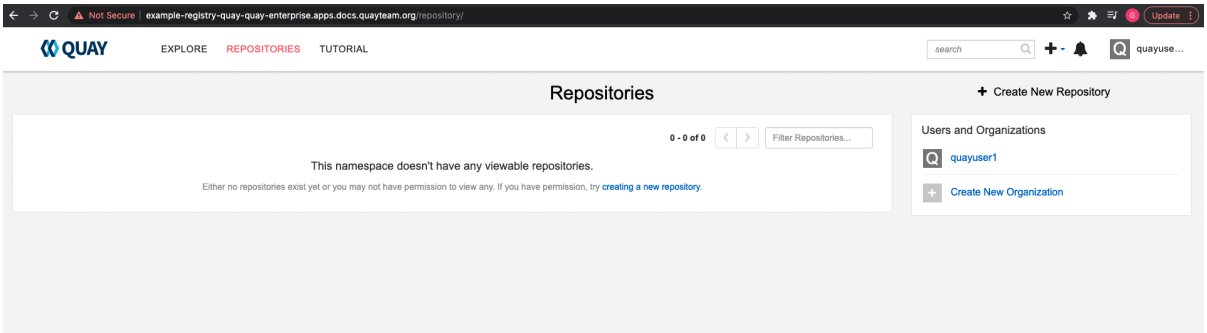
3. **Registry Endpoint** に値を設定したら、ブラウザーでこの URL に移動します。
4. Quay レジストリー UI で Create Account を選択し、ユーザーを作成します。



5. ユーザー名、パスワード、電子メールの詳細を入力して、**Create Account** をクリックします。



6. Quay レジストリーに自動的にログインします。



第5章 OPENSHIFT での QUAY の設定

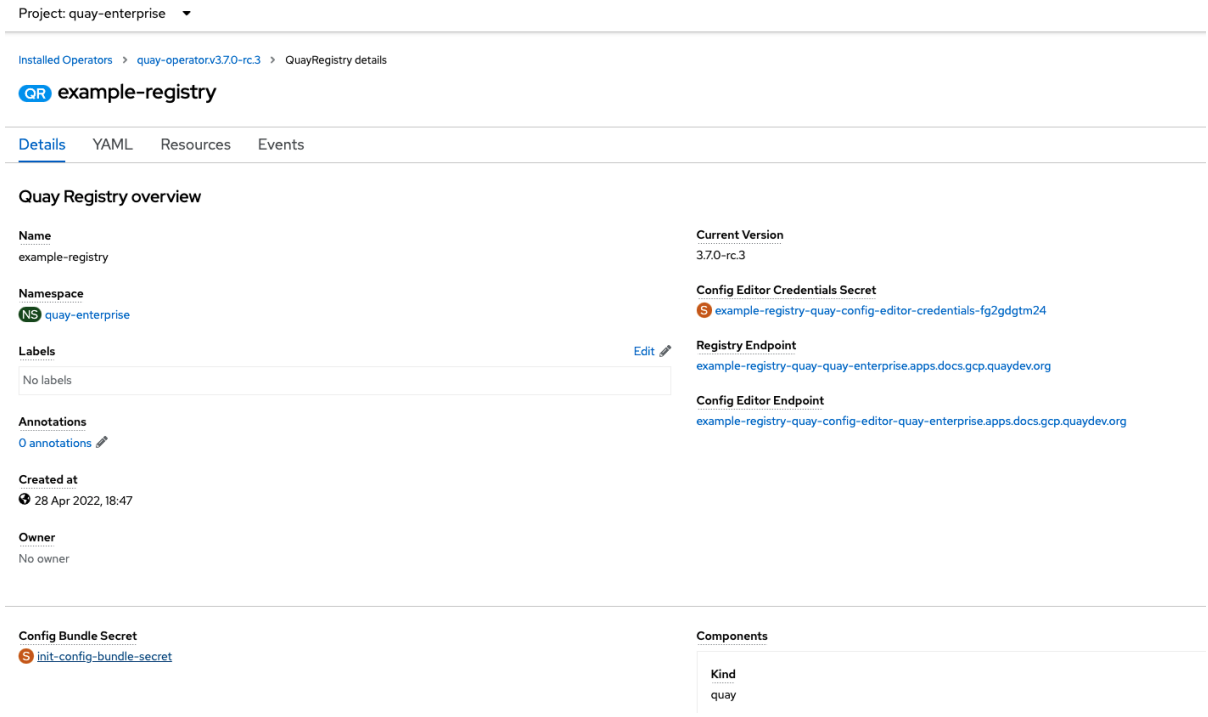
デプロイされたら、Quay 設定バンドルシークレット **spec.configBundleSecret** を編集して Quay アプリケーションを設定し、QuayRegistry リソースの **spec.components** オブジェクトのコンポーネントの管理ステータスを変更することもできます。

または、設定エディター UI を使用して、セクション *6章 設定ツールを使用した OpenShift における Quay の再設定* で説明されているように Quay アプリケーションを設定できます。

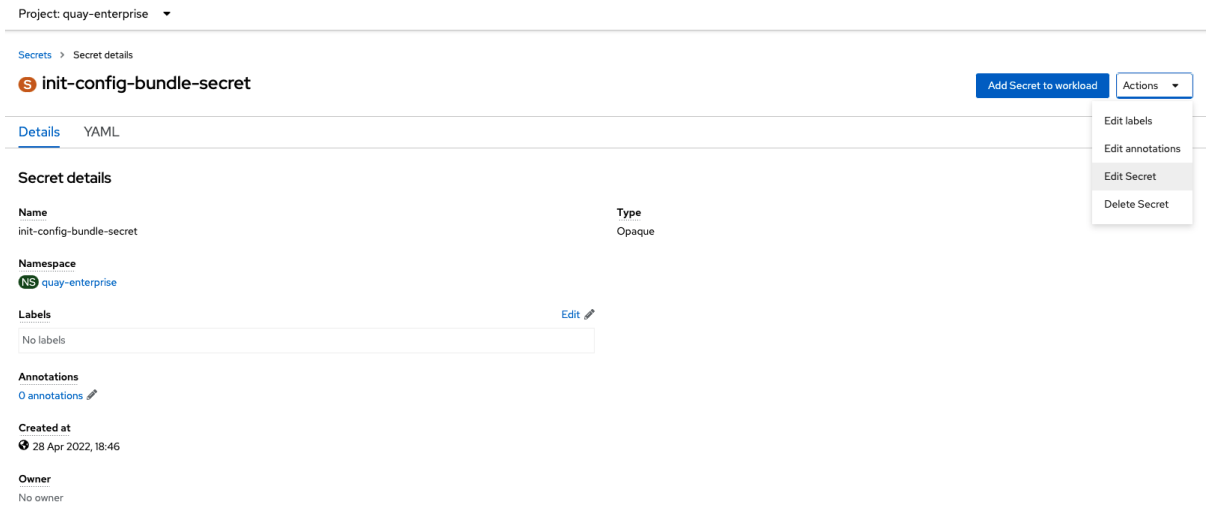
5.1. OPENSHIFT コンソールでの設定バンドルシークレットの編集

手順

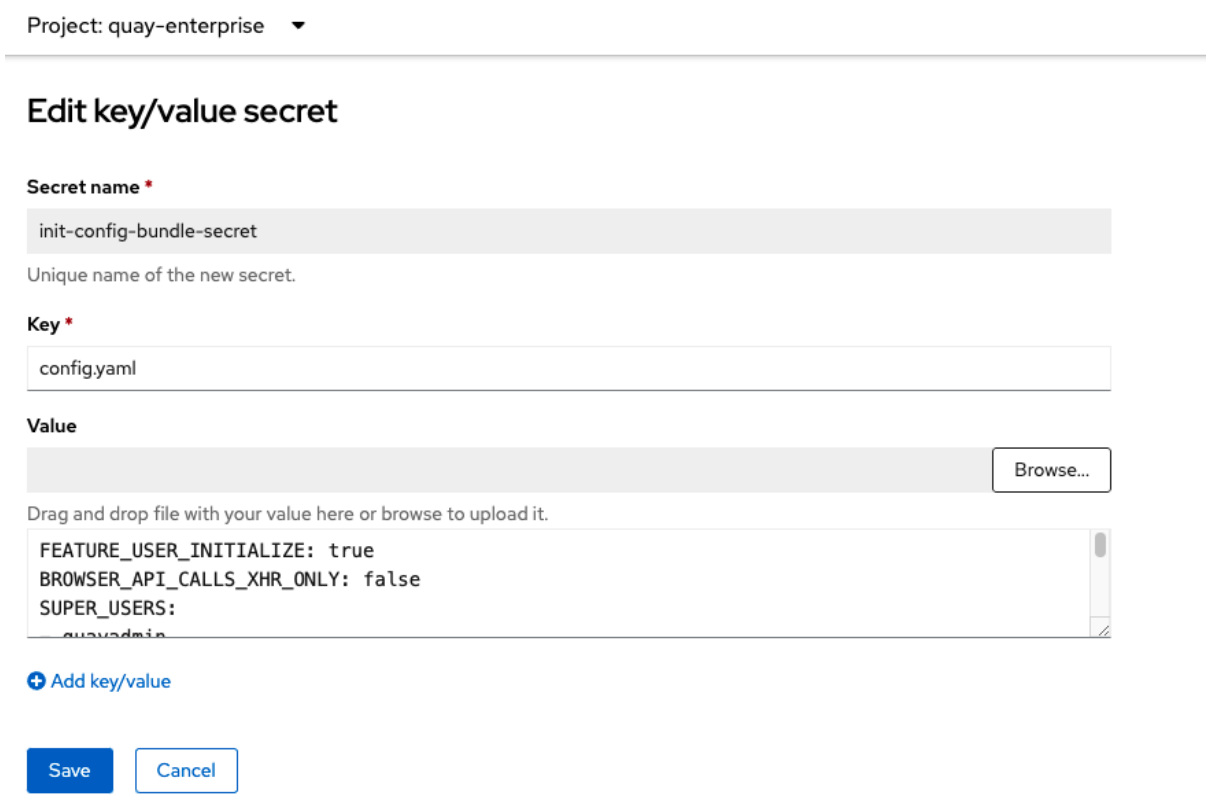
1. Quay Registry の概要画面で、Config Bundle Secret のリンクをクリックします。



2. シークレットを編集するには、**Actions → Edit Secret** をクリックします



3. 設定を変更して変更を保存します



4. デプロイメントを監視して正常に完了し、設定の変更が反映されていることを確認します。

5.2. QUAYREGISTRY エンドポイントおよびシークレットの決定

oc describe quayregistry または **oc get quayregistry -o yaml** を使用して QuayRegistry リソースを検査し、現在のエンドポイントおよびシークレットを判別します。

```
$ oc get quayregistry example-registry -n quay-enterprise -o yaml

apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: example-registry
  namespace: quay-enterprise
  ...
spec:
  components:
  - kind: quay
    managed: true
  ...
  - kind: clairpostgres
    managed: true
  configBundleSecret: init-config-bundle-secret
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-fg2gdgtm24
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.gcp.quaydev.org
  currentVersion: 3.7.0
  lastUpdated: 2022-05-11 13:28:38.199476938 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org
```

関連するフィールドは以下のとおりです。

- **registryEndpoint**: レジストリーの URL、レジストリー UI へのブラウザーアクセス、およびレジストリー API エンドポイント
- **configBundleSecret**: **config.yaml** ファイルと SSL 証明書を含む設定バンドルシークレット
- **configEditorEndpoint**: 設定エディターツールの URL、設定ツールへのブラウザーアクセス、および設定 API
- **configEditorCredentialsSecret**: ユーザー名 (通常は **quayconfig**) および設定エディターツールのパスワードが含まれるシークレット

設定エディターツールのユーザー名とパスワードを確認するには、以下を実行します。

1. シークレットを取得します。

```
$ oc get secret -n quay-enterprise example-registry-quay-config-editor-credentials-fg2gdgtm24 -o yaml

apiVersion: v1
data:
  password: SkZwQkVKTUN0a1BUZmp4dA==
  username: cXVheWNvbmZpZw==
kind: Secret
```

2. ユーザー名をデコードします。

```
$ echo 'cXVheWNvbmZpZw==' | base64 --decode

quayconfig
```

3. パスワードをデコードします。

```
$ echo 'SkZwQkVKTUN0a1BUZmp4dA==' | base64 --decode

JFpBEJMCtkPTfjxt
```

5.3. 既存設定のダウンロード

現在の設定にアクセスする方法は複数あります。

1. 設定エディターのエンドポイントを使用して、設定エディターのユーザー名とパスワードを指定します。

```
$ curl -k -u quayconfig:JFpBEJMCtkPTfjxt https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org/api/v1/config
```

```
{
   "config.yaml": {
      "ALLOW_PULLS_WITHOUT_STRICT_LOGGING": false,
      "AUTHENTICATION_TYPE": "Database",
      ...
      "USER_RECOVERY_TOKEN_LIFETIME": "30m"
```

```
    },
    "certs": {
        "extra_ca_certs/service-ca.crt":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURVVENDQWptZ0F3SUJBZ0lJRE9k
WFhuUXFjMUF3RFFZSktvWklodmNOQVFFTEJRQXdOakUwTURJR0ExVUUKQXd3cmIzQ
mxibk5vYVdaMExYTmxjblpwWTJVdGMyVnlkbWx1WnkxemFXZHVaWEpBTVRZek1UYzNPRE
V3TXpBZQpGdzB5TVRBNU1UWXdOelF4TkRKYUZ..."
    }
}
```

2. 設定バンドルシークレットの使用

    a. シークレットデータを取得します。

    ```
    $ oc get secret -n quay-enterprise init-config-bundle-secret -o jsonpath='{.data}'
    ```

    出力例

    ```
    {
        "config.yaml": "RkVBVFVSRV9VU0 ... MDAwMAo="
    }
    ```

    b. データをデコードします。

    ```
    $ echo 'RkVBVFVSRV9VU0 ... MDAwMAo=' | base64 --decode
    ```

    ```
    FEATURE_USER_INITIALIZE: true
    BROWSER_API_CALLS_XHR_ONLY: false
    SUPER_USERS:
    - quayadmin
    FEATURE_USER_CREATION: false
    FEATURE_QUOTA_MANAGEMENT: true
    FEATURE_PROXY_CACHE: true
    FEATURE_BUILD_SUPPORT: true
    DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
    ```

5.4. 設定バンドルを使用したカスタム SSL 証明書の設定

カスタム SSL 証明書は、最初のデプロイ前、または Red Hat Quay が OpenShift 上にデプロイされた後に、config bundle secret を作成または更新することで設定できます。既存のデプロイメントに証明書を追加する場合は、設定を変更しなくても、新規の設定バンドルシークレットに既存の **config.yaml** を含める必要があります。

5.4.1. TLS をアンマネージドに設定

Quay Registry yaml で、**kind: tls** を **managed: false** に設定します。

```
- kind: tls
  managed: false
```

イベントでは、適切な設定をするまで、変更がブロックされていることがわかります。

```
- lastTransitionTime: '2022-03-28T12:56:49Z'
  lastUpdateTime: '2022-03-28T12:56:49Z'
  message: >-
    required component `tls` marked as unmanaged, but `configBundleSecret`
    is missing necessary fields
  reason: ConfigInvalid
  status: 'True'
```

5.4.2. 設定バンドルに証明書を追加

手順

1. 埋め込みデータまたはファイルを使用してシークレットを作成します。
   a. 設定の詳細を Secret リソース YAML ファイルに直接組み込みます。以下に例を示します。

   custom-ssl-config-bundle.yaml

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: custom-ssl-config-bundle-secret
     namespace: quay-enterprise
   data:
     config.yaml: |
       FEATURE_USER_INITIALIZE: true
       BROWSER_API_CALLS_XHR_ONLY: false
       SUPER_USERS:
       - quayadmin
       FEATURE_USER_CREATION: false
       FEATURE_QUOTA_MANAGEMENT: true
       FEATURE_PROXY_CACHE: true
       FEATURE_BUILD_SUPPORT: true
       DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
     extra_ca_cert_my-custom-ssl.crt: |
       -----BEGIN CERTIFICATE-----
       MIIDsDCCApigAwIBAgIUCqlzkHjF5i5TXLFy+sepFrZr/UswDQYJKoZIhvcNAQEL

   BQAwbzELMAkGA1UEBhMCSUUxDzANBgNVBAgMBkdBTFdBWTEPMA0GA1UEBwwG
   R0FM
       ....
       -----END CERTIFICATE-----
   ```

   次に、YAML ファイルからシークレットを作成します。

   ```
   $ oc create  -f custom-ssl-config-bundle.yaml
   ```

   b. または、必要な情報が含まれるファイルを作成し、それらのファイルからシークレットを作成できます。

   ```
   $ oc create secret generic custom-ssl-config-bundle-secret \
     --from-file=config.yaml \
     --from-file=extra_ca_cert_my-custom-ssl.crt=my-custom-ssl.crt
   ```

2. 作成した Secret を参照する QuayRegistry YAML ファイル **quayregistry.yaml** を作成または更新します。以下はその例です。

quayregistry.yaml

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

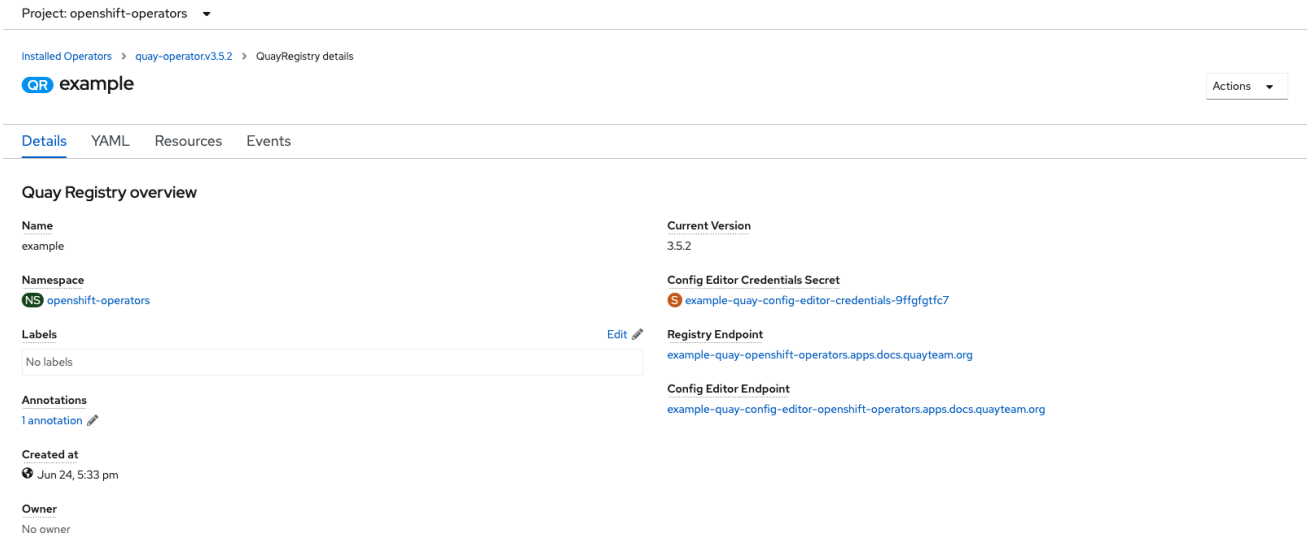
3. YAML ファイルを使用してレジストリーをデプロイまたは更新します。

```
oc apply -f quayregistry.yaml
```

第6章 設定ツールを使用した OPENSHIFT における QUAY の再設定

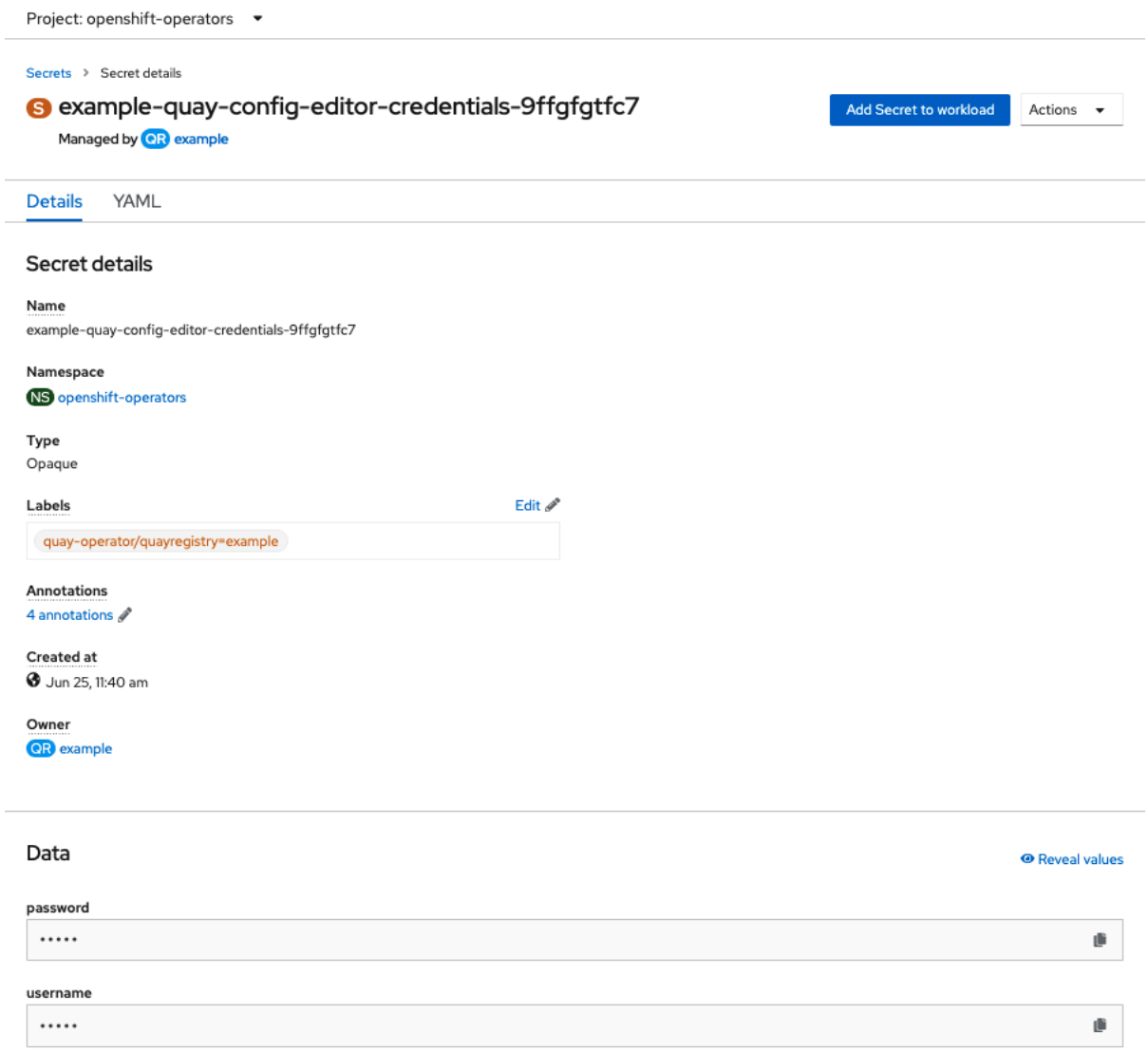
6.1. 設定エディターへのアクセス

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

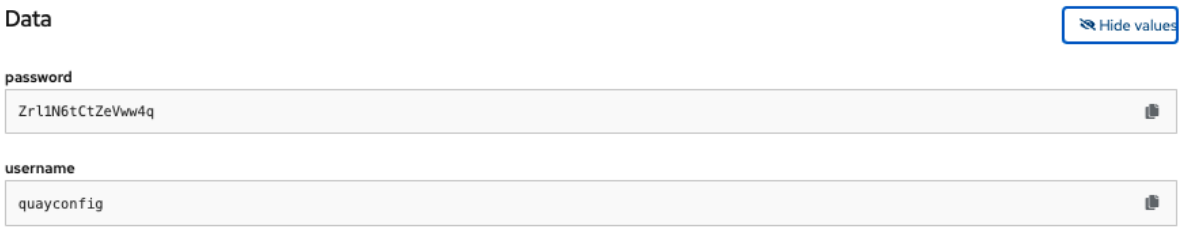


6.1.1. 設定エディターの認証情報の取得

1. 設定エディターシークレットのリンクをクリックします。

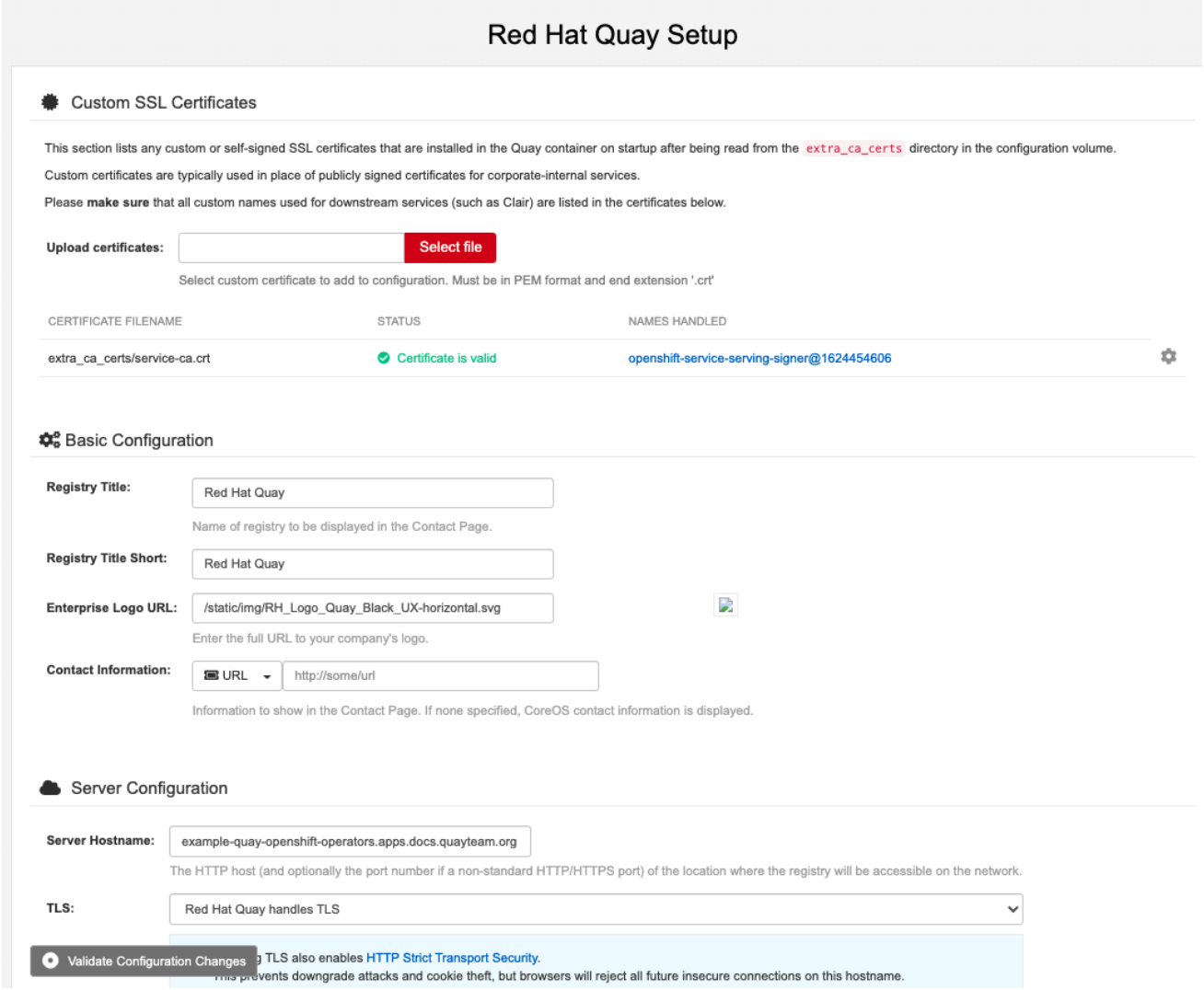


2. Secret details 画面の Data セクションで、**Reveal values** をクリックし、設定エディターへのログインに使用する認証情報を表示します。



6.1.2. 設定エディターへのログイン

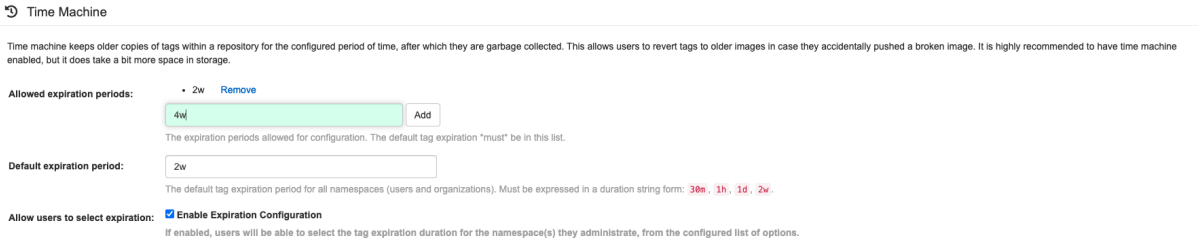
設定エディターエンドポイントを参照し、設定ツールにアクセスする時に使用するユーザー名 (通常は **quayconfig**)、および対応するパスワードを入力します。



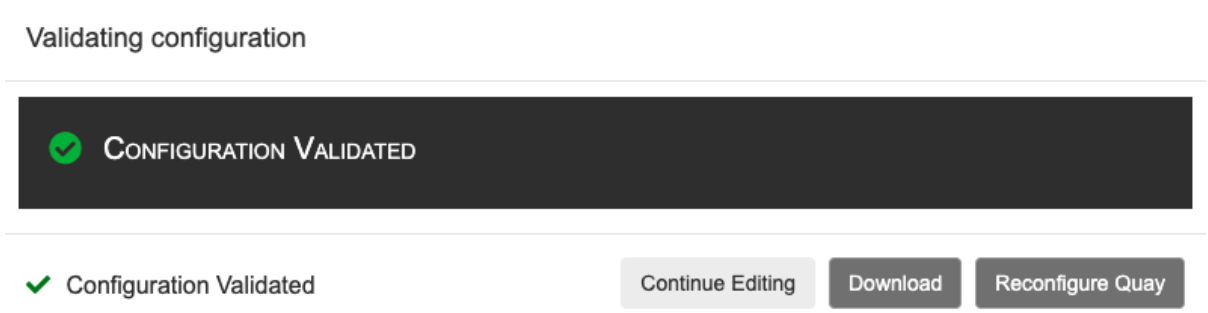
6.1.3. 設定の変更

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

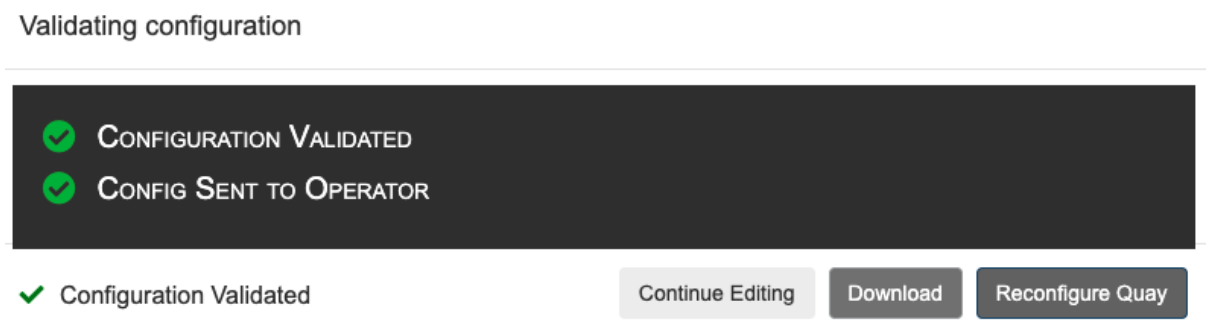
1. 時間マシン機能に関する有効期限 (**4w** など) を追加します。

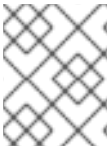


2. **Validate Configuration Changes** を選択して、変更が有効であることを確認します。
3. **Reconfigure Quay** ボタンを押して変更を適用します。



4. 設定ツールは、変更が Quay に送信されていることを通知します。





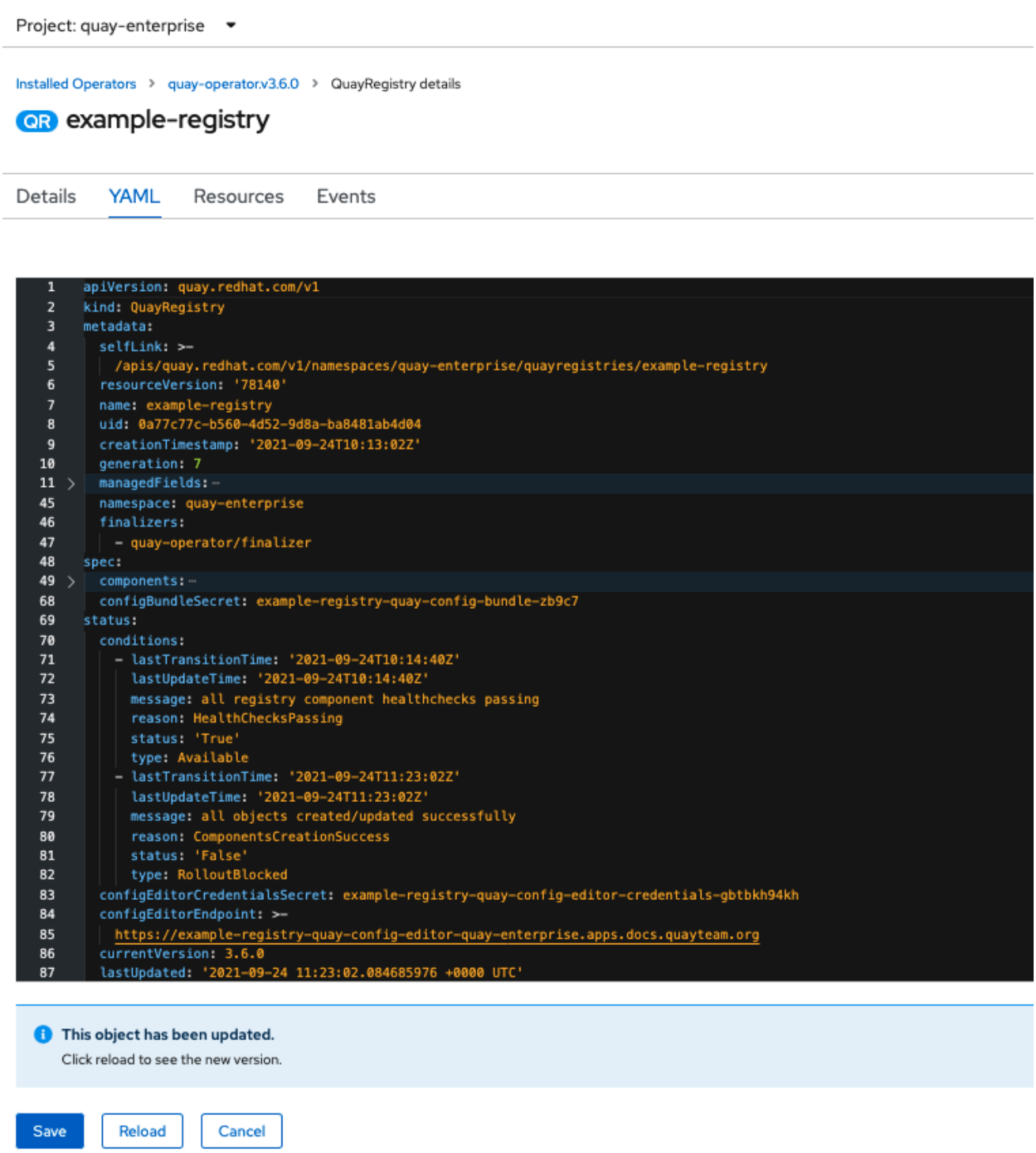
注記

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

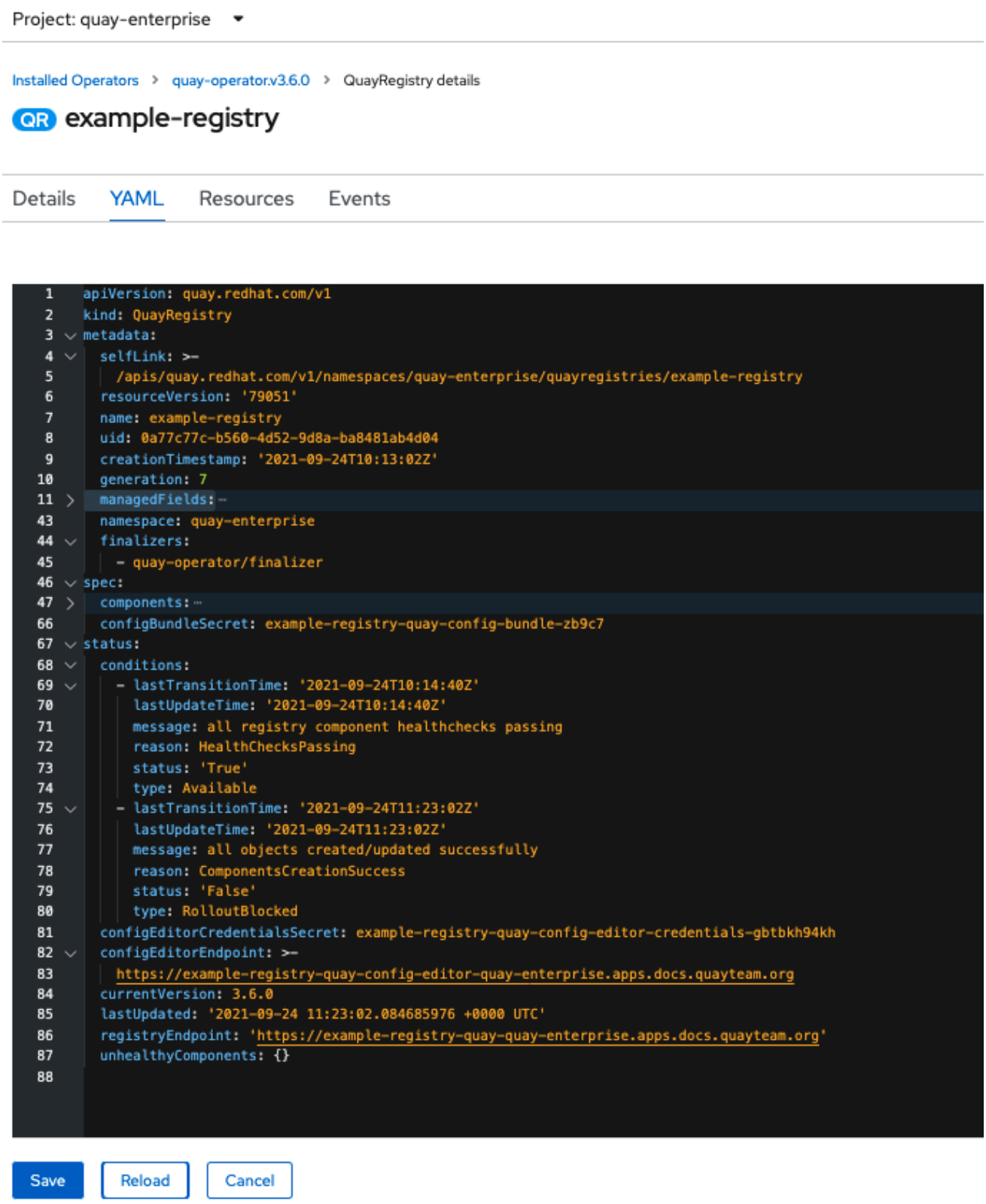
6.2. UI での再設定の監視

6.2.1. QuayRegistry リソース

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

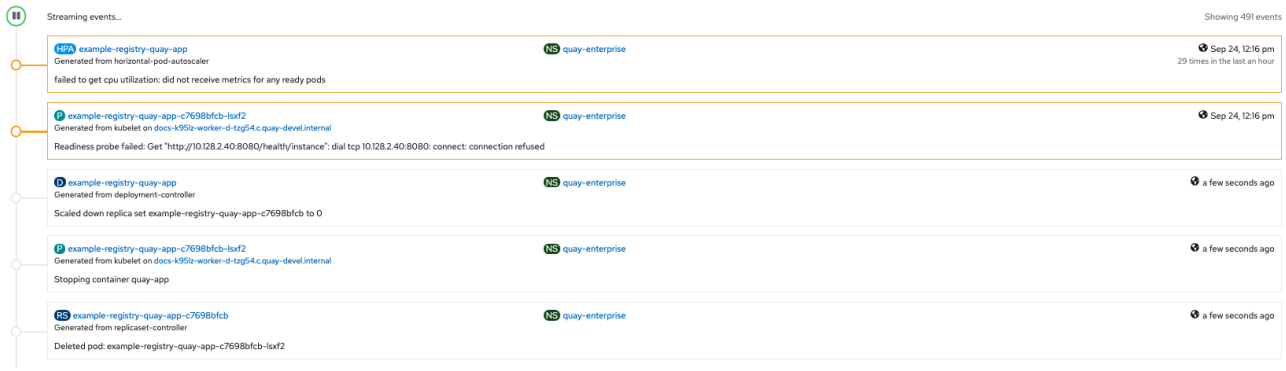


ステータスが変わるたびに、データをリロードして更新されたバージョンを表示するように求められます。最終的に、Operator は変更を調整し、正常でないコンポーネントが報告されることはありません。

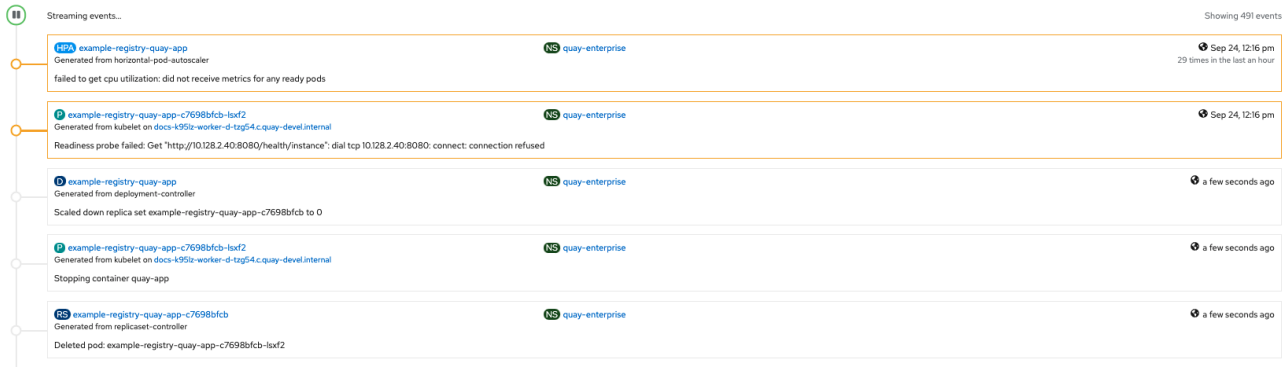


6.2.2. イベント

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



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



6.3. 再設定後に更新された情報へのアクセス

6.3.1. UI で更新された設定ツールの認証情報へのアクセス

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

6.3.2. UI で更新された config.yaml へのアクセス

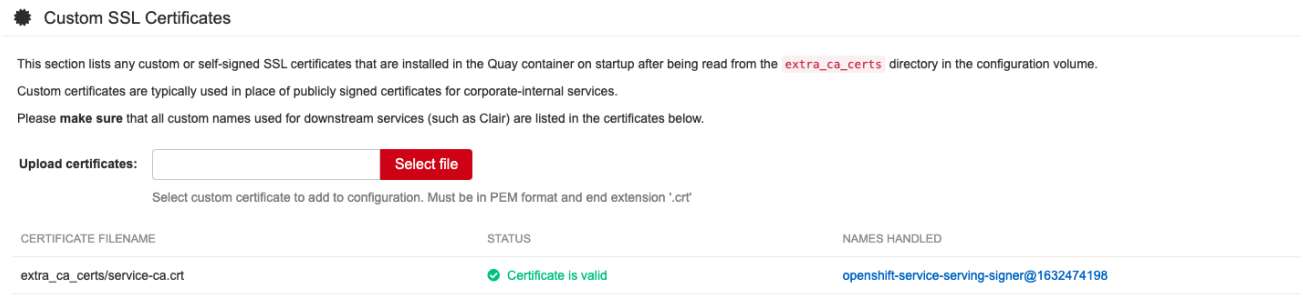
設定バンドルを使用して、更新された **config.yaml** ファイルにアクセスします。

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

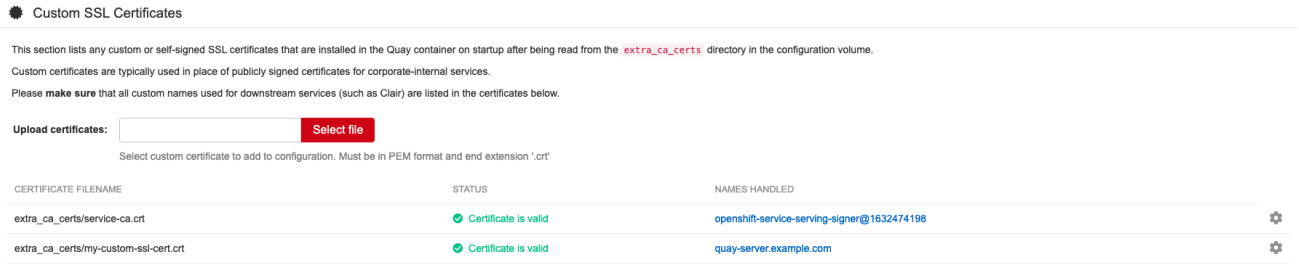
```
...
SERVER_HOSTNAME: example-quay-openshift-operators.apps.docs.quayteam.org
SETUP_COMPLETE: true
SUPER_USERS:
- quayadmin
TAG_EXPIRATION_OPTIONS:
- 2w
- 4w
...
```

6.4. カスタム SSL 証明書 UI

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



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



6.5. レジストリーへの外部アクセス

OpenShift で実行している場合は、**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
```

第7章 QUAY OPERATOR の機能

7.1. コンソールでのモニタリングおよびアラート

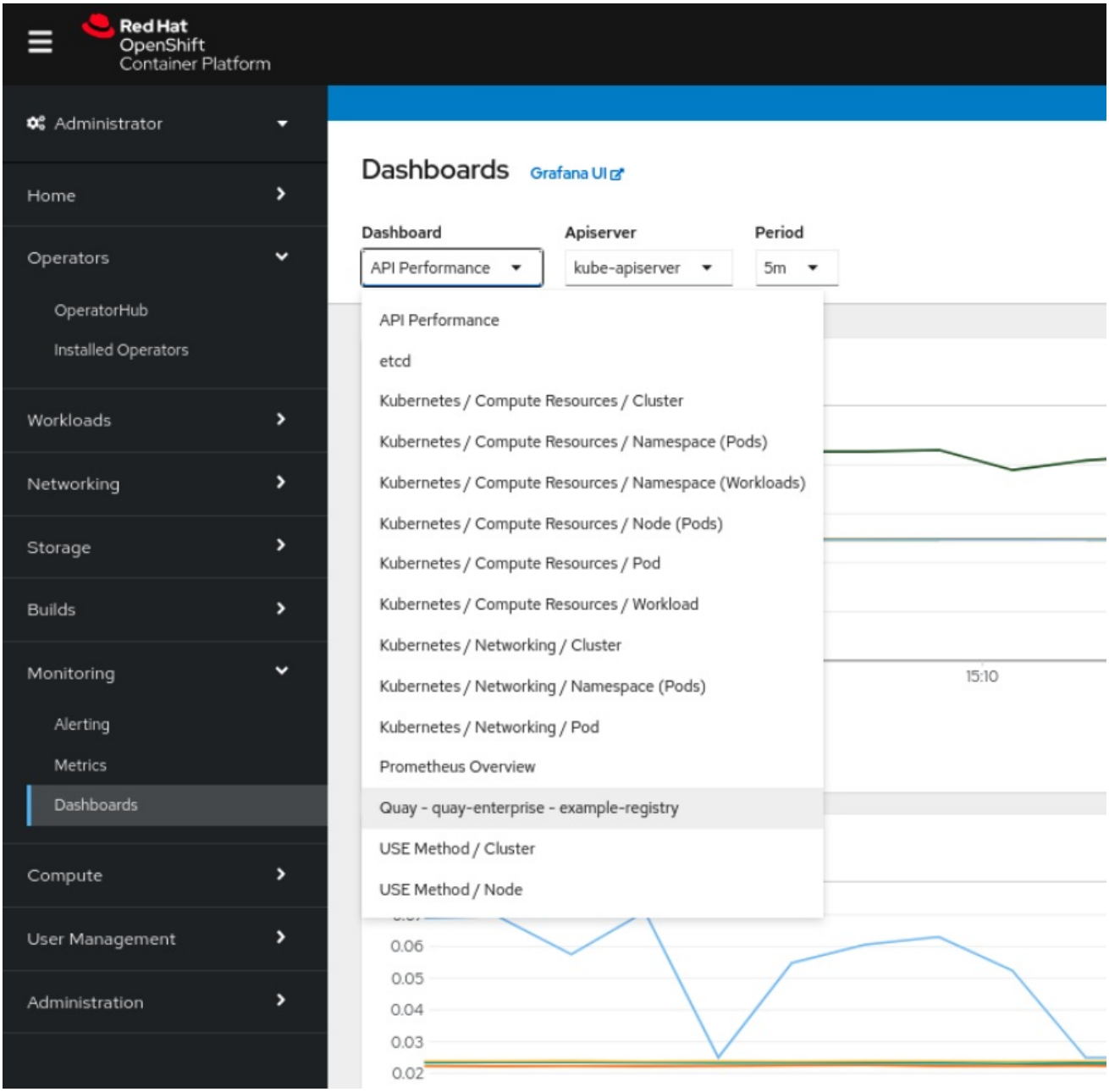
Red Hat Quay は、OpenShift コンソールから Operator を使用してデプロイされた Quay インスタンスのモニタリングのサポートを提供します。新規のモニタリング機能には、Grafana ダッシュボード、個別のメトリクスへのアクセス、Quay Pod を頻繁に再起動するために通知するアラートなどが含まれます。

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。

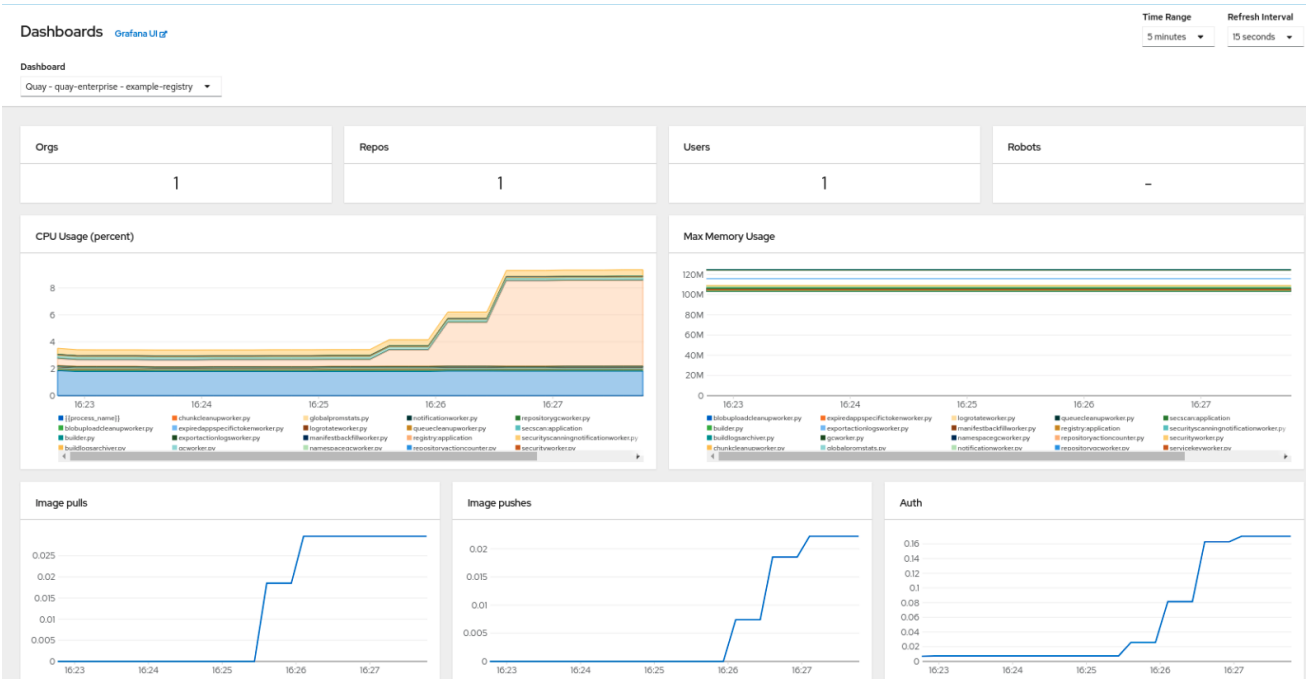
7.1.1. ダッシュボード

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。



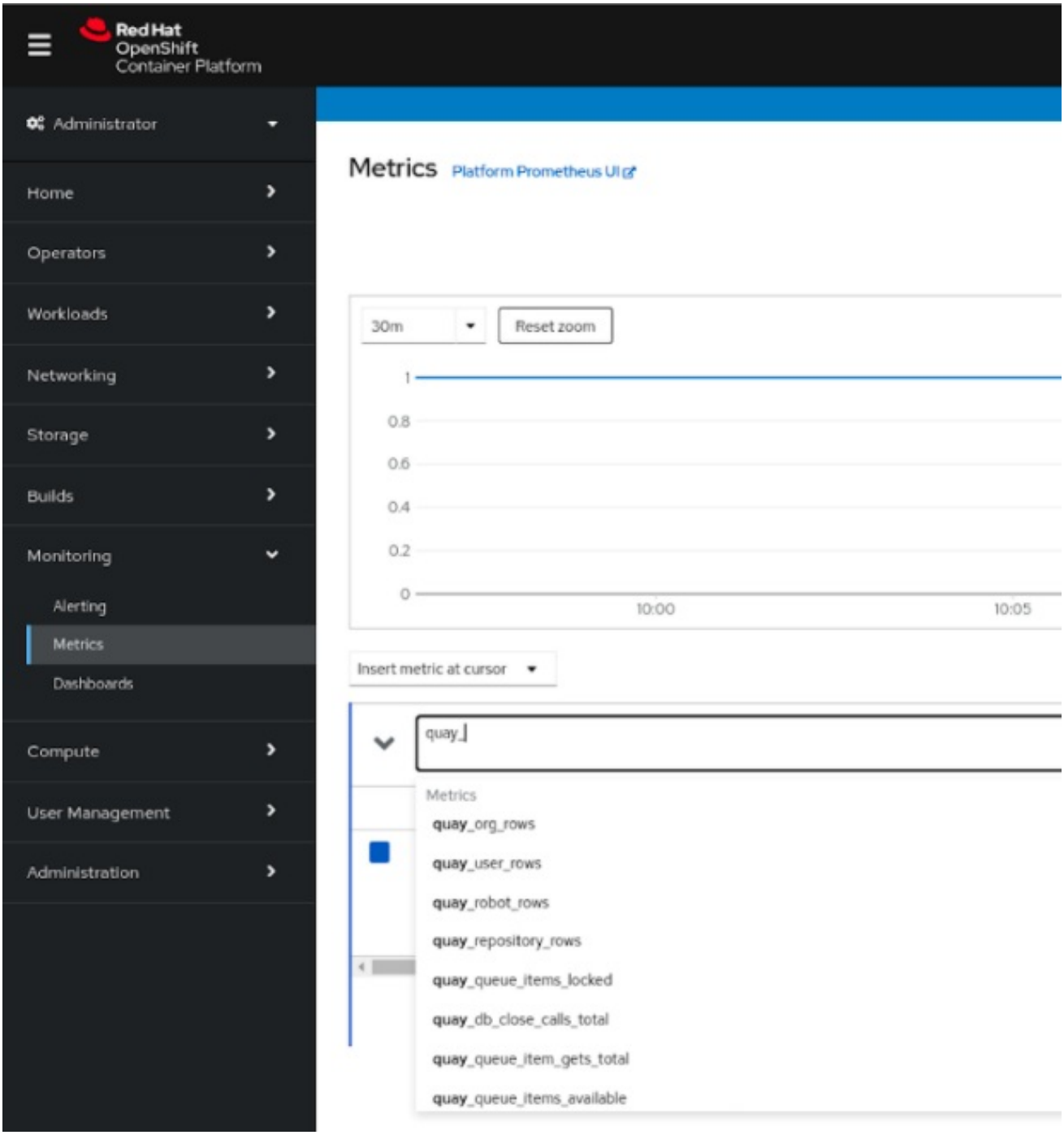
ダッシュボードには、以下を含むさまざまな統計が表示されます。

- 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

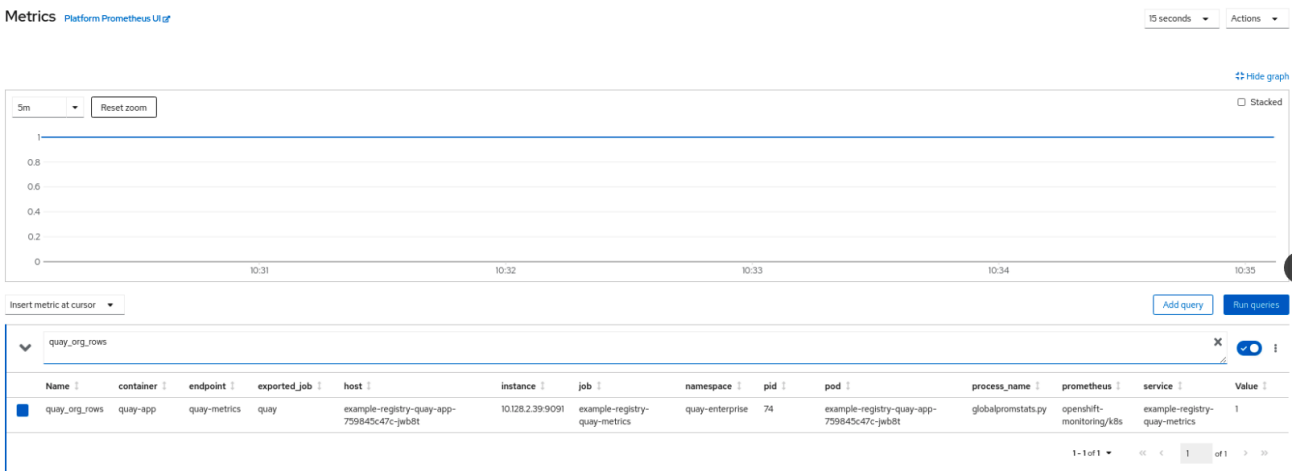


7.1.2. Metrics

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。



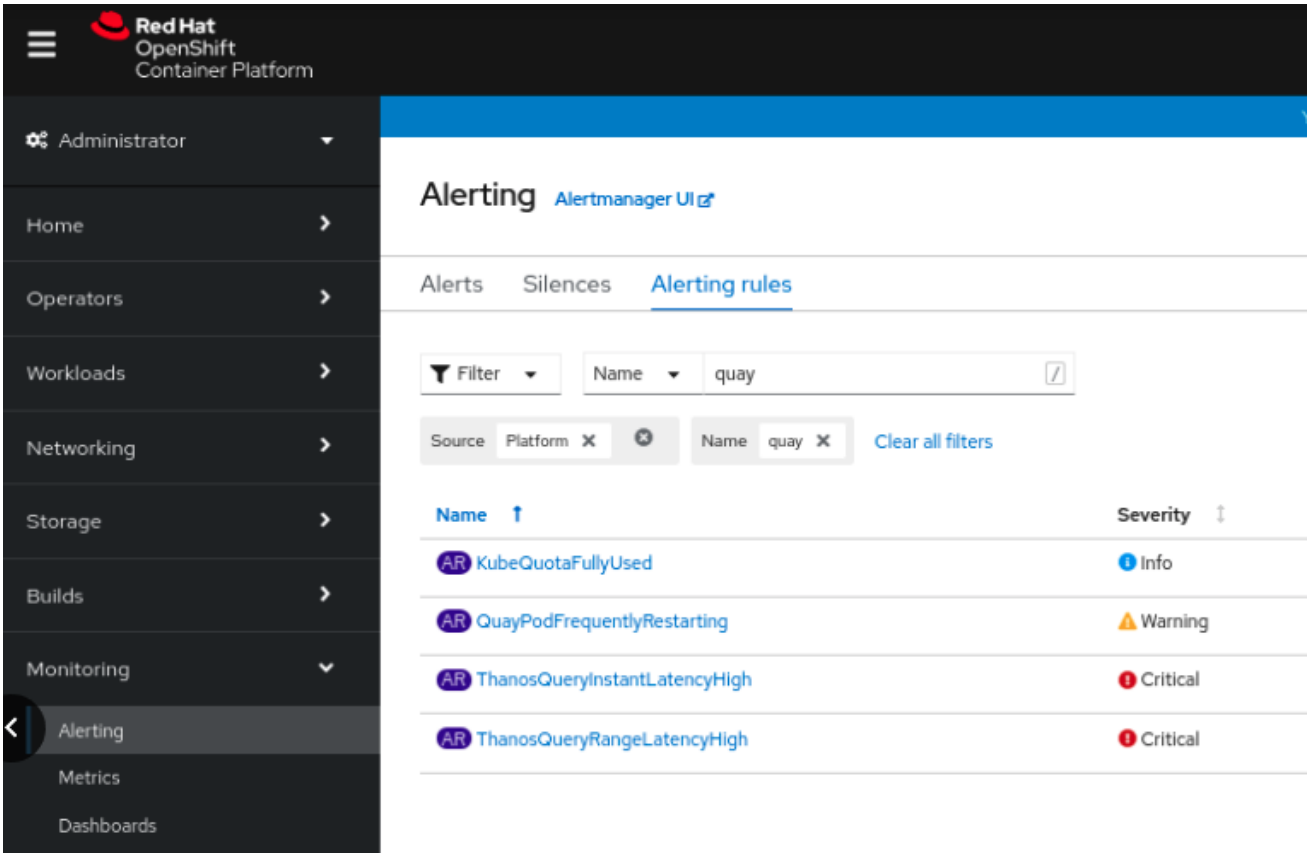
quay_org_rows など、サンプルメトリクスを選択します。



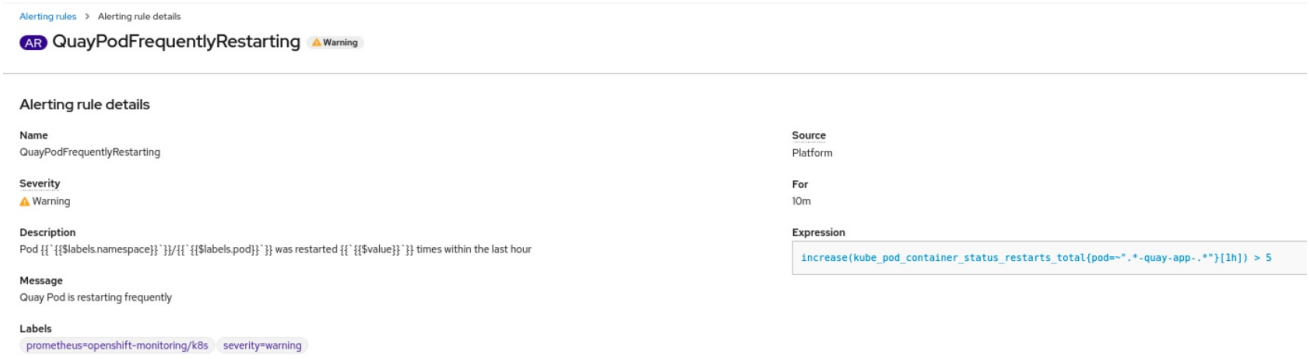
このメトリクスには、レジストリー内の組織の数が表示されます。これはダッシュボードに直接表示されます。

7.1.3. 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 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。



7.2. エアギャップされた OPENSHIFT クラスターにおける CLAIR の脆弱性データベースの手動更新

Clair は、異なる脆弱性データベースのフェッチおよび解析に使用されるロジックをカプセル化する **updaters** というパッケージを使用します。Clair は、異なる環境でのアップデーターの実行と、結果のインポートをサポートします。これは、Clair クラスターがインターネットと直接対話できないようにするインストールをサポートします。

エアギャップされた OpenShift クラスターで Clair の脆弱性データベースを手動で更新するには、以下の手順に従います。

- **clairctl** プログラムを取得します。
- Clair 設定を取得します。
- **clairctl** を使用して、インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートします。
- エアギャップされた OpenShift クラスターの Clair 設定を更新して、Clair データベースへのアクセスを許可します。
- インターネットアクセスのあるシステムからアップデーターバンドルを転送し、これをエアギャップされた環境内で利用できるようにします。
- **clairctl** を使用してアップデーターバンドルを、エアギャップされた OpenShift クラスター用の Clair インスタンスにインポートします。

7.2.1. clairctl の取得

OpenShift クラスターの Clair デプロイメントから **clairctl** プログラムを取得するには、以下のように **oc cp** コマンドを使用します。

```
$ oc -n quay-enterprise cp example-registry-clair-app-64dd48f866-6ptgw:/usr/bin/clairctl ./clairctl
$ chmod u+x ./clairctl
```

スタンドアロンの Clair のデプロイメントでは、以下のように **podman cp** コマンドを使用します。

```
$ sudo podman cp clairv4:/usr/bin/clairctl ./clairctl
$ chmod u+x ./clairctl
```

7.2.2. Clair 設定の取得

7.2.2.1. OpenShift 設定の Clair

OpenShift Operator を使用してデプロイされた Clair インスタンスの設定ファイルを取得するには、適切な namespace を使用して設定シークレットを取得およびデコードし、これをファイルに保存します。以下に例を示します。

```
$ kubectl get secret -n quay-enterprise example-registry-clair-config-secret  -o "jsonpath=
{$.data['config\.yaml']}" | base64 -d > clair-config.yaml
```

以下は、Clair 設定ファイルからの抜粋です。

clair-config.yaml

```
http_listen_addr: :8080
introspection_addr: ""
log_level: info
indexer:
    connstring: host=example-registry-clair-postgres port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    scanlock_retry: 10
```

```
    layer_scan_concurrency: 5
    migrations: true
    scanner:
      package: {}
      dist: {}
      repo: {}
    airgap: false
matcher:
    connstring: host=example-registry-clair-postgres port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    max_conn_pool: 100
    indexer_addr: ""
    migrations: true
    period: null
    disable_updaters: false
notifier:
    connstring: host=example-registry-clair-postgres port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    migrations: true
    indexer_addr: ""
    matcher_addr: ""
    poll_interval: 5m
    delivery_interval: 1m
    ...
```

7.2.2.2. スタンドアロン Clair 設定

スタンドアロンの Clair デプロイメントでは、設定ファイルは、**podman run** コマンドの CLAIR_CONF 環境変数に指定されたものです。以下に例を示します。

```
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:v3.7.10
```

7.2.3. アップデータバンドルのエクスポート

インターネットにアクセスできる Clair インスタンスから、適切な設定ファイルと共に **clairctl** を使用してアップデーターバンドルをエクスポートします。

```
$ ./clairctl --config ./config.yaml export-updaters updates.gz
```

7.2.4. エアギャップされた OpenShift クラスターでの Clair データベースへのアクセスの設定

- **kubectl** を使用して Clair データベースサービスを判別します。

```
$ kubectl get svc -n quay-enterprise

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
example-registry-clair-app          ClusterIP      172.30.224.93    <none>
```

```
80/TCP,8089/TCP                 4d21h
example-registry-clair-postgres   ClusterIP   172.30.246.88   <none>        5432/TCP
4d21h
...
```

- 以下のように Clair データベースポートを転送し、これがローカルマシンからアクセスできるようにします。

```
$ kubectl port-forward -n quay-enterprise service/example-registry-clair-postgres 5432:5432
```

- Clair 設定ファイルを更新し、複数の **connstring** フィールドの **host** の値を **localhost** に置き換えます。以下に例を示します。

clair-config.yaml

```
    ...
    connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    ...
```

注記

kubectl port-forward を使用する代わりに **kubefwd** を使用できます。この方法では、Clair 設定ファイルの **connstring** フィールドを、**localhost** を使用するために変更する必要はありません。

7.2.5. アップデーターバンドルのエアギャップされた環境へインポート

アップデーターバンドルをエアギャップされた環境に転送した後に、**clairctl** を使用して、バンドルを OpenShift Operator によってデプロイされた Clair データベースにインポートします。

```
$ ./clairctl --config ./clair-config.yaml import-updaters updates.gz
```

7.3. 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 での実行をサポートしています。

第8章 高度なコンセプト

8.1. 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、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。

8.1.1. 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** の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
```

ただし、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予期せず 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、infra ノードにテイントを適用し、制御したい Pod に許容値を追加します。

```
$ 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
```

8.1.2. 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 を展開している場合は、インストールした Operator と、デプロイのために作成した特定の名前空間を削除します。

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 で作成された後続のリソースは、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。

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

Quay Operator のインストール時に、適切なプロジェクト namespace を明示的に指定します (この場合は **quay-registry**)。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
```

8.1.4. 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たら、デプロイメントが準備されるのを待ちます。Quay Pod を一覧表示する場合は、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
```

8.2. OPERATOR が単一 NAMESPACE にインストールされている場合のモニタリングの有効化

Red Hat Quay Operator が単一 namespace にインストールされている場合、モニタリングコンポーネントはアンマネージドになります。モニタリングを設定するには、OpenShift Container Platform でユーザー定義の namespace についてこれを有効にする必要があります。詳細については、OCP ドキュメントのモニタリングスタックの設定 および ユーザー定義プロジェクトのモニタリングの有効化 を参照してください。

以下の手順では、OCP ドキュメント に基づいて Quay のモニタリングを設定する方法を説明します。

8.2.1. クラスターモニタリング設定マップの作成

1. **cluster-monitoring-config** ConfigMap オブジェクトが存在するかどうかを確認します。

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config

Error from server (NotFound): configmaps "cluster-monitoring-config" not found
```

2. ConfigMap オブジェクトが存在しない場合は、以下を行います。

a. 以下の YAML マニフェストを作成します。この例では、このファイルは **cluster-monitoring-config.yaml** という名前です。

```
$ cat cluster-monitoring-config.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
```

b. ConfigMap オブジェクトを作成します。

```
$ oc apply -f cluster-monitoring-config.yaml configmap/cluster-monitoring-config created
```

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config

NAME                        DATA   AGE
cluster-monitoring-config   1      12s
```

8.2.2. ユーザー定義のワークロードモニタリング設定マップの作成

1. **user-workload-monitoring-config** ConfigMap オブジェクトが存在するかどうかを確認します。

```
$ oc -n openshift-user-workload-monitoring get configmap user-workload-monitoring-config

Error from server (NotFound): configmaps "user-workload-monitoring-config" not found
```

2. ConfigMap オブジェクトが存在しない場合は、以下を行います。

a. 以下の YAML マニフェストを作成します。この例では、このファイルは **user-workload-monitoring-config.yaml** という名前です。

```
$ cat user-workload-monitoring-config.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

b. ConfigMap オブジェクトを作成します。

```
$ oc apply -f user-workload-monitoring-config.yaml

configmap/user-workload-monitoring-config created
```

8.2.3. ユーザー定義プロジェクトのモニタリングの有効化

1. ユーザー定義プロジェクトのモニタリングが実行されているかどうかを確認します。

```
$ oc get pods -n openshift-user-workload-monitoring

No resources found in openshift-user-workload-monitoring namespace.
```

2. **cluster-monitoring-config** ConfigMap を編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

3. **enableUserWorkload: true** を設定して、クラスターでユーザー定義プロジェクトのモニタリングを有効にします。

```
apiVersion: v1
data:
  config.yaml: |
    enableUserWorkload: true
kind: ConfigMap
metadata:
  annotations:
```

4. ファイルを保存して変更を適用し、適切な Pod が実行されていることを確認します。

```
$ oc get pods -n openshift-user-workload-monitoring

NAME                                  READY   STATUS    RESTARTS   AGE
prometheus-operator-6f96b4b8f8-gq6rl  2/2     Running   0          15s
prometheus-user-workload-0            5/5     Running   1          12s
prometheus-user-workload-1            5/5     Running   1          12s
thanos-ruler-user-workload-0          3/3     Running   0          8s
thanos-ruler-user-workload-1          3/3     Running   0          8s
```

8.2.4. Quay メトリクスを公開するための Service オブジェクトの作成

1. Service オブジェクトの YAML ファイルを作成します。

```
$ cat quay-service.yaml

apiVersion: v1
kind: Service
metadata:
  annotations:
  labels:
    quay-component: monitoring
    quay-operator/quayregistry: example-registry
  name: example-registry-quay-metrics
  namespace: quay-enterprise
spec:
  ports:
  - name: quay-metrics
```

```
    port: 9091
    protocol: TCP
    targetPort: 9091
  selector:
    quay-component: quay-app
    quay-operator/quayregistry: example-registry
  type: ClusterIP
```

2. Service オブジェクトを作成します。

```
$  oc apply -f quay-service.yaml

service/example-registry-quay-metrics created
```

8.2.5. ServiceMonitor オブジェクトを作成します。

ServiceMonitor リソースを作成して、メトリクスをスクレープするように OpenShift Monitoring を設定します。

1. ServiceMonitor リソースの YAML ファイルを作成します。

```
$ cat quay-service-monitor.yaml

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    quay-operator/quayregistry: example-registry
  name: example-registry-quay-metrics-monitor
  namespace: quay-enterprise
spec:
  endpoints:
  - port: quay-metrics
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      quay-component: monitoring
```

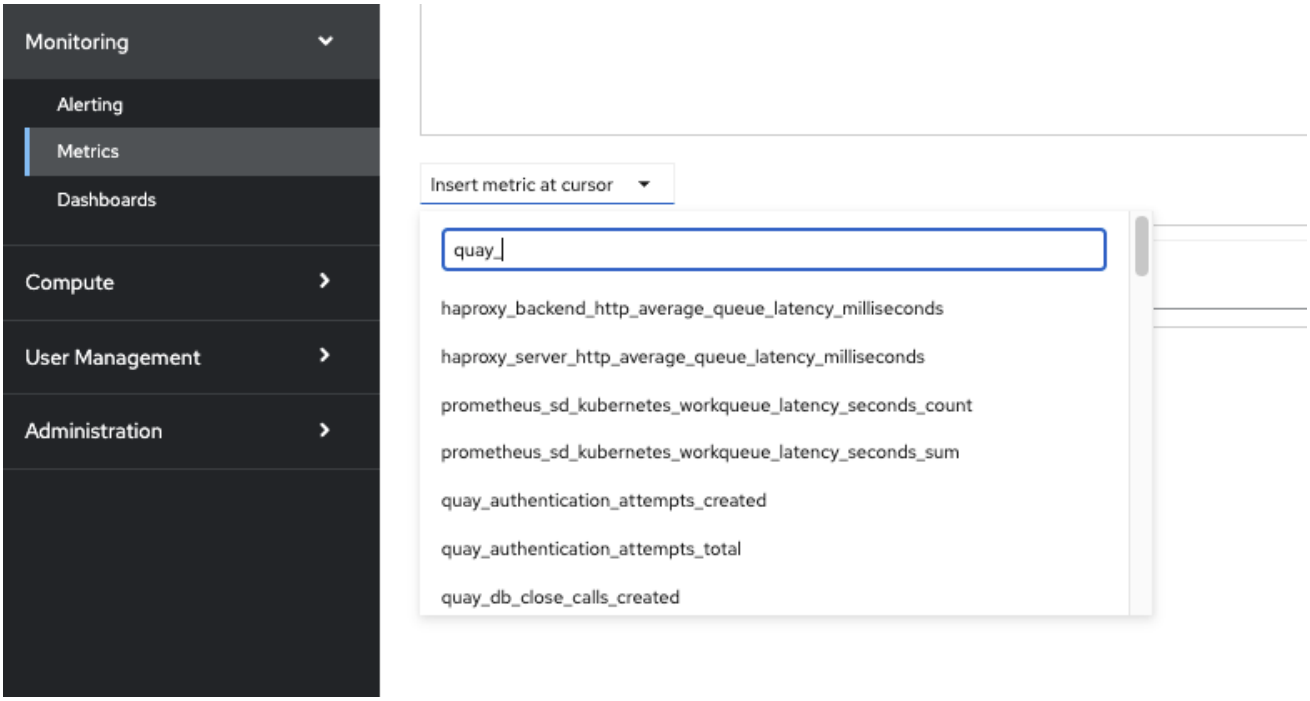
2. ServiceMonitor を作成します。

```
$  oc apply -f quay-service-monitor.yaml

servicemonitor.monitoring.coreos.com/example-registry-quay-metrics-monitor created
```

8.2.6. OpenShift でのメトリクスの表示

OpenShift コンソールでメトリクスには Monitoring → Metrics からアクセスできます。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。



たとえば、ユーザーをレジストリーに追加した場合は、**quay-users_rows** メトリクスを選択します。

8.3. 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、新規ストレージプールに PVC を追加することができます。

8.3.1. 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. 永続ボリューム要求 (PVC) の新しいサイズを入力し、**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必要があります。

注記

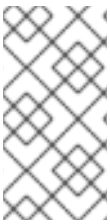
CSI ボリュームの拡張は、テクノロジープレビュー機能としてのみ利用できます。詳細は https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.6/html/storage/expanding-persistent-volumes を参照してください。

8.3.2. 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. プールを管理する Pod 数を選択し、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. **Deploy** をクリックします。

数分後に、追加の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。

8.4. 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。これは開発または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。

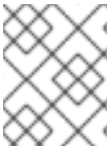
特定の状況では、Operator で使用されるデフォルトイメージを上書きすることが役に立つ場合があります。これは、Quay Operator の **ClusterServiceVersion** に 1 つ以上の環境変数を設定して実行できます。

8.4.1. 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:) で参照される **必要** があります。

8.4.2. 実行中の Operator へのオーバーライドの適用

Operator Lifecycle Manager (OLM) を使用して Quay Operator をクラスターにインストールした場合は、**ClusterServiceVersion** オブジェクトを変更することで、マネージドコンポーネントのコンテナーイメージを簡単に上書きすることができます。これは、クラスター内で実行中の Operator を示す OLM の表現です。Kubernetes UI または **kubectl/oc** 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。

8.5. 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
```

8.5.1. 高度な Clair 設定

8.5.1.1. 管理されていない Clair 設定

Red Hat Quay 3.7 を使用すると、ユーザーは Red Hat Quay OpenShift Container Platform Operator でアンマネージド Clair 設定を実行できます。この機能により、ユーザーはアンマネージド Clair データベースを作成したり、アンマネージドデータベースなしでカスタム Clair 設定を実行したりできます。

8.5.1.1.1. Clair データベースの管理を解除

アンマネージド Clair データベースにより、Red Hat Quay は geo-replicated environment で作業できます。この環境では、Operator の複数のインスタンスが同じデータベースと通信する必要があります。管理されていない Clair データベースは、ユーザーがクラスターの外部に存在する高可用性 (HA) Clair データベースを必要とする場合にも使用できます。

手順

- Quay Operator で、QuayRegistry カスタムリソースの **clairpostgres** コンポーネントを unmanaged に設定します。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay370
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clairpostgres
      managed: false
```

8.5.1.1.2. カスタム Clair データベースの設定

Red Hat Quay Operator for OpenShift Container Platform を使用すると、ユーザーは **configBundleSecret** パラメーターを編集して独自の Clair 設定を提供できます。

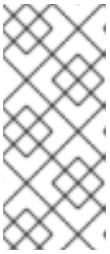
手順

1. **clair-config.yaml** を含む Quay 設定バンドルシークレットを作成します。

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file extra_ca_cert_rds-ca-2019-root.pem=./rds-ca-2019-root.pem --from-file clair-config.yaml=./clair-config.yaml --from-file ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
```

clair-config.yaml 設定の例:

```
indexer:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    layer_scan_concurrency: 6
    migrations: true
    scanlock_retry: 11
log_level: debug
matcher:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    migrations: true
metrics:
    name: prometheus
notifier:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    migrations: true
```



注記

- データベース証明書は、**clair-config.yaml** の Clair アプリケーション Pod の **/run/certs/rds-ca-2019-root.pem** の下にマウントされます。**clair-config.yaml** を設定するときに指定する必要があります。
- **clair-config.yaml** の例は、OpenShift 設定の Clair にあります。

2. **clair-config.yaml** を **configBundleSecret** という名前のバンドルシークレットに追加します。

```
apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>
  clair-config.yaml: <base64 encoded Clair config>
  extra_ca_cert_<name>: <base64 encoded ca cert>
  clair-ssl.crt: >-
  clair-ssl.key: >-
```

注記

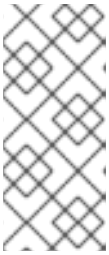
更新されると、提供された **clair-config.yaml** が Clair Pod にマウントされます。提供されていないフィールドには、Clair 設定モジュールを使用してデフォルトが自動的に入力されます。

適切に設定した後、Clair アプリケーション Pod は **Ready** 状態に戻るはずです。

8.5.1.2. managed データベースを使用したカスタム Clair 設定の実行

場合によっては、ユーザーは **managed** データベースを使用してカスタム Clair 設定を実行したい場合があります。これは、以下のシナリオで役に立ちます。

- ユーザーがアップデーターを無効にしたい場合
- ユーザーがエアギャップ環境で実行している場合



注記

- エアギャップ環境で Quay を実行している場合は、**clair-config.yaml** の **airgap** パラメーターを **true** に設定する必要があります。
- エアギャップ環境で Quay を実行している場合は、すべてのアップデーターを無効にする必要があります。

clairpostgres が **managed** に設定されている場合は、カスタム Clair データベースの設定の手順を使用してデータベースを設定します。

エアギャップ環境での Clair の実行の詳細は、エアギャップ OpenShift クラスターでの Clair データベースへのアクセスの設定 を参照してください。

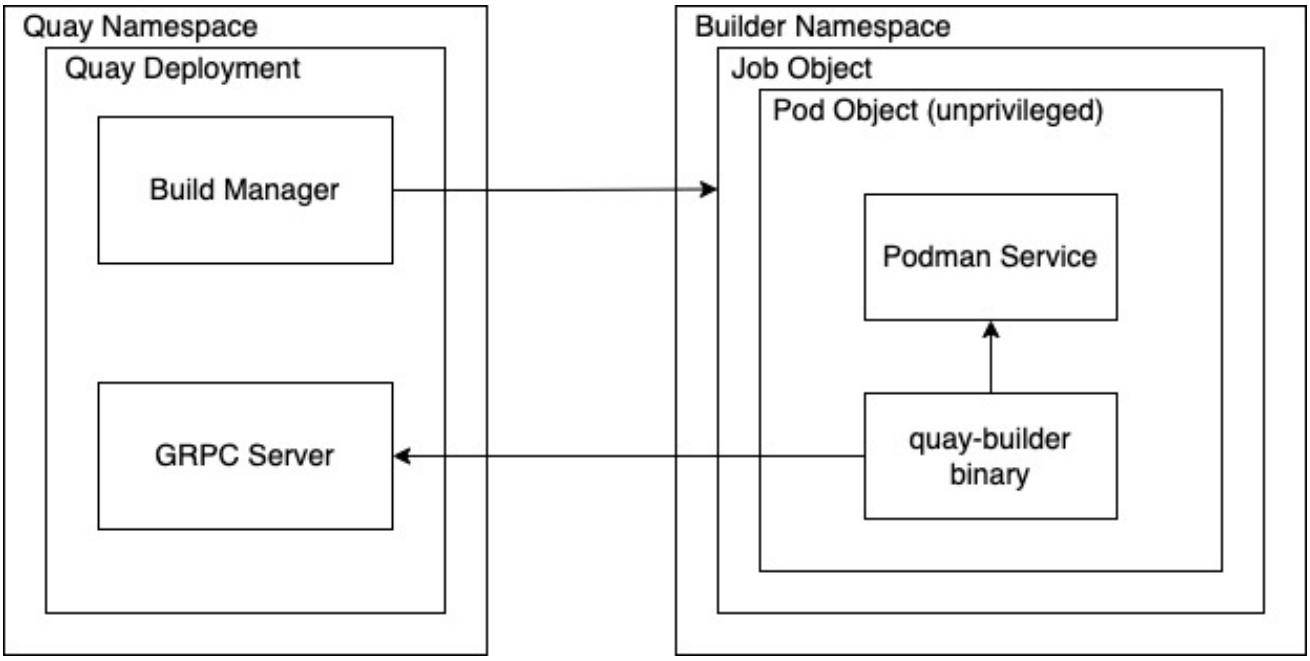
第9章 RED HAT QUAY ビルドの機能強化

Red Hat Quay 3.7 より前は、Quay は Pod によって起動された仮想マシンで **podman** コマンドを実行していました。仮想プラットフォームでビルドを実行するには、ネストされた仮想化を有効にする必要があります。これは、Red Hat Enterprise Linux または OpenShift Container Platform では機能しません。その結果、ビルドはベアメタルクラスターで実行する必要があり、これはリソースの非効率的な使用です。

Red Hat Quay 3.7 では、仮想マシンレイヤーを含まないビルドオプションを追加することで、ビルドの実行に必要なベアメタル制約が削除されました。その結果、ビルドは仮想化されたプラットフォームで実行できます。以前のビルド設定を実行するための下位互換性も利用できます。

9.1. RED HAT QUAY の拡張ビルドアーキテクチャー

前のイメージは、拡張ビルド機能の想定される設計フローとアーキテクチャーを示しています。



この機能拡張により、ビルドマネージャーは最初に **Job Object** を作成します。次に、**Job Object** は **quay-builder-image** を使用して Pod を作成します。 **quay-builder-image** には、**quay-builder binary** サービスおよび **Podman** サービスが含まれます。作成された Pod は **unprivileged** として実行されます。次に、**quay-builder binary** は、ステータスを伝達し、ビルドマネージャーからビルド情報を取得しながら、イメージをビルドします。

9.2. RED HAT QUAY ビルドの制限

特権のないコンテキストで Red Hat Quay でビルドを実行すると、以前のビルド戦略で機能していた一部のコマンドが失敗する可能性があります。ビルド戦略を変更しようとすると、ビルドのパフォーマンスの問題と信頼性の問題が発生する可能性があります。

コンテナーでビルドを直接実行しても、仮想マシンを使用する場合と同じように分離されることはありません。ビルド環境を変更すると、以前は機能していたビルドが失敗する可能性もあります。

9.3. OPENSHIFT を使用した RED HAT QUAY ビルダー環境の作成

9.3.1. OpenShift TLS コンポーネント

tls コンポーネントを使用すると、TLS 設定を制御できます。

注記

TLS コンポーネントが Operator によって管理されていると、Red Hat Quay 3.7 はビルダーをサポートしません。

tls を **unmanaged** に設定する場合は、独自の **ssl.cert** ファイルと **ssl.key** ファイルを提供します。このとき、クラスターでビルダーをサポートする場合は、Quay ルートとビルダールート名の両方を証明書の SAN リストに追加するか、ワイルドカードを使用する必要があります。ビルダールートを追加するには、次の形式を使用します。

```
[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443
```

9.3.2. OpenShift Container Platform ビルダー向けの Red Hat Quay の使用

次の手順では、Red Hat Quay にビルダー機能を実装する方法について説明します。

前提条件

- ビルダーには SSL 証明書が必要です。詳細については、Red Hat Quay コンテナーへの TLS 証明書の追加 を参照してください。
- AWS S3 ストレージを使用している場合は、ビルダーを実行する前に、AWS コンソールでストレージバケットを変更する必要があります。必要なパラメーターについては、次のセクションの AWS S3 ストレージバケットの変更を参照してください。

手順

- この手順は、クラスターが既にプロビジョニングされており、Quay Operator が実行されていることを前提としています。
- この手順は、OpenShift Container Platform で仮想 namespace を設定するためのものです。

9.3.2.1. 仮想ビルダー向けの OpenShift Container Platform の準備

1. クラスター管理者アカウントを使用して、Red Hat Quay クラスターにログインします。
2. 仮想ビルダーが実行される新しいプロジェクトを作成します (例: **virtual-builders**)。

```
$ oc new-project virtual-builders
```

3. ビルドの実行に使用する **ServiceAccount** をこの **Project** に作成します。

```
$ oc create sa -n virtual-builders quay-builder
```

4. 作成したサービスアカウントに編集権限を付与して、ビルドを実行できるようにします。

```
$ oc adm policy -n virtual-builders add-role-to-user edit system:serviceaccount:virtual-builders:quay-builder
```

5. Quay ビルダーに **anyuid scc** 権限を付与します。

```
$ oc adm policy -n virtual-builders add-scc-to-user anyuid -z quay-builder
```

注記

このアクションには、クラスター管理者特権が必要です。非特権ビルドまたはルートレスビルドを機能させるには、ビルダーを Podman ユーザーとして実行する必要があるため、これが必要です。

6. Quay ビルダーサービスアカウントのトークンを取得します。

   a. OpenShift Container Platform 4.10 以前のバージョンを使用している場合は、以下のコマンドを入力します。

```
oc sa get-token -n virtual-builders quay-builder
```

   b. OpenShift Container Platform 4.11 以降を使用している場合は、以下のコマンドを入力します。

```
$ oc create token quay-builder -n virtual-builders
```

出力例

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJscldSdEUtYWsifQ...
```

7. ビルダールートを決定します。

```
$ oc get route -n quay-enterprise
```

出力例

```
NAME                          HOST/PORT                                                  PATH
SERVICES                      PORT   TERMINATION     WILDCARD
...
example-registry-quay-builder         example-registry-quay-builder-quay-enterprise.apps.docs.quayteam.org                example-registry-quay-app             grpc
edge/Redirect   None
...
```

8. 拡張子が .crt の自己署名 SSL 証明書を生成します。

```
$ oc extract cm/kube-root-ca.crt -n openshift-apiserver ca.crt
```

```
$ mv ca.crt extra_ca_cert_build_cluster.crt
```

9. コンソールで設定バンドルのシークレットを見つけ、Actions → Edit Secret を選択して、適切なビルダー設定を追加します。

```
FEATURE_USER_INITIALIZE: true
```

```
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- <superusername>
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: <sample_build_route> 1
BUILD_MANAGER:
 - ephemeral
 - ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  JOB_REGISTRATION_TIMEOUT: 3600 2
  ORCHESTRATOR:
   REDIS_HOST: <sample_redis_hostname> 3
   REDIS_PASSWORD: ""
   REDIS_SSL: false
   REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
   - EXECUTOR: kubernetesPodman
     NAME: openshift
     BUILDER_NAMESPACE: <sample_builder_namespace> 4
     SETUP_TIME: 180
     MINIMUM_RETRY_THRESHOLD:
     BUILDER_CONTAINER_IMAGE: <sample_builder_container_image> 5
     # Kubernetes resource options
     K8S_API_SERVER: <sample_k8s_api_server> 6
     K8S_API_TLS_CA: <sample_crt_file> 7
     VOLUME_SIZE: 8G
     KUBERNETES_DISTRIBUTION: openshift
     CONTAINER_MEMORY_LIMITS: 300m 8
     CONTAINER_CPU_LIMITS: 1G 9
     CONTAINER_MEMORY_REQUEST: 300m 10
     CONTAINER_CPU_REQUEST: 1G 11
     NODE_SELECTOR_LABEL_KEY: ""
     NODE_SELECTOR_LABEL_VALUE: ""
     SERVICE_ACCOUNT_NAME: <sample_service_account_name>
     SERVICE_ACCOUNT_TOKEN: <sample_account_token> 12
```

1. ビルドルートは、Open Shift Operators namespace の名前で **oc get route -n** を実行することにより取得されます。たとえば、ポートはルートの最後に指定する必要があり、**[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443** の形式に従う必要があります。

2. **JOB_REGISTRATION_TIMEOUT** パラメーターの設定が低すぎると、**failed to register job to build manager: rpc error: code = Unauthenticated desc = Invalid build token: Signature has expired** エラーが発生する可能性があります。このパラメーターは少なくとも 240 に設定することをお勧めします。

3. Redis ホストにパスワードまたは SSL 証明書がある場合は、それに応じて更新する必要があります。

4. 仮想ビルダーの namespace の名前と一致するように設定します (例: **virtual-builders**)。

5. 早期アクセスの場合、**BUILDER_CONTAINER_IMAGE** は現在 **quay.io/projectquay/quay-builder:3.7.0-rc.2** です。ただし、早期アクセス期間中に変更

される可能性があります。このような事態が発生した場合は、お客様に注意を促します。

6 **oc cluster-info** を実行して取得します。

7 カスタム CA 証明書を手動で作成して追加する必要があります (例 **K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt**)。

8 指定しない場合、デフォルトは 5120Mi です。

9 仮想ビルドの場合は、クラスターに十分なリソースがあることを確認する必要があります。指定しない場合、デフォルトは 1000m です。

10 指定しない場合、デフォルトは 3968Mi です。

11 指定しない場合、デフォルトは 500m です。

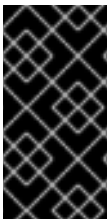
12 **oc create sa** の実行時に取得されます。

サンプル設定

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org:443
BUILD_MANAGER:
  - ephemeral
  - ALLOWED_WORKER_COUNT: 1
    ORCHESTRATOR_PREFIX: buildman/production/
    JOB_REGISTRATION_TIMEOUT: 3600
    ORCHESTRATOR:
      REDIS_HOST: example-registry-quay-redis
      REDIS_PASSWORD: ""
      REDIS_SSL: false
      REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
    EXECUTORS:
      - EXECUTOR: kubernetesPodman
        NAME: openshift
        BUILDER_NAMESPACE: virtual-builders
        SETUP_TIME: 180
        MINIMUM_RETRY_THRESHOLD:
        BUILDER_CONTAINER_IMAGE: quay.io/projectquay/quay-builder:3.7.0-rc.2
        # Kubernetes resource options
        K8S_API_SERVER: api.docs.quayteam.org:6443
        K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt
        VOLUME_SIZE: 8G
        KUBERNETES_DISTRIBUTION: openshift
        CONTAINER_MEMORY_LIMITS: 1G
        CONTAINER_CPU_LIMITS: 1080m
        CONTAINER_MEMORY_REQUEST: 1G
        CONTAINER_CPU_REQUEST: 580m
        NODE_SELECTOR_LABEL_KEY: ""
```

```
      NODE_SELECTOR_LABEL_VALUE: ""
      SERVICE_ACCOUNT_NAME: quay-builder
      SERVICE_ACCOUNT_TOKEN:
"eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJscldSdEUtYWsifQ"
```

9.3.2.2. SSL 証明書を手動で追加



重要

- 設定ツールの既知の問題のため、ビルダーを適切に実行するには、カスタム SSL 証明書を手動で追加する必要があります。次の手順を使用して、カスタム SSL 証明書を手動で追加します。SSL 証明書の作成の詳細については、Red Hat Quay コンテナーへの TLS 証明書の追加 を参照してください。

9.3.2.2.1. 証明書を作成して署名

1. 認証局を作成し、証明書に署名します。詳細については、認証局の作成と証明書への署名 を参照してください。

注記

- Quay レジストリーの URL に **alt_name** を追加します。
- config.yaml で指定されている **BUILDMAN_HOSTNAME** に **alt_name** を追加します。

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 = example-registry-quay-quay-enterprise.apps.docs.quayteam.org
DNS.2 = example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org
```

サンプルコマンド

```
$ openssl genrsa -out rootCA.key 2048
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
$ openssl genrsa -out ssl.key 2048
$ openssl req -new -key ssl.key -out ssl.csr
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
ssl.cert -days 356 -extensions v3_req -extfile openssl.cnf
```

9.3.2.2.2. TLS をアンマネージドに設定

Quay Registry yaml で、**kind: tls** を **managed: false** に設定します。

```
- kind: tls
  managed: false
```

イベントでは、適切な設定をするまで、変更がブロックされていることがわかります。

```
- lastTransitionTime: '2022-03-28T12:56:49Z'
  lastUpdateTime: '2022-03-28T12:56:49Z'
  message: >-
    required component `tls` marked as unmanaged, but `configBundleSecret`
    is missing necessary fields
  reason: ConfigInvalid
  status: 'True'
```

9.3.2.2.3. 一時的なシークレットの作成

1. CA 証明書のデフォルトの namespace にシークレットを作成します。

```
$ oc create secret generic -n quay-enterprise temp-crt --from-file
extra_ca_cert_build_cluster.crt
```

2. ssl.key ファイルおよび ssl.cert ファイルのデフォルトの namespace にシークレットを作成します。

```
$ oc create secret generic -n quay-enterprise quay-config-ssl --from-file ssl.cert --from-file
ssl.key
```

9.3.2.2.4. シークレットデータを config.yaml にコピー

1. コンソール UI の **Workloads → Secrets** で新しいシークレットを見つけます。
2. シークレットごとに、YAML ビューを見つけます。

```
kind: Secret
apiVersion: v1
metadata:
  name: temp-crt
  namespace: quay-enterprise
  uid: a4818adb-8e21-443a-a8db-f334ace9f6d0
  resourceVersion: '9087855'
  creationTimestamp: '2022-03-28T13:05:30Z'
...
data:
  extra_ca_cert_build_cluster.crt: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0l....
type: Opaque
```

```
kind: Secret
apiVersion: v1
metadata:
  name: quay-config-ssl
  namespace: quay-enterprise
```

```
  uid: 4f5ae352-17d8-4e2d-89a2-143a3280783c
  resourceVersion: '9090567'
  creationTimestamp: '2022-03-28T13:10:34Z'
...
data:
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

3. UI で、またはコマンドラインから次のようなコマンドを実行して、Quay Registry 設定バンドルのシークレットを見つけます。

```
$ oc get quayregistries.quay.redhat.com -o jsonpath="{.items[0].spec.configBundleSecret}
{'\n'}"  -n quay-enterprise
```

4. OpenShift コンソールで、設定バンドルシークレットの YAML タブを選択し、作成した 2 つのシークレットからデータを追加します。

```
kind: Secret
apiVersion: v1
metadata:
  name: init-config-bundle-secret
  namespace: quay-enterprise
  uid: 4724aca5-bff0-406a-9162-ccb1972a27c1
  resourceVersion: '4383160'
  creationTimestamp: '2022-03-22T12:35:59Z'
...
data:
  config.yaml: >-
    RkVBVFVSRV9VU0VSX0lOSVRJQUxJWkU6IHRydWUKQlJ...
  extra_ca_cert_build_cluster.crt: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0ldw....
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

5. **Save** をクリックします。Pod が再起動していることを確認する必要があります。

```
$ oc get pods -n quay-enterprise
```

出力例

```
NAME                                        READY   STATUS          RESTARTS   AGE
...
example-registry-quay-app-6786987b99-vgg2v          0/1     ContainerCreating   0          2s
example-registry-quay-app-7975d4889f-q7tvl          1/1     Running             0          5d21h
example-registry-quay-app-7975d4889f-zn8bb          1/1     Running             0          5d21h
example-registry-quay-app-upgrade-lswsn             0/1     Completed           0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv   0/1     ContainerCreating   0          2s
```

```
example-registry-quay-config-editor-c6c4d9ccd-2mwg2   1/1   Running       0
5d21h
example-registry-quay-database-66969cd859-n2ssm       1/1   Running       0       6d1h
example-registry-quay-mirror-764d7b68d9-jmlkk         1/1   Terminating   0       5d21h
example-registry-quay-mirror-764d7b68d9-jqzwg         1/1   Terminating   0       5d21h
example-registry-quay-redis-7cc5f6c977-956g8          1/1   Running       0       5d21h
```

6. Quay レジストリーが再設定されたら、Quay アプリ Pod が実行されていることを確認します。

```
$ oc get pods -n quay-enterprise
```

出力例

```
example-registry-quay-app-6786987b99-sz6kb            1/1   Running       0       7m45s
example-registry-quay-app-6786987b99-vgg2v            1/1   Running       0       9m1s
example-registry-quay-app-upgrade-lswsn               0/1   Completed     0       6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv  1/1   Running       0       9m1s
example-registry-quay-database-66969cd859-n2ssm       1/1   Running       0       6d1h
example-registry-quay-mirror-758fc68ff7-5wxlp         1/1   Running       0       8m29s
example-registry-quay-mirror-758fc68ff7-lbl82         1/1   Running       0       8m29s
example-registry-quay-redis-7cc5f6c977-956g8          1/1   Running       0       5d21h
```

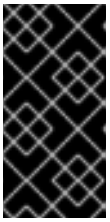
7. ブラウザーで、レジストリーエンドポイントにアクセスし、証明書が適切に更新されていることを確認します。

```
Common Name (CN) example-registry-quay-quay-enterprise.apps.docs.quayteam.org
Organisation (O) DOCS
Organisational Unit (OU) QUAY
```

9.3.2.3. UI を使用してビルドトリガーを作成

1. Quay リポジトリーにログインします。
2. **Create New Repository** をクリックして、**testrepo** などの新しいレジストリーを作成します。
3. **Repositories** ページで、左側のペインの **Builds** タブをクリックします。または、対応する URL を直接使用します。次に例を示します。

```
https://example-registry-quay-quay-
enterprise.apps.docs.quayteam.org/repository/quayadmin/testrepo?tab=builds
```



重要

場合によっては、ビルダーでホスト名の解決に問題が発生することがあります。この問題は、ジョブオブジェクトで **default** に設定されている **dnsPolicy** に関連している可能性があります。現在、この問題に対する回避策はありません。これは、Red Hat Quay の将来のバージョンで解決される予定です。

4. **Create Build Trigger→ Custom Git Repository Push** をクリックします。
5. Git リポジトリーのクローン作成に使用する HTTPS または SSH スタイルの URL を入力し、**Continue** をクリックします。以下に例を示します。

```
https://github.com/gabriel-rh/actions_test.git
```

6. **Tag manifest with the branch or tag name**を確認し、**Continue** をクリックします。
7. トリガーが呼び出されたときにビルドする Dockerfile の場所 (たとえば **/Dockerfile**) を入力し、**Continue** をクリックします。
8. Docker ビルドのコンテキストの場所 (たとえば /) を入力し、**Continue** をクリックします。
9. 必要に応じて、ロボットアカウントを作成します。それ以外の場合は、**Continue** をクリックします。
10. **Continue** をクリックして、パラメーターを確認します。
11. **Builds** ページで、トリガー名の **Options** アイコンをクリックし、**Run Trigger Now** をクリックします。
12. Git リポジトリーからコミット SHA を入力し、**Start Build** をクリックします。
13. ビルドのステータスを確認するには、**Build History** ページで commit をクリックするか、**oc get pods -n virtual-builders** を実行します。

```
 $ oc get pods -n virtual-builders
NAME                                          READY   STATUS    RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Running   0          7s
```

```
$ oc get pods -n virtual-builders
NAME                                          READY   STATUS        RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Terminating   0          9s
```

```
$ oc get pods -n virtual-builders
No resources found in virtual-builders namespace.
```

14. ビルドが完了したら、左側のペインのタグで **Tags** のステータスを確認できます。

注記

早期アクセスにより、完全なビルドログとビルドのタイムスタンプは現在利用できません。

9.3.2.4. AWS S3 ストレージバケットの変更

AWS S3 ストレージを使用している場合は、ビルダーを実行する前に、AWS コンソールでストレージバケットを変更する必要があります。

1. s3.console.aws.com で AWS コンソールにログインします。
2. 検索バーで **S3** を検索し、**S3** をクリックします。
3. バケットの名前 (**myawsbucket** など) をクリックします。
4. **Permissions** タブをクリックします。
5. **Cross-origin resource sharing (CORS)**の下に、次のパラメーターを含めます。

```
[
    {
        "AllowedHeaders": [
            "Authorization"
        ],
        "AllowedMethods": [
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    },
    {
        "AllowedHeaders": [
            "Content-Type",
            "x-amz-acl",
            "origin"
        ],
        "AllowedMethods": [
            "PUT"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    }
]
```

第10章 GEO レプリケーション

Geo レプリケーションでは、地理的に分散した複数の Red Hat Quay デプロイメントを、クライアントやユーザーの視点から、単一のレジストリーとして動作させることができます。グローバルに分散された Red Hat Quay のセットアップにおいて、プッシュとプルのパフォーマンスが大幅に向上します。イメージデータはバックグラウンドで非同期的に複製され、クライアントには透過的なフェイルオーバー/リダイレクトが行われます。

Red Hat Quay 3.7 では、Geo レプリケーションを使用した Red Hat Quay のデプロイメントは、スタンドアロンおよびオペレーターデプロイメントによってサポートされます。

10.1. GEO レプリケーション機能

- Geo レプリケーションが設定されていると、コンテナーイメージのプッシュはその Red Hat Quay インスタンスの推奨ストレージエンジンに書き込まれます (通常はリージョン内の最も近いストレージバックエンド)。
- 最初のプッシュの後、イメージデータはバックグランドで他のストレージエンジンに複製されます。
- レプリケーションロケーションのリストは設定可能で、それらは異なるストレージバックエンドにすることができます。
- イメージプルでは、プルのパフォーマンスを最大化するために、常に利用可能な最も近いストレージエンジンを使用します。
- レプリケーションがまだ完了していない場合、プルは代わりにソースストレージのバックエンドを使用します。

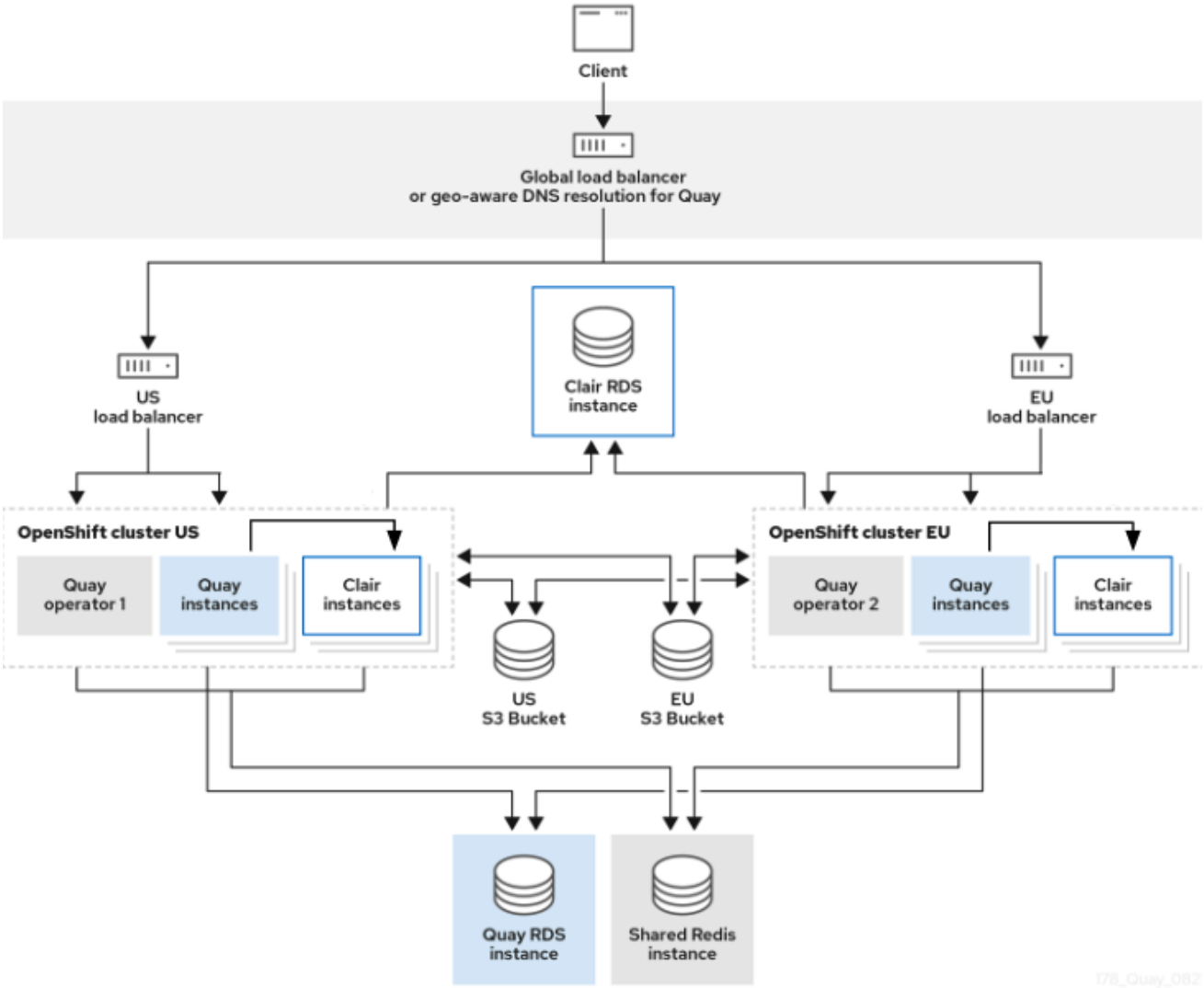
10.2. GEO レプリケーションの要件と制約

- geo レプリケーション設定では、Red Hat Quay では、すべてのリージョンが他のすべてのリージョンのオブジェクトストレージに対して読み取り/書き込みできる必要があります。オブジェクトストレージは、他のすべてのリージョンから地理的にアクセスできる必要があります。
- 1 つの geo レプリケーションサイトでオブジェクトストレージシステムに障害が発生した場合に、そのサイトの Red Hat Quay デプロイメントをシャットダウンして、クライアントがグローバルロードバランサーにより、ストレージシステムで問題のない残りのサイトにリダイレクトされるようにする必要があります。そうしないと、クライアントでプルとプッシュの失敗が発生します。
- Red Hat Quay は、接続されたオブジェクトストレージシステムの内部の正常性または可用性に関する認識はありません。1 つのサイトのオブジェクトストレージシステムが利用できなくなった場合に、残りのサイトの残りのストレージシステム (複数可) に自動的にリダイレクトされません。
- Geo レプリケーションは非同期です。サイトが完全に失われると、そのサイトのオブジェクトストレージシステムに保存されているが、障害発生時に残りのサイトにまだ複製されていないデータが失われます。
- 1 つのデータベース、つまりすべてのメタデータと Quay の設定がすべてのリージョンで共有されます。
  geo レプリケーションはデータベースをレプリケートしません。障害が発生した場合、geo レプリケーションが有効になっている Red Hat Quay は別のデータベースにフェイルオーバーしません。

- 1 つの Redis キャッシュは Quay のセットアップ全体で共有され、すべての Quay Pod からアクセスできる必要があります。
- ストレージバックエンド以外のすべてのリージョンで同じ設定を使用する必要があります。これは、**QUAY_DISTRIBUTED_STORAGE_PREFERENCE** 環境変数を使用して明示的に設定できます。
- Geo レプリケーションでは、各リージョンにオブジェクトストレージが必要です。ローカルストレージや NFS では動作しません。
- 各リージョンは、各リージョンのすべてのストレージエンジンにアクセスできる必要があります (ネットワークパスが必要)。
- また、ストレージプロキシーオプションを使用することもできます。
- ストレージのバックエンド全体 (すべての blob) が複製されます。これは、組織、リポジトリー、イメージに限定することができるリポジトリーミラーリングとは対照的です。
- すべての Quay インスタンスは、ロードバランサーを介して同じエントリーポイントを共有する必要があります。
- すべての Quay インスタンスは、共通の設定ファイルで定義された同じスーパーユーザーのセットを持つ必要があります。
- Geo レプリケーションでは、Clair 設定を **unmanaged** に設定する必要があります。管理されていない Clair データベースにより、Red Hat Quay オペレーターは、Operator の複数のインスタンスが同じデータベースと通信する必要がある地理的に複製された環境で作業できます。詳細は、Advanced Clair configuration を参照してください。
- Geo レプリケーションには、SSL/TSL 証明書とキーが必要です。詳細は、Using SSL to protect connections to Red Hat Quay を参照してください。

上記の要件を満たすことができない場合は、代わりに 2 つ以上の異なる Quay のデプロイメントを使用し、リポジトリーミラーリング機能を利用する必要があります。

10.3. RED HAT QUAY OPERATOR を使用した GEO レプリケーション



上記の例では、Red Hat Quay Operator は、共通のデータベースと共通の Redis インスタンスを使用して、2 つの別々のリージョンにデプロイされています。ローカライズされたイメージストレージは各リージョンで提供され、最も近くにある利用可能なストレージエンジンからイメージプルが提供されます。コンテナーイメージのプッシュは、Quay インスタンスの推奨ストレージエンジンに書き込まれ、次にバックグラウンドで他のストレージエンジンに複製されます。

Operator は現在、Clair セキュリティースキャナーとそのデータベースを別々に管理しているため、Geo レプリケーションの設定を活用して、Clair データベースを管理しないようにすることができます。代わりに、外部共有データベースが使用されます。Red Hat Quay と Clair は、PostgreSQL のいくつかのプロバイダーとベンダーをサポートしています。これらは Red Hat Quay 3.x test matrix にあります。さらに、Operator は、デプロイメントに挿入できるカスタム Clair 設定もサポートします。これにより、ユーザーは外部データベースの接続資格情報を使用して Clair を設定できます。

10.3.1. Openshift での Geo レプリケーションの設定

手順

1. Quaypostgres インスタンスをデプロイします。
   a. データベースにログインします。
   b. Quay のデータベースを作成します。

```
CREATE DATABASE quay;
```

c. データベース内で pg_trm 拡張機能を有効にします。

```
\c quay;
CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

2. Redis インスタンスをデプロイします。

注記

- クラウドプロバイダーに独自のサービスがある場合は、Redis インスタンスをデプロイする必要がない場合があります。
- Builder を利用している場合は、Redis インスタンスをデプロイする必要があります。

a. Redis 用の VM をデプロイします。

b. Quay が実行されているクラスターからアクセスできることを確認してください。

c. ポート 6379/TCP が開いている必要があります。

d. インスタンス内で Redis を実行します。

```
sudo dnf install -y podman
podman run -d --name redis -p 6379:6379 redis
```

3. クラスターごとに 1 つずつ、2 つのオブジェクトストレージバックエンドを作成します。理想的には、一方のオブジェクトストレージバケットは 1 番目のクラスター (プライマリー) の近くにあり、もう一方は 2 番目のクラスター (セカンダリー) の近くにあります。

4. 環境変数のオーバーライドを使用して、同じ設定バンドルでクラスターをデプロイし、個々のクラスターに適切なストレージバックエンドを選択します。

5. クラスターへの単一のエントリーポイントを提供するように、ロードバランサーを設定します。

10.3.1.1. 設定

config.yaml ファイルはクラスター間で共有され、一般的な PostgreSQL、Redis、およびストレージバックエンドの詳細が含まれます。

config.yaml

```
SERVER_HOSTNAME: <georep.quayteam.org or any other name> 1
DB_CONNECTION_ARGS:
  autorollback: true
  threadlocals: true
DB_URI: postgresql://postgres:password@10.19.0.1:5432/quay 2
BUILDLOGS_REDIS:
  host: 10.19.0.2
  port: 6379
```

```
USER_EVENTS_REDIS:
  host: 10.19.0.2
  port: 6379
DISTRIBUTED_STORAGE_CONFIG:
  usstorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPGVMASAAMQABCDEFG
      bucket_name: georep-test-bucket-0
      secret_key: AYWfEaxX/u84XRA2vUX5C987654321
      storage_path: /quaygcp
  eustorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPGVMASAAMQWERTYUIOP
      bucket_name: georep-test-bucket-1
      secret_key: AYWfEaxX/u84XRA2vUX5Cuj12345678
      storage_path: /quaygcp
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS:
  - usstorage
  - eustorage
DISTRIBUTED_STORAGE_PREFERENCE:
  - usstorage
  - eustorage
FEATURE_STORAGE_REPLICATION: true
```

1. ルートには適切な **SERVER_HOSTNAME** を使用する必要があり、グローバルロードバランサーのホスト名と一致する必要があります。
2. OpenShift Operator を使用してデプロイされた Clair インスタンスの設定ファイルを取得するには、Retrieving the Clair config 参照してください。

configBundleSecret を作成します。

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

各クラスターで、**configBundleSecret** を設定し、**QUAY_DISTRIBUTED_STORAGE_PREFERENCE** 環境変数のオーバーライドを使用して、そのクラスターに適切なストレージを設定します。

注記

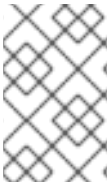
両方のデプロイメント間の **config.yaml** ファイルは一致する必要があります。一方のクラスターに変更を加える場合は、もう一方のクラスターでも変更する必要があります。

米国のクラスター

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

```
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: postgres
      managed: false
    - kind: clairpostgres
      managed: false
    - kind: redis
      managed: false
    - kind: quay
      managed: true
      overrides:
        env:
        - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
          value: usstorage
    - kind: mirror
      managed: true
      overrides:
        env:
        - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
          value: usstorage
```

+



注記

TLS は管理されておらず、ルートは管理されているため、設定ツールを使用するか、設定バンドルで直接証明書を提供する必要があります。詳細は、Configuring TLS and routes を参照してください。

ヨーロッパのクラスター

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: georep-config-bundle
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: postgres
      managed: false
    - kind: clairpostgres
      managed: false
    - kind: redis
      managed: false
    - kind: quay
      managed: true
```

```
    overrides:
      env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: eustorage
  - kind: mirror
    managed: true
    overrides:
      env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: eustorage
```

+

注記

TLS は管理されておらず、ルートは管理されているため、設定ツールを使用するか、設定バンドルで直接証明書を提供する必要があります。詳細は、Configuring TLS and routes を参照してください。

10.3.2. Geo レプリケーションのための複合ストレージ

Red Hat Quay の Geo レプリケーションは、異なる複数のレプリケーションターゲットの使用をサポートしています。たとえば、パブリッククラウドの AWS S3 ストレージとオンプレミスの Ceph ストレージを使用する、などです。これは、すべての Red Hat Quay Pod とクラスターノードからすべてのストレージバックエンドへのアクセスを許可するという重要な要件を複雑にします。その結果、以下が推奨されています。

- VPN を使用して、内部ストレージの可視化を防ぐ。**または**
- Quay が使用する指定のバケットへのアクセスのみを許可するトークンペアを使用する。

これにより、Red Hat Quay のパブリッククラウドインスタンスはオンプレミスのストレージにアクセスできるようになりますが、ネットワークは暗号化され、保護され、ACL を使用することで、セキュリティー要件を満たすことができます。

これらのセキュリティー対策を実施できない場合は、2 つの異なる Red Hat Quay レジストリーをデプロイし、Geo レプリケーションの代わりにリポジトリーミラーリングを使用することが推奨されます。

第11章 RED HAT QUAY OPERATOR によって管理される RED HAT QUAY のバックアップおよび復元

OpenShift Container Platform で Red Hat Quay Operator によって管理される場合、このセクション内のコンテンツを使用して Red Hat Quay をバックアップおよび復元します。

11.1. RED HAT QUAY のバックアップ

この手順では、Red Hat Quay Operator を使用して OpenShift Container Platform にデプロイされた Red Hat Quay のバックアップを作成する方法を説明します。

前提条件

- Red Hat Quay Operator を使用して、OpenShift Container Platform で正常に Red Hat Quay がデプロイメントされている (状況条件 **Available** が **true** に設定されている)。
- コンポーネント **quay**、**postgres**、および **objectstorage** は **managed: true** に設定されている。
- コンポーネント **clair** が **managed: true** に合、コンポーネント **clairpostgres** も **managed: true** に設定されている (Red Hat Quay Operator v3.7 以降で開始)。

注記

デプロイメントに部分的に管理されていないデータベースまたはストレージコンポーネントが含まれ、Postgres または S3 互換オブジェクトストレージの外部サービスを使用している場合、Red Hat Quay デプロイメントを実行するには、サービスプロバイダーまたはベンダーのドキュメントを参照してデータのバックアップを作成してください。このガイドで説明されているツールは、外部 Postgres データベースまたはオブジェクトストレージのバックアップの開始点として参照できます。

11.1.1. Red Hat Quay 設定のバックアップ

1. **QuayRegistry** カスタムリソースをエクスポートしてバックアップします。

```
$ oc get quayregistry <quay-registry-name> -n <quay-namespace> -o yaml > quay-registry.yaml
```

2. 作成される **quayregistry.yaml** を編集し、ステータスセクションおよび以下のメタデータフィールドを削除します。

```
metadata.creationTimestamp
metadata.finalizers
metadata.generation
metadata.resourceVersion
metadata.uid
```

3. 管理対象キーシークレットをバックアップします。

注記

Red Hat Quay 3.7.0 より前のバージョンを実行している場合は、この手順を省略できます。一部のシークレットは Quay の初回デプロイ時に自動的に生成されます。これらは **QuayRegistry** リソースの名前空間で、**<quay-registry-name>-quay-registry-managed-secret-keys** というシークレットに保存されます。

```
$ oc get secret -n <quay-namespace> <quay-registry-name>-quay-registry-managed-secret-keys -o yaml > managed-secret-keys.yaml
```

4. 作成された **managed-secret-keys.yaml** ファイルを編集し、エントリー **metadata.ownerReferences** を削除します。**managed-secret-keys.yaml** ファイルは、以下のようになります。

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: <quayname>-quay-registry-managed-secret-keys
  namespace: <quay-namespace>
data:
  CONFIG_EDITOR_PW: <redacted>
  DATABASE_SECRET_KEY: <redacted>
  DB_ROOT_PW: <redacted>
  DB_URI: <redacted>
  SECRET_KEY: <redacted>
  SECURITY_SCANNER_V4_PSK: <redacted>
```

data プロパティーの情報はすべて同じままにする必要があります。

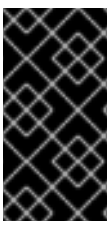
5. 現在の Quay 設定をバックアップします。

```
$ oc get secret -n <quay-namespace>  $(oc get quayregistry <quay-registry-name> -n <quay-namespace>  -o jsonpath='{.spec.configBundleSecret}') -o yaml > config-bundle.yaml
```

6. Quay Pod 内にマウントされた **/conf/stack/config.yaml** ファイルをバックアップします。

```
$ oc exec -it quay-pod-name -- cat /conf/stack/config.yaml > quay-config.yaml
```

11.1.2. Red Hat Quay デプロイメントのスケールダウン



重要

この手順は、Red Hat Quay デプロイメントの状態の整合性のあるバックアップを作成するために必要になります。Postgres データベースや S3 互換オブジェクトストレージが外部サービスによって提供されるセットアップ (Operator の管理対象外) を含め、この手順を省略しないでください。

1. **Operator バージョン 3.7 以降:** Red Hat Quay の自動スケーリングを無効にし、Red Hat Quay、ミラーワーカー、および Clair (管理対象の場合) のレプリカ数をオーバーライドすることで Red Hat Quay デプロイメントを縮小します。**QuayRegistry** リソースは、以下のようになります。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: false (1)
    - kind: quay
      managed: true
      overrides: (2)
        replicas: 0
    - kind: clair
      managed: true
      overrides:
        replicas: 0
    - kind: mirror
      managed: true
      overrides:
        replicas: 0
    …
```

(1) Quay、Clair、ミラーリングワーカーの自動スケーリングの無効化

(2) データベースおよびオブジェクトストレージにアクセスするコンポーネントのレプリカ数を 0 に設定

2. **Operator バージョン 3.6 以前**: まず Red Hat Quay Operator をスケールダウンしてから、マネージド Red Hat Quay リソースをスケールダウンして Red Hat Quay デプロイメントを縮小します。

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-operator-namespace>|awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-app/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-mirror/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/clair-app/ {print $1}') -n <quay-namespace>
```

3. **registry-quay-app**、**registry-quay-mirror**、および **registry-clair-app** Pod (どのコンポーネントを Red Hat Quay Operator がマネージするように設定したかにより異なります) が非表示になるまで待機します。以下のコマンドを実行してステータスを確認できます。

```
$ oc get pods -n <quay-namespace>
```

出力例:

```
$ oc get pod

quay-operator.v3.7.1-6f9d859bd-p5ftc               1/1     Running     0             12m
quayregistry-clair-postgres-7487f5bd86-xnxpr       1/1     Running     1 (12m ago)   12m
```

```
quayregistry-quay-app-upgrade-xq2v6               0/1     Completed   0          12m
quayregistry-quay-config-editor-6dfdcfc44f-hlvwm  1/1     Running     0          73s
quayregistry-quay-database-859d5445ff-cqthr       1/1     Running     0          12m
quayregistry-quay-redis-84f888776f-hhgms          1/1     Running     0          12m
```

11.1.3. Red Hat Quay 管理対象データベースのバックアップ

注記

Red Hat Quay デプロイメントが外部 (管理対象外) Postgres データベースで設定されている場合は、これらのデータベースの一貫したバックアップを作成する方法についてベンダーのドキュメントを参照してください。

1. Quay PostgreSQL Pod 名を特定します。

```
$ oc get pod -l quay-component=postgres -n <quay-namespace> -o
jsonpath='{.items[0].metadata.name}'
```

出力例:

```
quayregistry-quay-database-59f54bb7-58xs7
```

2. Quay データベース名を取得します。

```
$ oc -n <quay-namespace> rsh $(oc get pod -l app=quay -o NAME -n <quay-namespace>
|head -n 1) cat /conf/stack/config.yaml|awk -F"/" '/^DB_URI/ {print $4}'
quayregistry-quay-database
```

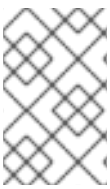
3. バックアップデータベースをダウンロードします。

```
$ oc exec quayregistry-quay-database-59f54bb7-58xs7 -- /usr/bin/pg_dump -C quayregistry-
quay-database  > backup.sql
```

11.1.3.1. Red Hat Quay 管理対象オブジェクトストレージのバックアップ

このセクションの手順は、以下の設定に適用されます。

- スタンドアロンのマルチクラウドオブジェクトゲートウェイ設定
- OpenShift Data Foundations ストレージでは、Red Hat Quay Operator が ObjectStorageBucketClaim API 経由で S3 オブジェクトストレージバケットをプロビジョニングしている必要があります。



注記

Red Hat Quay デプロイメントが外部 (管理対象外) オブジェクトストレージで設定されている場合は、Quay のストレージバケットのコンテンツのコピーを作成する方法についてベンダーのドキュメントを参照してください。

1. **AWS_ACCESS_KEY_ID** をデコードし、エクスポートします。

```
$ export AWS_ACCESS_KEY_ID=$(oc get secret -l app=noobaa -n <quay-namespace>  -o jsonpath='{.items[0].data.AWS_ACCESS_KEY_ID}' |base64 -d)
```

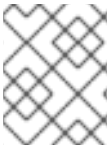
2. **AWS_SECRET_ACCESS_KEY_ID** をデコードし、エクスポートします。

```
$ export AWS_SECRET_ACCESS_KEY=$(oc get secret -l app=noobaa -n <quay-namespace> -o jsonpath='{.items[0].data.AWS_SECRET_ACCESS_KEY}' |base64 -d)
```

3. 新しいディレクトリーを作成し、すべての Blob をそのディレクトリーにコピーします。

```
$ mkdir blobs

$ aws s3 sync --no-verify-ssl --endpoint https://$(oc get route s3 -n openshift-storage  -o jsonpath='{.spec.host}')  s3://$(oc get cm -l app=noobaa -n <quay-namespace> -o jsonpath='{.items[0].data.BUCKET_NAME}') ./blobs
```



注記

AWS コマンドラインユーティリティーの代わりに、rclone または sc3md を使用することもできます。

11.1.4. Red Hat Quay デプロイメントのバックアップのスケーリング

1. **Operator バージョン 3.7 以降:** 自動スケーリングを再度有効にし、必要な場合は Quay、ミラーワーカー、および Clair のレプリカオーバーライドを適宜削除して、Red Hat Quay デプロイメントをスケールアップします。**QuayRegistry** リソースは、以下のようになります。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: true (1)
    - kind: quay (2)
      managed: true
    - kind: clair
      managed: true
    - kind: mirror
      managed: true
    …
```

(1) Quay、Clair、ミラーリングワーカーの自動スケーリングの再有効化 (必要に応じて)

(2) Quay コンポーネントのバックアップをスケーリングするためにレプリカオーバーライドを再削除

2. **Operator バージョン 3.6 以前の場合:** Red Hat Quay Operator を再度スケールアップして Red Hat Quay デプロイメントをスケールアップします。

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-operator-namespace> |
awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>
```

3. Red Hat Quay デプロイメントのステータスを確認します。

```
$ oc wait quayregistry registry --for=condition=Available=true -n <quay-namespace>
```

出力例:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: registry
  namespace: <quay-namespace>
  ...
spec:
  ...
status:
  - lastTransitionTime: '2022-06-20T05:31:17Z'
    lastUpdateTime: '2022-06-20T17:31:13Z'
    message: All components reporting as healthy
    reason: HealthChecksPassing
    status: 'True'
    type: Available
```

11.2. RED HAT QUAY の復元

この手順は、Red Hat Quay Operator がデータベースを管理する際に Red Hat Quay を復元するために使用されます。これは、Red Hat Quay レジストリーをバックアップした後に実行する必要があります。詳細については、Red Hat Quay のバックアップ を参照してください。

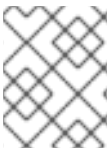
前提条件

- Red Hat Quay が、Red Hat Quay Operator を使用して OpenShift Container Platform にデプロイされている。
- Red Hat Quay Operator によって管理される Red Hat Quay 設定のバックアップが、Red Hat Quay のバックアップ セクションの手順に従って作成されている。
- Red Hat Quay データベースがバックアップされている。
- Red Hat Quay で使用されるオブジェクトストレージバケットがバックアップされている。
- コンポーネント **quay**、**postgres**、および **objectstorage** は **managed: true** に設定されている。
- コンポーネント **clair** が **managed: true** に設定されている場合、コンポーネント **clairpostgres** も **managed: true** に設定されている (Red Hat Quay Operator v3.7 以降で開始)。
- OpenShift Container Platform クラスターのターゲット namespace で、Red Hat Quay Operator によって管理される Red Hat Quay デプロイメントを実行していない。

注記

デプロイメントに部分的に管理されていないデータベースまたはストレージコンポーネントが含まれ、Postgres または S3 互換オブジェクトストレージの外部サービスを使用している場合、Red Hat Quay デプロイメントを実行するには、サービスプロバイダーまたはベンダーのドキュメントを参照して、Red Hat Quay を複弁する前にバックアップからデータを復元してください。

11.2.1. バックアップからの Red Hat Quay およびその設定の復元



注記

これらの手順では、Red Hat Quay のバックアップ ガイドのプロセスに従い、同じ名前のバックアップファイルを作成していることを前提としています。

1. バックアップされた Red Hat Quay 設定と生成されたキーをバックアップから復元します。

```
$ oc create -f ./config-bundle.yaml

$ oc create -f ./managed-secret-keys.yaml
```

重要

エラー **Error from server (AlreadyExists): error when creating "./config-bundle.yaml": secrets "config-bundle-secret" already exists** が発生した場合、**$ oc delete Secret config-bundle-secret -n <quay-namespace>** を使用して既存リソースを削除し、**$ oc create -f ./config-bundle.yaml** で再作成する必要があります。

2. **QuayRegistry** カスタムリソースを復元します。

```
$ oc create -f ./quay-registry.yaml
```

3. Red Hat Quay デプロイメントのステータスを確認し、これが利用可能になるまで待機します。

```
$ oc wait quayregistry registry --for=condition=Available=true -n <quay-namespace>
```

11.2.2. Red Hat Quay デプロイメントのスケールダウン

1. **Operator バージョン 3.7 以降:** Red Hat Quay の自動スケーリングを無効にし、Red Hat Quay、ミラーワーカー、および Clair (管理対象の場合) のレプリカ数をオーバーライドすることで Quay デプロイメントを縮小します。**QuayRegistry** リソースは、以下のようになります。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
```

```
    managed: false (1)
  - kind: quay
    managed: true
    overrides: (2)
      replicas: 0
  - kind: clair
    managed: true
    overrides:
      replicas: 0
  - kind: mirror
    managed: true
    overrides:
      replicas: 0
  …
```

(1) Quay、Clair、ミラーリングワーカーの自動スケーリングの無効化

(2) データベースおよびオブジェクトストレージにアクセスするコンポーネントのレプリカ数を 0 に設定

2. **Operator バージョン 3.6 以前**: まず Red Hat Quay Operator をスケールダウンしてから、管理対象の Red Hat Quay リソースをスケールダウンして Red Hat Quay デプロイメントを縮小します。

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-operator-namespace>|awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>

$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-app/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-mirror/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/clair-app/ {print $1}') -n <quay-namespace>
```

3. **registry-quay-app**、**registry-quay-mirror**、および **registry-clair-app** Pod (どのコンポーネントを Operator の管理対象として設定したかにより異なります) が非表示になるまで待機します。以下のコマンドを実行してステータスを確認できます。

```
$ oc get pods -n <quay-namespace>
```

出力例:

```
registry-quay-config-editor-77847fc4f5-nsbbv   1/1     Running           0          9m1s
registry-quay-database-66969cd859-n2ssm        1/1     Running           0          6d1h
registry-quay-redis-7cc5f6c977-956g8           1/1     Running           0          5d21h
```

11.2.3. Red Hat Quay データベースの復元

1. Quay データベース Pod を特定します。

```
$ oc get pod -l quay-component=postgres -n  <quay-namespace> -o jsonpath='{.items[0].metadata.name}'
```

出力例:

```
quayregistry-quay-database-59f54bb7-58xs7
```

2. ローカル環境および Pod にコピーして、バックアップをアップロードします。

```
$ oc cp ./backup.sql -n <quay-namespace> registry-quay-database-66969cd859-
n2ssm:/tmp/backup.sql
```

3. データベースに対してリモートターミナルを開きます。

```
$ oc rsh -n <quay-namespace> registry-quay-database-66969cd859-n2ssm
```

4. psql を入力します。

```
bash-4.4$ psql
```

5. 以下のコマンドを実行してデータベースを一覧表示できます。

```
postgres=# \l
```

出力例:

```
                                  List of databases
        Name         |         Owner          | Encoding |  Collate   |   Ctype    |   Access
privileges
-----------------------------+----------------------------+----------+------------+------------+---------------
--------
postgres                    | postgres                   | UTF8     | en_US.utf8 | en_US.utf8 |
quayregistry-quay-database | quayregistry-quay-database | UTF8     | en_US.utf8 |
en_US.utf8 |
```

6. データベースを削除します。

```
postgres=# DROP DATABASE "quayregistry-quay-database";
```

出力例:

```
DROP DATABASE
```

7. postgres CLI を終了して bash-4.4 を再入力します。

```
\q
```

8. PostgreSQL データベースをバックアップデータベースにリダイレクトします。

```
sh-4.4$ psql < /tmp/backup.sql
```

9. bash を終了します。

```
sh-4.4$ exit
```

11.2.4. Red Hat Quay オブジェクトストレージデータの復元

1. **AWS_ACCESS_KEY_ID** をエクスポートします。

```
$ export AWS_ACCESS_KEY_ID=$(oc get secret -l app=noobaa -n <quay-namespace>  -o jsonpath='{.items[0].data.AWS_ACCESS_KEY_ID}' |base64 -d)
```

2. **AWS_SECRET_ACCESS_KEY** をエクスポートします。

```
$ export AWS_SECRET_ACCESS_KEY=$(oc get secret -l app=noobaa -n <quay-namespace> -o jsonpath='{.items[0].data.AWS_SECRET_ACCESS_KEY}' |base64 -d)
```

3. 以下のコマンドを実行して、すべての Blob をバケットにアップロードします。

```
$ aws s3 sync --no-verify-ssl --endpoint https://$(oc get route s3 -n openshift-storage  -o jsonpath='{.spec.host}') ./blobs  s3://$(oc get cm -l app=noobaa -n <quay-namespace> -o jsonpath='{.items[0].data.BUCKET_NAME}')
```

注記

AWS コマンドラインユーティリティーの代わりに、rclone または sc3md を使用することもできます。

11.2.5. Red Hat Quay デプロイメントのスケールアップ

1. **Operator バージョン 3.7 以降:** 自動スケーリングを再度有効にし、必要な場合は Quay、ミラーワーカー、および Clair のレプリカオーバーライドを適宜削除して、Red Hat Quay デプロイメントをスケールアップします。**QuayRegistry** リソースは、以下のようになります。

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
  - kind: horizontalpodautoscaler
    managed: true (1)
  - kind: quay (2)
    managed: true
  - kind: clair
    managed: true
  - kind: mirror
    managed: true
  …
```

(1) Red Hat Quay、Clair、ミラーリングワーカーの自動スケーリングの再有効化 (必要に応じて)

(2) Red Hat Quay コンポーネントのバックアップをスケーリングするためにレプリカオーバーライドを再削除

2. **Operator バージョン 3.6 以前の場合:** Red Hat Quay Operator を再度スケールアップして Red Hat Quay デプロイメントをスケールアップします。

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-operator-namespace> |
awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>
```

3. Red Hat Quay デプロイメントのステータスを確認します。

```
$ oc wait quayregistry registry --for=condition=Available=true -n <quay-namespace>
```

出力例:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: registry
  namespace: <quay-namespace>
  ...
spec:
  ...
status:
  - lastTransitionTime: '2022-06-20T05:31:17Z'
    lastUpdateTime: '2022-06-20T17:31:13Z'
    message: All components reporting as healthy
    reason: HealthChecksPassing
    status: 'True'
    type: Available
```

第12章 QUAY OPERATOR のアップグレードの概要

Quay Operator は、**シンクロナイズドバージョニング** スキームに従います。つまり、Operator の各バージョンは 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。

12.1. 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と、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** オブジェクトで確認できます。**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。これが望ましくない場合は、**Manual** 承認ストラテジーを選択する必要があります。

12.2. QUAY OPERATOR 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 を参照してください。

一般的に、Red Hat Quay は以前の (N-1) マイナーバージョンからのアップグレードのみをサポートしています。たとえば、Red Hat Quay 3.0.5 から最新バージョンの 3.5 への直接アップグレードはサポートされていません。代わりに、ユーザーは次のようにアップグレードする必要があります。

1. 3.0.5 → 3.1.3
2. 3.1.3 → 3.2.2
3. 3.2.2 → 3.3.4
4. 3.3.4 → 3.4.z
5. 3.4.z → 3.5.z

必要なデータベースの移行が正しく実行され、適切な順序でアップグレードが行われるようにするために必要です。

場合によっては、Red Hat Quay は、以前の (N-2、N-3) マイナーバージョンからの直接のシングルステップアップグレードをサポートします。通常の以前のマイナーバージョンのみのアップグレードに対するこの例外は、古いリリースのお客様のアップグレード手順を簡素化します。次のアップグレードパ

スがサポートされています。

1. 3.3.z → 3.6.z
2. 3.4.z → 3.6.z
3. 3.4.z → 3.7.z
4. 3.5.z → 3.7.z

3.7 へのアップグレードを希望する Quay のスタンドアロンデプロイメントのユーザーは、スタンドアロンアップグレード ガイドを参照してください。

12.2.1. Quay のアップグレード

Quay をあるマイナーバージョンから次のマイナーバージョン (たとえば、3.4 → 3.5) に更新するには、Quay Operator の更新チャネルを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、ユーザーが最初にインストール時に選択した major-minor 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。承認戦略が **Automatic** に設定されている場合、Quay Operator は自動的に最新の **z** ストリームにアップグレードします。これにより、ダウンタイムがほとんどない (またはまったくない) 新しい **z** ストリームへの Quay の自動更新が行われます。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。

12.2.2. 3.3.z または 3.4.z から 3.6 に直接アップグレードする際の注意事項

12.2.2.1. エッジルーティングを有効にしてアップグレード

- 以前は、エッジルーティングを有効にして 3.3.z バージョンの Red Hat Quay を実行している場合、ユーザーは 3.4.z バージョンの Red Hat Quay にアップグレードできませんでした。これは、Red Hat Quay 3.6 のリリースで解決されました。
- 3.3.z から 3.6 にアップグレードするときに、Red Hat Quay 3.3.z デプロイメントで **tls.termination** が **none** に設定されている場合は、TLS エッジターミネーションを使用して HTTPS に変更され、デフォルトのクラスターワイルドカード証明書が使用されます。以下に例を示します。

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: quay33
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    enableRepoMirroring: true
    image: quay.io/quay/quay:v3.3.4-2
    ...
    externalAccess:
      hostname: quayv33.apps.devcluster.openshift.com
      tls:
        termination: none
    database:
...
```

12.2.2.2. サブジェクト別名のないカスタム TLS 証明書/キーペアを使用したアップグレード

Red Hat Quay 3.3.4 から Red Hat Quay 3.6 に直接アップグレードするときに、サブジェクト代替名 (SAN) なしで独自の TLS 証明書/キーペアを使用しているお客様には問題があります。Red Hat Quay 3.6 へのアップグレード中、デプロイメントはブロックされ、Quay TLS 証明書に SAN が必要であることを示す Quay Operator Pod ログからのエラーメッセージが表示されます。

可能であれば、SAN 内の正しいホスト名を使用して TLS 証明書を再生成する必要があります。考えられる回避策には、アップグレード後に **quay-app**、**quay-upgrade**、**quay-config-editor** Pod で環境変数を定義して、CommonName のマッチングを有効にすることが含まれます。

```
GODEBUG=x509ignoreCN=0
```

GODEBUG=x509ignoreCN=0 フラグは、SAN が存在しない場合に、X.509 証明書の CommonName フィールドをホスト名として扱うという従来の動作を有効にします。ただし、この回避策は再デプロイメント後も持続しないため、お勧めしません。

12.2.2.3. Quay Operator を使用して 3.3.z または 3.4.z から 3.6 にアップグレードする場合の Clair v4 の設定

OpenShift 上の新しい Red Hat Quay デプロイメントに Clair v4 をセットアップするには、Quay Operator を使用することが強く推奨されます。デフォルトでは、Quay Operator は、Red Hat Quay のデプロイとともに Clair のデプロイメントをインストールまたはアップグレードし、Clair のセキュリティースキャンを自動的に設定します。

OpenShift で Clair v4 をセットアップする手順は、Red Hat Quay OpenShift デプロイメントでの Clair のセットアップ を参照してください。

12.2.3. 3.3.z から 3.6 にアップグレードする際の Swift 設定

Red Hat Quay 3.3.z から 3.6.z にアップグレードすると、ユーザーが **Switch auth v3 requires tenant_id (string) in os_options** エラーを受け取る場合があります。回避策として、**DISTRIBUTED_STORAGE_CONFIG** を手動で更新して、**os_options** および **tenant_id** パラメーターを追加できます。

```
DISTRIBUTED_STORAGE_CONFIG:
  brscale:
  - SwiftStorage
  - auth_url: http://****/v3
    auth_version: "3"
    os_options:
      tenant_id: ****
      project_name: ocp-base
      user_domain_name: Default
    storage_path: /datastorage/registry
    swift_container: ocp-svc-quay-ha
    swift_password: *****
    swift_user: *****
```

12.2.4. Operator の更新チャネルの変更

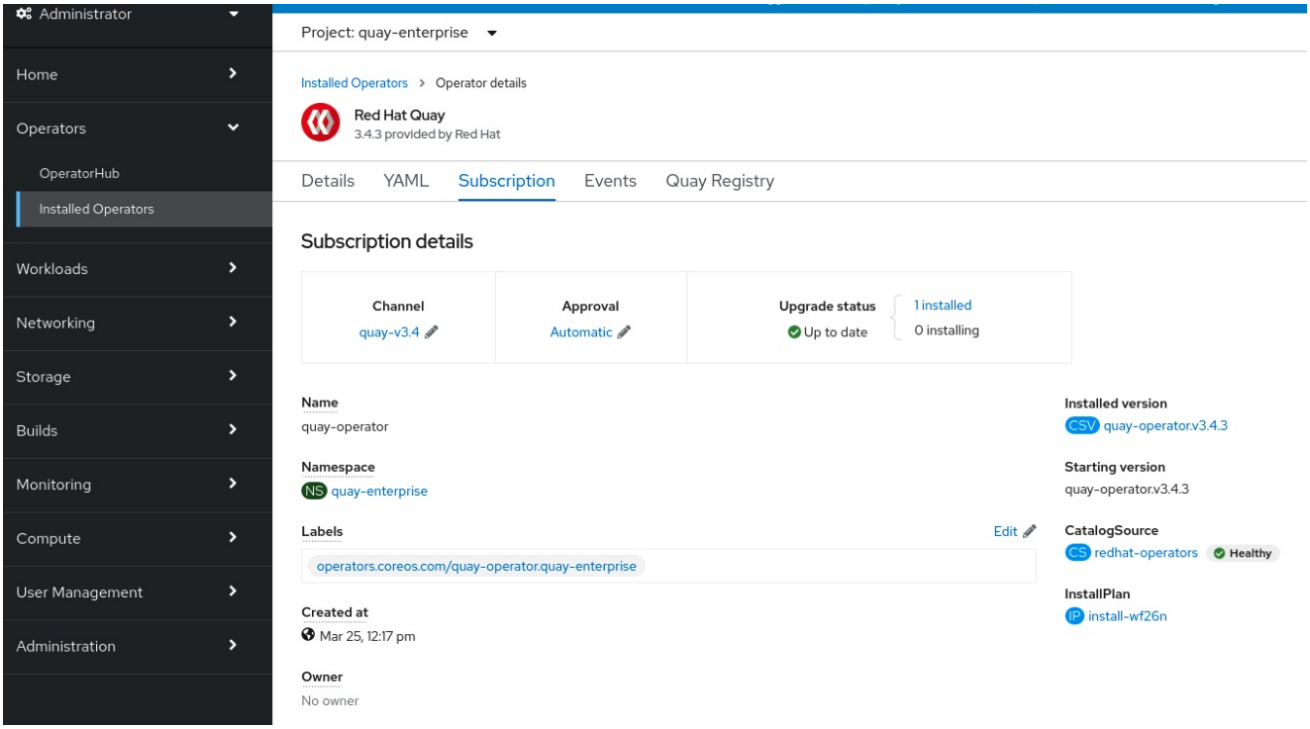
インストールされた Operator のサブスクリプションは、Operator の更新を追跡し、受信するために使用される更新チャネルを指定します。Quay Operator をアップグレードして新規チャネルからの更新の追跡および受信を開始するには、インストールされた Quay Operator の **Subscription** タブで更新チャ

ネルを変更します。**Automatic** 承認ストラテジーのある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。

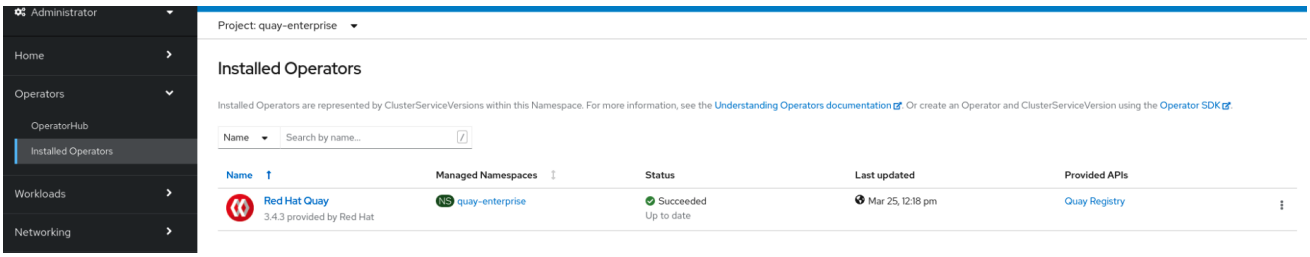
12.2.5. 保留中の Operator アップグレードの手動による承認

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は、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、Installed Operators を一覧表示したページに戻り、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** などの 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提供します。



12.3. QUAYREGISTRY のアップグレード

Quay Operator を起動すると、監視するように設定されている namespace にある **QuayRegistries** を探します。見つかった場合は、次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、通常通りリコンサイルを行います。
- **status.currentVersion** が Operator のバージョンと等しい場合は、通常通り調整を行います。

- **status.currentVersion** が Operator のバージョンと一致しない場合は、アップグレードできる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。

12.4. QUAY 3.7 の機能の有効化

12.4.1. クォータ管理設定

クォータ管理は **FEATURE_QUOTA_MANAGEMENT** プロパティーでサポートされるようになり、デフォルトでオフになっています。クォータ管理を有効にするには、**config.yaml** の機能フラグを **true** に設定します。

```
FEATURE_QUOTA_MANAGEMENT: true
```

12.4.2. Red Hat Quay を使用してリモート組織設定をプロキシーする

Red Hat Quay を使用したリモート組織のプロキシーが、**FEATURE_PROXY_CACHE** プロパティーでサポートされるようになりました。プロキシーキャッシュを有効にするには、**confg.yaml** の機能フラグを **true** に設定します。

```
FEATURE_PROXY_CACHE: true
```

12.4.3. Red Hat Quay ビルドの機能強化

ビルドは仮想化プラットフォームで実行できます。以前のビルド設定を実行するための下位互換性も利用できます。仮想ビルドを有効にするには、**config.yaml** の機能フラグを **true** に設定します。

```
FEATURE_BUILD_SUPPORT: true
```

12.4.4. Red Hat Quay Operator を使用した Geo レプリケーション

Geo レプリケーションを使用した Red Hat Quay のデプロイメントが、Operator デプロイメントでサポートされるようになりました。ジオレプリケーションを有効にするには、**config.yaml** の機能フラグを **true** に設定します。

```
FEATURE_STORAGE_REPLICATION: true
```

12.5. QUAY 3.6 の機能の有効化

12.5.1. 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 3.6 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は、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。

12.5.2. OCI および Helm サポート

Helm アーティファクトおよび OCI アーティファクトのサポートが、Red Hat Quay 3.6 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
```

12.6. QUAYECOSYSTEM のアップグレード

アップグレードは、**QuayEcosystem** API を使用して限られた設定を行っていた旧バージョンの Operator からサポートされています。移行が予期せず行われるようにするには、移行を行うために特別なラベルを **QuayEcosystem** に適用する必要があります。Operator が管理するための新しい **QuayRegistry** が作成されますが、古い **QuayEcosystem** は手動で削除されるまで残り、何か問題が発生した場合にロールバックして Quay にアクセスできるようになります。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"** の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. すべてが正しく動作したと確信したら、**QuayEcosystem** を削除しても構いません。Kubernetes のガベージコレクションがすべての古いリソースをクリーンアップします。

12.6.1. 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時に問題が発生した場合は、以下の手順を実行して **QuayEcosystem** の使用に戻します。

1. 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>
```

2. **Route** を使用して外部アクセスを提供していた場合は、UI や **kubectl** を使用して元の **Service** を指すように **Route** を変更します。

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は、アップグレードプロセスを終了し、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。

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

Quay Operator は、**QuayEcosystem** コンポーネントの移行に失敗したり、サポートされていない場合、ログや **status.conditions** にエラーを報告します。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はなく、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。

External Access

パススルー **Route** アクセスのみが自動移行でサポートされます。他の方法には手動移行が必要です。

- ホスト名のない **LoadBalancer**: **QuayEcosystem** にラベル **"quay-operator/migration-complete": "true"** が付けられた後、Kubernetes が **Service** をガベージコレクションしてロードバランサーを削除するのを防ぐため、**QuayEcosystem** を削除する **前** に、既存の **Service** から **metadata.ownerReferences** フィールドを削除します。新規 **Service** は **metadata.name** 形式の **<QuayEcosystem-name>-quay-app** で作成されます。既存の **Service** の **spec.selector** を新しい **Service** の **spec.selector** に合わせて編集することで、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しい Pod に誘導されるようになります。これで古い **Service** を管理します。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。新しい **Service** が提供する **status.loadBalancer** エンドポイントを指すように、DNS 設定を変更します。

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。

関連情報

- Red Hat Quay Operator 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。