Red Hat Quay 3.6

使用 Quay Operator 在 OpenShift 上部署 Red Hat Quay

使用 Quay Operator 在 OpenShift 上部署 Red Hat Quay

Last Updated: 2023-02-28

Red Hat Quay 3.6 使用 Quay Operator 在 OpenShift 上部署 Red Hat Quay

使用 Quay Operator 在 OpenShift 上部署 Red Hat Quay

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

使用 Red Hat Quay Operator 在 OpenShift Cluster 上部署 Red Hat Quay

目录

前言

Red Hat Quay 是一个企业级容器 registry。使用 Red Hat Quay 构建和存储容器镜像，然后将其在整个企业中进行部署。

Red Hat Quay Operator 提供了在 OpenShift 集群上部署和管理 Red Hat Quay 的简单方法。

从 Red Hat Quay 3.4.0 开始，Operator 已完全重写，以便提供更好的体验，并支持更多第 2 天操作。因此，新的 Operator 更易于使用，并且更意见。与之前 Operator 版本相关的主要区别如下：

- **QuayEcosystem** 自定义资源已被 **QuayRegistry** 自定义资源替代
- 默认安装选项生成完全支持的 Quay 环境，其中包含支持所有受管依赖关系（数据库、缓存、对象存储等），供生产环境使用（某些组件可能不可用）
- Quay 配置提供了一个强大的验证库，供 Quay 应用和配置工具共享，以实现一致性
- 现在，可以使用 **ObjectBucketClaim** Kubernetes API 管理对象存储（Red Hat OpenShift Data Foundation 可用于在 OpenShift 上提供此 API 的支持的实现）
- 自定义部署 Pod 用于测试和开发场景的容器镜像

第 1 章 RED HAT QUAY OPERATOR 简介

本文档概述了在 OpenShift 中使用 Red Hat Quay Operator 配置、部署、管理和升级 Red Hat Quay 的步骤。

它显示如何：

- 安装 Red Hat Quay Operator
- 配置对象存储，可以是受管或非受管
- 配置其他非受管组件（如果需要），包括数据库、Redis、路由、TLS 等。
- 使用 Operator 在 OpenShift 上部署 Red Hat Quay registry
- 使用 Operator 支持的高级功能
- 通过升级 Operator 来升级 registry

1.1. QUAYREGISTRY API

Quay Operator 提供 **QuayRegistry** 自定义资源 API，以声明性地管理集群中的 **Quay** 容器 registry。使用 OpenShift UI 或命令行工具与 API 交互。

- 创建 **QuayRegistry** 将会导致 Operator 部署和配置在集群中运行 Quay 所需的所有必要资源。
- 编辑 **QuayRegistry** 将使 Operator 协调更改并创建/升级/委派对象，以匹配所需配置。
- 删除 **QuayRegistry** 将导致所有之前创建的资源垃圾回收，**Quay** 容器 registry 将不再可用。

QuayRegistry API 非常简单，以下几节中概述了这些字段。

1.2. QUAY OPERATOR 组件

Quay 是一个强大的容器 registry 平台，因此有多个依赖项。它们包括数据库、对象存储、Redis 等。Quay Operator 管理对 Quay 的意见部署及其对 Kubernetes 的依赖项。这些依赖项被视为 *组件*，并通过 **QuayRegistry** API 配置。

在 **QuayRegistry** 自定义资源中，**spec.components** 字段配置组件。每个组件包含两个字段： **kind** - 组件的名称，**managed** - 布尔值（无论组件生命周期是由 Operator 处理）。默认情况下（显示此字段），所有组件都将在协调后自动填充，以获得可见性：

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

```
  kind: route
- managed: true
  kind: mirror
- managed: true
  kind: monitoring
- managed: true
  kind: tls
```

1.3. 使用受管组件

除非 **QuayRegistry** 自定义资源指定其他，否则 Operator 将对以下受管组件使用默认值：

- Postgres ： 要存储 registry 元数据，请使用 Software Collections中的 Postgres 10 版本
- **Redis：** 处理 Quay 构建器协调和一些内部日志记录
- **objectstorage：** 用于存储镜像层 blob，使用 Noobaa/RHOCS 提供的 **ObjectBucketClaim** Kubernetes API
- **Clair：** 提供镜像漏洞策略扫描
- **HorizontalPodAutoscaler：** 根据 memory/cpu 消耗调整 Quay pod 的数量
- **mirror：** 配置存储库镜像 worker（支持可选存储库镜像）
- **Route：** 从外部 OpenShift 提供指向 Quay registry 的外部入口点
- **monitoring：** 功能包括 Grafana 仪表板、对单个指标的访问以及通知的提示以经常重启 Quay Pod
- **TLS：** 配置 Red Hat Quay 或 OpenShift 是否处理 TLS

Operator 将处理 Red Hat Quay 使用受管组件所需的配置和安装工作。如果 Quay Operator 对环境的意见进行了建议，您可以为 Operator 提供 **非受管** 资源(overrides)的 Operator，如以下部分所述。

1.4. 将非受管组件用于依赖项

如果您有现有的组件，如 Postgres、Redis 或对象存储，则您要首先在 Quay 配置捆绑包(**config.yaml**)中配置它们，然后在 **QuayRegistry** 中引用捆绑包（作为 Kubernetes **Secret**），同时表示哪些组件是非受管状态。

注意

Quay 配置编辑器也可用于创建或修改现有配置捆绑包，并简化更新 Kubernetes **Secret** 的过程，特别是用于多项更改。当通过配置编辑器更改 Quay 配置并发送到 Operator 时，将更新 Quay 部署来反映新配置。

1.5. 配置捆绑包 SECRET

spec.configBundle Secret 字段是对与 **QuayRegistry** 相同的命名空间中的 **metadata.name** 字段的引用。此 **Secret** 必须包含 **config.yaml** 键/值对。此 **config.yaml** 文件是一个 Quay config YAML 文件。此字段是可选的，如果未提供，则会由 Operator 自动填充。如果提供，它会充当以后与任何受管组件的其他字段合并的基本配置字段，以形成最终输出 **Secret**，然后挂载到 Quay 应用 Pod 中。

1.6. OPENSHIFT 上的 RED HAT QUAY 的先决条件

在 OpenShift 上开始部署 Red Hat Quay Operator 之前，请考虑以下事项：

1.6.1. OpenShift 集群

您需要一个特权帐户到要在其上部署 Red Hat Quay Operator 的 OpenShift 4.5 或更高版本的集群中。该帐户必须在集群范围内创建命名空间。

1.6.2. 资源要求

每个 Red Hat Quay 应用程序 pod 都具有以下资源要求：

- 8Gi 内存
- 2000 毫秒的 CPU.

Red Hat Quay Operator 将根据它管理的 Red Hat Quay 部署至少创建一个应用程序 pod。确保您的 OpenShift 集群有足够的计算资源来满足这些要求。

1.6.3. Object Storage

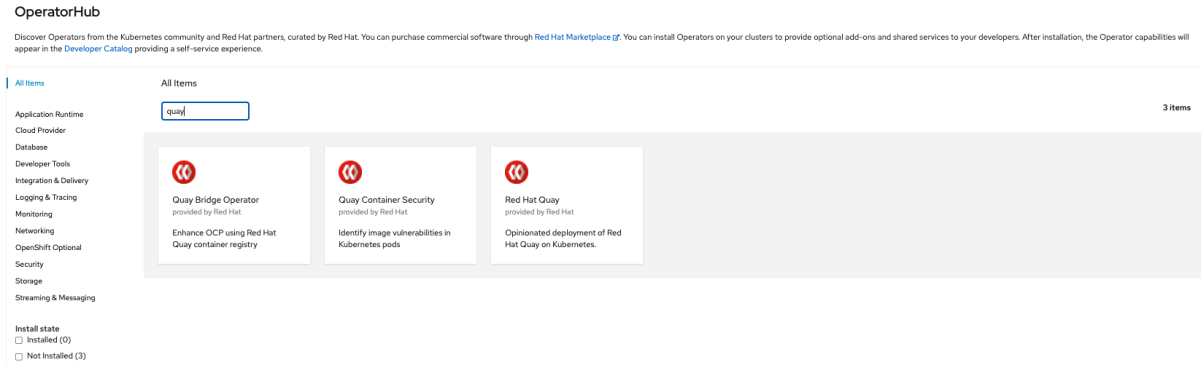
默认情况下，Red Hat Quay Operator 使用 **ObjectBucketClaim** Kubernetes API 来置备对象存储。消耗此 API 将 Operator 与特定供应商的实现分离。Red Hat OpenShift Data Foundation 通过其 NooBaa 组件提供此 API，在此示例中将使用它们。

Red Hat Quay 可以手动配置为使用以下任何支持的云存储选项：

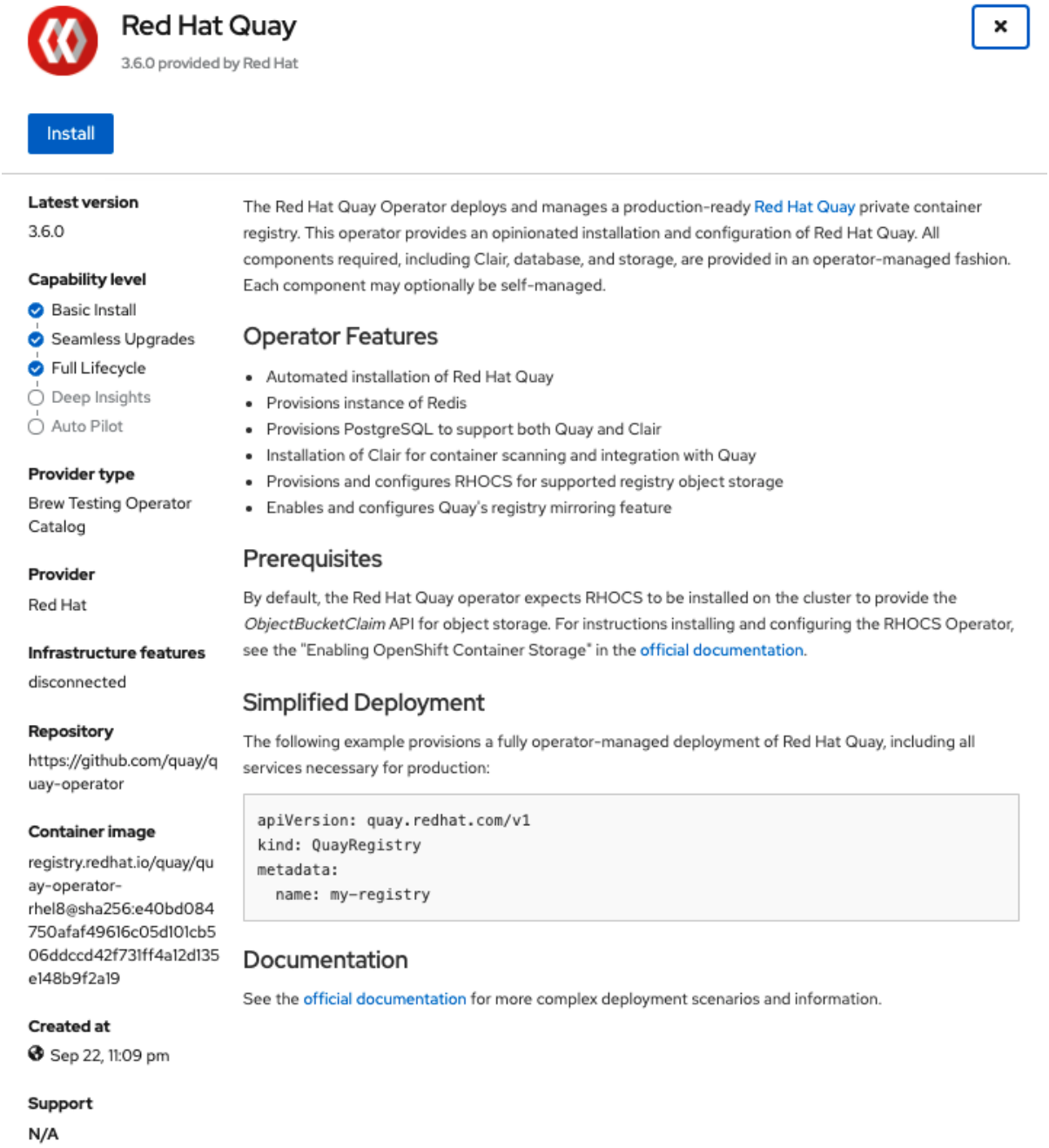
- Amazon S3（请参阅 S3 IAM Bucket Policy ）以了解有关为 Red Hat Quay 配置 S3 存储桶策略的详情。
- Azure Blob Storage
- Google Cloud Storage
- Ceph 对象网关(RADOS)
- OpenStack Swift
- CloudFront + S3

第 2 章 从 OPERATORHUB 安装 QUAY OPERATOR

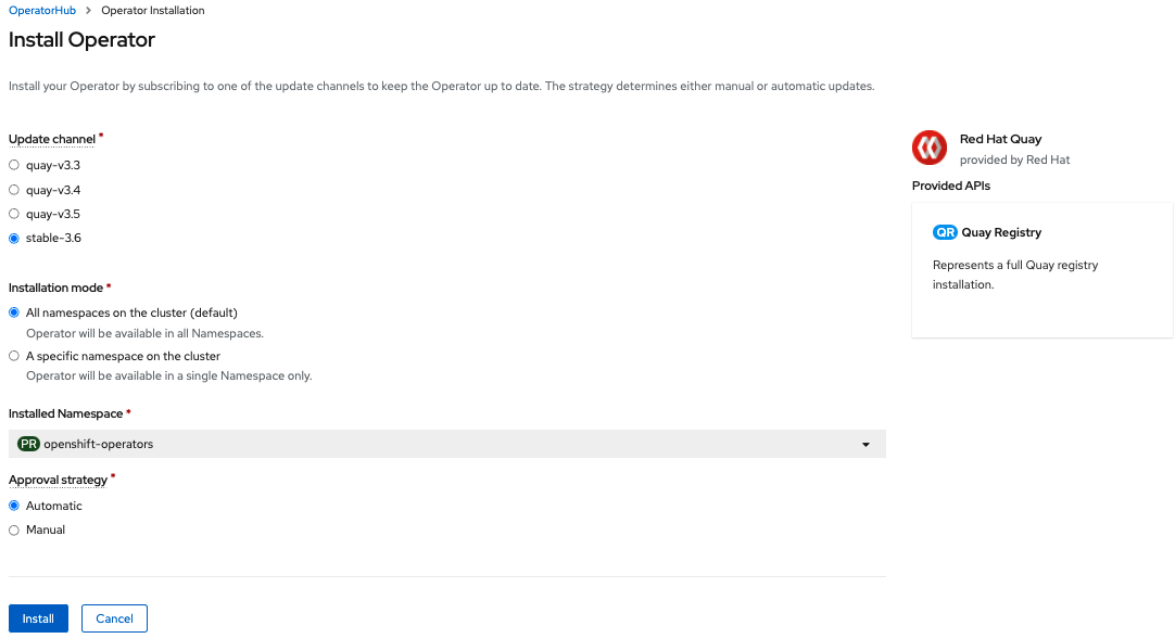
1. 使用 OpenShift 控制台 Select Operators → OperatorHub，然后选择 Red Hat Quay Operator。如果有多个，请确定使用 Red Hat Certified Operator 而不是社区版本。



2. 安装页面概述了功能和先决条件：



3. 选择 Install。此时会出现 Operator 安装页面。



4. 以下选择可用来自定义安装：

   - **更新频道：** 选择更新频道，例如，为最新版本选择 **stable-3.6**。
   - **安装模式：** 如果您希望 Operator 在整个集群范围内可用，请选择 **All namespaces on the cluster**。如果只想在一个命名空间中部署，请选择 **A specific namespace on the cluster**。建议您wide 安装 Operator 集群。如果选择单一命名空间，则默认无法使用监控组件。
   - **Approval Strategy：** 选择批准自动或手动更新。建议自动更新策略。

5. 选择 Install。

6. 短时间后，会在 Installed Operators 页面中看到成功安装 Operator。

第 3 章 在部署前配置 QUAY

Operator 可以在 OpenShift 上部署时管理所有 Red Hat Quay 组件，这是默认配置。另外，您还可以在外部管理一个或多个组件，在其中需要对集合进行更多控制，然后让 Operator 管理剩余的组件。

配置非受管组件的标准模式为：

1. 使用适当的设置创建 **config.yaml** 配置文件
2. 使用配置文件创建 Secret

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

3. 创建 QuayRegistry YAML 文件 **quayregistry.yaml**，标识非受管组件以及引用所创建的 Secret，例如：

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. 使用 YAML 文件部署 registry：

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

3.1. 预配置 QUAY 以实现自动化

Quay 具有多个支持自动化的配置选项。这些选项可以在部署前设置，以尽可能地缩短与用户界面交互的需要。

3.1.1. 允许 API 创建第一个用户

将配置选项 **FEATURE_USER_INITIALIZE** 设置为 **true**，以便您可以使用 API **/api/v1/user/initialize** 来创建第一个用户。此 API 端点不需要身份验证，这与需要由现有组织中 OAuth 应用生成的 OAuth 令牌的所有其他 registry API 调用不同。

部署 Quay 后，您可以使用 API 来创建用户，例如 **quayadmin**，只要还没有创建其他用户。如需更多信息，请参阅 使用 API 创建第一个用户 部分

3.1.2. 启用常规 API 访问

将配置选项 **BROWSER_API_CALLS_XHR_ONLY** 设置为 **false**，以允许常规访问 Quay registry API。

3.1.3. 添加超级用户

虽然在部署后无法创建用户，但可以方便地确保第一个用户是一个具有完整权限的管理员。使用 **SUPER_USER** 配置对象，可以提前配置此设置。

3.1.4. 限制用户创建

配置了超级用户后，您可以限制创建新用户到超级用户组的能力。将 **FEATURE_USER_CREATION** 设置为 **false** 以限制用户创建。

3.1.5. 推荐的自动化配置

创建包含适当设置的 **config.yaml** 配置文件：

config.yaml

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

3.1.6. 使用初始配置部署 Operator

1. 使用配置文件创建 Secret

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

2. 创建 QuayRegistry YAML 文件 **quayregistry.yaml**，标识非受管组件以及引用所创建的 Secret，例如：

quayregistry.yaml

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

3. 部署 registry：

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

4. 使用 API 创建第一个用户 **quayadmin**

3.2. 配置对象存储

在安装 Red Hat Quay 之前，您需要配置对象存储，无论您是允许 Operator 管理存储还是自行管理它。

如果您希望 Operator 负责管理存储，请参阅受管存储部分，以了解有关 安装和配置 NooBaa / RHOCS Operator 的信息。

如果您使用单独的存储解决方案，请在配置 Operator 时将 **objectstorage 设置为非受管状态**。请参见以下部分。非受管存储，以获取有关配置现有存储的详细信息。

3.2.1. 非受管存储

本节提供了一些非受管存储的配置示例，以方便。有关设置对象存储的完整详情，请参阅 Red Hat Quay 配置指南。

3.2.1.1. AWS S3 存储

```
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 云存储

```
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 存储

```
DISTRIBUTED_STORAGE_CONFIG:
  azureStorage:
    - AzureStorage
    - azure_account_name: azure_account_name_here
      azure_account_key: azure_account_key_here
      azure_container: azure_container_here
      sas_token: some/path/
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - azureStorage
```

3.2.1.4. NooBaa 非受管存储

1. 在 Storage → Object Bucket Claims 的控制台中创建一个 NooBaa Object Bucket Claim。

2. 检索 Object Bucket Claim Data 的详情，包括 Access Key、Bucket Name、Endpoint(hostname) 和 Secret Key。

3. 使用 Object Bucket Claim 信息创建 **config.yaml** 配置文件：

```
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
```

3.2.2. 受管存储

如果您希望 Operator 为 Quay 管理对象存储，您的集群需要能够通过 **ObjectBucketClaim** API 提供对象存储。使用 Red Hat OpenShift Data Foundation(ODF)Operator，有两个支持的选项：

- 由本地 Kubernetes **PersistentVolume** 存储支持的 Multi-Cloud Object Gateway 的独立实例
  - 不可用
  - 包括在 Quay 订阅中
  - 不需要单独订阅 ODF
- 具有横向扩展对象服务和 Ceph 的生产部署 ODF
  - 高可用性
  - 为 ODF 需要单独订阅

要使用独立实例选项，请继续阅读以下内容。有关 ODF 的生产部署，请参考 官方文档。

注意

Operator 使用 50 GiB 自动分配对象存储磁盘空间。此数字代表在大中型 Red Hat Quay 安装中的可用存储量，但可能不适用于您的用例。重新定义 RHOCS 卷大小目前没有被 Operator 处理。如需了解更多详细信息，请参阅调整受管存储的大小部分。

3.2.2.1. 关于单机对象网关

作为 Red Hat Quay 订阅的一部分，用户有权使用 Red Hat OpenShift Data Foundation Operator 的 *Multi-Cloud Object Gateway* (MCG)组件（以前称为 OpenShift Container Storage Operator）。此网关组件允许您为基于 Kubernetes **PersistentVolume**的块存储存储接口提供 S3 兼容对象存储接口。其用法仅限于由 Operator 管理的 Quay 部署，以及 MCG 实例的确切规格，如下所示。

由于 Red Hat Quay 不支持本地文件系统存储，因此用户可以将网关与 Kubernetes **PersistentVolume** 结合使用，以提供受支持的部署。**PersistentVolume** 直接挂载到网关实例上，作为对象存储的后备存储，并且支持任何基于块的 **StorageClass**。

由于 **PersistentVolume** 的性质，这不是横向扩展、高度可用的解决方案，并不取代 Red Hat OpenShift Data Foundation(ODF)等横向扩展存储系统。只有网关的单一实例正在运行。如果运行网关的 pod 因重新调度、更新或计划外停机而不可用，则会导致连接的 Quay 实例临时降级。

3.2.2.1.1. 创建独立对象网关

要安装 ODF（以前称为 OpenShift Container Storage）Operator 并配置单个实例多云网关服务，请按照以下步骤操作：

1. 打开 OpenShift 控制台并选择 Operators → OperatorHub，然后选择 OpenShift Data Foundation Operator。
2. 选择 Install。接受所有默认选项，然后再次选择 Install。
3. 在一分钟内，Operator 将安装并创建命名空间 **openshift-storage**。当 **Status** 列标记为 **Succeeded** 时，您可以确认它已完成。

> 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. 创建 NooBaa 对象存储。将以下 YAML 保存到名为 **noobaa.yaml** 的文件。

```
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
```

这将创建 *多云对象网关* 的单一实例部署。

5. 使用以下命令应用配置：

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

6. 几分钟后，您应该会看到 MCG 实例已完成置备（PAS**E** 列将设置为 **Ready**）：

```
$ 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. 接下来，为网关配置后备存储。将以下 YAML 保存到名为 **noobaa-pv-backing-store.yaml** 的文件。

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) 对象存储服务的整体容量，根据需要进行调整

(2) 用于请求的 **PersistentVolume 的 StorageClass**，删除此属性以使用集群默认值

8. 使用以下命令应用配置：

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

这会为网关创建后备存储配置。Quay 中的所有镜像将通过网关在上述配置创建的 **PersistentVolume** 中存储为对象。

9. 最后，运行以下命令，使 **PersistentVolume** 存储 Operator 发布的所有 **ObjectBucketClaims** 的默认设置。

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

为 Red Hat Quay 设置 *Multi-Cloud Object Gateway* 实例的流程。请注意，这个配置不能在安装 Red Hat OpenShift Data Foundation 的集群中并行运行。

3.3. 配置数据库

3.3.1. 使用现有的 Postgres 数据库

1. 使用所需的数据库字段创建配置文件 **config.yaml** ：

config.yaml:

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

2. 使用配置文件创建 Secret：

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

3. 创建 QuayRegistry YAML 文件 **quayregistry.yaml**，将 **postgres** 组件标记为非受管状态，并引用所创建的 Secret：

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. 按照以下部分所述部署 registry。

3.3.2. 数据库配置

您使用 DB_URI 字段和 DB_CONNECTION_ARGS 结构中的所需 DB_URI 字段和可选连接参数来配置与数据库的连接。DB_CONNECTION_ARGS 下定义的一些键值对是通用的，另一些则特定于数据库。特别是，SSL 配置取决于您要部署的数据库，下面提供了 PostgreSQL 和 MySQL 示例。

3.3.2.1. 数据库 URI

表 3.1. 数据库 URI

字段	类型	描述
DB_URI （必需）	字符串	用于访问数据库的 URI，包括任何凭证

例如：

```
postgresql://quayuser:quaypass@quay-server.example.com:5432/quay
```

3.3.2.2. 数据库连接参数

表 3.2. 数据库连接参数

字段	类型	描述
DB_CONNECTION_ARGS	对象	数据库的可选连接参数，如超时和 SSL
.autorollback	布尔值	是否使用线程连接 应该为 **true**
.threadlocals	布尔值	是否使用自动重新连接 应该 **ALWAYS** 为 **true**

3.3.2.2.1. PostgreSQL SSL 连接参数

下面提供了 PostgreSQL SSL 配置示例：

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

sslmode 选项确定是否存在与服务器协商安全 SSL TCP/IP 连接的安全 SSL TCP/IP 连接。有六个模式：

- **disable：** 只尝试非 SSL 连接
- **Allow：** 首先尝试非 SSL 连接 ; 如果失败，请尝试 SSL 连接
- **prefer:**（默认）首先尝试 SSL 连接；如果失败，请尝试非 SSL 连接
- **Requires：** 仅尝试 SSL 连接。如果存在 root CA 文件，验证证书的方式与是否指定了 verify-ca 的方式相同
- **verify-ca：** 仅尝试 SSL 连接，并验证服务器证书是否由可信证书颁发机构(CA)发出。
- **verify-full：** 仅尝试 SSL 连接，验证服务器证书是否由受信任的 CA 发布，并且请求的服务器主机名与证书中的服务器主机名匹配

有关 PostgreSQL 的有效参数的更多信息，请参阅 https://www.postgresql.org/docs/current/libpq-connect.html

3.3.2.2.2. MySQL SSL 连接参数

MySQL SSL 配置示例如下：

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

有关 MySQL 的有效连接参数的信息，请参考 https://dev.mysql.com/doc/refman/8.0/en/connecting-using-uri-or-key-value-pairs.html。

3.3.3. 使用受管 PostgreSQL

建议：

- 应使用 Postgres 镜像或您自己的备份基础架构提供的工具定期执行数据库备份。Operator 目前不确保 Postgres 数据库被备份。
- 必须使用 Postgres 工具和程序从备份中恢复 Postgres 数据库。请注意，当数据库恢复正在进行时，您的 Quay **Pod** 不应该运行。
- Operator 使用 50 GiB 自动分配数据库磁盘空间。此数字代表在大中型 Red Hat Quay 安装中的可用存储量，但可能不适用于您的用例。Operator 当前没有处理数据库卷的大小。

3.4. 配置 TLS 和路由

通过新的受管组件 **tls**，增加了对 OpenShift Container Platform Edge-Termination 路由的支持。这会将 **路由** 组件与 TLS 分开，并允许用户单独配置它们。**EXTERNAL_TLS_TERMINATION : true** 是建议的设置。managed **tls** 意味着使用默认的集群通配符证书。非受管 **tls** 意味着用户提供的 cert/key 对将注入到 **Route** 中。

SSL.cert 和 **ssl.key** 现在被移到一个单独的持久的 Secret 中，这样可确保在每次协调时不会重新生成 cert/key 对。现在，它们被格式化为 **边缘路由**，并挂载到 Quay 容器中的同一目录中。

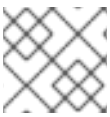
在配置 TLS 和路由时，可以多次修改，但适用以下规则：

- 如果 TLS **管理**，则必须 **管理**路由
- 如果 TLS 是 **非受管状态**，则必须提供 certs，使用 config 工具或直接在配置捆绑包中提供

下表概述有效选项：

表 3.3. TLS 和路由的有效配置选项

选项	Route	TLS	提供的证书	结果
我的负载均衡器可以处理 TLS	Managed	Managed	否	带有默认通配符证书的边缘路由
Red Hat Quay 处理 TLS	Managed	Unmanaged	是	带有在 pod 中挂载的 certs 的 passthrough 路由
Red Hat Quay 处理 TLS	Unmanaged	Unmanaged	是	证书在 quay pod 中进行设置，但必须手动创建路由



注意

当由 Operator 管理 TLS 时，Red Hat Quay 3.6 不支持构建器。

3.4.1. 使用 TLS 证书创建配置捆绑包 secret，密钥对：

要添加您自己的 TLS 证书和密钥，请在配置捆绑包 secret 中包括它们，如下所示：

```
$ 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. 配置其他组件

3.5.1. 使用外部 Redis

如果要使用外部 Redis 数据库，请将组件设置为 **QuayRegistry** 实例中的非受管：

1. 使用所需的 redis 字段创建配置文件 **config.yaml** ：

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

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

2. 使用配置文件创建 Secret

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

3. 创建 QuayRegistry YAML 文件 **quayregistry.yaml**，它将 redis 组件标记为非受管状态，并引用所创建的 Secret：

```
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. 部署 registry

3.5.1.1. Redis 配置字段

3.5.1.1.1. 构建日志

表 3.4. 构建日志配置

字段	类型	描述
BUILDLOGS_REDIS （必需）	对象	构建日志缓存的 redis 连接详情

字段	类型	描述
.主机 （必需）	字符串	Redis 可以访问的主机名 **示例：** **quay-server.example.com**
.port （必需）	Number	Redis 可访问的端口 **示例：** **6379**
.password	字符串	Redis 可访问的端口 **示例：** **strongpassword**

3.5.1.1.2. 用户事件

表 3.5. 用户事件配置

字段	类型	描述
USER_EVENTS_REDIS （必需）	对象	用户事件处理的 redis 连接详情
.主机 （必需）	字符串	Redis 可以访问的主机名 **示例：** **quay-server.example.com**
.port （必需）	Number	Redis 可访问的端口 **示例：** **6379**
.password	字符串	Redis 可访问的端口 **示例：** **strongpassword**

3.5.1.1.3. redis 配置示例

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

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

3.5.2. 禁用 Horizontal Pod Autoscaler

HorizontalPodAutoscalers 已添加到 Clair、Quay 和 Mirror pod 中，以便在负载高峰期间自动扩展。

因为 HPA 默认配置为 **受管**，Quay 的 pod 数量将 Clair 和存储库镜像设置为 2。这有助于在通过 Operator 更新/配置 Quay 或重新调度事件期间出现停机的问题。

如果要禁用自动扩展或创建自己的 **HorizontalPodAutoscaler**，只需在 **QuayRegistry** 实例中将组件指定为 Unmanaged：

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

3.5.3. 禁用 Route 组件

要防止 Operator **创建路由** ：

1. 将组件标记为 **QuayRegistry** 中的非受管：

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

2. 通过编辑 **config.yaml** 文件，指定您希望 Quay 在配置中处理 TLS：

config.yaml

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

如果您无法正确配置非受管路由，您会看到类似如下的错误：

```
{
  {
    "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"
}
```


注意

禁用默认路由 **意味着您** 现在负责创建路由、**Service** 或 **Ingress**，以便能访问 Quay 实例，以及您使用的任何 DNS 都必须与 Quay 配置中 **SERVER_HOSTNAME** 匹配。

3.5.4. 非受管监控

如果在单一命名空间中安装 Quay Operator，则监控组件会自动设置为 'unmanaged'。要在这种情况下启用监控，请查看 第 8.2 节 “在单一命名空间中安装 Operator 时启用监控” 部分。

显式禁用监控：

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

3.5.5. 非受管镜像

显式禁用镜像：

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

```
components:
  - kind: mirroring
    managed: false
```

第 4 章 使用 QUAY OPERATOR 部署 QUAY

Operator 可从命令行或 OpenShift 控制台部署，但基本步骤相同。

4.1. 从命令行部署 RED HAT QUAY

1. 创建一个命名空间，如 quay-enterprise。

2. 如果要预先配置部署的各个方面，为 config 捆绑包创建 secret

3. 在名为 quayregistry.yaml的文件中创建 QuayRegistry 自定义资源

   a. 对于最小部署，使用所有默认值：

   quayregistry.yaml:

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

   b. 如果要具有一些组件非受管，请在 spec 字段中添加此信息。例如，最小部署可能类似如下：

   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. 如果您已创建了配置捆绑包，如 init-config-bundle-secret，在 quayregistry.yaml 文件中引用它：

quayregistry.yaml:

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

4. 在指定的命名空间中创建 QuayRegistry ：

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

5. 如需有关如何跟踪部署进度的信息，请参阅 监控和调试部署过程。

6. 等待 status.registryEndpoint 填充。

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

4.1.1. 使用命令行查看创建的组件

使用 oc get pods 命令查看部署的组件：

```
$ 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. Pod 横向自动扩展(HPA)

默认部署显示了以下正在运行的 pod：

- Quay 应用本身的两个 pod(example-registry-quay-app-*')
- 用于 Quay 日志记录的 Redis pod(example-registry-quay-redis-*)
- Quay 用于元数据存储的 PostgreSQL 的一个数据库 pod(example-registry-quay-database-*)
- 用于 Quay 配置编辑器的一个 pod(example-registry-quay-config-editor-*)
- 两个 Quay 镜像 pod(example-registry-quay-mirror-*)
- Clair 应用的两个 pod(example-registry-clair-app-*)
- Clair 的 PostgreSQL pod(example-registry-clair-postgres-*)

因为 HPA 默认配置为 受管，Quay 的 pod 数量将 Clair 和存储库镜像设置为 2。这有助于在通过 Operator 更新/配置 Quay 或重新调度事件期间出现停机的问题。

```
$ 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. 使用 API 创建第一个用户

当使用 API 创建第一个用户时，必须满足以下条件：

- 配置选项 FEATURE_USER_INITIALIZE 必须设置为 true
- 数据库中没有用户

如需有关预配置部署的更多信息，请参阅 预配置 Quay 以实现自动化

4.1.3.1. 调用 API

使用 status.registryEndpoint URL，调用 /api/v1/user/initialize API，传递用户名、密码和电子邮件地址。您还可以通过指定 "access_token": true 来请求 OAuth 令牌。

```
$  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}'
```

```
{"access_token":"6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED",
"email":"quayadmin@example.com","encrypted_password":"1nZMLH57RIE5UGdL/yYpDOHL
qiNCgimb6W9kfF8MjZ1xrfDpRyRs9NUnUuNuAitW","username":"quayadmin"}
```

如果成功，该方法返回一个带有用户名、电子邮件和加密密码的对象。如果用户已存在于数据库中，则返回错误：

```
$  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": "quayuser2", "password":"quaypass123", "email":
"quayuser2@example.com"}'
```

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

密码必须至少为 8 个字符，且不包含空格：

```
 $  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":"pass123", "email":
"quayadmin@example.com"}'
```

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

4.1.3.2. 使用 OAuth 令牌

现在，您可以在指定返回的 OAuth 代码调用 Quay API 的剩余部分。例如，获取当前用户的列表：

```
$ curl -X GET -k -H "Authorization: Bearer
6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://example-registry-quay-quay-
enterprise.apps.docs.quayteam.org/api/v1/superuser/users/
```

```
{
  "users": [
    {
      "kind": "user",
      "name": "quayadmin",
      "username": "quayadmin",
      "email": "quayadmin@example.com",
      "verified": true,
      "avatar": {
        "name": "quayadmin",
        "hash": "3e82e9cbf62d25dec0ed1b4c66ca7c5d47ab9f1f271958298dea856fb26adc4c",
        "color": "#e7ba52",
        "kind": "user"
      },
      "super_user": true,
      "enabled": true
    }
  ]
}
```

在本例中，quayadmin 用户的详细信息返回，因为它是目前创建的唯一用户。

4.1.3.2.1. 创建机构

要创建机构，使用 POST 调用 api/v1/organization/ 端点：

```
$ curl -X POST -k --header 'Content-Type: application/json' -H "Authorization: Bearer 6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/organization/ --data '{"name": "testorg", "email": "testorg@example.com"}'
```

```
"Created"
```

4.1.3.2.2. 获取机构详情

检索您创建的机构详情：

```
$ curl -X GET -k --header 'Content-Type: application/json' -H "Authorization: Bearer 6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://min-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/organization/testorg
```

```
{
  "name": "testorg",
  "email": "testorg@example.com",
  "avatar": {
    "name": "testorg",
    "hash": "5f113632ad532fc78215c9258a4fb60606d1fa386c91b141116a1317bf9c53c8",
    "color": "#a55194",
    "kind": "user"
  },
  "is_admin": true,
  "is_member": true,
  "teams": {
    "owners": {
      "name": "owners",
      "description": "",
      "role": "admin",
      "avatar": {
        "name": "owners",
        "hash": "6f0e3a8c0eb46e8834b43b03374ece43a030621d92a7437beb48f871e90f8d90",
        "color": "#c7c7c7",
        "kind": "team"
      },
      "can_view": true,
      "repo_count": 0,
      "member_count": 1,
      "is_synced": false
    }
  },
  "ordered_teams": [
    "owners"
  ],
  "invoice_email": false,
  "invoice_email_address": null,
```

```
    "tag_expiration_s": 1209600,
    "is_free_account": true
}
```

4.1.4. 监控和调试部署过程

Red Hat Quay 3.6 提供了新的功能，可在部署阶段排除问题。QuayRegistry 对象中的状态可帮助您在部署过程中监控组件的健康状态，以帮助您调试可能出现的问题：

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

部署后，QuayRegistry 对象会显示基本配置：

```
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: ""
```

- 使用 oc get pods 命令查看部署组件的当前状态：

```
$ 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
```

在部署进行过程中，QuayRegistry 对象会显示当前状态。在本实例中，数据库迁移就位，其他组件也等到此完成后才会等待。

```
  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
```

当部署过程成功完成时，QuayRegistry 对象中的状态不会显示不健康的组件：

```
 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: 3.6.0
   lastUpdated: 2021-09-14 10:52:46.104181633 +0000 UTC
   registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
   unhealthyComponents: {}
```

4.2. 从 OPENSHIFT 控制台部署 RED HAT QUAY

1. 创建一个命名空间，如 quay-enterprise。

2. 选择 Operators → Installed Operators，然后选择 Quay Operator 来导航到 Operator 详情视图。

3. 在 'Provided API' 下点 ' Quay Registry' 标题上的 'Create Instance'。

4.

（可选）更改 QuayRegistry 的 'Name'。这将影响 registry 的主机名。所有其他字段都填充了默认值。

5. 单击 'Create' 以提交 QuayRegistry 以供 Quay Operator 部署。

6. 您应该重定向到 QuayRegistry 列表视图。点刚才创建的 QuayRegistry 来查看详情视图。

7. 'Registry Endpoint' 有值后，点它通过 UI 访问新的 Quay registry。现在，您可以选择"创建帐户"来创建用户并登录。

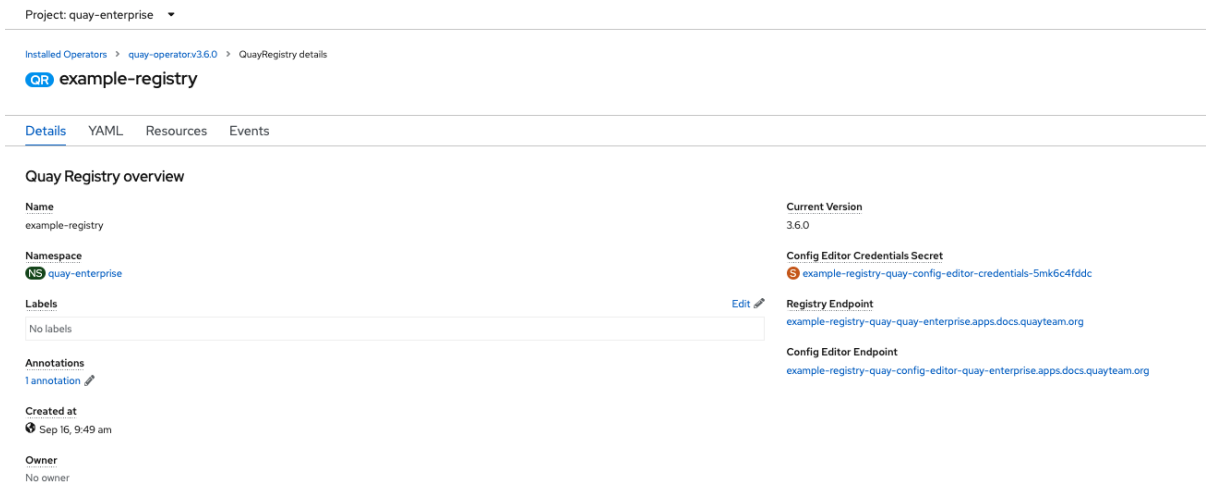
4.2.1. 使用 Quay UI 创建第一个用户

注意

此流程假设 FEATURE_USER_CREATION 配置选项尚未设为 false。如果错误为 false，则 UI 上的 Create Account 功能将被禁用，您必须使用 API 来创建第一个用户。

1. 在 OpenShift 控制台中，使用适当的命名空间 / 项目导航到 Operators → Installed Operators。

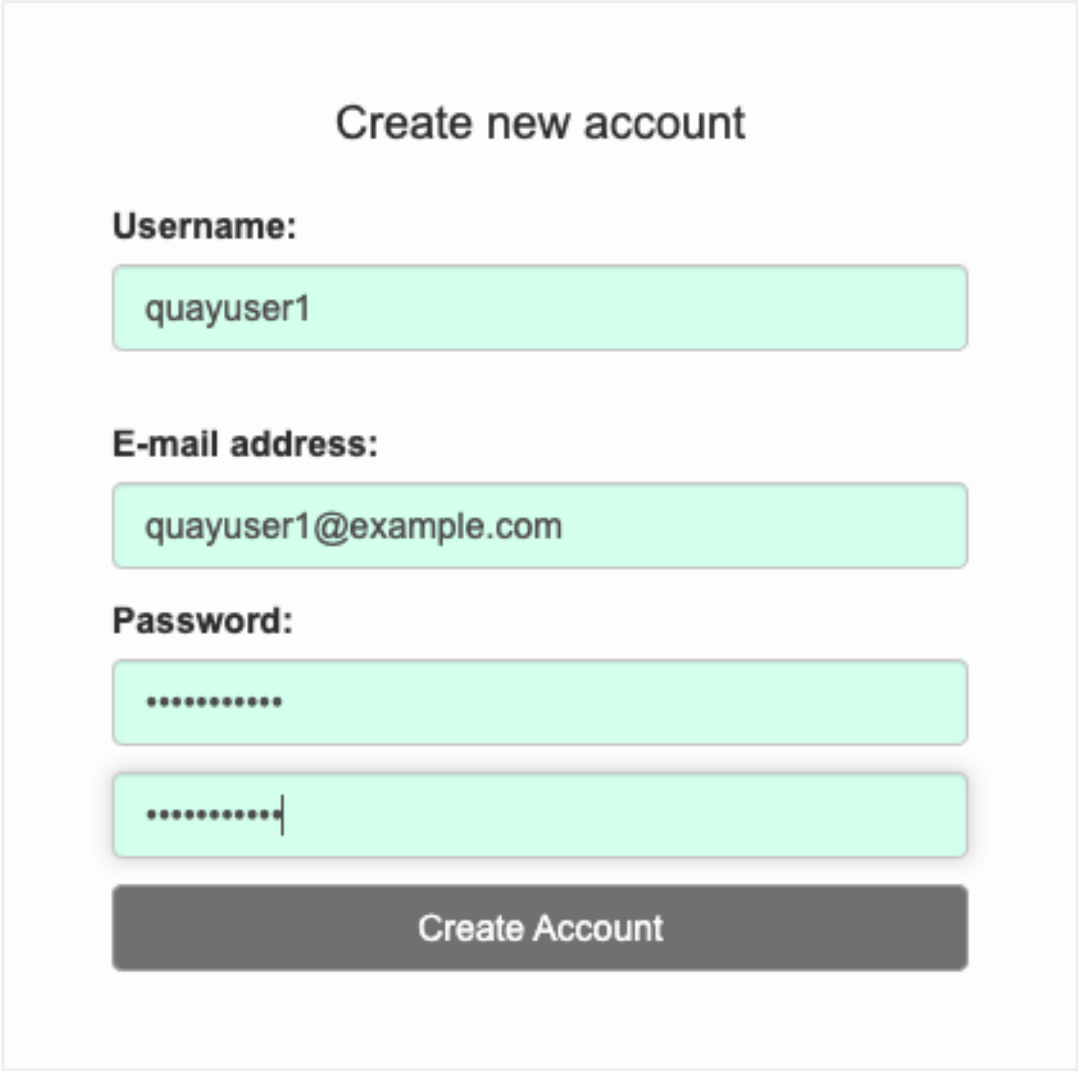
2. 点击新安装的 QuayRegistry，以查看详情：



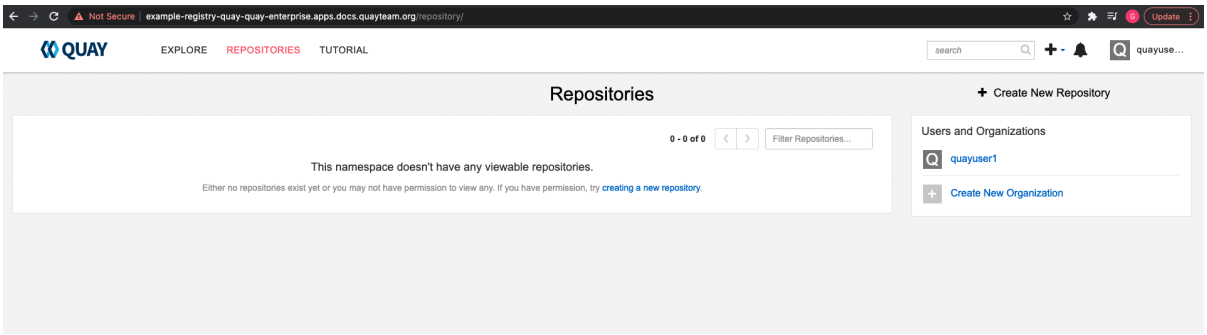
3. **当 Registry 端点 具有值后，在浏览器中导航到此 URL**

4. 选择 **Quay registry UI 中的"Create Account"来创建用户**

5. **输入用户名、密码、电子邮件并点击 Create Account**



6. **您会自动登录到 Quay registry**



第 5 章 使用命令行和 API 在 OPENSHIFT 上配置 QUAY

部署后，您可以通过编辑 Quay 配置捆绑包 secret spec.configBundleSecret 配置 Quay 应用程序，您也可以更改 QuayRegistry 资源的 spec.components 对象中的组件的受管状态。

Operator 不会监视 spec.configBundleSecret 资源是否有更改，因此建议对新的 Secret 资源进行配置更改，并且 spec.configBundleSecret 字段已更新，以反映更改。如果与新配置存在问题，则简单地将 spec.configBundleSecret 的值恢复到旧的 Secret。

更改配置的步骤涉及：

1. 确定当前的端点和 secret
2. 如果已在 OpenShift 上部署了 Red Hat Quay，请下载现有配置捆绑包
3. 创建或更新 config.yaml 配置文件
4. 装配 Quay 所需的任何 SSL 证书或服务所需的自定义 SSL 证书
5. 使用配置文件和任何证书创建新配置捆绑包 secret
6. 创建或更新 registry，引用新的配置捆绑包 secret，并指定任何覆盖来管理组件
7. 监控部署以确保成功完成并且配置更改生效

另外，您可以使用配置编辑器 UI 来配置 Quay 应用，如 *第 6 章 使用配置工具在 OpenShift 中重新配置 Quay* 部分所述。

5.1. 确定 QUAYREGISTRY 端点和 SECRET

您可以使用 oc describe quayregistry 或 oc get quayregistry -o yaml 来检查 QuayRegistry 资源，以确定当前的端点和 secret：

```
$ 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:
  ...
  configBundleSecret: example-registry-quay-config-bundle-fjpnm
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-kk55dc7299
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.quayteam.org
  currentVersion: 3.6.0
  lastUpdated: 2021-09-21 11:18:13.285192787 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
  unhealthyComponents: {}
```

相关字段为：

- registryEndpoint ：registry的 URL，用于浏览器访问 registry UI，以及 registry API 端点
- configBundleSecret: config bundle secret，其中包含 config.yaml 文件和任何 SSL 证书
- configEditorEndpoint ：配置编辑器工具的 URL，用于浏览器访问配置工具，以及配置 API
- configEditorCredentialsSecret:包含用户名的 secret（通常是 quayconfig）以及配置编辑器工具的密码

要确定配置编辑器工具的用户名和密码：

1. 检索 secret：

```
$ oc get secret -n quay-enterprise example-registry-quay-config-editor-credentials-
kk55dc7299 -o yaml

apiVersion: v1
data:
```

```
  password: SkZwQkVKTUN0a1BUZmp4dA==
  username: cXVheWNvbmZpZw==
kind: Secret
```

2. 解码用户名：

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

quayconfig
```

3. 解码密码：

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

JFpBEJMCtkPTfjxt
```

5.2. 下载现有配置

访问当前配置的方法有很多：

1. 使用配置编辑器端点，为配置编辑器指定用户名和密码：

```
$ 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":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURVVENDQWptZ0F3SUJBZ0lJRE9kWFhuUXFjMUF3RFFZSktvWklodmNOQVFFTEJRQXdOakUwTURJR0ExVUUKQXd3cmIzQmxibk5vYVdaMExYTmxjblpwWTJVdGMyVnlkbWx1WnkxemFXZHVaWEpBTVRZek1UYzNPREV3TXpBZQpGdzB5TVRBNU1UWXdOelF4TkRKYUZ..."
   }
}
```

2. 使用 config bundle secret

a. 获取 secret 数据：

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

```
{
  "config.yaml":
"QUxMT1dfUFVMTFNfV0lUSE9VVF9TVFJJQ1RfTE9HR0lORzogZmFsc2UKQVVUSE
VOVElDQVRJT05fVFlQRTogRGF0YWJhc2UKQVZBVEFSX0tJTkQ6IGxvY2FsCkRBV
EFCQVNFX1NFQ1JFVF9LRVk6IHhlOEc1VDBNbklaGxNQzNkTjd3MWR5WWxwVm
o0a0R2enlxZ3l6Ulp5ZjFpODBmWWU3VDUxU1FPZ3hXelpocFlqYlVxNzRKaDllVVVE
VWpyCkRFR
...
OgotIDJ3ClRFQU1fUkVTWU5DX1NUQUxFX1RJTUU6IDYwbQpURVNUSU5HOiBmY
WxzZQpVU0VSX1JFQ09WRVJZX1RPS0VOX0xJRkVUSU1FOiAzMG0K",
  "extra_ca_cert_service-ca.crt":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURVVENDQWptZ0F3SUJBZ0
lJRE9kWFhuUXFjMUF3RFFZSktvWklodmNOQVFFTEJRQXdOakUwTURJR0ExVUU
KQXd3cmlzQmxibk5vYVdaMExYTmxjblpwWTJVdGMyVnlkbWx1WnkxemFXZHVaW
EpBTVRZek1UYzNPREV3TXpBZQpGdzB5TVRBNU1UWXdOelF4TkRKYUZ3MHI
...
XSW1jaApkQXZTWGpFUnZOZEZzN3pHK1VzTmZwN0ZlQkJVWkY4L2RZNWJCR2M
wWTVaY0J6bFNjQT09Ci0tLS0tRU5EIENFUlRJRkIDQVRFLS0tLS0K"
}
```

b. 解码数据：

```
$ echo 'QUxMT1dfUFVMTFN...U1FOiAzMG0K' | base64 --decode
```

```
ALLOW_PULLS_WITHOUT_STRICT_LOGGING: false
AUTHENTICATION_TYPE: Database
...
TAG_EXPIRATION_OPTIONS:
- 2w
TEAM_RESYNC_STALE_TIME: 60m
TESTING: false
USER_RECOVERY_TOKEN_LIFETIME: 30m
```

5.3. 使用配置捆绑包配置自定义 SSL 证书

您可以通过创建新的配置捆绑包 secret，在初始部署前或在 OpenShift 上部署 Red Hat Quay 后配置自定义 SSL 证书。如果您要在现有部署中添加证书，则必须在新 config bundle secret 中包含完整的 config.yaml，即使您没有进行任何配置更改。

1. 使用嵌入式数据或使用文件创建 secret：

a. 直接将配置详情嵌入到 Secret 资源 YAML 文件中，例如：

custom-ssl-config-bundle.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-ssl-config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: |
    ALLOW_PULLS_WITHOUT_STRICT_LOGGING: false
    AUTHENTICATION_TYPE: Database
    ...
  extra_ca_cert_my-custom-ssl.crt: |
    -----BEGIN CERTIFICATE-----
    MIIDsDCCApigAwIBAgIUCqlzkHjF5i5TXLFy+sepFrZr/UswDQYJKoZIhvcNAQEL

BQAwbzELMAkGA1UEBhMCSUUxDzANBgNVBAgMBkdBTFdBWTEPMA0GA1UEB
wwGR0FM
    ....
    -----END CERTIFICATE-----
```

接下来，从 YAML 文件创建 secret：

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

b. 另外，您可以创建包含所需信息的文件，然后从这些文件创建 secret：

```
$ 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. 创建或更新 QuayRegistry YAML 文件 quayregistry.yaml，引用所创建的 Secret，例如：

quayregistry.yaml

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

3. **使用 YAML 文件部署或更新 registry：**

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

5.4. 卷大小覆盖

从 Red Hat Quay v3.6.2 开始，您可以指定为受管组件置备的存储资源所需的大小。Clair 和 Quay PostgreSQL 数据库的默认大小为 50Gi。现在，您可以预先选择足够大的容量，可能是因为性能的原因，或者在您的存储后端没有调整大小功能的情况下选择。

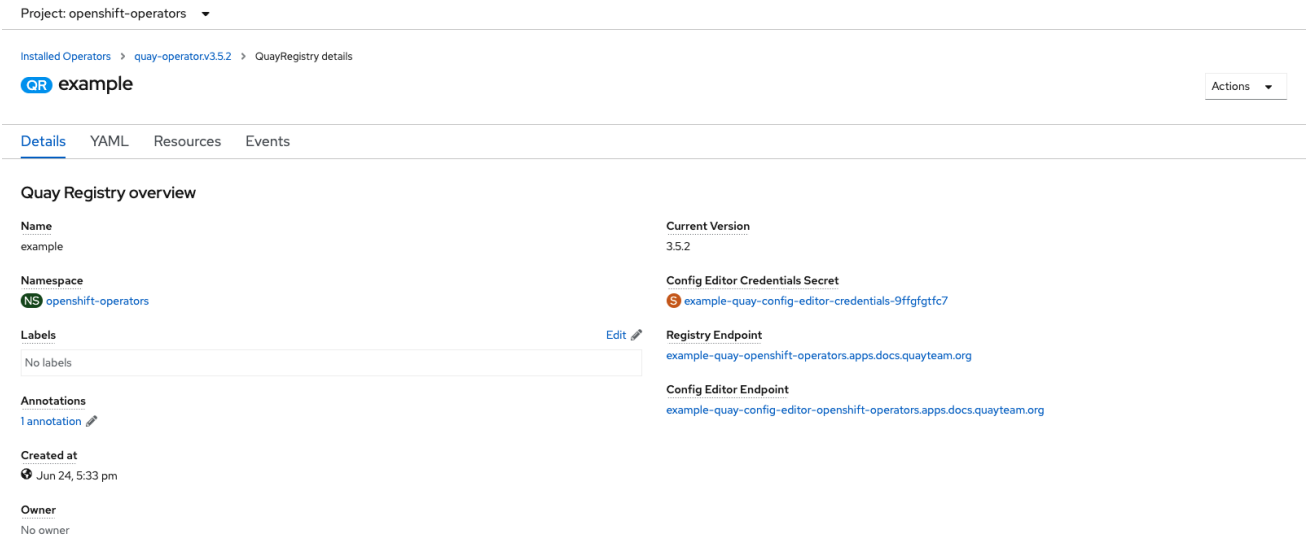
在以下示例中，Clair 和 Quay PostgreSQL 数据库的卷大小已设置为 70Gi ：

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay-example
  namespace: quay-enterprise
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clair
      managed: true
      overrides:
        volumeSize: 70Gi
    - kind: postgres
      managed: true
      overrides:
        volumeSize: 70Gi
```

第 6 章 使用配置工具在 OPENSHIFT 中重新配置 QUAY

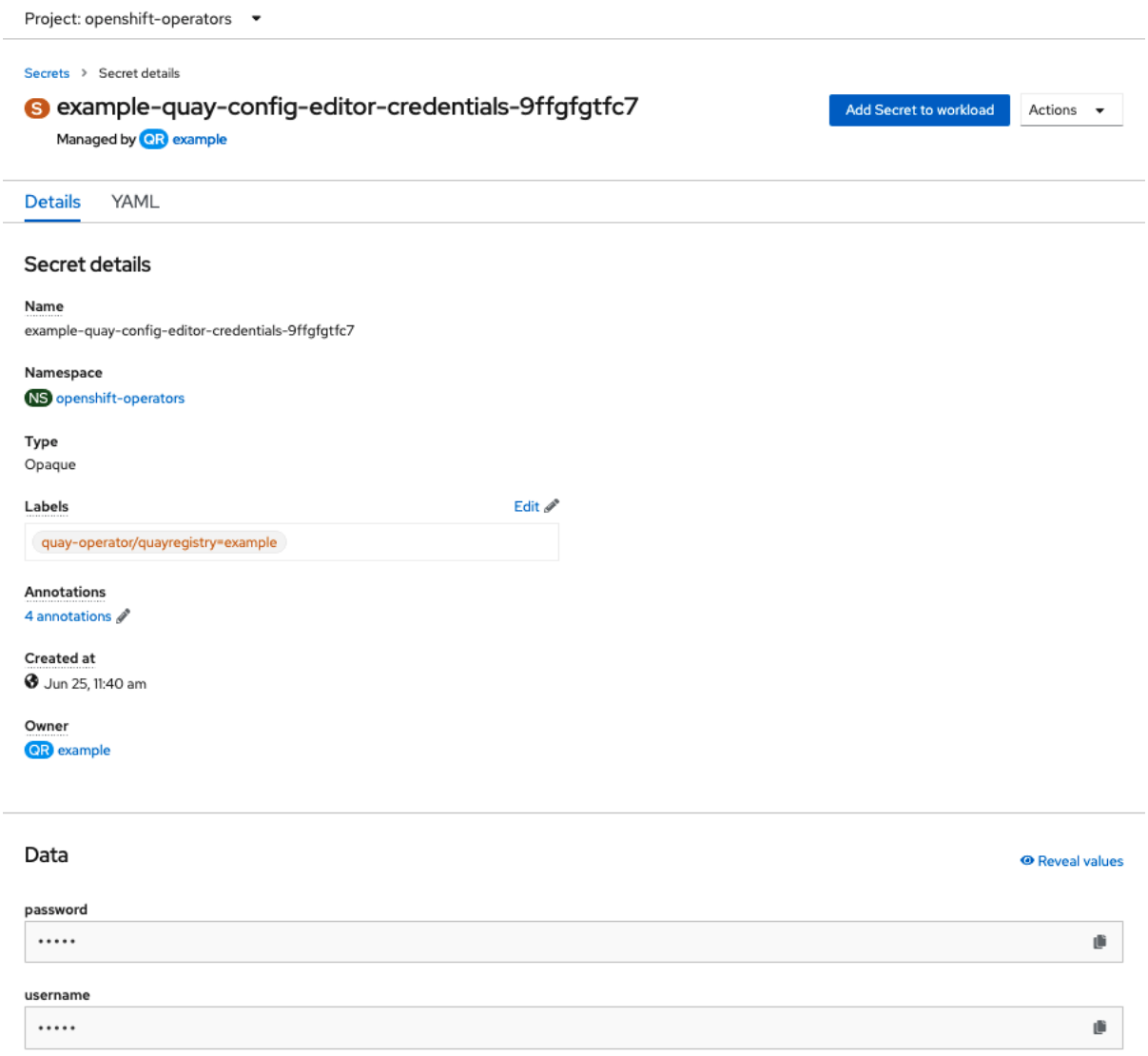
6.1. 访问配置编辑器

在 QuayRegistry 屏幕的 Details 部分中，可以使用 config 编辑器的端点，以及指向含有登录配置编辑器凭证的 secret 的链接：

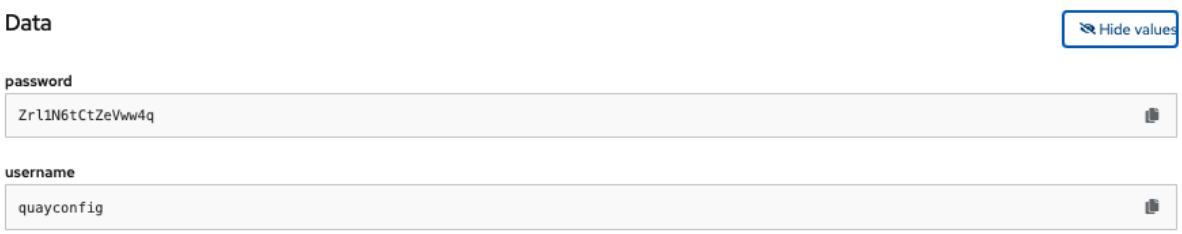


6.1.1. 检索配置编辑器凭证

1. 点击配置编辑器 secret 的链接：

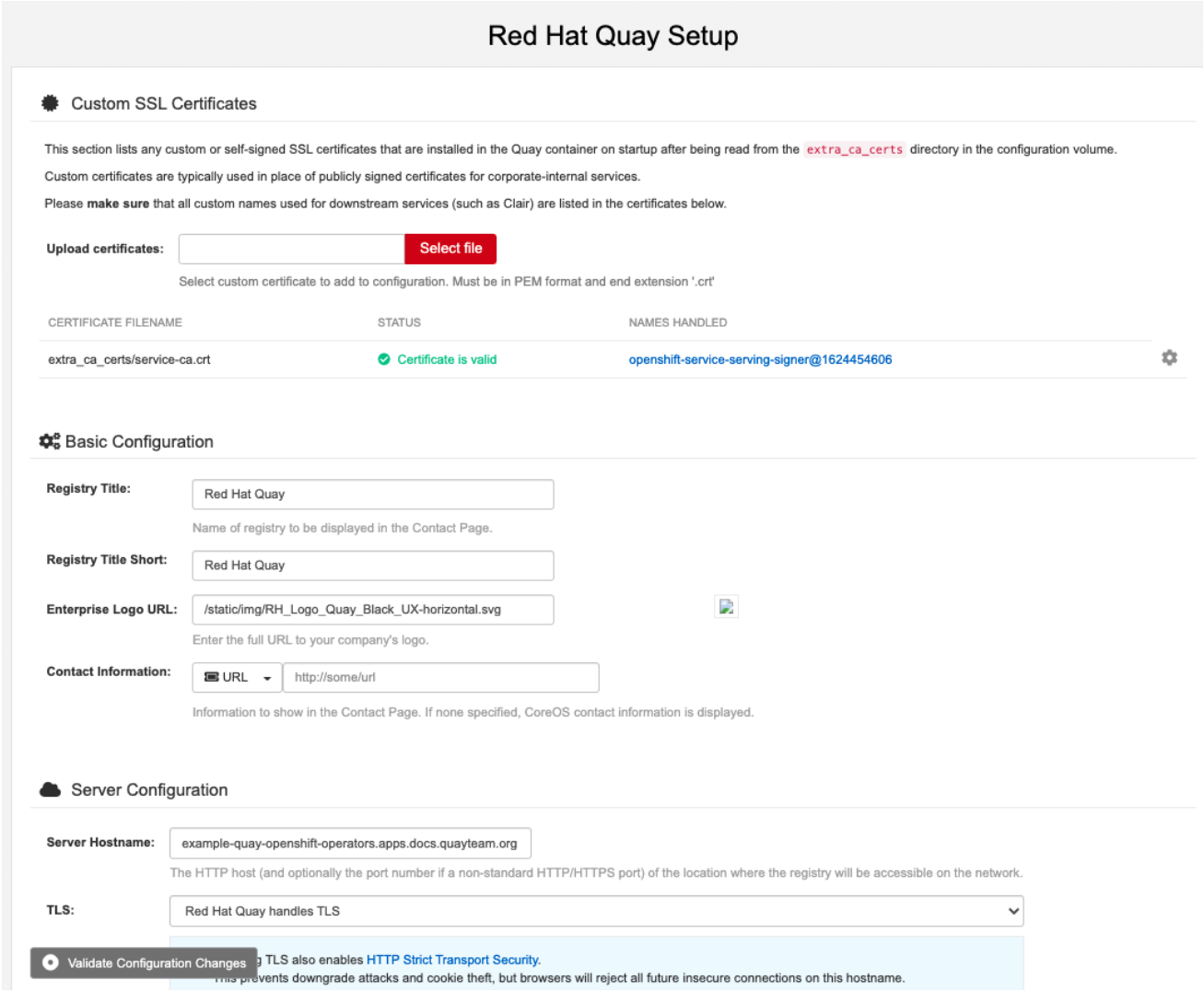


2. **在 Secret Details 屏幕的 Data 部分，点 Reveal values 来查看登录到配置编辑器的凭证：**



6.1.2. 登录到配置编辑器

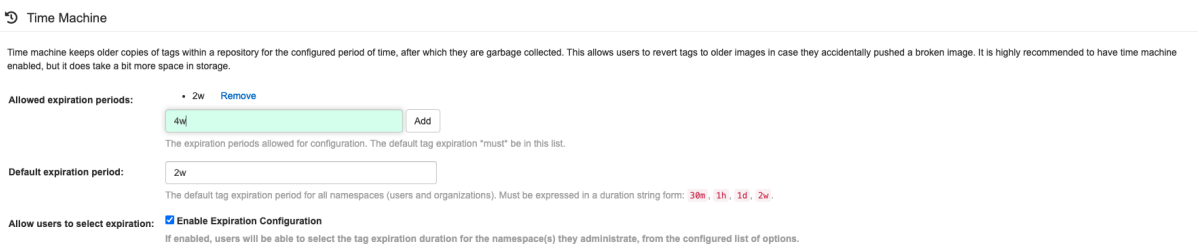
浏览到配置编辑器端点，然后输入用户名（通常是 quayconfig ）以及访问配置工具的对应密码：



6.1.3. 更改配置

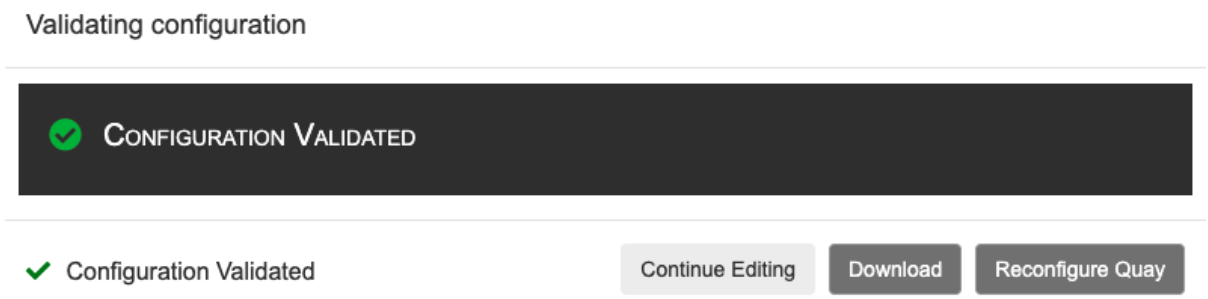
在本例中，通过配置编辑器工具添加超级用户：

1. 为时间机器功能添加过期周期，如 **4w** ：

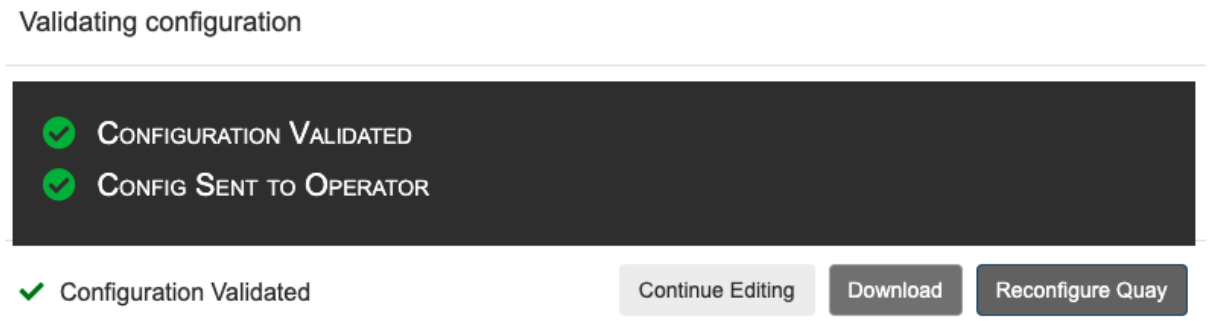


2. 选择 **Validate Configuration Changes** 以确保更改有效

3. 按 **Reconfigure Quay** 按钮应用更改：



4. 配置工具通知您已向 Quay 提交了更改：



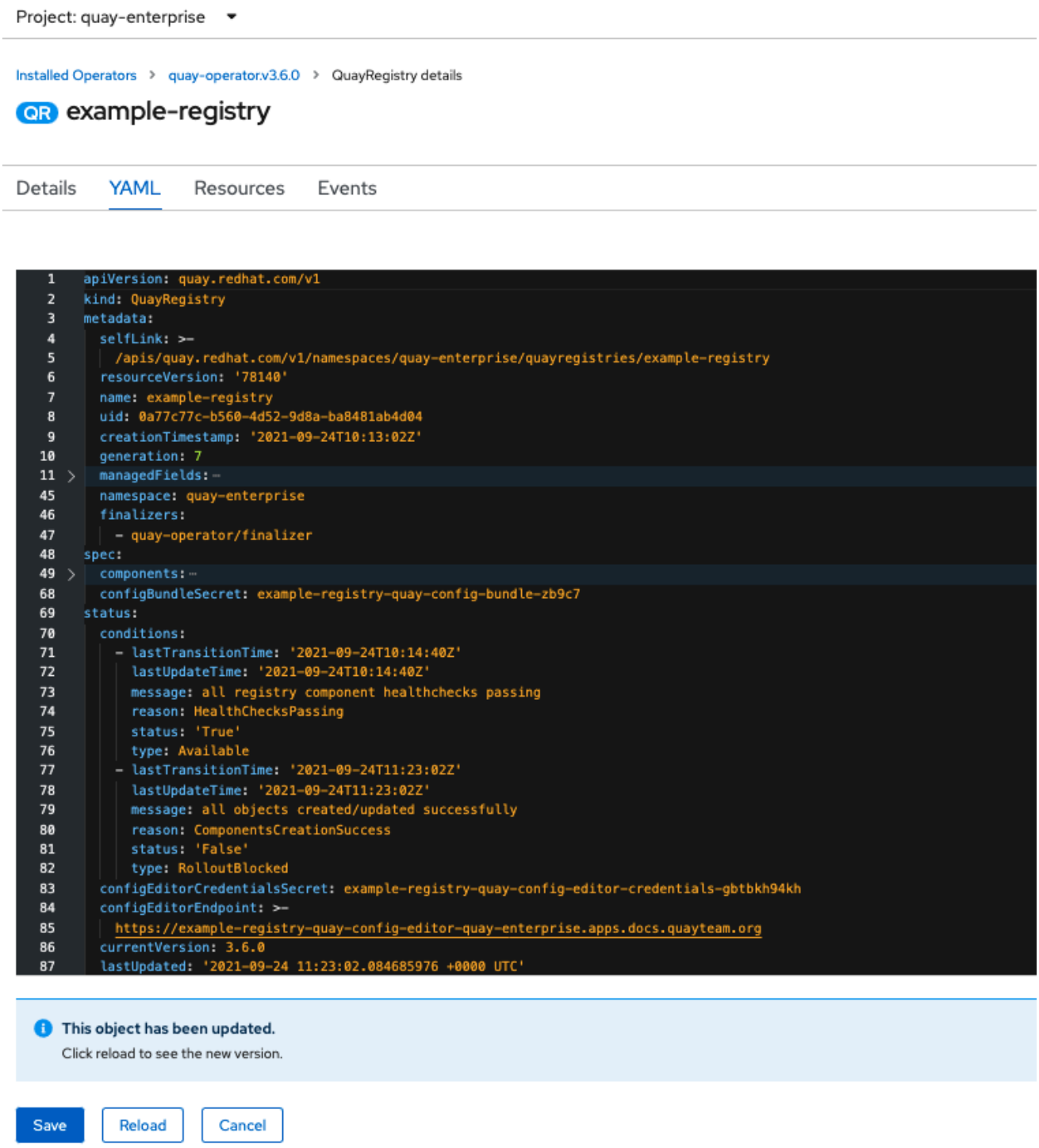
注意

使用配置工具 UI 配置 Red Hat Quay 可能会导致 registry 短时间不可用，同时应用更新的配置。

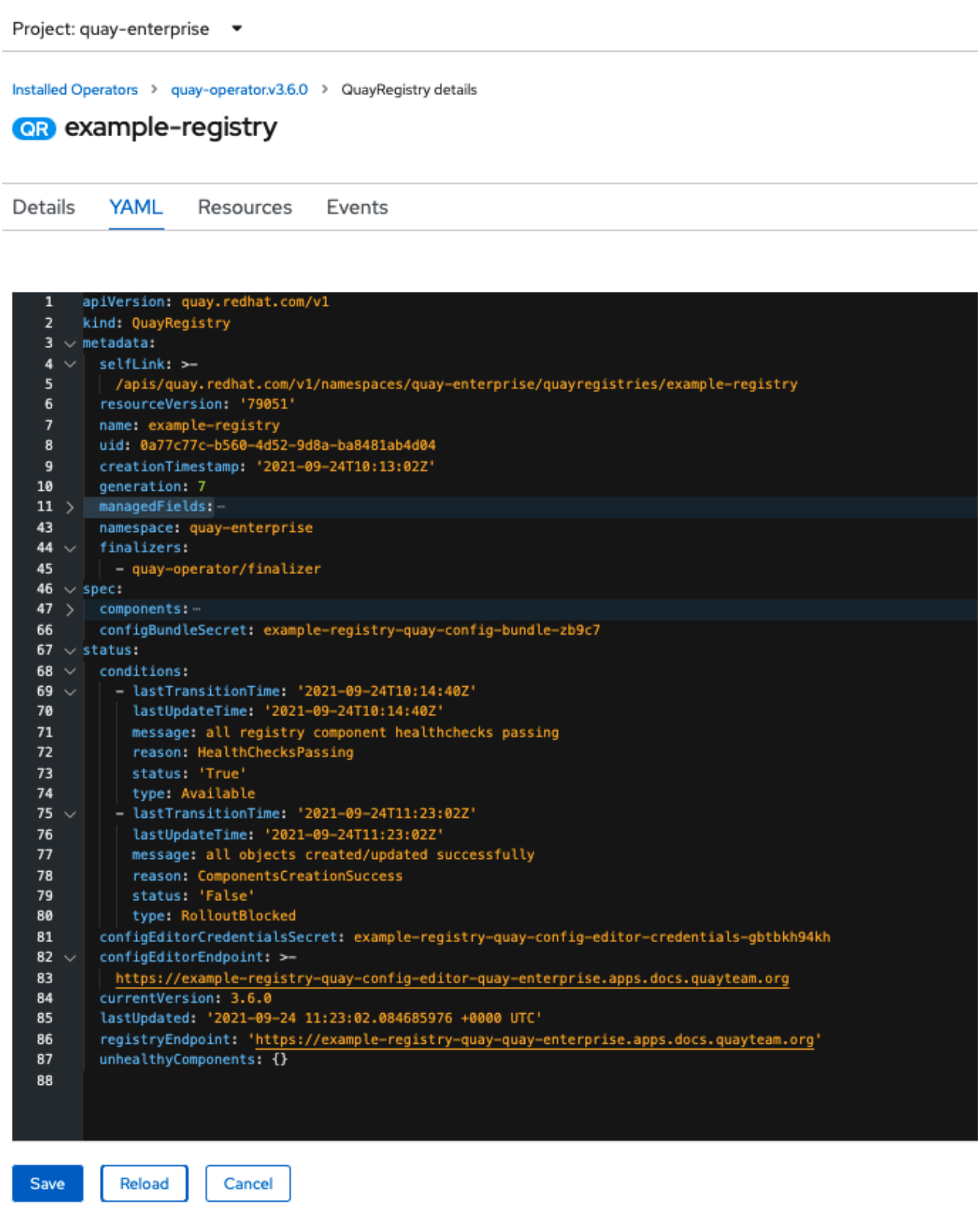
6.2. 在 UI 中监控重新配置

6.2.1. QuayRegistry 资源

重新配置 Operator 后，您可以跟踪特定 QuayRegistry 实例的 YAML 选项卡中的重新部署进度，本例中为 **example-registry** ：

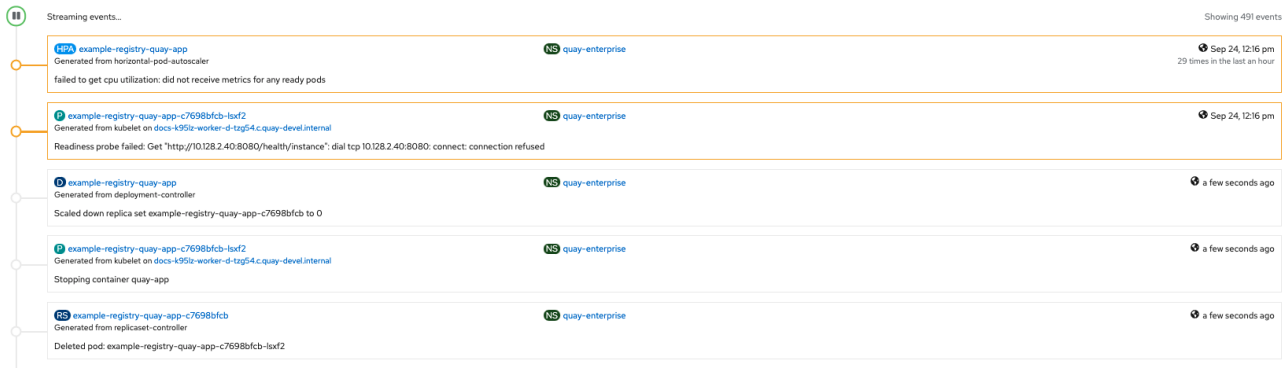


每次状态发生变化时，系统都会提示您重新载入数据以查看更新的版本。最后，Operator 将协调更改，且不会报告不健康的组件。

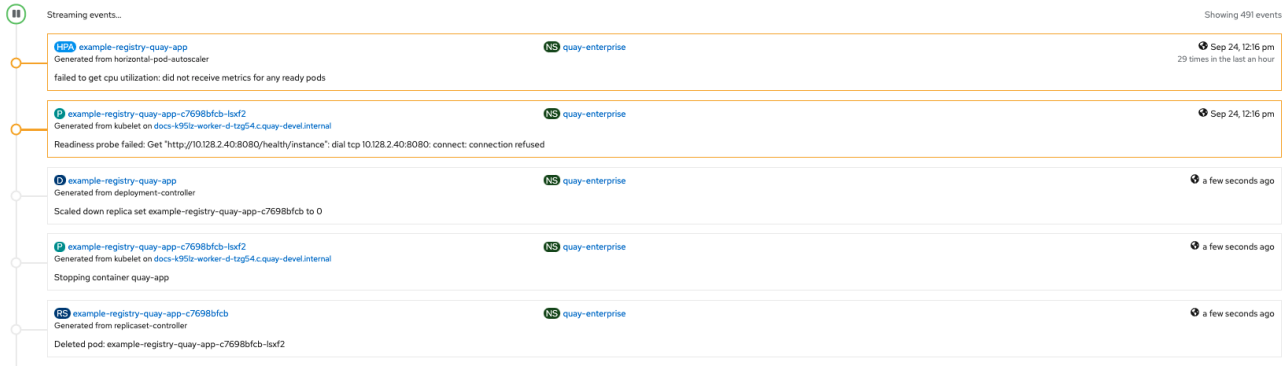


6.2.2. 事件

QuayRegistry 的 Events 选项卡显示与重新部署相关的一些事件：



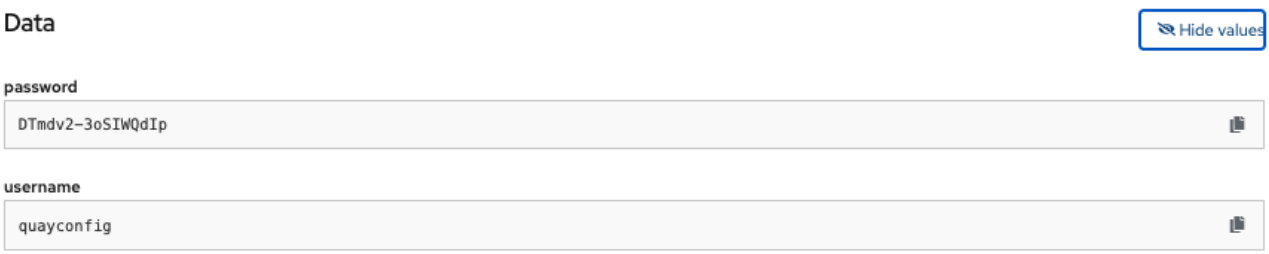
在 OpenShift 控制台 Home → Events 下，针对受重新配置影响的所有资源的流传输事件：



6.3. 在重新配置后访问更新的信息

6.3.1. 在 UI 中访问更新的配置工具凭证

由于已经为配置工具创建了新 pod，因此系统将创建一个新 secret，并在下一次尝试登录时需要使用更新的密码：



6.3.2. 在 UI 中访问更新的 config.yaml

使用 config 捆绑包访问更新的 config.yaml 文件。

1. 在 QuayRegistry 详情屏幕上点 Config Bundle Secret

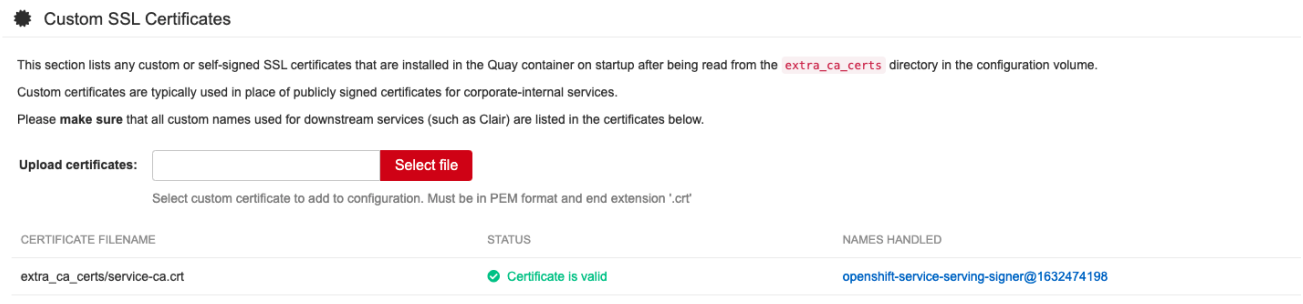
2. 在 Secret Details 屏幕的 Data 部分，点 Reveal values 查看 config.yaml 文件

3. 检查是否应用了更改。在这种情况下，4w 应该位于 TAG_EXPIRATION_OPTIONS 列表中：

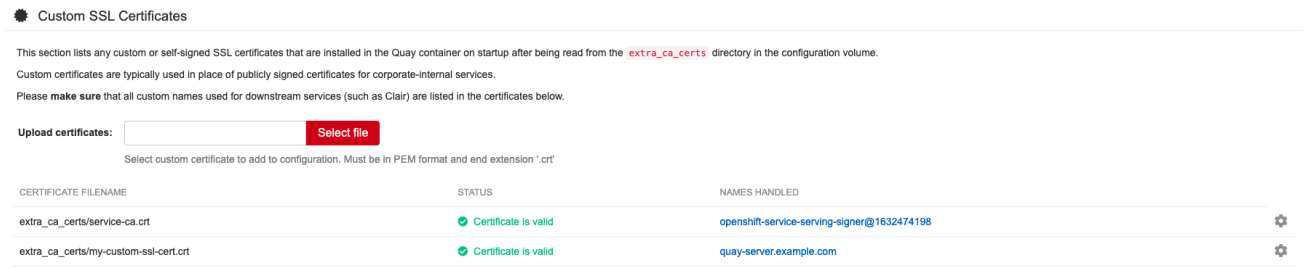
```
...
SERVER_HOSTNAME: example-quay-openshift-operators.apps.docs.quayteam.org
SETUP_COMPLETE: true
SUPER_USERS:
- quayadmin
TAG_EXPIRATION_OPTIONS:
- 2w
- 4w
...
```

6.4. 自定义 SSL 证书 UI

配置工具可用于加载自定义证书，以便于访问外部数据库等资源。选择要上传的自定义证书，确保它们采用 PEM 格式，扩展为 .crt。



配置工具还显示任何上传的证书的列表。上传自定义 SSL 证书后，它将显示在列表中：



6.5. 外部访问 REGISTRY

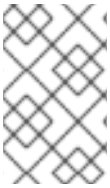
在 OpenShift 上运行时，Routes API 可用，并将自动用作受管组件。创建 QuayRegistry 后，可在 QuayRegistry 的 status 块中找到外部访问点：

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

第 7 章 QUAY OPERATOR 的功能

7.1. 控制台监控和警报

Red Hat Quay 3.6 支持从 OpenShift 控制台内部监控使用 Operator 部署的 Quay 实例。新的监控功能包括 Grafana 仪表板、对单个指标的访问以及通知的提示来经常重启 Quay Pod。

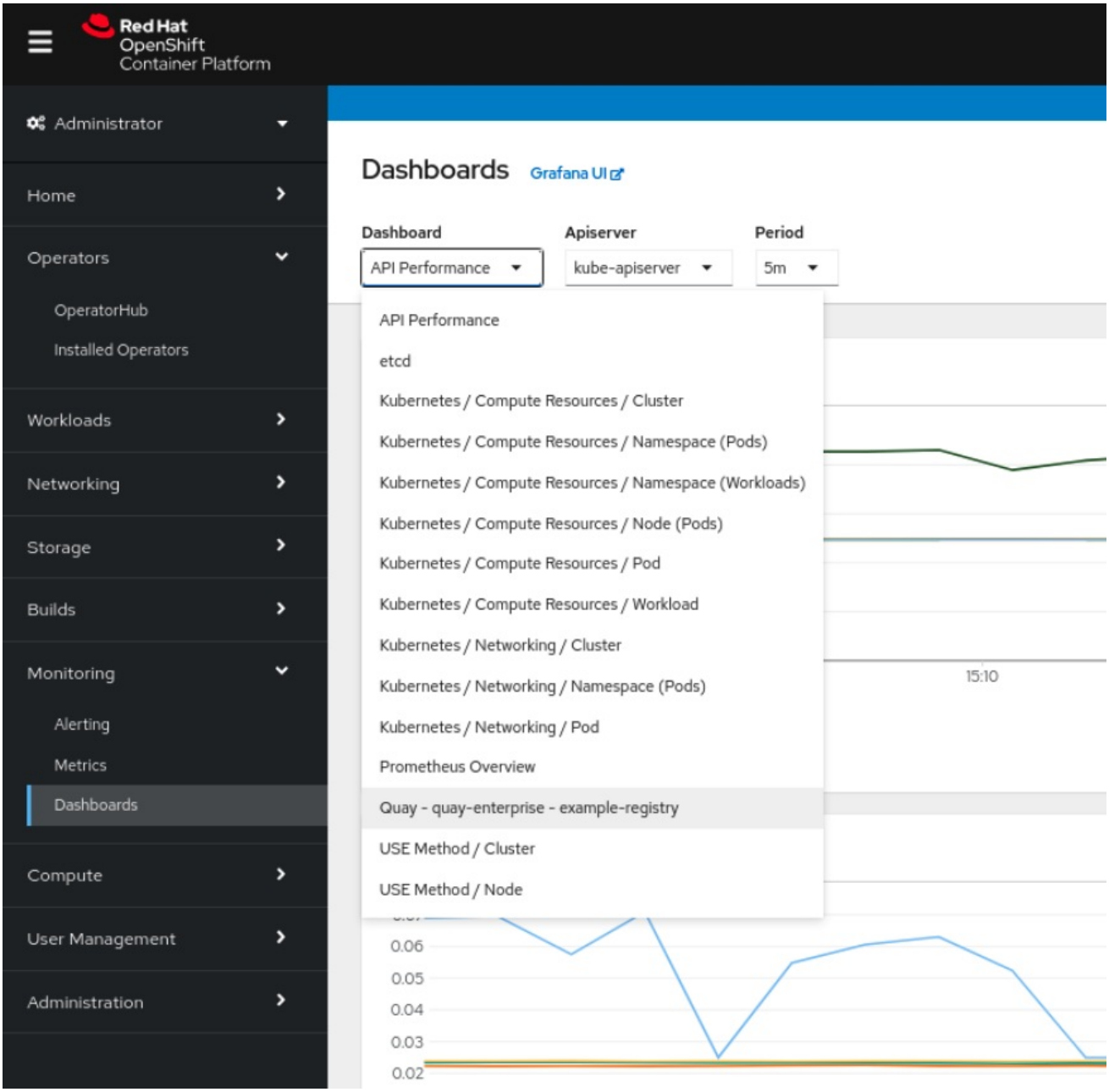


注意

要启用监控功能，必须在"所有命名空间"模式下安装 Operator。

7.1.1. Dashboard

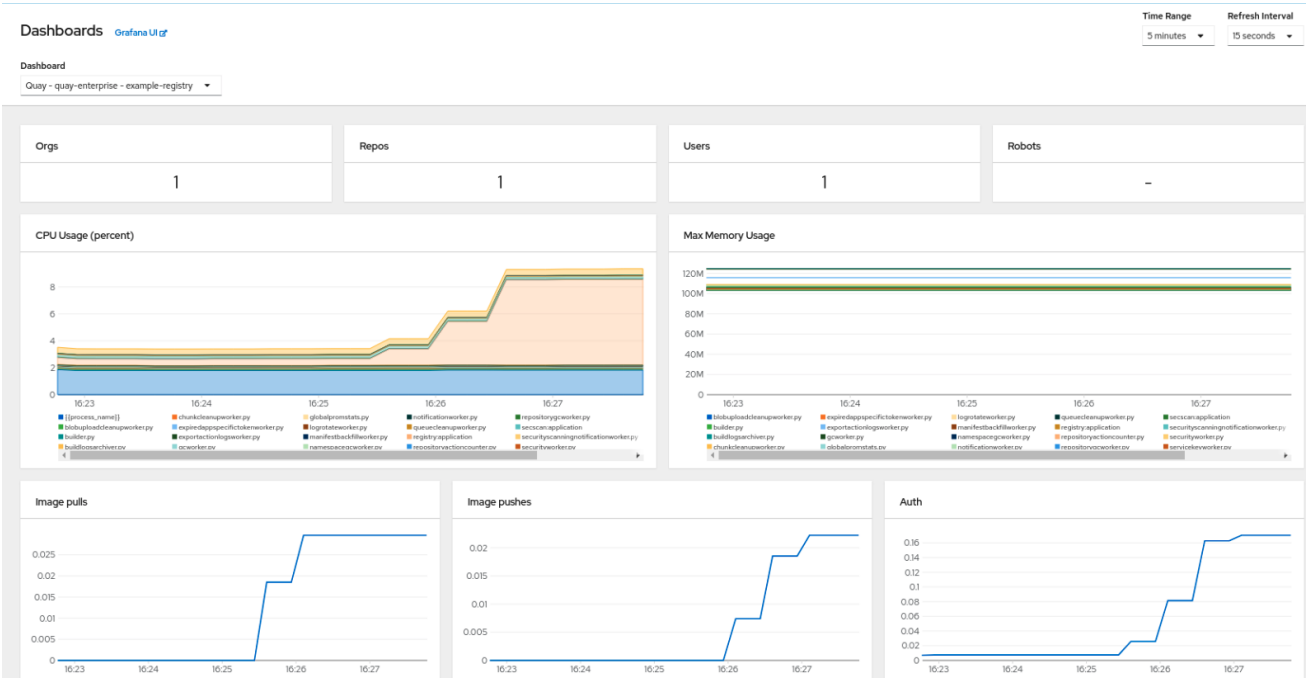
在 OpenShift 控制台中，导航到 Monitoring → Dashboards 并搜索所需 Quay registry 实例的仪表板：



仪表板显示各种统计信息，包括：

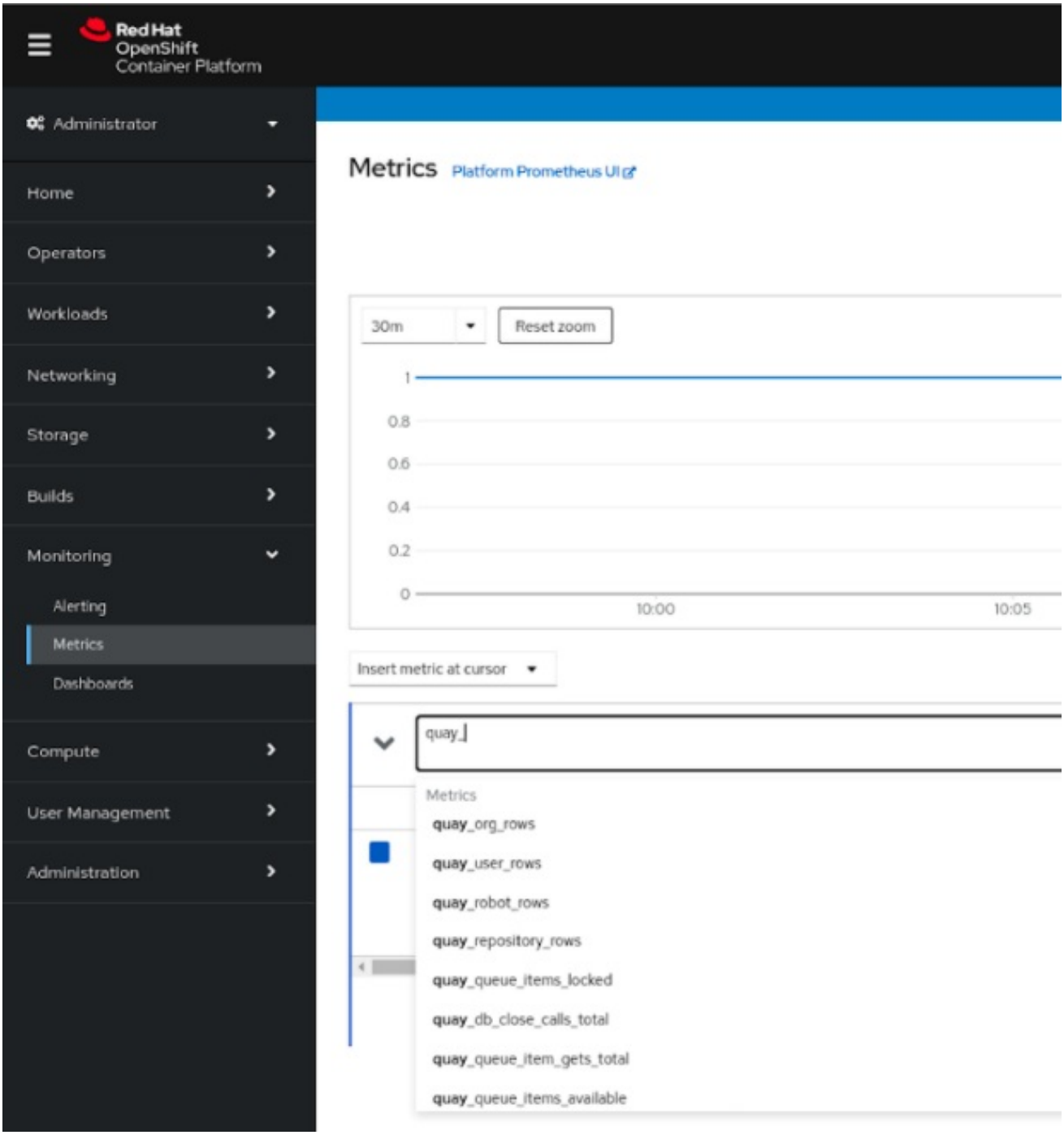
- 机构、存储库、用户和 Robot 帐户的数量
- CPU 使用量和最大内存用量
- 镜像拉取和验证请求的率

- **API 请求率**
- **延迟**

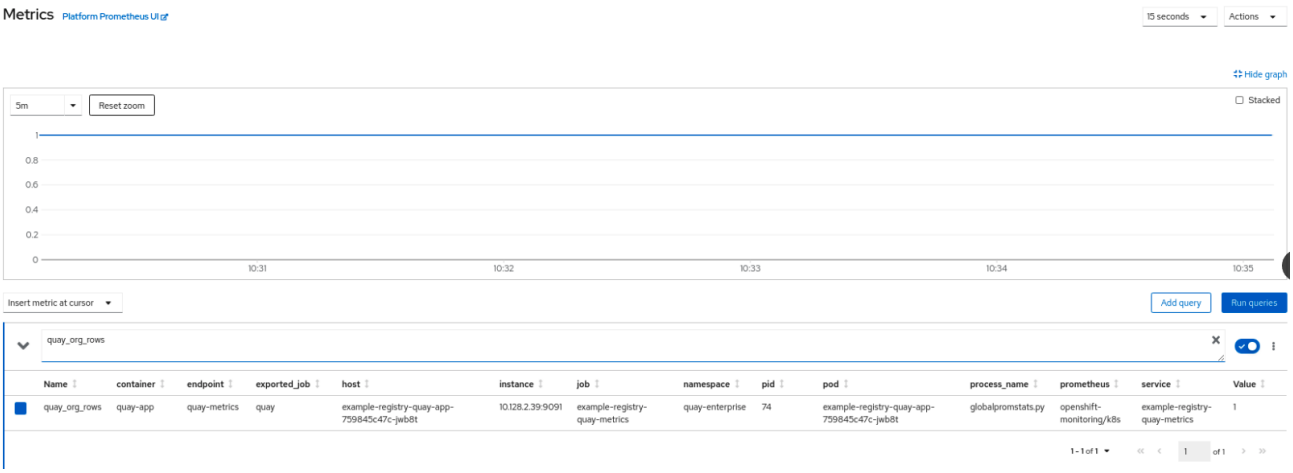


7.1.2. 指标

您可以通过在 UI 中访问 Monitoring → Metrics 来查看 Quay 仪表板后面的底层指标。在 Expression 字段中输入文本 quay_ 以查看可用指标列表：



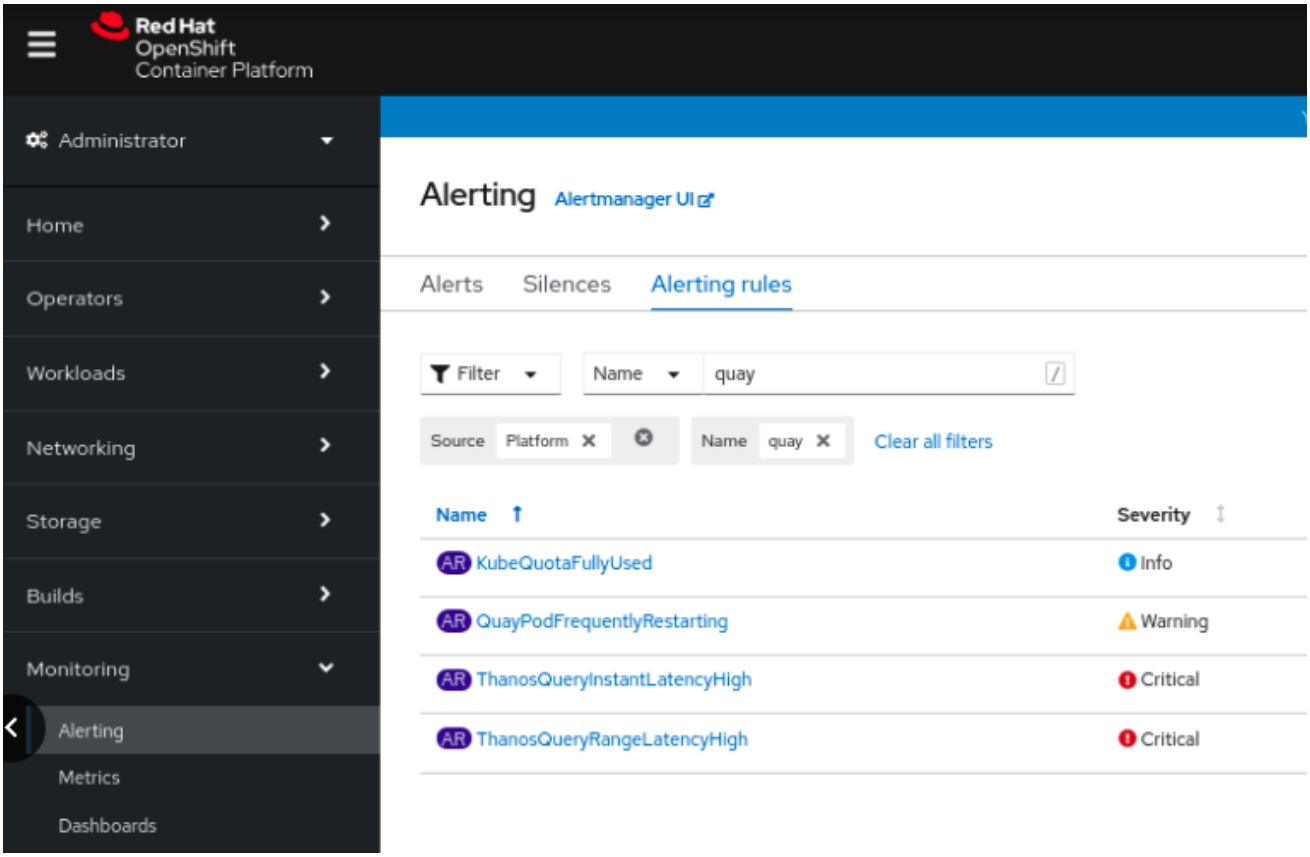
选择一个示例指标，如 **quay_org_rows** ：



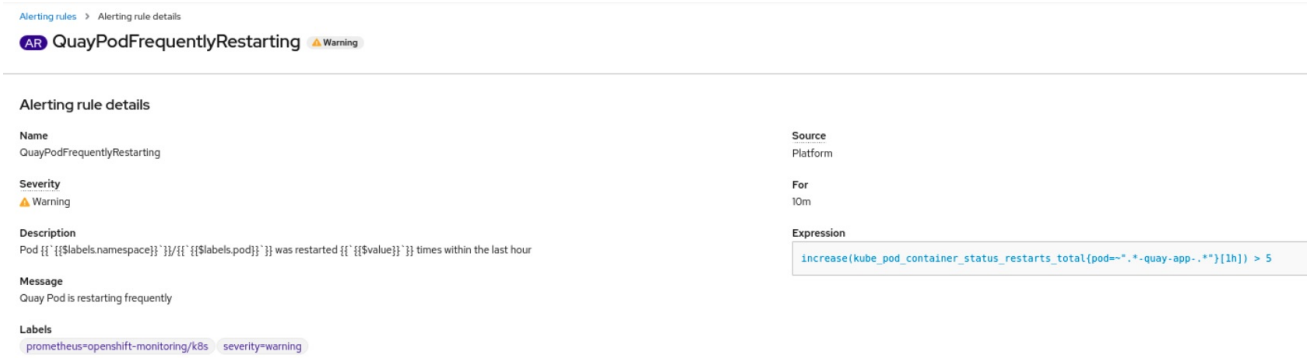
此指标显示 registry 中的组织数量，也会直接位于仪表板中。

7.1.3. 警报

如果 Quay Pod 经常重启，会引发警报。可以通过从 Consol UI 中的 Monitoring → Alerting 访问 Alerting 规则并搜索特定于 Quay 的警报来配置警报：



选择 **QuayPodFrequentlyRestarting** 规则详情来配置警报：



7.2. 在 AIR-GAPPED OPENSHIFT 集群中为 CLAIR 手动更新漏洞数据库

Clair 使用名为 updaters 的软件包，用于封装获取和解析不同漏洞数据库的逻辑。Clair 支持在不同环境中运行更新器并导入结果。这旨在支持不允许 Clair 集群直接与互联网对话的安装。

要手动更新 air-gapped OpenShift 集群中的 Clair 漏洞数据库，请使用以下步骤：

- 获取 clairctl 程序
- 检索 Clair 配置
- 使用 clairctl 从可访问互联网的 Clair 实例导出 updaters 捆绑包
- 更新 air-gapped OpenShift 集群中的 Clair 配置，以允许访问 Clair 数据库
- 从有互联网访问的系统中传输更新者捆绑包，使其可在 air-gapped 环境中可用
- 使用 clairctl 将 updaters 捆绑包导入到 air-gapped OpenShift 集群的 Clair 实例中

7.2.1. 获取 clairctl

要从 OpenShift 集群中的 Clair 部署中获取 clairctl 程序，请使用 oc cp 命令，例如：

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

对于独立 Clair 部署，请使用 podman cp 命令，例如：

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

7.2.2. 检索 Clair 配置

7.2.2.1. OpenShift 配置的 Clair

要检索使用 OpenShift Operator 部署的 Clair 实例的配置文件，请使用适当的命名空间检索和解码配置 secret，并将它保存到文件中，例如：

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

Clair 配置文件摘录如下：

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. 独立 Clair 配置

对于独立 Clair 部署，配置文件是在 podman run 命令中 CLAIR_CONF 环境变量中指定的环境变量指定的，例如：

```
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:v3.6.8
```

7.2.3. 导出更新器捆绑包

从可访问互联网的 Clair 实例中，使用正确的配置文件使用 clairctl 导出 updaters 捆绑包：

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

7.2.4. 配置 air-gapped OpenShift 集群中的 Clair 数据库访问

- 使用 kubectl 来确定 Clair 数据库服务：

```
$ 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
...
```

- 转发 **Clair 数据库端口，使其可从本地机器访问，例如：**

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

- **更新 Clair 配置文件，将多个 connstring 字段中的值替换为 localhost，例如：**

clair-config.yaml

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

注意

作为使用 kubectl port-forward 的替代选择，您可以使用 kubefwd 替代。使用此方法时，不需要修改 Clair 配置文件中的 connstring 字段，以使用 localhost。

7.2.5. 将 updaters 捆绑包导入到 air-gapped 环境中

将 updaters 捆绑包传送到 air-gapped 环境后，使用 clairctl 将捆绑包导入到 OpenShift Operator 部署的 Clair 数据库中：

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

7.3. FIPS 就绪情况和合规性

NIST（美国国家标准与技术研究所开发的系统信息处理标准）被认为是安全和保护敏感数据的金级标准，特别是在大型银行、医疗保健和公共领域。Red Hat Enterprise Linux 和 Red Hat OpenShift Container Platform 通过提供一个 FIPS 模式来支持这个标准，该系统只允许使用某些 FIPS 验证的加密模块，如 openssl。这样可确保 FIPS 合规性。

从版本 3.5 开始，Red Hat Quay 支持以 FIPS 模式在 RHEL 和 OCP 上运行。另外，Red Hat Quay 本身还承诺只使用验证或者被 NIST 验证的加密库。Red Hat Quay 3.5 根据 RHEL 8.3 加密库有待处理的 FIPS 140-2 验证。完成该验证后，Red Hat Quay 将正式的 FIPS 兼容。

第 8 章 高级概念

8.1. 在基础架构节点上部署 QUAY

默认情况下，在使用 Operator 部署 registry 时，Quay 相关的 pod 会放置在任意 worker 节点上。OpenShift Container Platform 文档演示了如何使用机器集配置节点来只托管基础架构组件（请参阅 https://docs.openshift.com/container-platform/4.7/machine_management/creating-infrastructure-machinesets.html）。

如果您不使用 OCP MachineSet 资源来部署 infra 节点，本节演示了如何为基础架构需要手动标记和污点节点。

在配置了基础架构节点（手动或使用机器集）后，您可以使用节点选择器和容限来控制 Quay pod 在这些节点中的放置。

8.1.1. 基础架构使用的标签和污点节点

在本例中，集群中有三个 master 节点和 6 个 worker 节点：

```
$ 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
```

为基础架构添加最后三个 worker 节点添加标签：

```
$ 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=
```

现在，当您列出集群中的节点时，最后的 3 个 worker 节点会带有 infra 的角色：

```
$ 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
```

当将 infra 节点分配为 worker 时，用户工作负载可能会意外地分配给 infra 节点。要避免这种情况，您可以将污点应用到 infra 节点，然后为您要控制的 pod 添加容限。

```
$ 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. 使用节点选择器和容限创建项目

如果您已经使用 Quay Operator 部署了 Quay，请删除已安装的 Operator 和您创建的任何特定命名空间。

创建 Project 资源，指定节点选择器和容限，如下例所示：

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"}
      ]
```

使用 **oc apply** 命令来创建项目：

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

quay-registry 命名空间中创建的任何后续资源现在都应调度到专用基础架构节点。

8.1.3. 在命名空间中安装 Quay Operator

安装 **Quay Operator** 时，在本例中为 **quay-registry** 时明确指定适当的项目命名空间。这将导致 **Operator pod** 本身登录三个基础架构节点之一：

```
$ 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. 创建 registry

如前所述创建注册表，然后等待部署就绪。当您列出 **Quay pod** 时，您应该会看到它们只调度到您为基础架构考虑的三个节点上：

```
$ 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. 在单一命名空间中安装 OPERATOR 时启用监控

当在单一命名空间中安装 Red Hat Quay Operator 时，监控组件为非受管状态。要配置监控，您需要为 OpenShift Container Platform 中的用户定义的命名空间启用它。如需更多信息，请参阅 配置监控堆栈 和 为用户定义的项目启用监控的 OCP 文档。

以下步骤演示了如何根据 OCP 文档为 Quay 配置监控。

8.2.1. 创建集群监控配置映射

1. 检查 **cluster-monitoring-config ConfigMap** 对象是否存在：

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

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

2. 如果 **ConfigMap** 对象不存在：

   a. 创建以下 **YAML** 清单。在本例中，该文件名为 **cluster-monitoring-config.yaml** ：

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

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

   b. 创建 **ConfigMap** 对象：

```
$ 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. 创建用户定义的工作负载监控配置映射

1. 检查 **user-workload-monitoring-config ConfigMap** 对象是否存在：

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

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

2. **如果 ConfigMap 对象不存在：**

a. 创建以下 **YAML 清单。在本例中，该文件名为 user-workload-monitoring-config.yaml ：**

```
$ 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. 创建 **ConfigMap** 对象：

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

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

8.2.3. 为用户定义的项目启用监控

1. 检查用户定义的项目的监控是否正在运行：

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

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

2. 编辑 **cluster-monitoring-config ConfigMap：**

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

3. 将 **enableUserWorkload: true** 设置为在集群中为用户定义的项目启用监控：

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

4. 保存文件以应用更改，然后检查适当的 **pod** 是否正在运行：

```
$ 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. 创建 Service 对象以公开 Quay 指标

1. 为 **Service** 对象创建 **YAML** 文件：

```
$ 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. 创建 **Service** 对象：

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

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

8.2.5. 创建 ServiceMonitor 对象

通过创建 **ServiceMonitor** 资源，将 **OpenShift Monitoring** 配置为提取指标。

1. 为 **ServiceMonitor** 资源创建 **YAML** 文件：

```
$ 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
```

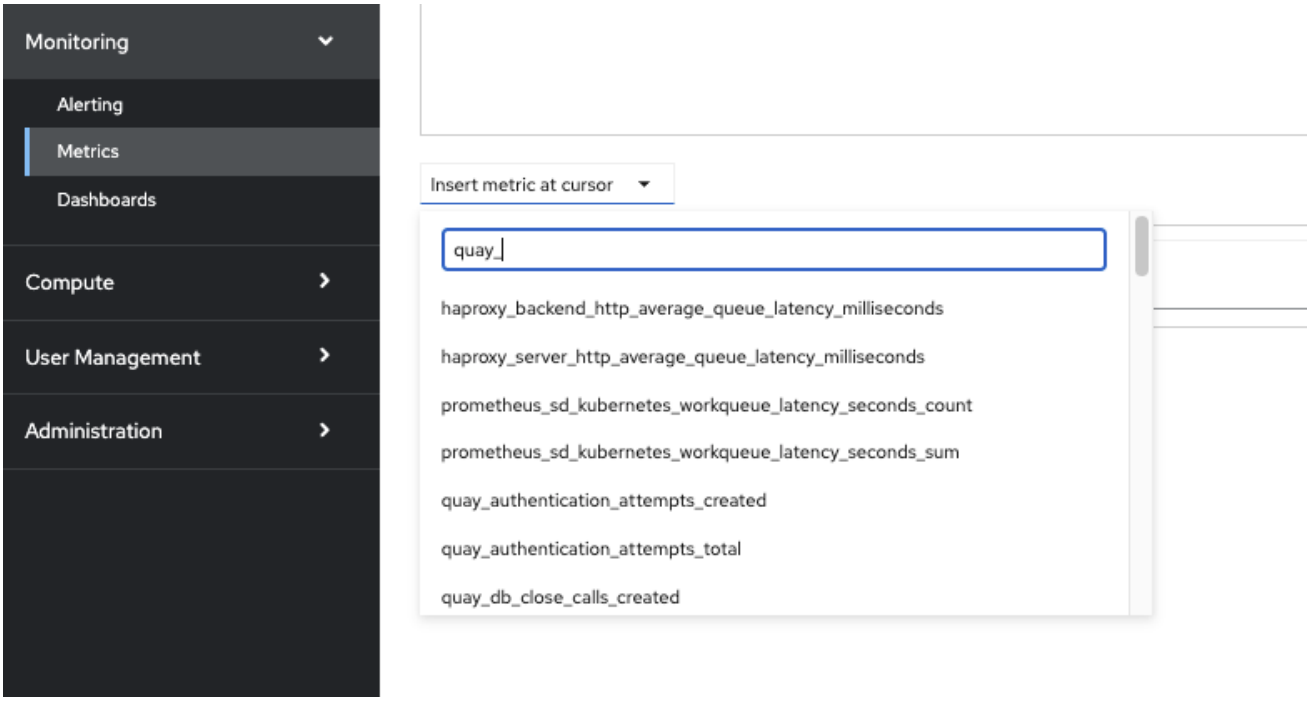
2. 创建 **ServiceMonitor**：

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

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

8.2.6. 查看 OpenShift 中的指标

您可以在 OpenShift 控制台中访问 **Monitoring → Metrics** 下的指标。在 **Expression** 字段中输入文本 **quay_** 以查看可用指标列表：



例如，如果您已将用户添加到 registry，请选择 **quay-users_rows** 指标：

8.3. 重新定义受管存储大小

Quay Operator 在创建 NooBaa 对象(50 Gib)时，使用 RHOCS 提供的默认值来创建默认对象存储。扩展此存储的方法有两种：您可以重新定义现有 PVC 大小，或将更多 PVC 添加到新的存储池中。

8.3.1. resize Noobaa PVC

1. 登录到 OpenShift 控制台并选择 Storage → Persistent Volume Claims。
2. 选择名为 noobaa-default-backing-store-noobaa-pvc-* 的 PersistentVolumeClaim。
3. 在 Action 菜单中，选择 Expand PVC。
4. 输入 Persistent Volume Claim 的新大小，然后选择 Expand。

几分钟后（取决于 PVC 的大小），扩展的大小应该在 PVC 的 Capacity 字段中反映。

注意

扩展 CSI 卷只是一个技术预览功能。如需更多信息，请参阅 https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/storage/expanding-persistent-volumes。

8.3.2. 添加另一个存储池

1. 登录到 OpenShift 控制台并选择 Networking → Routes。确保已选中 openshift-storage 项目。
2. 单击 noobaa-mgmt Route 的 Location 字段。
3. 登录到 Noobaa 管理控制台。
4. 在主仪表板上，在 Storage Resources 下，选择 Add Storage Resources。

5. 选择 **Deploy Kubernetes Pool**

6. **输入新的池名称。点击 Next。**

7. 选择管理池的 **Pod** 数量并设置每个节点的大小。点击 **Next。**

8. 单击 **Deploy。**

几分钟后，额外的存储池将添加到 Noobaa 资源中，可供 Red Hat Quay 使用。

8.4. 自定义默认 OPERATOR 镜像

注意

对于生产环境 **Quay** 环境，不支持使用此机制，强烈建议仅用于开发/测试。使用 **Quay Operator** 的非默认镜像时，无法保证部署正常工作。

在某些情况下，覆盖 Operator 使用的默认镜像可能会很有用。这可以通过在 Quay Operator ClusterServiceVersion 中设置一个或多个环境变量来实现。

8.4.1. 环境变量

Operator 中使用以下环境变量覆盖组件镜像：

环境变量	组件
RELATED_IMAGE_COMPONENT_QUAY	**base**
RELATED_IMAGE_COMPONENT_CLAIR	**Clair**
RELATED_IMAGE_COMPONENT_POSTGRES	**Postgres** 和 **clair** 数据库
RELATED_IMAGE_COMPONENT_REDIS	**redis**

注意

清单 必须引用 覆盖镜像(@sha256:)，而不是标签(:latest)。

8.4.2. 将覆盖应用到正在运行的 Operator

当通过 Operator Lifecycle Manager(OLM)在集群中安装 Quay Operator 时，受管组件容器镜像可以通过修改 ClusterServiceVersion 对象来轻松覆盖，这是 OLM 在集群中运行 Operator 的表示。使用 Kubernetes UI 或 kubectl/oc 查找 Quay Operator 的 ClusterServiceVersion ：

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

使用 UI、oc edit 或任何其他方法修改 Quay ClusterServiceVersion，使其包含上方概述的环境变量以指向覆盖镜像：

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

```
- name: RELATED_IMAGE_COMPONENT_QUAY
  value:
quay.io/projectquay/quay@sha256:c35f5af964431673f4ff5c9e90bdf45f19e38b8742b5903d41c1
0cc7f6339a6d
- name: RELATED_IMAGE_COMPONENT_CLAIR
  value:
quay.io/projectquay/clair@sha256:70c99feceb4c0973540d22e740659cd8d616775d3ad1c1698dd
f71d0221f3ce6
- name: RELATED_IMAGE_COMPONENT_POSTGRES
  value: centos/postgresql-10-
centos7@sha256:de1560cb35e5ec643e7b3a772ebaac8e3a7a2a8e8271d9e91ff023539b4dfb33
- name: RELATED_IMAGE_COMPONENT_REDIS
  value: centos/redis-32-
centos7@sha256:06dbb609484330ec6be6090109f1fa16e936afcf975d1cbc5fff3e6c7cae7542
```

请注意，这是在 Operator 级别上完成的，因此每个 QuayRegistry 都使用相同的覆盖进行部署。

8.5. AWS S3 CLOUDFRONT

如果您将 AWS S3 CloudFront 用于后端 registry 存储，请指定私钥，如下例所示：

```
$ 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
```

第 9 章 在 OPENSHIFT CONTAINER PLATFORM 部署中备份和恢复 RED HAT QUAY

使用本节中的内容在 OpenShift Container Platform 部署中备份和恢复 Red Hat Quay。

9.1. 备份 RED HAT QUAY

此流程只适用于 OpenShift Container Platform 和 NooBaa 部署。

先决条件

- 在 OpenShift Container Platform 上部署 Red Hat Quay。

流程

1. 通过导出 QuayRegistry 自定义资源来备份它：

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

2. 编辑生成的 quayregistry.yaml 并删除 status 部分及以下元数据字段：

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

3. 备份受管密钥 secret：

注意

如果您正在运行早于 Red Hat Quay 3.0 的版本，可以跳过这一步。首次部署 Quay 时会自动生成一些 secret。它们存储在 QuayRegistry 命名空间中的名为 &lt ;quay-registry-name>-quay-registry-managed-secret-keys 的 secret 中。

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

4. 编辑生成的 managed-secret-keys.yaml 文件并删除所有所有者引用。您的 managed-secret-keys.yaml 文件应类似于如下：

```
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 属性下的所有信息都应保持不变。

5. 备份当前的 Quay 配置：

```
$ 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. 备份在 Quay pod 中挂载的 /conf/stack/config.yaml 文件：

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

7. 缩减 Quay Operator：

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

8. 缩减 Quay 命名空间：

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace> -l quay-component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

9. 等待 registry-quay-app pod 消失。您可以运行以下命令来检查其状态：

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

输出示例：

```
registry-quay-config-editor-77847fc4f5-nsbbv   1/1     Running          0          9m1s
registry-quay-database-66969cd859-n2ssm        1/1     Running          0          6d1h
registry-quay-mirror-758fc68ff7-5wxlp          1/1     Running          0          8m29s
registry-quay-mirror-758fc68ff7-lbl82          1/1     Running          0          8m29s
registry-quay-redis-7cc5f6c977-956g8           1/1     Running          0          5d21h
```

10. 识别 Quay PostgreSQL pod 名称：

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

exampe 输出：

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

1. 获取 Quay 数据库名称：

```
$ 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
```

2. 下载备份数据库：

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

3. 解码并导出 AWS_ACCESS_KEY_ID ：

```
$ 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)
```

4. 解码并导出 AWS_SECRET_ACCESS_KEY_ID ：

```
$ 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)
```

5. 创建新目录并将所有 blob 复制到其中：

```
$ 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
```

注意

您还可以使用 rclone 或 sc3md，而不是 AWS 命令行工具。

1. 扩展 Quay Operator:

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

2. 扩展 Quay 命名空间：

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-namespace> -l quay-component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

3. 检查 Operator 的状态：

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

输出示例：

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

```
  namespace: <quay-namespace>
  ...
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
    0          5d21h
```

9.2. 恢复 RED HAT QUAY

当 Red Hat Quay Operator 管理数据库时，这个流程用于恢复 Red Hat Quay。它应该在执行 Quay registry 备份后执行。

先决条件

- Red Hat Quay 使用 Quay Operator 在 OpenShift Container Platform 上部署。
- 您的 Red Hat Quay 数据库已被备份。

流程

1. 恢复备份的 Quay 配置和随机生成的密钥：

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

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

注意

如果您收到错误 Error from server (AlreadyExists)：创建 "./config-bundle.yaml": secrets "config-bundle-secret" already exists 时的错误，您必须使用 $ oc delete Secret config-bundle-secret -n <quay -namespace > 重新创建它。

2. **恢复 QuayRegistry 自定义资源：**

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

3. **缩减 Quay Operator：**

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

4. **缩减 Quay 命名空间：**

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace> -l quay-component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

5. **识别您的 Quay 数据库 pod：**

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

输出示例：

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

6. **通过从本地环境复制到本地环境和 pod 来上传备份：**

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

7. **打开到数据库的远程终端：**

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

8. **Enter psql:**

```
bash-4.4$ psql
```

9. **您可以运行以下命令来列出数据库：**

```
postgres=# \l
```

输出示例：

```
                                  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 |
```

10. **丢弃数据库：**

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

输出示例：

```
DROP DATABASE
```

11. **退出 postgres CLI 以重新输入 bash-4.4 ：**

```
\q
```

12. **将您的 PostgreSQL 数据库重定向到备份数据库：**

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

13. **退出 bash：**

```
sh-4.4$ exit
```

14. **导出 AWS_ACCESS_KEY_ID ：**

```
$ 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)
```

15. **导出 AWS_SECRET_ACCESS_KEY ：**

```
$ 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)
```

16. **运行以下命令将所有 blob 上传到存储桶：**

```
$ 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}')
```

17. **扩展 Quay Operator:**

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

18. **扩展 Quay 命名空间：**

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-namespace> -l
quay-component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-
namespace>
```

19. **检查 Operator 的状态，并确保它重新上线：**

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

输出示例：

```
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
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
    0          5d21h
```

第 10 章 升级 QUAY OPERATOR 概述

Quay Operator 遵循一个 *同步的版本* 方案，这意味着每个 Operator 版本都绑定到 Quay 版本及其管理的组件。QuayRegistry 自定义资源中没有设置要部署的 Quay 版本的字段。Operator 只了解如何部署所有组件的单一版本。选择此方案以确保所有组件都良好工作，并降低 Operator 如何管理 Kubernetes 上许多不同版本的 Quay 的生命周期的复杂性。

10.1. OPERATOR LIFECYCLE MANAGER

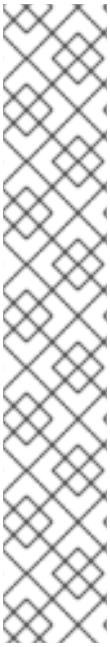
应使用 Operator Lifecycle Manager(OLM) 来安装和升级 Quay Operator。当使用默认的 approvalStrategy 创建订阅时，OLM 会在新版本可用时自动升级 Quay Operator。

警告

当 Quay Operator 通过 Operator Lifecycle Manager 安装时，可能会将其配置为支持自动或手动升级。这个选项显示在安装过程中 Quay Operator 的 Operator Hub 页面中。它还可以通过 approvalStrategy 字段在 Quay Operator Subscription 对象中找到。选择 Automatic 意味着每当发布新 Operator 版本时自动升级您的 Quay Operator。如果这不是理想的选择，则应选择 手动批准策略。

10.2. 升级 QUAY OPERATOR

在 OpenShift 上升级已安装的 Operator 的标准方法包括在 升级已安装的 Operator 中。



注意

通常，Red Hat Quay 只支持从一个次版本升级到下一个次版本，例如 3.4 → 3.5。但是，对于 3.6，支持多个升级路径：

- 3.3.z → 3.6
- 3.4.z → 3.6
- 3.5.z → 3.6

有关 Quay 独立部署的用户，要升级到 3.6，请参阅 独立升级 指南。

10.2.1. 升级 Quay

要将 Quay 从一个次要版本升级到下一个次要版本，如 3.4 → 3.5，您需要更改 Quay Operator 的更新频道。

对于 z 流升级，例如 3.4.2 → 3.4.3，会在用户在安装过程中初始选择的主次频道中发布更新。执行 z 流升级的步骤取决于以 上所述的批准Strategy。如果批准策略被设置为 Automatic，Quay Operator 会自动升级到最新的 z 流。这会导致自动将 Quay 更新部署到较新的 z 流，而无需停机。否则，必须手动批准更新，然后才能开始安装。

10.2.2. 有关直接从 3.3.z 或 3.4.z 升级到 3.6 的备注

10.2.2.1. 启用边缘路由升级

- 在以前的版本中，当运行启用了边缘路由的 3.3.z 版本时，用户无法升级到 Red Hat Quay 的 3.4.z 版本。这个问题已通过 Red Hat Quay 3.6 的发行版本解决。
- 当从 3.3.z 升级到 3.6 时，如果在 Red Hat Quay 3.3.z 部署中将 tls.termination 设置为 none，它将改为使用 TLS 边缘终止的 HTTPS，并使用默认集群通配符证书。例如：

```
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:
...
```

10.2.2.2. 使用自定义 TLS 证书/密钥对升级，无需主题备用名称

当直接从 Red Hat Quay 3.3.4 升级到 Red Hat Quay 3.6 时，用户可以使用自己的 TLS 证书/密钥对而无需 Subject Alternative Name(SAN)的用户有问题。在升级到 Red Hat Quay 3.6 时，部署会被阻止，来自 Quay Operator pod 日志的错误消息表示 Quay TLS 证书必须具有 SAN。

如果可能，您应该使用 SAN 中的正确主机名重新生成 TLS 证书。一个可能的解决方法涉及在 quay-app、quay-upgrade 和 quay-config-editor pod 中定义环境变量来启用 CommonName 匹配：

```
GODEBUG=x509ignoreCN=0
```

GODEBUG=x509ignoreCN=0 标志可在没有 SANs 时将 X.509 证书上的 CommonName 字段视为主机名的传统行为。但是，我们不推荐使用这个临时解决方案，因为它不会在重新部署之间保留。

10.2.2.3. 使用 Quay Operator 从 3.3.z 或 3.4.z 升级到 3.6 时配置 Clair v4

要在 OpenShift 上的新 Red Hat Quay 部署上设置 Clair v4，强烈建议使用 Quay Operator。默认情况下，Quay Operator 将安装或升级 Clair 部署以及 Red Hat Quay 部署，并自动配置 Clair 安全扫描。

有关在 OpenShift 上设置 Clair v4 的说明，请参阅在 Red Hat Quay OpenShift 部署 上设置 Clair。

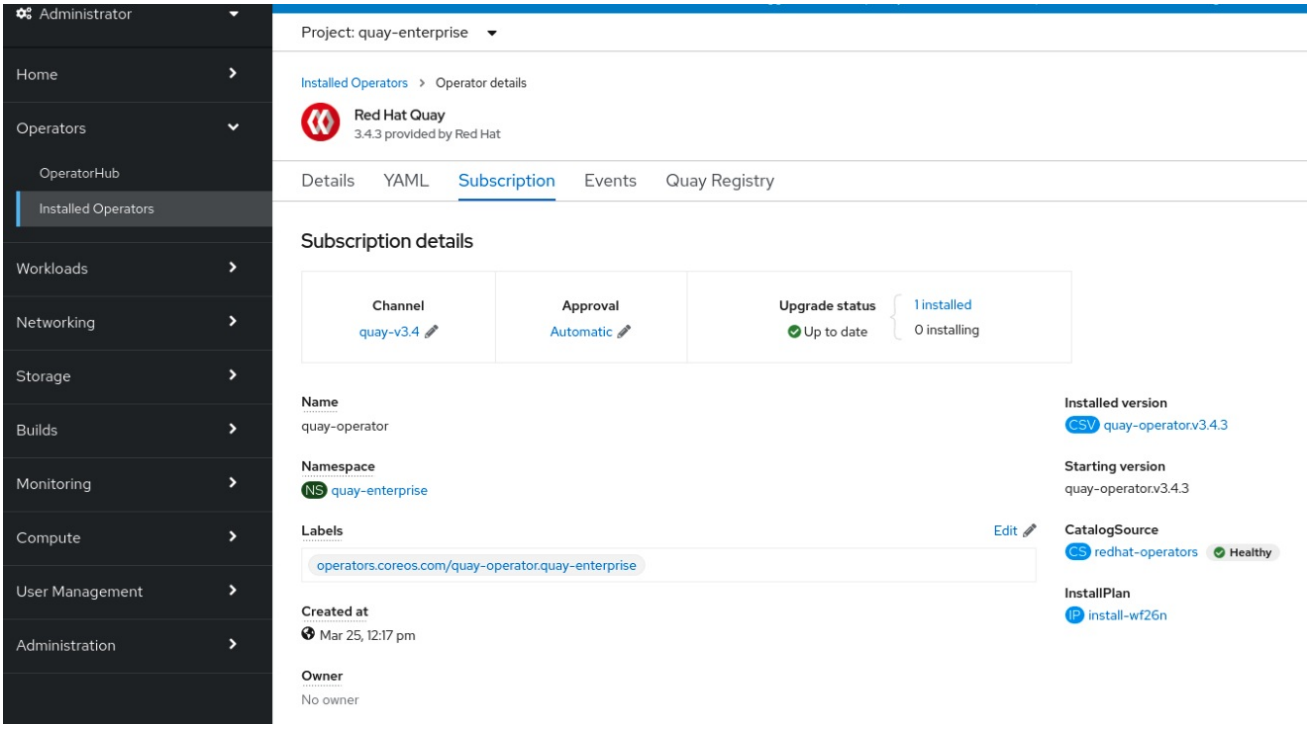
10.2.3. 更改 Operator 的更新频道

已安装的 Operator 的订阅指定一个更新频道，用于跟踪和接收 Operator 的更新。要升级 Quay Operator 以开始跟踪并从更新频道接收更新，请更改已安装 Quay Operator 的 Subscription 选项卡中的更新频道。对于带有 自动批准策略 的订阅，升级会自动开始，并可在列出 Installed Operators 的页面上进行监控。

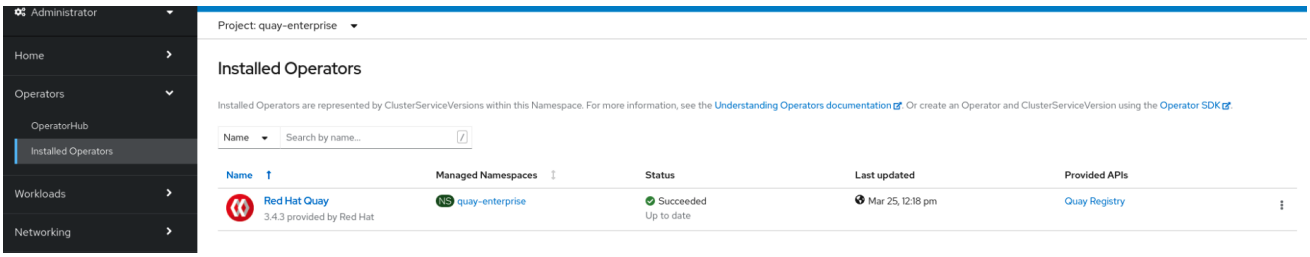
10.2.4. 手动批准待处理的 Operator 升级

如果已安装的 Operator 的订阅被设置为 Manual，则当其当前更新频道中发布新更新时，在开始安装前必须手动批准更新。如果 Quay Operator 有待处理的升级，则此状态将显示在 Installed Operators 列表中。在 Quay Operator 的 Subscription 选项卡中，您可以预览安装计划，并查看列出可用于升级的资源。如果满意，点 Approve 并返回到列出 Installed Operators 的页面，以监控升级的进度。

下图显示了 UI 中的 Subscription 选项卡，包括 更新频道、批准策略、Upgrade 状态和 InstallPlan：



Installed Operators 列表提供当前 Quay 安装的高级别概述：



10.3. 升级 QUAYREGISTRY

当 Quay Operator 启动时，它会立即查找它配置为监视的命名空间中的任何 QuayRegistries。当找到时，会使用以下逻辑：

- 如果 status.currentVersion 没有设置，则正常协调。
- 如果 status.currentVersion 等于 Operator 版本，请正常协调。
- 如果 status.currentVersion 没有等于 Operator 版本，请检查是否升级它。如果可以，执行升级任务，并在完成后将 status.currentVersion 设置为 Operator 的版本。如果无法升级，返回错误并只保留 QuayRegistry 及其部署的 Kubernetes 对象。

10.4. 在 QUAY 3.6 中启用功能

10.4.1. 控制台监控和警报

在 OpenShift 控制台中对监控 Quay 3.6 的支持要求在所有命名空间中安装 Operator。如果您之前在特定命名空间中安装了 Operator，请删除 Operator 本身，并在升级完成后为所有命名空间重新安装它。

10.4.2. OCI 和 Helm 支持

现在，Red Hat Quay 3.6 中默认启用对 Helm 和一些 OCI 工件的支持。如果要显式启用该功能，例如，如果您是从没有默认启用的版本升级，则需要重新配置 Quay 部署，以使用以下属性启用 OCI 工件：

```
FEATURE_GENERAL_OCI_SUPPORT: true
```

10.5. 升级 QUAYECO 系统

以前的 Operator 版本支持升级，这些 Operator 将 QuayEcosystem API 用于一组有限的配置。为确保迁移不会意外发生，需要将一个特殊的标签应用到 QuayEco 系统 供迁移。为 Operator 创建一个新的 QuayRegistry，但旧的 QuayEcosystem 将保留到手动删除，以确保您可以回滚并仍能访问 Quay，以出现问题。要将现有的 QuayEcosystem 迁移到新的 QuayRegistry 中，请按照以下步骤操作：

1. 将 "quay-operator/migrate": "true" 添加到 QuayEcosystem 的 metadata.labels。

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

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

2. 等待 QuayEcosystem 使用相同的 metadata.name 创建 QuayRegistry。QuayEcosystem 将标记为标签 "quay-operator/migration-complete": "true "。

3. 设置了新的 QuayRegistry 的 status.registryEndpoint 后，访问 Quay 并确认所有数据和设置已成功迁移。

4. 当您确定一切正常工作时，您可以删除 QuayEcosystem 和 Kubernetes 垃圾回收会清理所有旧资源。

10.5.1. 恢复 QuayEco 系统升级

如果在从 QuayEcosystem 自动升级到 QuayRegistry 时出现问题，请按照以下步骤恢复至使用 QuayEcosystem ：

1. 使用 UI 或 kubectl 来删除 QuayRegistry ：

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

2. 如果使用 Route 提供外部访问，请使用 UI 或 kubectl 将路由更改为指向原始服务。

注意

如果您的 QuayEcosystem 管理 Postgres 数据库，升级过程会将数据迁移到由升级 Operator 管理的新 Postgres 数据库。旧数据库不会更改或删除，但 Quay 在迁移完成后将不再使用它。如果数据迁移过程中出现问题，升级过程将退出，建议您将数据库作为非受管组件继续进行。

10.5.2. 升级支持的 QuayEcosystem 配置

如果迁移 QuayEcosystem 组件失败或不受支持，Quay Operator 会在日志中报告错误，并且 status.conditions。所有非受管组件都应成功迁移，因为 Quay 的 config.yaml 中还没有提供任何

Kubernetes 资源，且所有必要的值都已在 Quay 的 config.yaml 中提供。

数据库

不支持临时数据库（必须设置volumeSize 字段）。

Redis

不需要任何特殊操作。

外部访问

自动迁移只支持 passthrough Route 访问。其他方法需要手动迁移。

- LoadBalancer 没有自定义主机名：在 QuayEcosystem 标记为标签 "quay-operator/migration-complete": "true" 后，在删除 QuayEcosystem *前* 从现有服务中删除 metadata.ownerReferences 字段以防止 Kubernetes 垃圾回收服务并删除负载均衡器。 使用 metadata.name 格式 <quay Ecosystem-name>-quay-app 创建一个新服务。编辑现有服务的 spec.selector 以匹配新服务的 spec.selector，以便向旧负载均衡器端点的流量定向到新 pod。您现在负责旧服务 ；Quay Operator 不会管理它。

- LoadBalancer/NodePort/Ingress with custom hostname: 使用 metadata.name 格式 <quay Ecosystem-name>-quay-app 创建类型为 LoadBalancer 的新服务。将您的 DNS 设置更改为指向新服务提供的 status.loadBalancer 端点。

Clair

不需要任何特殊操作。

Object Storage

QuayEcosystem 没有受管对象存储组件，因此对象存储始终标记为非受管状态。不支持本地存储。

仓库镜像

不需要任何特殊操作。

其他资源

- 如需有关 Red Hat Quay Operator 的更多信息，请参阅 上游 quay-operator 项目。