Red Hat Quay 3.11

Red Hat Quay Operator の OpenShift Container Platform へのデプロイ

Red Hat Quay Operator の OpenShift Container Platform へのデプロイ

Last Updated: 2024-04-12

Red Hat Quay 3.11 Red Hat Quay Operator の OpenShift Container Platform へのデプロイ

Red Hat Quay Operator の OpenShift Container Platform へのデプロイ

法律上の通知

概要

Red Hat Quay Operator を OpenShift Container Platform クラスターにデプロイします。

目次

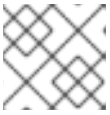
はじめに

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 のリリースに伴い、エクスペリエンスの強化と Day 2 運用に対するサポートの追加を目的として、Red Hat Quay Operator が再記述されました。その結果、Red Hat Quay Operator の操作性が向上し、デフォルト部分が増加して使いやすくなりました。Red Hat Quay 3.4.0 と前のバージョンとの主な違いは次のとおりです。

- **QuayEcosystem** カスタムリソースが **QuayRegistry** カスタムリソースに置き換えられました。
- デフォルトのインストールオプションで、実稼働環境での使用に対応した管理対象の依存関係 (データベース、キャッシュ、オブジェクトストレージなど) が含まれる、完全にサポートされた Red Hat Quay 環境が生成されます。



注記

一部のコンポーネントは、高可用性を備えていない可能性があります。

- Red Hat Quay の設定用の新しい検証ライブラリー。
- オブジェクトストレージは、**ObjectBucketClaim** Kubernetes API を使用して Red Hat Quay Operator が管理できるようになりました。

注記

Red Hat OpenShift Data Foundation を使用して、この API がサポートされる実装を OpenShift Container Platform 上で提供できます。

- テストおよび開発シナリオ用にデプロイされた Pod で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

この章の内容を使用して、以下を実行します。

- Red Hat Quay Operator を使用して OpenShift Container Platform に Red Hat Quay をインストールする
- 管理対象または管理対象外のオブジェクトストレージを設定する
- データベース、Redis、ルート、TLS などの管理対象外のコンポーネントを設定する
- Red Hat Quay Operator を使用して Red Hat Quay レジストリーを OpenShift Container Platform にデプロイする
- Red Hat Quay でサポートされている高度な機能を使用する
- Red Hat Quay Operator を使用して Red Hat Quay レジストリーをアップグレードする

1.1. RED HAT QUAY OPERATOR コンポーネント

Red Hat Quay には多くの依存関係があります。依存関係には、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。Red Hat Quay Operator は、Red Hat Quay の独自のデプロイメントとその Kubernetes への依存関係を管理します。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、**maned** (コンポーネントのライフサイクルが Red Hat Quay Operator によって処理されるかどうかを指定するブール値) の 2 つのフィールドが含まれています。

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入力されます。

QuayRegistry リソースの例

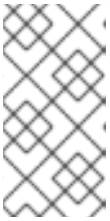
```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
  spec:
    configBundleSecret: config-bundle-secret
    components:
    - kind: quay
      managed: true
    - kind: postgres
      managed: true
    - kind: clair
      managed: true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

1.2. 管理コンポーネントの使用

QuayRegistry カスタムリソースで特に指定しない限り、Red Hat Quay Operator は以下の管理コンポーネントにデフォルトを使用します。

- **quay:** OpenShift Container Platform 上の Red Hat Quay のデプロイメントに関するオーバーライド (環境変数やレプリカの数など) を保持します。このコンポーネントは Red Hat Quay 3.7 以降の新しいコンポーネントであり、管理対象外に設定できません。
- **postgres:** レジストリーメタデータの保存には、Red Hat Quay 3.9 以降、 Software Collections の PostgreSQL 13 のバージョンが使用されます。



注記

Red Hat Quay 3.8 → 3.9 にアップグレードする場合、Operator は PostgreSQL 10 から PostgreSQL 13 へのアップグレードを自動的に処理します。このアップグレードは必須です。PostgreSQL 10 は 2022 年 11 月 10 日に最終リリースとなり、サポートされなくなりました。

- **clair:** イメージの脆弱性スキャンを提供します。
- **redis:** ライブビルダーログと Red Hat Quay チュートリアルを保存します。ガベージコレクションに必要なロックメカニズムも含まれます。
- **horizontalpodautoscaler:** メモリー/CPU 消費量に応じて **Quay** Pod の数を調整します。
- **objectstorage:** イメージレイヤー Blob を保存するために、Noabaa または Red Hat OpenShift Data Foundation が提供する **ObjectBucketClaim** Kubernetes API を利用します。
- **route:** OpenShift Container Platform の外部から Red Hat 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:** 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ようにリポジトリーミラーワーカーを設定します。
- **monitoring:** Grafana ダッシュボード、個々のメトリクスへのアクセス、**Quay** Pod の頻繁な再起動に関する通知などの機能があります。
- **tls:** Red Hat Quay または OpenShift Container Platform が SSL/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。これは、Red Hat Quay のデプロイに使用される PostgreSQL データベースとは別のデータベースです。

Red Hat Quay Operator は、Red Hat Quay がマネージド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。Red Hat Quay Operator が実行するデプロイが環境に適さない場合は、Red Hat Quay Operator に **unmanaged** リソース (オーバーライド) を指定できます。これについては次のセクションで説明します。

1.3. 依存関係向けのマネージド外コンポーネントの使用

Red Hat Quay で使用する PostgreSQL、Redis、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 Red Hat Quay 設定バンドル (**config.yaml**) 内でそれらを設定します。次に、どのコンポーネントが管理対象外であるかを示しながら、**QuayRegistry** バンドル内で (Kubernetes **Secret** として) 参照する必要があります。

注記

アンマネージド PostgreSQL データベースを使用していて、バージョンが PostgreSQL 10 である場合は、PostgreSQL 13 へのアップグレードが強く推奨されます。PostgreSQL 10 は 2022 年 11 月 10 日に最終リリースとなり、サポートされなくなりました。詳細は、PostgreSQL のバージョン管理ポリシー を参照してください。

管理対象外コンポーネントの設定については、以下のセクションを参照してください。

- 既存の PostgreSQL データベースの使用
- 管理対象外 Horizontal Pod Autoscaler の使用
- 管理対象外ストレージの使用
- 管理対象外 NooBaa インスタンスの使用
- 管理対象外 Redis データベースの使用
- ルートコンポーネントの無効化
- モニタリングコンポーネントの無効化
- ミラーリングコンポーネントの無効化

1.4. 設定バンドルシークレット

spec.configBundleSecret フィールドは、**QuayRegistry** リソースと同じ namespace にある **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。

config.yaml ファイルは、Red Hat Quay の **config.yaml** ファイルです。このフィールドはオプションで、指定されない場合は Red Hat Quay Operator により自動入力されます。指定されている場合、後に管理コンポーネントの他のフィールドにマージされる設定フィールドのベースセットとして機能し、Red Hat 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。

1.5. OPENSHIFT CONTAINER PLATFORM 上の RED HAT QUAY の前提条件

Red Hat Quay Operator を使用して OpenShift Container Platform に Red Hat Quay をデプロイする前に、次の前提条件を考慮してください。

1.5.1. OpenShift Container Platform クラスター

Red Hat Quay Operator をデプロイするには、OpenShift Container Platform 4.5 以降のクラスターと管理アカウントへのアクセス権が必要です。管理アカウントには、クラスタースコープで namespaces を作成する権限が必要です。

1.5.2. 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。

- 8 Gi 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 Container Platform 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てください。

1.5.3. 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 を使用すると、Red Hat Quay Operator がベンダー固有の実装から切り離されます。この API は、Red Hat OpenShift Data Foundation の NooBaa コンポーネントを介して提供されます。NooBaa はこのドキュメント全体で例として使用されています。

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は S3 IAM Bucket Policy を参照)
- Microsoft Azure Blob Storage
- Google Cloud Storage
- Ceph Object Gateway(RADOS)
- OpenStack Swift
- CloudFront + S3

1.5.4. StorageClass

Red Hat Quay Operator を使用して **Quay** および **Clair** PostgreSQL データベースをデプロイする場合、デフォルトの **StorageClass** がクラスター内に設定されます。

Red Hat Quay Operator によって使用されるデフォルトの **StorageClass** は、**Quay** および **Clair** データベースに必要な永続ボリューム要求をプロビジョニングします。これらの PVC はデータを永続的に保存するために使用され、Red Hat Quay レジストリーと Clair 脆弱性スキャナーが利用可能な状態を維持し、再起動や障害が発生してもその状態が維持されるようにします。

インストールを続行する前に、**Quay** および **Clair** コンポーネント用のストレージのシームレスなプロビジョニングを確保するために、クラスター内にデフォルトの **StorageClass** が設定されていることを確認してください。

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

以下の手順を使用して、OpenShift Container Platform OperatorHub から Red Hat Quay Operator をインストールします。

手順

1. OpenShift Container Platform コンソールを使用して、**Operators → OperatorHub** を選択します。
2. 検索ボックスに **Red Hat Quay** と入力し、Red Hat が提供する公式の Red Hat Quay Operator を選択します。機能、前提条件、デプロイメント情報について記載されている **Installation** ページに移動します。
3. **Install** 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。
    a. **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-3.11** を選択します。
    b. **インストールモード:**
        i. Red Hat Quay 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。Red Hat Quay Operator をクラスター全体に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。
        ii. 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。
            - **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. **Install** を選択します。

第3章 デプロイメント前に行う RED HAT QUAY の設定

Red Hat Quay Operator は、OpenShift Container Platform にデプロイされると、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が、セットアップをさらに制御する場合は、1 つ以上のコンポーネントを外部から管理できます。

次のパターンを使用して、管理対象外 Red Hat Quay コンポーネントを設定します。

手順

1. 適切な設定で **config.yaml** 設定ファイルを作成します。最小限の設定については、以下を参照してください。

```
$ touch config.yaml
```

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

2. 次のコマンドを入力して、設定ファイルを使用して **Secret** 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** を参照します。以下はその例です。

QuayRegistry YAML ファイルの例

```
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. 次のコマンドを入力し、**quayregistry.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 は、自動化を可能にするいくつかの設定オプションをサポートします。ユーザーはデプロイメント前にこれらのオプションを設定して、ユーザーインターフェイスとの対話の必要性を減らすことができます。

3.1.1. API による最初のユーザー作成の許可

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、**/api/v1/user/initialize** API を呼び出す必要があります。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 エンドポイントは認証を必要としません。

他のユーザーが作成されていない場合、ユーザーは Red Hat Quay のデプロイ後に API を使用して **quayadmin** などのユーザーを作成できます。詳細は、API を使用して最初のユーザーを作成する 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 への一般アクセスを許可するには、**BROWSER_API_CALLS_XHR_ONLY** 設定オプションを **false** に設定する必要があります。

3.1.3. スーパーユーザーの追加

Red Hat Quay をデプロイしたら、ユーザーを作成し、最初のユーザーに完全な権限を持つ管理者権限を付与できます。**SUPER_USER** 設定オブジェクトを使用すると、事前に完全な権限を設定できます。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後、**FEATURE_USER_CREATION** を **false** に設定することで、新しいユーザーを作成できる権限をスーパーユーザーグループに制限できます。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. Red Hat Quay 3.11 での新機能の有効化

新しい Red Hat Quay 3.11 の機能を使用するには、次の機能の一部またはすべてを有効にします。

```
# ...
FEATURE_UI_V2: true
FEATURE_UI_V2_REPO_SETTINGS: true
FEATURE_AUTO_PRUNE: true
ROBOTS_DISALLOW: false
# ...
```

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

ストレージを管理するのが Red Hat Quay Operator かユーザーかにかかわらず、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。

Red Hat Quay Operator にストレージの管理を任せたい場合は、マネージドストレージ セクションで、NooBaa および Red Hat OpenShift Data Foundations Operator のインストールと設定の詳細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、管理対象外ストレージ を参照してください。

3.2.1. 管理対象外ストレージの使用

このセクションでは、参考として管理対象外ストレージの設定例を示しています。オブジェクトストレージの設定方法について、詳しくは 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

Red Hat Quay デプロイメント用に AWS S3 ストレージを設定する場合は、次の例を使用します。

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

3.2.1.2. Google Cloud storage

Red Hat Quay デプロイメント用に Google Cloud ストレージを設定する場合は、次の例を使用します。

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

3.2.1.3. Microsoft Azure ストレージ

Red Hat Quay デプロイメント用に Microsoft Azure ストレージを設定する場合は、次の例を使用します。

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

(1) Microsoft 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 Microsoft 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

Red Hat Quay デプロイメント用に Ceph/RadosGW ストレージを設定する場合は、次の例を使用します。

```
DISTRIBUTED_STORAGE_CONFIG:
  radosGWStorage: #storage config name
    - RadosGWStorage #actual driver
    - access_key: access_key_here #parameters
      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: #must contain name of the storage config
    - radosGWStorage
```

3.2.1.5. Swift ストレージ:

Red Hat Quay デプロイメント用に Swift ストレージを設定する場合は、次の例を使用します。

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

3.2.1.6. NooBaa 管理対象外ストレージ

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. Red Hat Quay コンソールで、**Storage → Object Bucket Claims** に移動して、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)、および秘密鍵を含む 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) の情報を使用する **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. 管理対象外 NooBaa インスタンスの使用

以下の手順を使用して、Red Hat Quay デプロイメントに管理対象外 NooBaa インスタンスを使用します。

手順

1. Storage → Object Bucket Claims のコンソールで NooBaa Object Bucket Claim を作成します。
2. **Access Key**、**Bucket Name**、**Endpoint (hostname)**、および **Secret Key** 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。以下に例を示します。

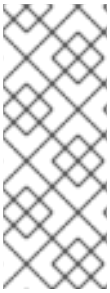
```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - RHOCSStorage
    - access_key: WmrXtSGk8B3nABCDEFGH
      bucket_name: my-noobaa-bucket-claim-8b844191-dc6c-444e-9ea4-87ece0abcdef
      hostname: s3.openshift-storage.svc.cluster.local
      is_secure: true
      port: "443"
      secret_key: X9P5SDGJtmSuHFCMSLMbdNCMfUABCDEFGH+C5QD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

3.2.3. マネージドストレージ

Red Hat Quay Operator に Red Hat Quay のオブジェクトストレージを管理させる場合、クラスターは **ObjectBucketClaim** API を通じてオブジェクトストレージを提供できる必要があります。Red Hat OpenShift Data Foundations Operator を使用する場合は、サポートされている 2 つのオプションを使用できます。

- ローカルの Kubernetes **PersistentVolume** ストレージでサポートされる Multi-Cloud Object Gateway のスタンドアロンインスタンス
  - 高可用性がない
  - Red Hat Quay サブスクリプションに含まれている
  - Red Hat OpenShift Data Foundation の別のサブスクリプションは不要
- スケールアウト Object Service と Ceph を備えた Red Hat OpenShift Data Foundation の実稼働環境用デプロイメント
  - 高可用性がある
  - Red Hat OpenShift Data Foundation の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。Red Hat OpenShift Data Foundation の実稼働環境用デプロイメントについて、詳しくは 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 Red Hat Quay 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。現在、Red Hat OpenShift Data Foundation ボリュームのサイズ変更は、Red Hat Quay Operator によって処理されません。詳細は、マネージドストレージのサイズ変更に関するセクションを参照してください。

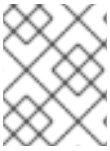
3.2.3.1. Red Hat Quay の Red Hat OpenShift Data Foundation Operator での Multicloud Object Gateway コンポーネントの使用

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 Red Hat 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 で管理される Red Hat Quay デプロイメントや、以下に示す Multicloud Object Gateway インスタンスの正確な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 Red Hat 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。

以下の手順を使用して、Local Storage Operator、Red Hat OpenShift Data Foundation をインストールし、スタンドアロンの Multicloud Object Gateway を作成して OpenShift Container Platform に Red Hat Quay をデプロイします。



注記

以下のドキュメントは、Red Hat OpenShift Data Foundation の公式ドキュメント と共通性 を共有します。

3.2.3.1.1. OpenShift Container Platform への Local Storage Operator のインストール

以下の手順を使用して、ローカルストレージデバイスに Red Hat OpenShift Data Foundation クラスターを作成する前に、**OperatorHub** から Local Storage Operator をインストールします。

1. **OpenShift Web コンソール** にログインします。
2. **Operators → OperatorHub** をクリックします。
3. 検索ボックスに **local storage** と入力して、Operators のリストから Local Storage Operator を見つけます。**Local Storage** をクリックします。
4. **Install** をクリックします。
5. Install Operator ページで、次のオプションを設定します。
   - 更新チャネルの場合は、**stable** を選択します。
   - インストールモードの場合は、**A specific namespace on the cluster**を選択します。
   - インストールされた namespace の場合は、**Operator recommended namespace openshift-local-storage** を選択します。
   - 更新の承認の場合は、**Automatic** を選択します。
6. **Install** をクリックします。

3.2.3.1.2. OpenShift Container Platform への Red Hat OpenShift Data Foundation のインストール

以下の手順を使用して、Red Hat OpenShift Data Foundation を OpenShift Container Platform にインストールします。

前提条件

- **cluster-admin** および Operator インストールのパーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift Container Platform クラスターにワーカーノードが少なくとも 3 つある。
- その他のリソース要件は、デプロイメントのプランニング ガイドを参照してください。

手順

1. **OpenShift Web コンソール** にログインします。
2. **Operators → OperatorHub** をクリックします。
3. 検索ボックスに **OpenShift Data Foundation** と入力します。**OpenShift Data Foundation** をクリックします。
4. **Install** をクリックします。

5. Install Operator ページで、次のオプションを設定します。
   - 更新チャネルの場合は、最新の安定したバージョンを選択します。
   - インストールモードの場合は、**A specific namespace on the cluster**を選択します。
   - インストールされた namespace の場合は、**Operator recommended Namespace: openshift-storage** を選択します。
   - 更新の承認の場合は、**Automatic** または **Manual** を選択します。
     Automatic (自動) 更新を選択した場合、Operator Lifecycle Manager (OLM) は介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。

     Manual 更新を選択した場合、OLM は更新要求を作成します。クラスター管理者は、Operator を新しいバージョンに更新できるように更新要求を手動で承認する必要があります。
   - コンソールプラグインの場合は、**Enable** を選択します。
6. **Install** をクリックします。
   Operator がインストールされると、**Web console update is available** メッセージを含むポップアップがユーザーインターフェイスに表示されます。このポップアップから **Refresh web console** をクリックして、反映するコンソールを変更します。
7. 次のセクション「OpenShift Container Platform UI を使用したスタンドアロン Multicloud Object Gateway の作成」に進み、Red Hat Quay の Multicloud Object Gateway コンポーネントを活用します。

3.2.3.1.3. OpenShift Container Platform UI を使用したスタンドアロン Multicloud Object Gateway の作成

スタンドアロン Multicloud Object Gateway を作成するには、次の手順を使用します。

前提条件

- Local Storage Operator がインストールされている。
- Red Hat OpenShift Data Foundation Operator がインストールされている。

手順

1. **OpenShift Web コンソール** で、**Operators → Installed Operators** をクリックし、インストールされた Operator をすべて表示します。
   namespace が **openshift-storage** であることを確認します。
2. **Create StorageSystem** をクリックします。
3. **Backing storage** ページで、以下を選択します。
   a. **Deployment type** の **Multicloud Object Gateway** を選択します。
   b. **Create a new StorageClass using the local storage devices**オプションを選択します。
   c. **Next** をクリックします。

注記

Local Storage Operator がまだインストールされていない場合は、インストールするように求められます。**Install** をクリックし、「OpenShift Container Platform への Local Storage Operator のインストール」で説明されている手順に従います。

4. **Create local volume set** ページで、以下の情報を入力します。

   a. **LocalVolumeSet** および **StorageClass** の名前を入力します。デフォルトで、ローカルボリュームセット名がストレージクラス名について表示されます。名前を変更できます。

   b. 以下のいずれかを選択します。

   - **すべてのノード上のディスク**
     すべてのノードにある選択したフィルターに一致する利用可能なディスクを使用します。

   - **選択したノード上のディスク**
     選択したノードにある選択したフィルターにのみ一致する利用可能なディスクを使用します。

   c. **Disk Type** の利用可能なリストから、**SSD/NVMe** を選択します。

   d. **Advanced** セクションを拡張し、以下のオプションを設定します。

ボリュームモード	デフォルトではファイルシステムが選択されています。Volume Mode で Filesystem が選択されていることを常に確認してください。
デバイスタイプ	ドロップダウンリストから 1 つ以上のデバイスタイプを選択します。
ディスクサイズ	デバイスの最小サイズ 100GB と、含める必要のあるデバイスの最大サイズを設定します。
ディスクの最大数の制限	これは、ノードで作成できる PV の最大数を示します。このフィールドが空のままの場合、PV は一致するノードで利用可能なすべてのディスクに作成されます。

   e. **Next** をクリックします。
   LocalVolumeSet の作成を確認するポップアップが表示されます。

   f. **Yes** をクリックして続行します。

5. **Capacity and nodes** ページで、以下を設定します。

   a. **Available raw capacity** には、ストレージクラスに関連付けられた割り当てられたすべてのディスクに基づいて容量の値が設定されます。これには少し時間がかかります。**Selected nodes** リストには、ストレージクラスに基づくノードが表示されます。

   b. **Next** をクリックして先に進みます。

6. オプション: **Connect to an external key management service** チェックボックスを選択します

b. オプション: **Connect to an external key management service** チェックボックスを選択します。これはクラスター全体の暗号化の場合はオプションになります。

   a. **Key Management Service Provider** ドロップダウンリストから、**Vault** または **Thales CipherTrust Manager (using KMIP)** を選択します。**Vault** を選択した場合は、次の手順に進みます。**Thales CipherTrust Manager (using KMIP)** を選択した場合は、手順 iii に進みます。

   b. **認証方法**を選択します。
   トークン認証方式の使用

      - Vault ('https://<hostname or ip>') サーバーの一意の **Connection Name**、ホストの **Address**、**Port** 番号および **Token** を入力します。

      - **Advanced Settings** をデプロイメントして、**Vault** 設定に基づいて追加の設定および証明書の詳細を入力します。

         - OpenShift Data Foundation 専用かつ特有のキーと値のシークレットパスを **Backend Path** に入力します。

         - オプション: **TLS Server Name** および **Vault Enterprise Namespace** を入力します。

         - それぞれの PEM でエンコードされた証明書ファイルをアップロードし、**CA 証明書**、**クライアント証明書**、および **クライアントの秘密鍵** を提供します。

         - **Save** をクリックして、手順 iv に進みます。
         Kubernetes 認証方式の使用

      - Vault ('https://<hostname or ip>') サーバーの一意の **Connection Name**、ホストの **Address**、**Port** 番号および **Role** 名を入力します。

      - **Advanced Settings** をデプロイメントして、Vault 設定に基づいて追加の設定および証明書の詳細を入力します。

         - Red Hat OpenShift Data Foundation 専用で一意の **Backend Path** にキーと値のシークレットパスを入力します。

         - 該当する場合は、**TLS Server Name** および **Authentication Path** を入力します。

         - PEM でエンコードされた、該当の証明書ファイルをアップロードし、**CA 証明書**、**クライアント証明書**、および **クライアントの秘密鍵** を提供します。

         - **Save** をクリックして、手順 iv に進みます。

   c. **Thales CipherTrust Manager (using KMIP)** を KMS プロバイダーとして使用するには、次の手順に従います。

      i. プロジェクト内のキー管理サービスの一意の **Connection Name** を入力します。

      ii. **Address** および **Port** セクションで、Thales CipherTrust Manager の IP と、KMIP インターフェイスが有効になっているポートを入力します。以下に例を示します。

         - **Address**: 123.34.3.2

         - **Port**: 5696

      iii. **クライアント証明書**、**CA 証明書**、および **クライアント秘密鍵** をアップロードします。

iv. StorageClass 暗号化が有効になっている場合は、上記で生成された暗号化および復号化に使用する一意の識別子を入力します。

v. **TLS Server** フィールドはオプションであり、KMIP エンドポイントの DNS エントリーがない場合に使用します。たとえば、**kmip_all_<port>.ciphertrustmanager.local** などです。

d. **Network** を選択します。

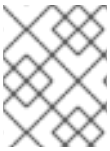
e. **Next** をクリックします。

7. **Review and create** ページで、設定の詳細を確認します。設定を変更するには、**Back** をクリックします。

8. **Create StorageSystem** をクリックします。

3.2.3.1.4. CLI を使用して Multicloud Object Gateway を作成する

以下の手順に従って、Red Hat OpenShift Data Foundation (以前の OpenShift Container Storage) Operator をインストールし、単一インスタンスの Multi-Cloud Gateway サービスを設定します。



注記

次の設定は、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。

手順

1. **OpenShift Web コンソール** で、**Operators → OperatorHub** を選択します。
2. **Red Hat OpenShift Data Foundation** を検索し、**Install** を選択します。
3. すべてのデフォルトのオプションを受け入れて、**Install** を選択します。
4. **Status** 列を表示して Operator がインストールされたことを確認します。この列には **Succeeded** とマークが付けられています。

警告

Red Hat OpenShift Data Foundation Operator のインストールが完了すると、ストレージシステムを作成するように求められます。この指示には従わないでください。代わりに、次の手順に従って NooBaa オブジェクトストレージを作成します。

5. マシン上で、次の情報を含む **noobaa.yaml** という名前のファイルを作成します。

```
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** の単一インスタンスデプロイメントが作成されます。

6. 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
```

出力例

```
noobaa.noobaa.io/noobaa created
```

7. 数分後に、**Multi-cloud Object Gateway** プロビジョニングが完了するはずです。次のコマンドを入力してステータスを確認できます。

```
$ oc get -n openshift-storage noobaas noobaa -w
```

出力例

```
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
```

8. 次の YAML ファイルを **noobaa-pv-backing-store.yaml** で名前作成して、ゲートウェイのバッキングストアを設定します。

```
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) 要求された **PersistentVolumes** に使用する **StorageClass**。クラスターのデフォルトを使用するには、このプロパティーを削除します。

9. 以下のコマンドを入力して設定を適用します。

```
$ oc create -f noobaa-pv-backing-store.yaml
```

出力例

```
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。Red Hat 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。

10. 以下のコマンドを実行して、**PersistentVolume** バッキングストアを、Red Hat Quay Operator が発行するすべての **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
```

第4章 トラフィック受信の設定

4.1. SSL/TLS とルートの設定

OpenShift Container Platform の **エッジターミネーション** ルートのサポートが、新しい管理対象コンポーネントの **tls** によって追加されました。これにより、**route** コンポーネントが SSL/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。

注記

- **tls** が管理対象の場合、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。
- **tls** が管理対象外の場合、ユーザーが指定したキーと証明書のペアがルートに挿入されます。

ssl.cert と **ssl.key** が別の永続的なシークレットに移動し、キーと証明書のペアが調整のたびに再生成されなくなりました。キーと証明書のペアは **edge** ルートとしてフォーマットされ、**Quay** コンテナー内の同じディレクトリーにマウントされます。

SSL/TLS とルートを設定する場合は複数の置換が可能ですが、次のルールが適用されます。

- SSL/TLS が **managed** になっている場合は、ルートも **managed** にする必要があります。
- SSL/TLS が **unmanaged** の場合は、設定バンドルで証明書を直接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4.1 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	証明書が提供されるか	結果
独自のロードバランサーが TLS を処理する	管理対象	管理対象	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	管理対象	管理対象外	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	管理対象外	管理対象外	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。

4.1.1. SSL/TLS 証明書とキーのペアを使用した設定バンドルシークレットの作成

次の手順を使用して、独自の SSL/TLS 証明書とキーペアを含む設定バンドルシークレットを作成します。

手順

- 次のコマンドを入力して、独自の SSL/TLS 証明書とキーペアを含む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 ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
```

第5章 OPENSHIFT CONTAINER PLATFORM 上の管理対象コンポーネントのリソースの設定

以下のコンポーネントに実行中の Pod がある場合は、Red Hat Quay on OpenShift Container Platform でリソースを手動調整できます。

- **quay**
- **clair**
- **mirroring**
- **clairpostgres**
- **postgres**

この機能により、ユーザーはより小規模なテストクラスターを実行したり、**Quay** Pod の機能が部分的に低下するのを避けるために事前にさらに多くのリソースを要求したりできるようになります。Kubernetes リソース単位 に応じて制限やリクエストを設定できます。

次のコンポーネントは、最小要件よりも低く設定しないでください。これにより、デプロイメントに問題が発生し、場合によっては Pod のデプロイメントが失敗する可能性があります。

- **quay**: 最低 6 GB、2vCPU
- **Clair**: 2 GB のメモリー、2 つの vCPU を推奨
- **clairpostgres**: 最低 200MB

リソース要求は、OpenShift Container Platform UI で設定することも、**QuayRegistry** YAML を直接更新して設定することもできます。

重要

これらのコンポーネントに設定されているデフォルト値は推奨値です。リソース要求の設定が高すぎるか低すぎると、それぞれリソースの使用効率やパフォーマンスが低する可能性があります。

5.1. OPENSHIFT CONTAINER PLATFORM UI を使用したリソース要求の設定

OpenShift Container Platform UI を使用してリソースを設定するには、次の手順に従います。

手順

1. OpenShift Container Platform 開発者コンソールで、**Operator → Installed Operators → Red Hat Quay** をクリックします。
2. **QuayRegistry** をクリックします。
3. レジストリーの名前 (例: **example-registry**) をクリックします。
4. **YAML** をクリックします。
5. **spec.components** フィールドでは、**.overrides.resources.limits** および

overrides.resources.requests フィールドに値を設定することで、**quay**、**clair**、ミラーリング **clairpostgres**、および **postgres** リソースのリソースをオーバーライドできます。以下に例を示します。

```
spec:
  components:
    - kind: clair
      managed: true
      overrides:
        resources:
          limits:
            cpu: "5"     # Limiting to 5 CPU (equivalent to 5000m or 5000 millicpu)
            memory: "18Gi"  # Limiting to 18 Gibibytes of memory
          requests:
            cpu: "4"     # Requesting 4 CPU
            memory: "4Gi"   # Requesting 4 Gibibytes of memory
    - kind: postgres
      managed: true
      overrides:
        resources:
          limits: {} (1)
          requests:
            cpu: "700m"   # Requesting 700 millicpu or 0.7 CPU
            memory: "4Gi"   # Requesting 4 Gibibytes of memory
    - kind: mirror
      managed: true
      overrides:
        resources:
          limits: (2)
          requests:
            cpu: "800m"   # Requesting 800 millicpu or 0.8 CPU
            memory: "1Gi"   # Requesting 1 Gibibyte of memory
    - kind: quay
      managed: true
      overrides:
        resources:
          limits:
            cpu: "4"    # Limiting to 4 CPU
            memory: "10Gi"   # Limiting to 10 Gibibytes of memory
          requests:
            cpu: "4Gi"   # Requesting 4 CPU
            memory: "10Gi"   # Requesting 10 Gibi of memory
    - kind: clairpostgres
      managed: true
      overrides:
        resources:
          limits:
            cpu: "800m"   # Limiting to 800 millicpu or 0.8 CPU
            memory: "3Gi"   # Limiting to 3 Gibibytes of memory
          requests: {}
```

(1) **limits** または **requests** フィールドを **{}** に設定すると、これらのリソースのデフォルト値が使用されます。

(2) **limits** または **requests** フィールドを空のままにしておくと、これらのリソースに制限は設定されません。

6. オプション: デフォルト値をオーバーライドした後は、**{}** で示される null を使用して、コンポーネントに割り当てられたデフォルト値にリセットできます。以下に例を示します。

```
# ...
  - kind: clairpostgres
    managed: true
    overrides:
      resources:
        limits: {}
        requests: {}
# ...
```

5.2. QUAYREGISTRY YAML の編集によるリソースリクエストの設定

レジストリーをデプロイした後で、Red Hat Quay を再設定してリソース要求を設定できます。これは、**QuayRegistry** YAML ファイルを直接編集し、レジストリーを再デプロイすることで実行できます。

手順

1. オプション: **QuayRegistry** YAML ファイルのローカルコピーがない場合は、次のコマンドを入力して取得します。

```
$ oc get quayregistry <registry_name> -n <namespace> -o yaml > quayregistry.yaml
```

2. この手順のステップ 1 で作成した **quayregistry.yaml** を開き、必要な変更を加えます。以下に例を示します。

```
  - kind: quay
    managed: true
    overrides:
      resources:
        limits: {}
        requests:
          cpu: "0.7"   # Requesting 0.7 CPU (equivalent to 500m or 500 millicpu)
          memory: "512Mi"   # Requesting 512 Mebibytes of memory
```

3. 変更を保存します。

4. 次のコマンドを実行して、更新された設定を使用して Red Hat Quay レジストリーを適用します。

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

出力例

```
quayregistry.quay.redhat.com/example-registry replaced
```

第6章 データベースの設定

6.1. 既存の POSTGRESQL データベースの使用

外部で管理されている PostgreSQL データベースを使用している場合、デプロイメントを成功させるには、**pg_trgm** 拡張機能を手動で有効にする必要があります。

既存の PostgreSQL データベースをデプロイするには、次の手順を使用します。

手順

1. 必要なデータベースフィールドを含む **config.yaml** ファイルを作成します。以下に例を示します。

   config.yaml ファイルの例:

   ```
   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 **Secret** 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 **unmanaged** とマークし、作成した **Secret** を参照する **QuayRegistry.yaml** ファイルを作成します。以下に例を示します。

   quayregistry.yaml ファイルの例

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

次のステップ

- 次のセクションに進み、レジストリーをデプロイします。

6.1.1. 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を説明します。

6.1.1.1. 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** 設定フィールドを説明します。

表6.1 データベース URI

フィールド	型	説明
DB_URI (必須)	String	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**

6.1.1.2. 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を説明します。

表6.2 データベース接続引数

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

6.1.1.2.1. PostgreSQL SSL/TLS 接続引数

SSL/TLS 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。次の例は、PostgreSQL SSL/TLS 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/TLS TCP/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6.3 SSL/TLS オプション

モード	説明
disable	この設定では、非 SSL/TLS 接続のみが試行されます。
allow	設定では、まず非 SSL/TLS 接続が試行されます。失敗すると、SSL/TLS 接続が試行されます。
prefer (デフォルト)	設定では、まず SSL/TLS 接続が試行されます。失敗すると、非 SSL/TLS 接続が試行されます。
require	設定では SSL/TLS 接続のみが試行されます。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が検証されます。
verify-ca	設定では SSL/TLS 接続のみが試行され、サーバー証明書が信頼された認証局 (CA) によって発行されたかどうかが検証されます。
verify-full	SSL/TLS 接続のみを試行し、サーバー証明書が信頼できる CA によって発行されていること、および要求されたサーバーのホスト名が証明書内のホスト名と一致することが確認されます。

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

6.1.1.2.2. MySQL SSL/TLS 接続引数

次の例は、MySQL SSL/TLS 設定のサンプルを示しています。

```
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。

6.1.2. マネージド PostgreSQL データベースの使用

Red Hat Quay 3.9 では、データベースが Red Hat Quay Operator によって管理されている場合、Red Hat Quay 3.8 から 3.9 に更新すると自動的に PostgreSQL 10 から PostgreSQL 13 にアップグレードされます。

重要

- マネージドデータベースを使用しているユーザーは、PostgreSQL データベースを 10 から 13 にアップグレードする必要があります。
- Red Hat Quay および Clair データベースが Operator によって管理されている場合、3.9.0 のアップグレードを成功させるには、各コンポーネントのデータベースのアップグレードが成功する必要があります。いずれかのデータベースのアップグレードが失敗すると、Red Hat Quay のバージョン全体のアップグレードが失敗します。この動作は想定されています。

Red Hat Quay Operator により PostgreSQL デプロイメントが PostgreSQL 10 から 13 にアップグレードされることを望まない場合は、**quayregistry.yaml** ファイルで PostgreSQL パラメーターを **manage: false** に設定する必要があります。データベースを管理対象外に設定する方法について、詳しくは既存 Postgres データベースの使用 を参照してください。

重要

- PostgreSQL 13 にアップグレードすることが強く推奨されます。PostgreSQL 10 は 2022 年 11 月 10 日に最終リリースとなり、サポートされなくなりました。詳細は、PostgreSQL のバージョン管理ポリシー を参照してください。

PostgreSQL データベースと Red Hat Enterprise Linux (RHEL) システムのバージョンを一致させるには、RHEL 8 の場合は PostgreSQL の RHEL 8 バージョンへの移行、RHEL 9 の場合は PostgreSQL の RHEL 9 バージョンへの移行 を参照してください。

Red Hat Quay 3.8 → 3.9 の手順の詳細は、Red Hat Quay Operator のアップグレードの概要 を参照してください。

6.1.2.1. PostgreSQL データベースの推奨事項

Red Hat Quay チームは、PostgreSQL データベースを管理するために以下を推奨します。

- Postgres SQL 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。Red Hat Quay Operator は現在、PostgreSQL データベースがバックアップされていることを保証していません。
- バックアップからの PostgreSQL データベースの復元は、PostgreSQL のツールと手順を使用して行う必要があります。データベースを復元している間は、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容量は、Red Hat Quay Operator によって 50 GiB が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。現在、データベースボリュームのサイズ変更は Red Hat Quay Operator で処理されません。

6.2. 外部 REDIS の設定

このセクションの内容に沿って、外部 Redis デプロイメントをセットアップします。

6.2.1. 管理対象外 Redis データベースの使用

外部 Redis データベースをセットアップするには、次の手順を実行します。

手順

1. 次の Redis フィールドを使用して **config.yaml** ファイルを作成します。

```
# ...
BUILDLOGS_REDIS:
    host: <quay-server.example.com>
    port: 6379
    ssl: false
# ...
USER_EVENTS_REDIS:
    host: <quay-server.example.com>
    port: 6379
    ssl: false
# ...
```

2. 次のコマンドを入力して、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 **unmanaged** に設定し、作成されたシークレットを参照する **quayregistry.yaml** ファイルを作成します。

```
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. Red Hat Quay レジストリーをデプロイします。

関連情報

Redis 設定フィールド

6.2.2. 管理対象外 Horizontal Pod Autoscaler の使用

Horizontal Pod Autoscaler (HPA) は **Clair**、**Quay**、および **Mirror** Pod に含まれるようになり、負荷の急増時に自動的にスケーリングされるようになりました。

HPA はデフォルトで管理対象となるように設定されているため、**Clair**、**Quay**、および **Mirror** の Pod 数は 2 に設定されています。これにより、Red Hat Quay Operator による Quay の更新または再設定時、またはイベントの再スケジューリング中のダウンタイムを容易に回避できます。

6.2.2.1. Horizontal Pod Autoscaler の無効化

自動スケーリングを無効にするか、独自の **HorizontalPodAutoscaler** を作成するには、**QuayRegistry** インスタンスでコンポーネントを **unamanged** として指定します。以下に例を示します。

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

6.2.3. Route コンポーネントの無効化

Red Hat Quay Operator がルートを作成できないようにするには、次の手順を使用します。

手順

1. コンポーネントを **quayregistry.yaml** ファイルで **managed: false** と設定します。

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

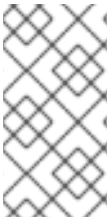
2. **config.yaml** ファイルを編集して、Red Hat Quay が SSL/TLS を処理するように指定します。以下に例を示します。

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

管理対象外ルートを正しく設定しないと、次のエラーが返されます。

```
{
  {
    "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

デフォルトルートを無効にするということは、Red Hat Quay インスタンスにアクセスするために **Route**、**Service**、または **Ingress** を作成する必要があることを意味します。さらに、使用する DNS は Red Hat Quay 設定の **SERVER_HOSTNAME** と一致する必要があります。

6.2.4. モニタリングコンポーネントの無効化

Red Hat Quay Operator を単一の namaspace にインストールすると、モニタリングコンポーネントは自動的に **managed: false** に設定されます。監視を明示的に無効にするには、次の参照を使用してください。

管理対象外のモニタリング

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

このシナリオでモニタリングを有効にするには、Red Hat Quay Operator が単一の namespace にインストールされている場合の監視の有効化 を参照してください。

6.2.5. ミラーリングコンポーネントの無効化

ミラーリングを無効にするには、次の YAML 設定を使用します。

管理対象外ミラーリングの YAML 設定の例

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

第7章 OPERATOR を使用した RED HAT QUAY のデプロイ

Red Hat Quay は、コマンドラインインターフェイスを使用して、または OpenShift Container Platform コンソールから OpenShift Container Platform にデプロイできます。手順は基本的に同じです。

7.1. コマンドラインからの RED HAT QUAY のデプロイ

コマンドラインインターフェイス (CLI) を使用して Red Hat Quay をデプロイするには、次の手順を実行します。

前提条件

- CLI を使用して OpenShift Container Platform にログインしている。

手順

1. 次のコマンドを入力して、namespace (例: **quay-enterprise)** を作成します。

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

2. オプション: Red Hat Quay デプロイメントで何かを事前に設定する場合は、設定バンドルの **Secret** を作成します。

```
$ oc create secret generic quay-enterprise-config-bundle --from-file=config-
bundle.tar.gz=/path/to/config-bundle.tar.gz
```

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. オプション: 一部のコンポーネントを管理対象外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。最小デプロイメントは次の例のようになります。

管理対象外コンポーネントを含む quayregistry.yaml の例

```
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. オプション: 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。

設定バンドルを含む quayregistry.yaml の例

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

d. オプション: プロキシーを設定している場合は、Red Hat 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。

プロキシーが設定された quayregistry.yaml の例

```
  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. 次のコマンドを入力して、指定の namespace に **QuayRegistry** を作成します。

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

5. 次のコマンドを入力して、**status.registryEndpoint** がいつ設定されるかを確認します。

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

関連情報

- Red Hat Quay のデプロイの進行状況を追跡する方法の詳細は、デプロイメントプロセスの監視およびデバッグ を参照してください。

7.1.1. 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。

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。

手順

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。

1. Red Hat Quay 設定ファイルを開き、以下の設定フィールドを更新します。

```
FEATURE_USER_INITIALIZE: true
SUPER_USERS:
    - quayadmin
```

2. 次のコマンドを入力して、Red Hat Quay サービスを停止します。

```
$ sudo podman stop quay
```

3. 次のコマンドを入力して、Red Hat Quay サービスを開始します。

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

4. 次の **CURL** コマンドを実行して、ユーザー名、パスワード、電子メール、およびアクセストークンを使用して新しいユーザーを生成します。

```
$ curl -X POST -k  http://quay-server.example.com/api/v1/user/initialize --header 'Content-
Type: application/json' --data '{ "username": "quayadmin", "password":"quaypass12345",
"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"} #
gitleaks:allow
```

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."}
```

5. 以下のコマンドを入力して、Red Hat Quay デプロイメントにログインします。

```
$ sudo podman login -u quayadmin -p quaypass12345 http://quay-server.example.com --tls-
verify=false
```

出力例

```
Login Succeeded!
```

7.1.2. コマンドラインで作成されたコンポーネントの表示

デプロイされた Red Hat Quay コンポーネントを表示するには、次の手順を使用します。

前提条件

- Red Hat Quay を OpenShift Container Platform にデプロイしている。

手順

1. 次のコマンドを入力して、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
```

出力例

```
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-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
```

7.1.3. Horizontal Pod 自動スケーリング

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。

- Red Hat Quay アプリケーション自体で使用する 2 つの Pod (**example-registry-quay-app-*`**)
- Red Hat Quay ロギングに使用する 1 つの Redis Pod (**example-registry-quay-redis-***)
- Red Hat Quay がメタデータストレージに使用する PostgreSQL の 1 つのデータベース Pod (**example-registry-quay-database-***)
- 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-***)

Horizontal PPod 自動スケーリングはデフォルトで **managed** に設定され、Quay、Clair、およびリポジトリーミラーリングの Pod 数は 2 に設定されます。これにより、Red Hat Quay Operator による Red Hat Quay の更新または再設定時、またはイベントの再スケジューリング中のダウンタイムを容易に回避できます。以下のコマンドを入力して、HPA オブジェクトに関する情報を表示できます。

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

出力例

```
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
```

関連情報

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。

7.1.4. 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。

手順

1. 次のコマンドを入力して、デプロイメントのステータスを確認します。

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

出力例

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: ""
```

2. **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
```

出力例

```
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-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
```

3. デプロイメントが進行中、**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
    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
```

4. 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は表示されません。

```
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
  currentVersion: {producty}
  lastUpdated: 2021-09-14 10:52:46.104181633 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
  unhealthyComponents: {}
```

7.2. OPENSHIFT CONTAINER PLATFORM 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。これはレジストリーのホスト名に影響します。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** リスト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。

7.2.1. Red Hat Quay UI を使用した最初のユーザーの作成

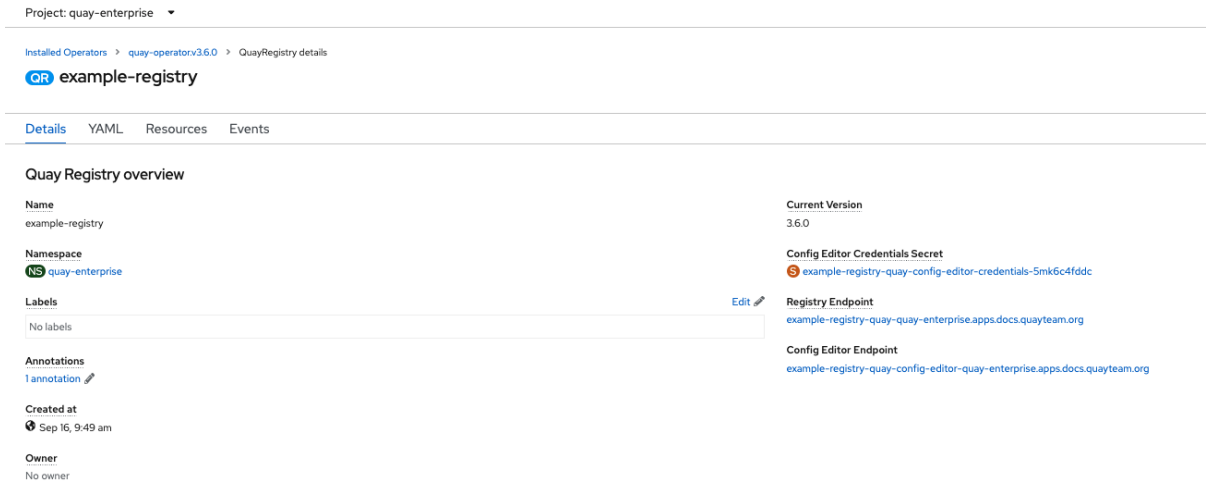
Red Hat Quay UI で最初のユーザーを作成するには、次の手順を実行します。

注記

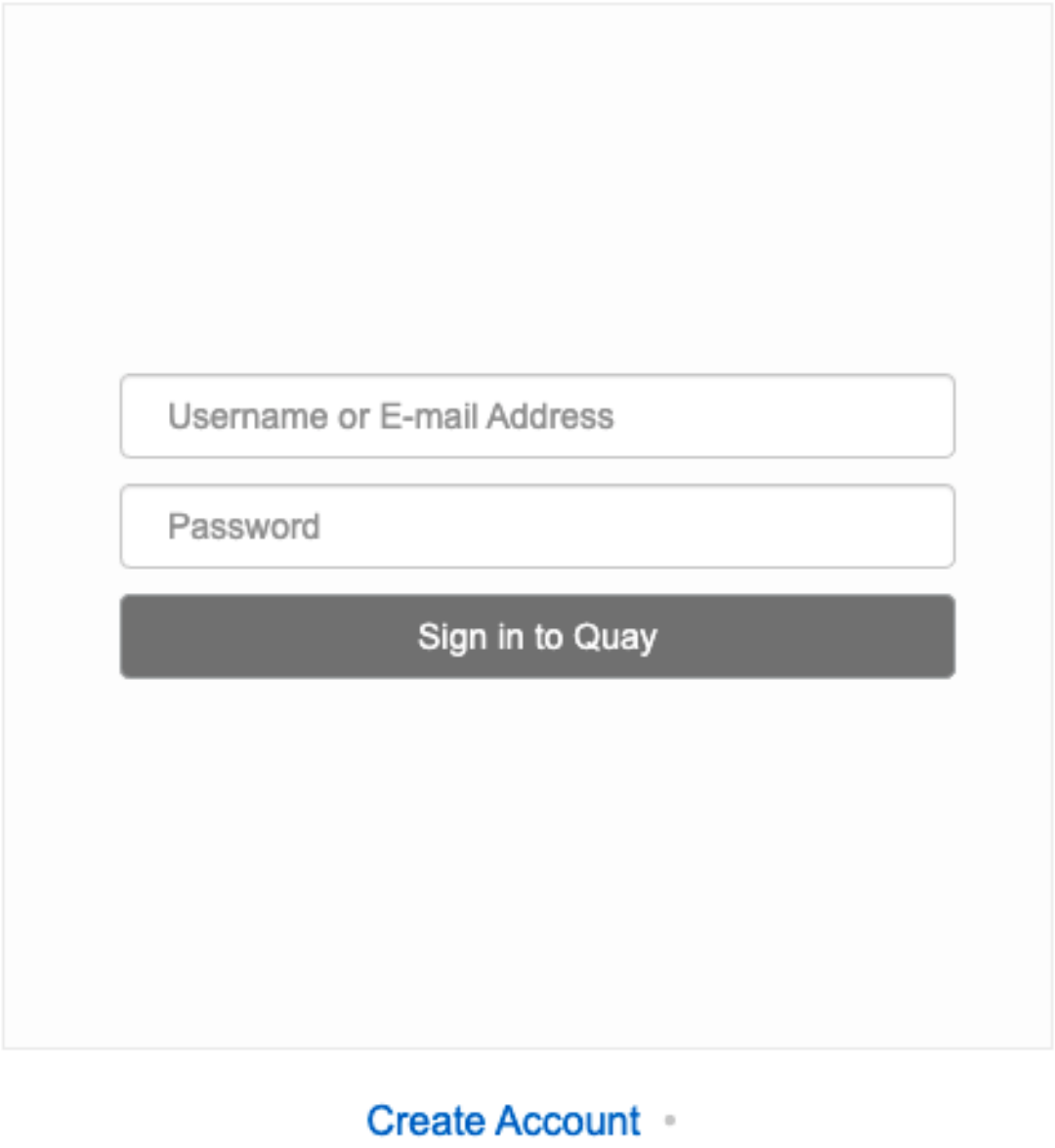
この手順では、**FEATURE_USER_CREATION** 設定オプションが **false** に設定されていることを前提としています。**false** の場合は、UI 上で **Create Account** 機能が無効になり、API を使用して最初のユーザーを作成する必要があります。

手順

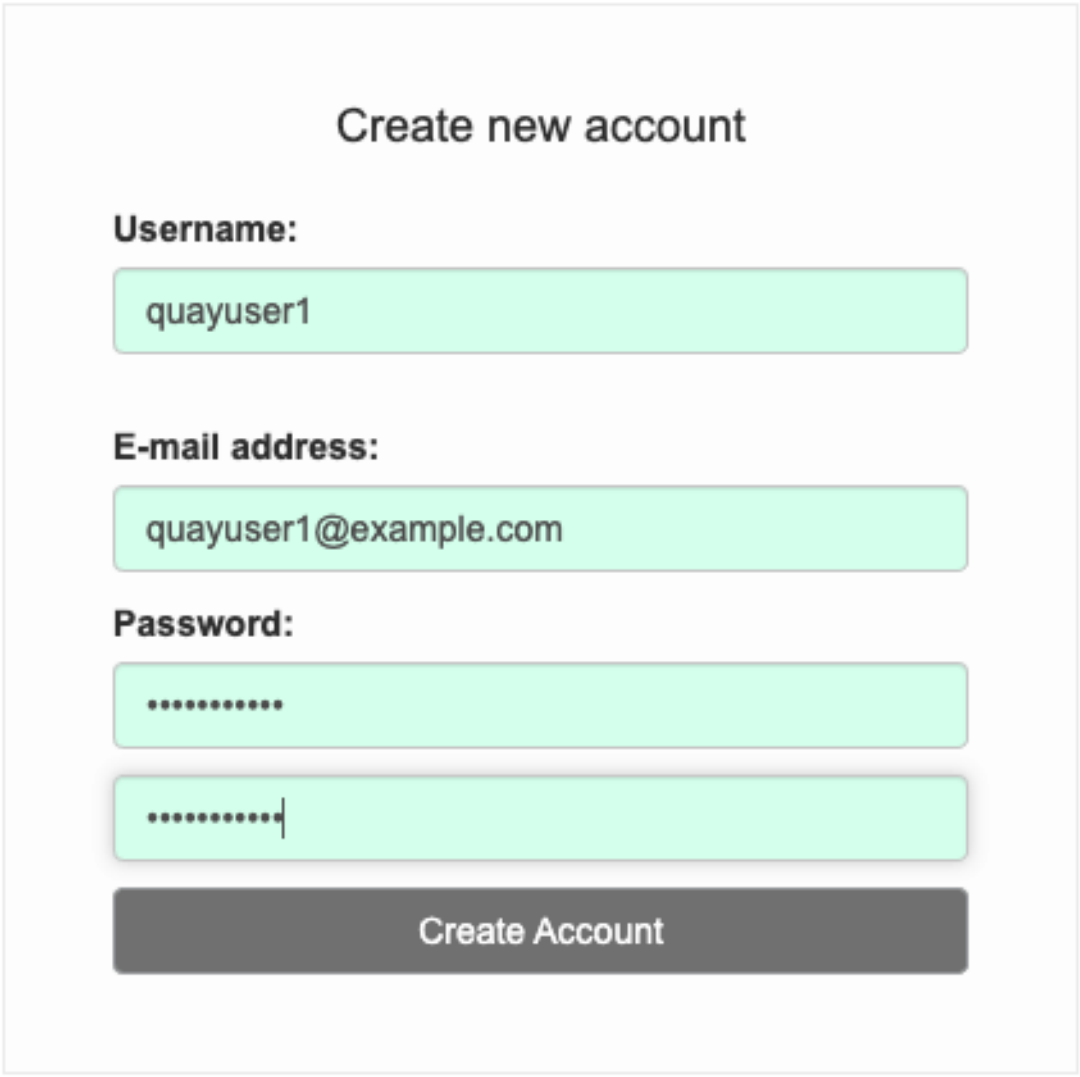
1. OpenShift Container Platform コンソールで、適切な namespace / プロジェクトを使用して **Operators → Installed Operators** に移動します。
2. 新しくインストールされた **QuayRegistry** オブジェクトをクリックして詳細を表示します。以下に例を示します。



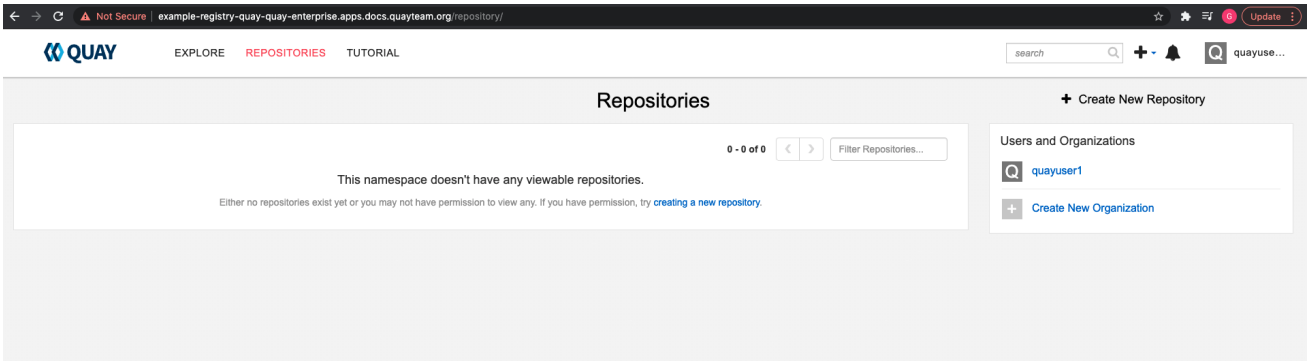
3. **Registry Endpoint** に値を設定したら、ブラウザーでこの URL に移動します。
4. Red Hat Quay レジストリー UI で **Create Account** を選択し、ユーザーを作成します。以下に例を示します。



5. **Username**、**Password**、**Email** の詳細を入力し、**Create Account** をクリックします。以下に例を示します。



最初のユーザーを作成すると、自動的に Red Hat Quay レジストリーにログインされます。以下に例を示します。



第8章 QUAYREGISTRY オブジェクトのステータスの表示

特定の Red Hat Quay デプロイメントのライフサイクル可観測性は、対応する **QuayRegistry** オブジェクトの **status** セクションで報告されます。Red Hat Quay Operator はこのセクションを常に更新します。Red Hat Quay または管理対象の依存関係において問題や状態の変更がないかを確認する場合に、こちらの場所を最初に探すようにしてください。

8.1. レジストリーエンドポイントの表示

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

8.2. 使用中の RED HAT QUAY のバージョンの表示

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

8.3. RED HAT QUAY デプロイメントの条件の表示

特定の条件は **status.conditions** で報告されます。

第9章 OPENSHIFT CONTAINER PLATFORM での RED HAT QUAY のカスタマイズ

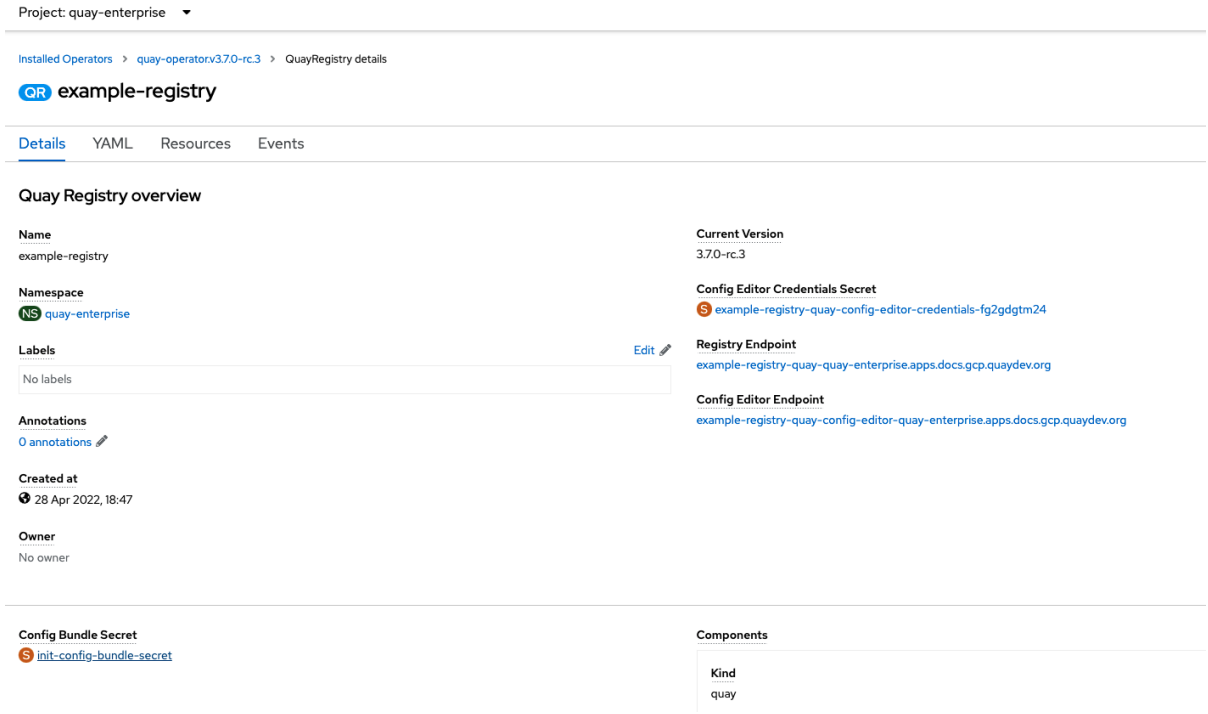
デプロイメント後に、Red Hat Quay 設定バンドルシークレット **spec.configBundleSecret** を編集して Red Hat Quay アプリケーションをカスタマイズできます。**QuayRegistry** リソースの **spec.components** オブジェクトで、コンポーネントの管理ステータスを変更したり、一部のコンポーネントのリソース要求を設定したりすることもできます。

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

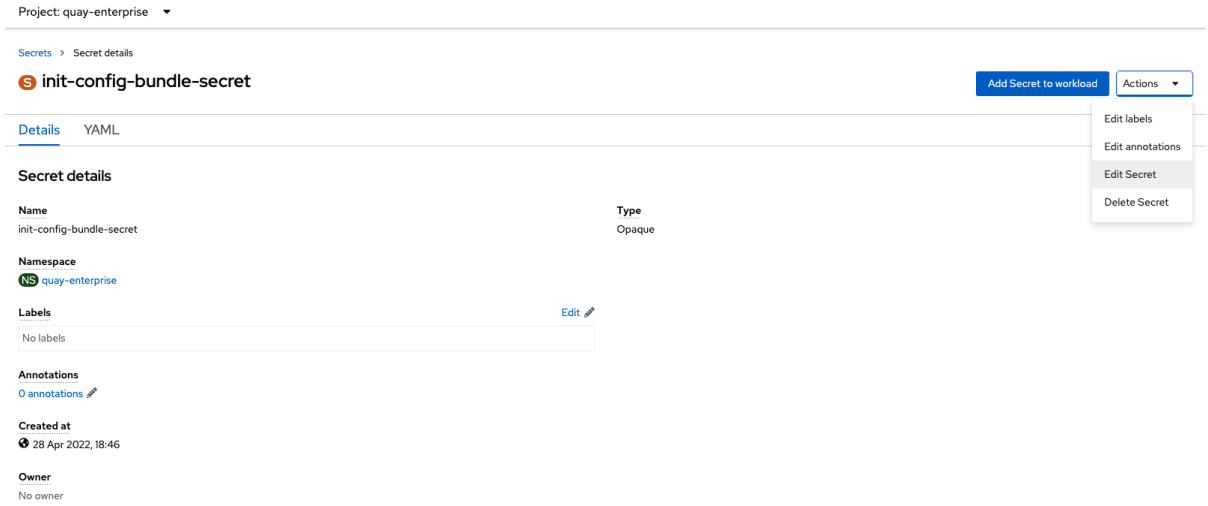
OpenShift Container Platform コンソールで設定バンドルシークレットを編集するには、次の手順を実行します。

手順

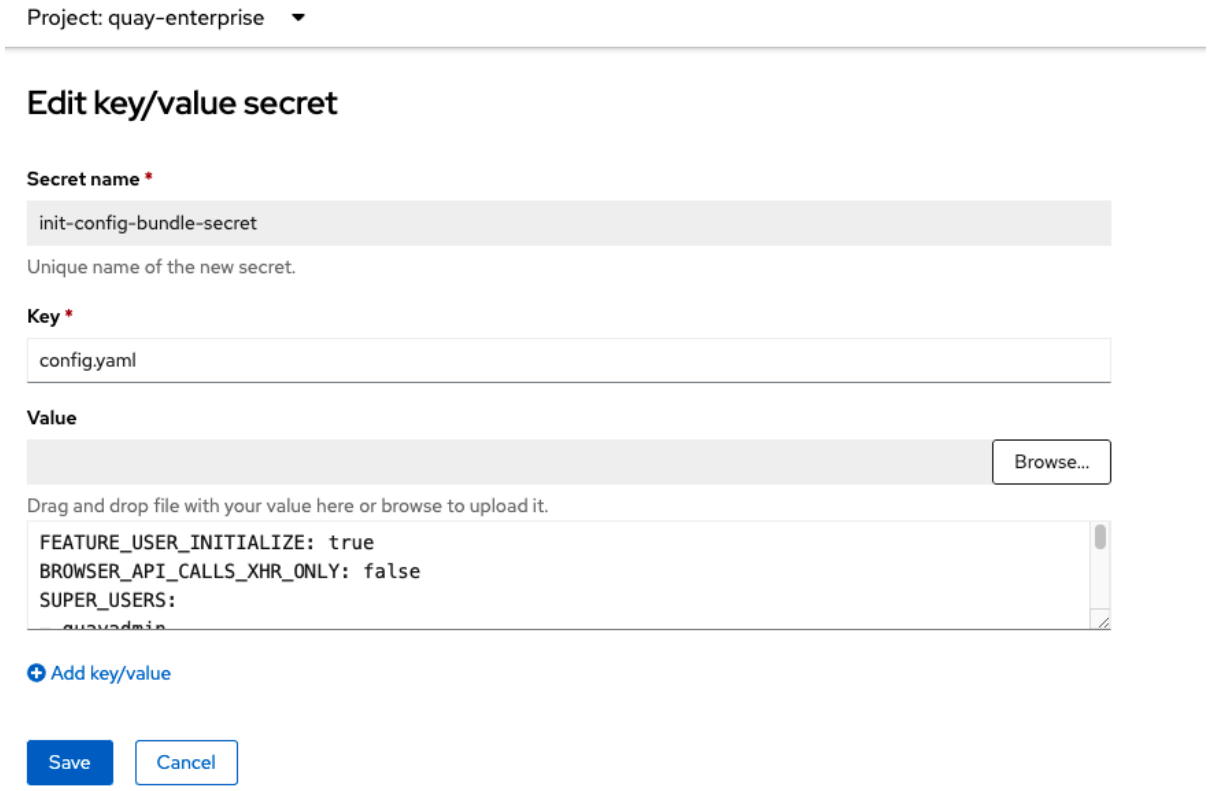
1. Red Hat Quay Registry の概要画面で、**Config Bundle Secret** のリンクをクリックします。



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. 設定を変更して保存します



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

9.2. QUAYREGISTRY エンドポイントおよびシークレットの決定

次の手順を実行して、**QuayRegistry** エンドポイントとシークレットを見つけます。

手順

1. 現在のエンドポイントとシークレットを見つけるには、次のコマンドを入力し、**oc description quayregistry** または **oc get quayregistry -o yaml** を使用して **QuayRegistry** リソースを調べます。

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

出力例

```
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 (1)
status:
  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
(2)
```

1. **config.yaml** ファイルと SSL/TLS 証明書を含む設定バンドルのシークレット。
2. 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。

9.3. 既存設定のダウンロード

以下は、各種ストラテジーを使用して既存の設定をダウンロードするための手順です。

9.3.1. 設定バンドルのシークレットを使用して既存の設定をダウンロード

設定バンドルのシークレットを使用して、既存の設定をダウンロードできます。

手順

1. 次のコマンドを入力して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="
}
```

2. 次のコマンドを入力してデータをデコードします。

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

出力例

```
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
```

9.4. 設定バンドルを使用したカスタム SSL/TLS 証明書の設定

カスタム SSL/TLS 証明書は、初期デプロイメントの前、または Red Hat Quay を OpenShift Container Platform にデプロイした後に設定できます。これは、設定バンドルのシークレットを作成または更新することで実行できます。

既存のデプロイメントに証明書を追加する場合は、設定を変更しない場合でも、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。

カスタム SSL/TLS 証明書を追加するには、次の手順を実行します。

手順

1. **QuayRegistry** YAML ファイルで、**kind: tls** を **manage:false** に設定します。以下はその例です。

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

2. **Events** ページには、適切な設定が行われるまで変更がブロックされていることが表示されます。以下に例を示します。

```
- 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'
```

3. 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-----
```

b. 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
```

..

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

a. 次のコマンドを入力して、**config.yaml** ファイルと **custom-ssl.crt** ファイルを含む汎用 **Secret** オブジェクトを作成します。

```
$ 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
```

b. 作成された **Secret** を参照する **QuayRegistry** YAML ファイルを作成または更新します。以下はその例です。

QuayRegistry YAML ファイルの例

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

c. 次のコマンドを入力し、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
```

次のステップ

- Red Hat Quay の機能