OpenShift Container Platform 4.14

镜像

在 OpenShift Container Platform 中创建和管理镜像及镜像流

Last Updated: 2024-12-17

OpenShift Container Platform 4.14 镜像

在 OpenShift Container Platform 中创建和管理镜像及镜像流

法律通告

摘要

本文档介绍在 OpenShift Container Platform 中创建和管理镜像及镜像流。另外还介绍如何使用模板。

目录

第 1 章 镜像概述

1.1. 了解容器、镜像和镜像流

当您设置为创建和管理容器化软件时，务必要理解容器、镜像和镜像流等重要概念。镜像包含一组准备就绪可运行的软件，容器是容器镜像的一个运行实例。镜像流提供了一种方法来存储相同基本镜像的不同版本。这些不同版本通过相同镜像名称的不同标签（tag）来表示。

1.2. 镜像

OpenShift Container Platform 中的容器基于 OCI 或 Docker 格式的容器 *镜像*创建。镜像是一种二进制文件，包含运行单一容器的所有要求以及描述其需求和功能的元数据。

您可以将其视为一种打包技术。容器只能访问其镜像中定义的资源，除非创建时授予容器其他访问权限。通过将同一镜像部署到跨越多个主机的多个容器内，并在它们之间进行负载平衡，OpenShift 容器平台可以为镜像中打包的服务提供冗余和横向扩展。

您可以直接使用 podman 或 **Docker** CLI 构建镜像，但 OpenShift Container Platform 也提供了构建程序（builder）镜像，这有助于通过将您的代码或配置添加到现有镜像来创建新镜像。

由于应用程序会随时间发展，因此单个镜像名称实际上可以指代同一镜像的许多不同版本。每个不同的镜像都会有一个代表它的唯一哈希值（一个较长的十六进制值，如 **fd44297e2ddb050ec4f...**），它通常会被缩短为一个 12 位长的值（如 **fd44297e2ddb**）。

您可以创建，管理，并使用容器镜像。

1.3. 镜像 REGISTRY

镜像 registry 是一个可存储和提供容器镜像的内容服务器。例如：

```
registry.redhat.io
```

registry 包含一个或多个镜像存储库的集合，其中包含一个或多个标记的镜像。红帽在 **registry.redhat.io** 上为订阅者提供了一个 registry。OpenShift Container Platform 还提供自己的 OpenShift 镜像 registry 来管理自定义容器镜像。

1.4. 镜像存储库

镜像存储库是相关容器镜像和标识它们的标签（tag）的集合。例如，OpenShift Container Platform Jenkins 镜像位于以下存储库中：

```
docker.io/openshift/jenkins-2-centos7
```

1.5. 镜像标签

镜像标签（tag）是应用于存储库中容器镜像的标签，用于将特定镜像与镜像流中的其他镜像区分开来。标签通常代表某种版本号。例如，这里 **:v3.11.59-2** 是标签：

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

您可以向镜像添加其他标签。例如，可为镜像分配 **:v3.11.59-2** 和 **:latest** 标签。

OpenShift Container Platform 提供 **oc tag** 命令，该命令类似于 **docker tag** 命令，但是在镜像流上运行，而非直接在镜像上运行。

1.6. 镜像 ID

镜像 ID 是 SHA（安全哈希算法）代码，可用于拉取（pull）镜像。SHA 镜像 ID 不能更改。特定 SHA 标识符会始终引用完全相同的容器镜像内容。例如：

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

1.7. 容器

OpenShift Container Platform 应用程序的基本单元称为容器。Linux 容器技术是一种轻量型机制，用于隔离运行中的进程，使它们只能跟指定的资源交互。容器一词被定义为容器镜像的特定运行或暂停实例。

在一个单一的主机上可以包括多个容器来运行多个不同的应用程序实例，且相互间无法看到其他应用程序的进程、文件、网络等。通常情况下，每个容器提供一项服务，常称为微服务，如 Web 服务器或数据库，但容器也可用于任意工作负载。

多年来，Linux 内核一直在整合容器技术的能力。Docker 项目为主机上的 Linux 容器开发了便捷的管理接口。最近，开放容器计划还为容器格式和容器运行时制定了开放标准。OpenShift Container Platform 和 Kubernetes 增加了在多主机安装之间编排 OCI 和 Docker 格式容器的功能。

尽管在使用 OpenShift Container Platorm 时，您不会直接与容器运行时交互，但了解这些容器运行时的功能和术语非常重要，有助于了解它们在 OpenShift Container Platform 中的作用以及您的应用程序在容器内的运作方式。

podman 等工具可用于替代 **docker** 命令行工具来直接运行和管理容器。利用 **podman**，您可独立于 OpenShift Container Platform 对容器进行试验。

1.8. 为什么使用镜像流

镜像流及其关联标签提供了一个用于从 OpenShift Container Platform 中引用容器镜像的抽象集。镜像流及其标签用于查看可用镜像，确保您使用所需的特定镜像，即使存储库中的镜像发生变化也是如此。

镜像流不含实际镜像数据，它提供了相关镜像的一个单独的虚拟视图，类似于镜像存储库。

您可配置构建（Build）和部署（Depolyment）来监测一个镜像流的通知。当新的镜像被添加时，执行相应的构建或部署。

例如，如果部署正在使用某个镜像并且创建了该镜像的新版本，则会自动执行部署以获取镜像的新版本。

但是，如果部署或构建所用的 imagestreamtag 没有更新，则即使更新了容器镜像 registry 中的容器镜像，构建或部署仍会继续使用之前的，已知良好的镜像。

源镜像可存储在以下任一位置：

- OpenShift Container Platform 集成的 registry。
- 一个外部 registry，如 registry.redhat.io 或 quay.io。
- OpenShift Container Platform 集群中的其他镜像流。

当您定义引用镜像流标签的对象时，如构建或部署配置，您将指向镜像流标签而不是存储库。您在构建或部署应用程序时，OpenShift Container Platform 会使用 imagestreamtag 来查询 Docker 存储库，以找到相应的镜像 ID，并使用正确的镜像。

镜像流元数据会与其他集群信息一起存储在 etcd 实例中。

使用镜像流有以下几大优势：

- 您可以添加标签、回滚标签和快速处理镜像，而无需使用命令行重新执行 push 操作。
- 当一个新镜像被推送（push）到 registry 时，可触发构建和部署。另外，OpenShift Container Platform 还针对 Kubernetes 对象等其他资源提供了通用触发器。
- 您可以为定期重新导入标记标签。如果源镜像已更改，则这个更改会被发现并反应在镜像流中。取决于构建或部署的具体配置，这可能会触发构建和/或部署流程。
- 您可使用细粒度访问控制来共享镜像，快速向整个团队分发镜像。
- 如果源更改，imagestreamtag 仍将指向已知良好的镜像版本，以确保您的应用程序不会意外中断。
- 您可以通过镜像流对象的权限配置安全性，以了解谁可以查看和使用镜像。
- 在集群级别上缺少读取或列出镜像权限的用户仍可使用镜像流来检索项目中标记的镜像。

您可以管理镜像流，将镜像流 与 Kubernetes 资源一起使用，并触发镜像流更新。

1.9. 镜像流标签

镜像流标签是指向镜像流中镜像的命名指针。镜像流标签与容器镜像标签类似。

1.10. 镜像流镜像

镜像流镜像允许您从标记了特定容器镜像的特定镜像流中检索该镜像。镜像流镜像是一个 API 资源对象，用于收集一些有关特定镜像 SHA 标识符的元数据。

1.11. 镜像流触发器

镜像流触发器（imagestream trigger）会在镜像流标签更改时引发特定操作。例如，导入可导致标签值变化。当有部署、构建或其他资源监听这些信息时，就会启动触发器。

1.12. 如何使用 CLUSTER SAMPLES OPERATOR

在初始启动时，Operator 会创建默认样本资源来初始化镜像流和模板的创建过程。您可以使用 Cluster Samples Operator 管理存储在 **openshift** 命名空间中的示例镜像流和模板。

作为集群管理员，您可以使用 Cluster Samples Operator：

- 配置 Operator。
- 使用带有备用 registry 的 Operator。

1.13. 关于模板

模板是要复制的对象的定义。您可以使用模板来构建和部署配置。

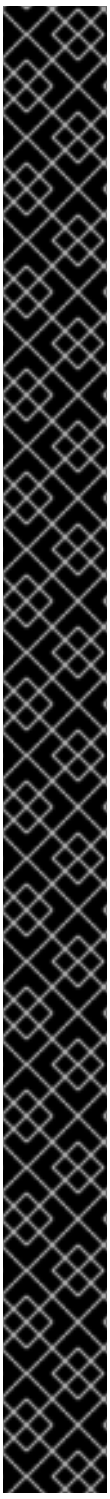
1.14. 如何使用 RUBY ON RAILS

作为开发人员，您可以使用 Ruby on Rails 进行：

- 编写应用程序：
  - 设置数据库。
  - 创建欢迎页面。
  - 为 OpenShift Container Platform 配置应用程序。
  - 将您的应用存储在 Git 中。
- 在 OpenShift Container Platform 中部署应用程序：
  - 创建数据库服务。
  - 创建 frontend 服务。
  - 为应用程序创建路由。

第 2 章 配置 CLUSTER SAMPLES OPERATOR

Cluster Samples Operator 运行在 **openshift** 命名空间中，用于安装和更新基于 Red Hat Enterprise Linux (RHEL) 的 OpenShift Container Platform 镜像流和 OpenShift Container Platform 模板。



CLUSTER SAMPLES OPERATOR 正在降级

- 从 OpenShift Container Platform 4.13 开始，Cluster Samples Operator 被降级。Cluster Samples Operator 将停止为非 Source-to-Image (Non-S2I) 镜像流和模板提供以下更新：
  - 新镜像流和模板
  - 更新现有镜像流和模板，除非它是 CVE 更新
- 根据 OpenShift Container Platform 生命周期策略日期和支持指南，Cluster Samples Operator 将提供对 Non-S2I 镜像流和模板的支持。
- Cluster Samples Operator 将继续支持 S2I 构建器镜像和模板，并接受更新。S2I 镜像流和模板包括：
  - Ruby
  - Python
  - Node.js
  - Perl
  - PHP
  - HTTPD
  - Nginx
  - EAP
  - Java
  - Webserver
  - .NET
  - Go
- 从 OpenShift Container Platform 4.16 开始，Cluster Samples Operator 将停止管理非 S2I 镜像流和模板。您可以联系镜像流或模板所有者以获取任何要求和将来的计划。此外，请参阅托管镜像流或模板的存储库列表。

2.1. 了解 CLUSTER SAMPLES OPERATOR

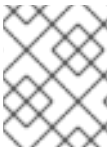
在安装过程中，Operator 会为自己创建默认配置对象，然后创建示例镜像流和模板，包括快速启动模板。

注意

为便于从需要凭证的其他 registry 中导入镜像流,集群管理员可在镜像导入所需的 **openshift** 命名空间中创建包含 Docker **config.json** 文件内容的额外 secret。

Cluster Samples Operator 配置是一个集群范围的资源，其部署包含在 **openshift-cluster-samples-operator** 命名空间中。

Cluster Samples Operator 的镜像包含关联的 OpenShift Container Platform 发行版本的镜像流和模板定义。在创建或更新每个示例时，Cluster Sample Operator 包含一个注解（annotation），用于注明 OpenShift Container Platform 的版本。Operator 使用此注解来确保每个示例与发行版本匹配。清单（inventory）以外的示例会与跳过的示例一样被忽略。对任何由 Operator 管理的示例进行的修改（版本注解被修改或删除），都将会被自动恢复。



注意

Jenkins 镜像实际上自安装后便已是镜像有效负载的一部分，并直接标记（tag）到镜像流中。

Cluster Samples Operator 配置资源包含一个终结器（finalizer），它会在删除时清除以下内容：

- Operator 管理的镜像流。
- Operator 管理的模板。
- Operator 生成的配置资源。
- 集群状态资源。

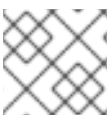
删除样本资源后，Samples Operator 会使用默认配置重新创建资源。

2.1.1. Cluster Samples Operator 使用管理状态

Cluster Samples Operator 默认配置为 **Managed**，或者配置了全局代理。在 **Managed** 状态下，Cluster Samples Operator 会主动管理其资源并保持组件的活跃状态，以便从 registry 中拉取示例镜像流和镜像，并确保安装了必要的示例模板。

在某些情况下，Cluster Samples Operator 会将自身引导为 **Removed**，包括：

- 如果 Cluster Samples Operator 在全新安装后初始启动时三分钟后仍无法访问 registry.redhat.io。
- 如果 Cluster Samples Operator 检测到它位于 IPv6 网络中。
- 镜像控制器配置参数 阻止使用默认镜像 registry 或通过 **samplesRegistry** 设置 指定的镜像 registry 创建镜像流。



注意

对于 OpenShift Container Platform，默认的镜像 registry 是 **registry.redhat.io**。

但是，如果 Cluster Samples Operator 检测到它位于 IPv6 网络上，并且配置了 OpenShift Container Platform 全局代理，则 IPv6 检查会替换所有检查。因此，Cluster Samples Operator 会将自身引导为 **Removed**。

重要

registry.redhat.io目前不支持 IPv6 安装。Cluster Samples Operator 从 registry.redhat.io 中提取大多数示例镜像流和镜像。

2.1.1.1. 受限网络安装

当无法访问 **registry.redhat.io** 时，当网络限制已存在时，Boostrapping 为 **Removed** 有助于进行受限网络安装。当网络访问受限时，Bootstrapping 为 **Removed** 可让集群管理员有更多时间决定是否需要样本，因为 Cluster Samples Operator 不提交当管理状态设置为 **Removed** 时示例镜像流导入失败的警报。当 Cluster Samples Operator 启动为 **Managed** 并尝试安装示例镜像流时，它会在安装失败时在初始安装后 2 小时启动警报。

2.1.1.2. 使用初始网络访问受限网络安装

当一个集群旨在作为受限网络或断开连接而安装的集群，如果其存在网络连接，则 Cluster Samples Operator 会从 **registry.redhat.io** 中安装内容，因为可以访问它。如果您希望 Cluster Samples Operator 仍然引导为 **Removed**，以便延迟样本安装，直到您决定了需要安装哪个样本、如何设置镜像等，请按照有关使用带有备用 registry 以及自定义节点（均在 additional resources 部分中链接）的 Samples Operator 的说明来覆盖 Cluster Samples Operator 默认配置，以便最初以 **Removed** 的形式出现。

您必须将以下额外 YAML 文件放到由 **openshift-install create manifest** 创建的 **openshift** 目录中：

带有 managementState: Removed 的 Cluster Samples Operator YAML 文件示例：

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  architectures:
  - x86_64
  managementState: Removed
```

2.1.2. Cluster Samples Operator 跟踪和错误恢复镜像流导入

在创建或更新示例镜像流后，Cluster Samples Operator 会监控每个镜像流标签镜像导入的进度。

如果导入失败，Cluster Samples Operator 会通过镜像流镜像导入 API（与 **oc import-image** 命令使用的 API 相同）。重新尝试导入大约每 15 分钟进行一次，直到导入成功，或者 Cluster Samples Operator 的配置已更改为镜像流被添加到 **skippedImagestreams** 列表中，或者管理状态变为 **Removed**。

其他资源

- 如果在安装过程中删除了 Cluster Samples Operator，您可以使用带有备用 registry 的 Cluster Samples Operator，以便导入内容，然后将 Cluster Samples Operator 设置为 **Managed** 来获取示例。
- 在带有初始网络访问权限的受限网络安装中确保 Cluster Samples Operator 引导为 **Removed**，直到您决定需要哪些样本，请按照自定义节点来覆盖 Cluster Samples Operator 默认配置，最初以 **Removed** 的形式出现。https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html-single/installation_configuration/#installing-customizing

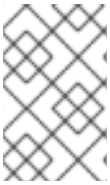
- 要在断开连接的环境中托管示例，请按照使用带有备用 registry 的 Cluster Samples Operator 的说明进行操作。

2.1.3. 协助镜像的 Cluster Samples Operator

在安装过程中，OpenShift Container Platform 在 **openshift-cluster-samples-operator** 命名空间中创建一个名为 **imagestreamtag-to-image** 的配置映射。**imagestreamtag-to-image** 配置映射包含每个镜像流标签的条目（填充镜像）。

配置映射中 data 字段中每个条目的键格式为 **<image_stream_name>_<image_stream_tag_name>**。

在断开连接的 OpenShift Container Platform 安装过程中，Cluster Samples Operator 的状态被设置为 **Removed**。如果您将其改为 **Managed**，它会安装示例。



注意

在网络限制或断开连接的环境中使用示例可能需要通过网络访问服务。某些示例服务包括：Github、Maven Central、npm、RubyGems、PyPi 等。这可能需要执行额外的步骤，让集群 samples operator 对象能够访问它们所需的服务。

您可以使用此配置映射作为导入镜像流所需的镜像的引用。

- 在 Cluster Samples Operator 被设置为 **Removed** 时，您可以创建镜像的 registry，或决定您要使用哪些现有镜像 registry。
- 使用新的配置映射作为指南来镜像您要镜像的 registry 的示例。
- 将没有镜像的任何镜像流添加到 Cluster Samples Operator 配置对象的 **skippedImagestreams** 列表中。
- 将 Cluster Samples Operator 配置对象的 **samplesRegistry** 设置为已镜像的 registry。
- 然后，将 Cluster Samples Operator 设置为 **Managed** 来安装您已镜像的镜像流。

如需了解详细信息，请参阅使用带有备用或镜像 registry 的 Cluster Samples Operator 镜像流 。

2.2. CLUSTER SAMPLES OPERATOR 配置参数

示例资源提供以下配置字段：

参数	描述
managementState	**Managed**: Cluster Samples Operator 根据配置要求更新示例。 **Unmanaged**：Cluster Samples Operator 忽略对**openshift** 命名空间中的配置资源对象和任何镜像流或模板的更新。 **Removed**：Cluster Samples Operator 删除**openshift** 命名空间中的 **Managed** 镜像流和模板集合。它忽略集群管理员创建的新示例或跳过列表中的任何示例。移除完成后，Cluster Samples Operator 会像处于 **Unmanaged** 状态一样工作，并忽略示例资源、镜像流或模板上的任何监控事件。

参数	描述
samplesRegistry	允许您指定镜像流可以访问哪个 registry 获取其镜像内容。对于 OpenShift Container Platform，**samplesRegistry** 默认为 **registry.redhat.io**。 **注意** 在未显式设置 **Samples Registry**（如空字符串）或该 registry 已设置为 registry.redhat.io 时，如果没有拉取（pull）访问的 secret，则不会开始创建或更新 RHEL 内容。这两种情况下，镜像导入将从需要凭证的 registry.redhat.io 中进行。 如果 **Samples Registry** 被除空字符串或 registry.redhat.io 以外的值覆盖，则创建或更新 RHEL 内容不受存在 pull secret 的限制。
architectures	用于选择架构类型的占位符。
skippedImagestreams	Cluster Samples Operator 清单中集群管理员希望 Operator 忽略或不予管理的镜像流。您可以在此参数中添加镜像流名称列表。例如: **["httpd","perl"]**。
skippedTemplates	Cluster Samples Operator 清单中集群管理员希望 Operator 忽略或不予管理的模板。

secret、镜像流和模板监控事件可在初始创建示例资源对象之前发生，Cluster Samples Operator 会检测到这些事件并对事件队列重新排序。

2.2.1. 配置限制

当 Cluster Samples Operator 开始支持多个架构时，则在处于 **Managed** 状态时不允许更改架构列表。

要更改构架值，集群管理员必须：

- 将 **Management State** 标记为 **Removed**，并保存更改。
- 在随后更改中，编辑构架并将 **Management State** 改回 **Managed**。

Cluster Samples Operator 在 **Removed** 状态下仍然会处理 secret。您可在切换到 **Removed** 之前，或在切换到 **Managed** 之前仍处于 **Removed** 时，或切换到 **Managed** 状态后创建 secret。如果在切换到 **Managed** 后创建 secret，创建示例会延迟到处理 secret 事件。如果您选择在切换前删除所有样本，则会有助于更改 registry。不需要在切换前删除所有示例。

2.2.2. Conditions

示例资源将在所处状态下保持以下条件：

条件	描述
SamplesExists	代表 **openshift** 命名空间中创建了样本。

条件	描述
ImageChangesInProgress	如果创建或更新了镜像流，但并非所有标记规范生成与标记状态生成均匹配，此条件则为 **True**。 所有生成均匹配，或者导入过程中发生不可恢复的错误时显示为 **False**，最后看到的错误位于消息字段中，待处理的镜像流列表位于原因字段中。 OpenShift Container Platform 中已弃用此条件。
ConfigurationValid	如果提交的改变在以前已被认为是不可以被改变的 (restricted)，则为 **True**，否则为 **False**。
RemovePending	代表有 **Management State: Removed** 设置待处理，但 Cluster Samples Operator 正在等待删除操作完成。
ImportImageErrorsExist	指明哪些镜像流在它们的一个标签的镜像导入阶段出错。 出错时显示为 **True**。出错的镜像流列表位于原因字段中。各个报告错误的详情位于消息字段中。
MigrationInProgress	当 Cluster Samples Operator 检测到对应版本与安装当前示例集的 Cluster Samples Operator 版本不同时，显示为 **True**。 OpenShift Container Platform 中已弃用此条件。

2.3. 访问 CLUSTER SAMPLES OPERATOR 配置

您可以使用提供的参数编辑文件来配置 Cluster Samples Operator。

先决条件

- 安装 OpenShift CLI (**oc**) 。

流程

- 访问 Cluster Samples Operator 配置：

```
$ oc edit configs.samples.operator.openshift.io/cluster -o yaml
```

Cluster Samples Operator 配置类似以下示例：

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
# ...
```

2.4. 从 CLUSTER SAMPLES OPERATOR 中删除已弃用的镜像流标签

Cluster Samples Operator 在镜像流中保留已弃用的镜像流标签，因为用户可能有使用已弃用的镜像流标签的部署。

您可以使用 **oc tag** 命令编辑镜像流标签，从而删除已弃用的镜像流标签。

注意

弃用的镜像流标签，示例供应商从镜像流中删除的标签不包括在初始安装中。

先决条件

- 已安装 **oc** CLI。

流程

- 使用 **oc tag** 命令编辑镜像流标签，从而删除已弃用的镜像流标签。

```
$ oc tag -d <image_stream_name:tag>
```

输**出示例**

```
Deleted tag default/<image_stream_name:tag>.
```

其他资源

- 有关配置凭证的更多信息，请参阅使用镜像 pull secret。

第 3 章 使用带有备用 REGISTRY 的 CLUSTER SAMPLES OPERATOR

您可以通过创建镜像 registry 来将 Cluster Samples Operator 与备用 registry 搭配使用。

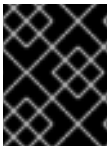
重要

您必须可以访问互联网来获取所需的容器镜像。在这一流程中，您要将镜像 registry 放在可访问您的网络以及互联网的镜像（mirror）主机上。

3.1. 关于镜像 REGISTRY

您可以镜像 OpenShift Container Platform 安装所需的镜像，以及容器镜像 registry 的后续产品更新，如 Red Hat Quay、JFrog Artifactory、Sonatype Nexus Repository 或 Harbor。如果您无法访问大型容器 registry，可以使用 *mirror registry for Red Hat OpenShift*，它是包括在 OpenShift Container Platform 订阅中的一个小型容器 registry。

您可以使用支持 Docker v2-2 的任何容器 registry，如 Red Hat Quay，*mirror registry for Red Hat OpenShift*, Artifactory, Sonatype Nexus Repository, 或 Harbor。无论您所选 registry 是什么，都会将互联网上红帽托管站点的内容镜像到隔离的镜像 registry 相同。镜像内容后，您要将每个集群配置为从镜像 registry 中检索此内容。



重要

OpenShift 镜像 registry 不能用作目标 registry，因为它不支持没有标签的推送，在镜像过程中需要这个推送。

如果选择的容器 registry 不是 *mirror registry for Red Hat OpenShift*，则需要集群中置备的每台机器都可以访问它。如果 registry 无法访问，安装、更新或常规操作（如工作负载重新定位）可能会失败。因此，您必须以高度可用的方式运行镜像 registry，镜像 registry 至少必须与 OpenShift Container Platform 集群的生产环境可用性相匹配。

使用 OpenShift Container Platform 镜像填充镜像 registry 时，可以遵循以下两种情况。如果您的主机可以同时访问互联网和您的镜像 registry，而不能访问您的集群节点，您可以直接从该机器中镜像该内容。这个过程被称为 *连接的镜像(mirror)*。如果没有这样的主机，则必须将该镜像文件镜像到文件系统中，然后将该主机或者可移动介质放入受限环境中。这个过程被称为 *断开连接的镜像*。

对于已镜像的 registry，若要查看拉取镜像的来源，您必须查看 **Trying 以访问** CRI-O 日志中的日志条目。查看镜像拉取源的其他方法（如在节点上使用 **crictl images** 命令）显示非镜像镜像名称，即使镜像是从镜像位置拉取的。

注意

红帽没有针对 OpenShift Container Platform 测试第三方 registry。

附加信息

有关查看 CRI-O 日志以查看镜像源的详情，请参阅 查看镜像拉取源。

3.1.1. 准备镜像主机

在创建镜像 registry 前，您必须准备镜像（mirror）主机。

3.1.2. 通过下载二进制文件安装 OpenShift CLI

您可以安装 OpenShift CLI(**oc**)来使用命令行界面与 OpenShift Container Platform 进行交互。您可以在 Linux、Windows 或 macOS 上安装 **oc**。

重要

如果安装了旧版本的 **oc**，则无法使用 OpenShift Container Platform 4.14 中的所有命令。下载并安装新版本的 **oc**。

在 Linux 上安装 OpenShift CLI
您可以按照以下流程在 Linux 上安装 OpenShift CLI(**oc**)二进制文件。

流程

1. 导航到红帽客户门户网站上的 OpenShift Container Platform 下载页面。
2. 从 **产品变体** 下拉列表中选择架构。
3. 从 **版本** 下拉列表中选择适当的版本。
4. 点 **OpenShift v4.14 Linux Client** 条目旁的 **Download Now** 来保存文件。
5. 解包存档：

```
$ tar xvf <file>
```

6. 将 **oc** 二进制文件放到 **PATH 中的目录中**。
要查看您的 **PATH**，请执行以下命令：

```
$ echo $PATH
```

验证

- 安装 OpenShift CLI 后，可以使用 **oc** 命令：

```
$ oc <command>
```

在 Windows 上安装 OpenShift CLI
您可以按照以下流程在 Windows 上安装 OpenShift CLI(**oc**)二进制文件。

流程

1. 导航到红帽客户门户网站上的 OpenShift Container Platform 下载页面。
2. 从 **版本** 下拉列表中选择适当的版本。
3. 点 **OpenShift v4.14 Windows Client** 条目旁的 **Download Now** 来保存文件。
4. 使用 ZIP 程序解压存档。
5. 将 **oc** 二进制文件移到 **PATH 中的目录中**。
要查看您的 **PATH**，请打开命令提示并执行以下命令：

```
C:\> path
```

验证

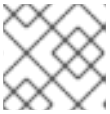
- 安装 OpenShift CLI 后，可以使用 **oc** 命令：

```
C:\> oc <command>
```

在 macOS 上安装 OpenShift CLI
您可以按照以下流程在 macOS 上安装 OpenShift CLI(**oc**)二进制文件。

流程

1. 导航到红帽客户门户网站上的 OpenShift Container Platform 下载页面。
2. 从 **版本** 下拉列表中选择适当的版本。
3. 点 **OpenShift v4.14 macOS Client** 条目旁的 **Download Now** 来保存文件。



注意

对于 macOS arm64，请选择 **OpenShift v4.14 macOS arm64 Client** 条目。

4. 解包和解压存档。
5. 将 **oc** 二进制文件移到 PATH 的目录中。
要查看您的 **PATH**，请打开终端并执行以下命令：

```
$ echo $PATH
```

验证

- 使用 **oc** 命令验证安装：

```
$ oc <command>
```

3.2. 配置允许对容器镜像进行镜像的凭证

创建容器镜像 registry 凭证文件，允许将红帽的镜像镜像到您的镜像环境中。

先决条件

- 您已将镜像 registry 配置为在断开连接的环境中使用。

流程

在安装主机上完成以下步骤：

1. 从 Red Hat OpenShift Cluster Manager 下载 **registry.redhat.io** pull secret。
2. 以 JSON 格式创建您的 pull secret 副本：

```
$ cat ./pull-secret | jq . > <path>/<pull_secret_file_in_json> 1
```

1 指定到存储 pull secret 的文件夹的路径，以及您创建的 JSON 文件的名称。

该文件类似于以下示例：

```
{
  "auths": {
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3. 为您的镜像 registry 生成 base64 编码的用户名和密码或令牌：

```
$ echo -n '<user_name>:<password>' | base64 -w0 1
BGVtbYk3ZHAtqXs=
```

1 通过 **<user_name>** 和 **<password>** 指定 registry 的用户名和密码。

4. 编辑 JSON 文件并添加描述 registry 的部分：

```
  "auths": {
    "<mirror_registry>": { 1
      "auth": "<credentials>", 2
      "email": "you@example.com"
    }
  },
```

1 对于 **<mirror_registry>**，指定 registry 域名，以及您的镜像 registry 用来提供内容的可选端口。例如：**registry.example.com** 或 **registry.example.com:8443**

2 使用 **<credentials>** 为您的镜像 registry 指定 base64 编码的用户名和密码。

该文件类似于以下示例：

```
{
```

```
  "auths": {
    "registry.example.com": {
      "auth": "BGVtbYk3ZHAtqXs=",
      "email": "you@example.com"
    },
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3.3. 镜像 OPENSHIFT CONTAINER PLATFORM 镜像存储库

镜像要在集群安装或升级过程中使用的 OpenShift Container Platform 镜像仓库。

先决条件

- 您的镜像主机可访问互联网。
- 您已将镜像 registry 配置为在受限网络中使用，并可访问您配置的证书和凭证。
- 您已从 Red Hat OpenShift Cluster Manager 下载了 pull secret，并已修改为包含镜像存储库的身份验证。
- 如果您使用自签名证书，已在证书中指定 Subject Alternative Name。

流程

在镜像主机上完成以下步骤：

1. 查看 OpenShift Container Platform 下载页面，以确定您要安装的 OpenShift Container Platform 版本，并决定 Repository Tags 页中的相应标签（tag）。
2. 设置所需的环境变量：
   a. 导出发行版本信息：

   ```
   $ OCP_RELEASE=<release_version>
   ```

   对于 **<release_version>**，请指定与 OpenShift Container Platform 版本对应的标签，用于您的架构，如 **4.5.4**。

   b. 导出本地 registry 名称和主机端口：

```
$ LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>'
```

对于 **<local_registry_host_name>**，请指定镜像存储库的 registry 域名；对于 **<local_registry_host_port>**，请指定用于提供内容的端口。

c. 导出本地存储库名称：

```
$ LOCAL_REPOSITORY='<local_repository_name>'
```

对于 **<local_repository_name>**，请指定要在 registry 中创建的仓库名称，如 **ocp4/openshift4**。

d. 导出要进行镜像的存储库名称：

```
$ PRODUCT_REPO='openshift-release-dev'
```

对于生产环境版本，必须指定 **openshift-release-dev**。

e. 导出 registry pull secret 的路径：

```
$ LOCAL_SECRET_JSON='<path_to_pull_secret>'
```

对于 **<path_to_pull_secret>**，请指定您创建的镜像 registry 的 pull secret 的绝对路径和文件名。

f. 导出发行版本镜像：

```
$ RELEASE_NAME="ocp-release"
```

对于生产环境版本，您必须指定 **ocp-release**。

g. 为您的集群导出构架类型：

```
$ ARCHITECTURE=<cluster_architecture> 1
```

1 指定集群的构架，如 **x86_64**, **aarch64**, **s390x**, 获 **ppc64le**。

h. 导出托管镜像的目录的路径：

```
$ REMOVABLE_MEDIA_PATH=<path> 1
```

1 指定完整路径，包括开始的前斜杠(/)字符。

3. 将版本镜像(mirror)到镜像 registry：

- 如果您的镜像主机无法访问互联网，请执行以下操作：

  i. 将可移动介质连接到连接到互联网的系统。

  ii. 查看要镜像的镜像和配置清单：

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON}  \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
```

```
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE} --dry-run
```

iii. 记录上一命令输出中的 **imageContentSources** 部分。您的镜像信息与您的镜像存储库相对应，您必须在安装过程中将 **imageContentSources** 部分添加到 **install-config.yaml** 文件中。

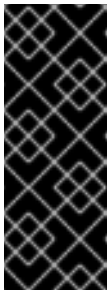
iv. 将镜像镜像到可移动介质的目录中：

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} --to-
dir=${REMOVABLE_MEDIA_PATH}/mirror
quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE}
```

v. 将介质上传到受限网络环境中，并将镜像上传到本地容器 registry。

```
$ oc image mirror -a ${LOCAL_SECRET_JSON} --from-
dir=${REMOVABLE_MEDIA_PATH}/mirror
"file://openshift/release:${OCP_RELEASE}*"
${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} 1
```

1 对于 **REMOVABLE_MEDIA_PATH**，您必须使用与镜像镜像时指定的同一路径。



重要

运行 **oc image mirror** 可能会导致以下错误： **error: unable to retrieve source image**。当镜像索引包括对镜像 registry 中不再存在的镜像的引用时，会发生此错误。镜像索引可能会保留旧的引用，以便为运行这些镜像的用户在升级图表中显示新的升级路径。作为临时解决方案，您可以使用 **--skip-missing** 选项绕过错误并继续下载镜像索引。如需更多信息，请参阅 Service Mesh Operator 镜像失败。

- 如果本地容器 registry 连接到镜像主机，请执行以下操作：

i. 使用以下命令直接将发行版镜像推送到本地 registry:

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON}  \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}
```

该命令将发行信息提取为摘要，其输出包括安装集群时所需的 **imageContentSources** 数据。

ii. 记录上一命令输出中的 **imageContentSources** 部分。您的镜像信息与您的镜像存储库相对应，您必须在安装过程中将 **imageContentSources** 部分添加到 **install-config.yaml** 文件中。

注意

镜像名称在镜像过程中被修补到 Quay.io， podman 镜像将在 bootstrap 虚拟机的 registry 中显示 Quay.io。

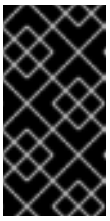
4. 要创建基于您镜像内容的安装程序，请提取内容并将其固定到发行版中：

   - 如果您的镜像主机无法访问互联网，请运行以下命令：

     ```
     $ oc adm release extract -a ${LOCAL_SECRET_JSON} --icsp-file=<file> --
     command=openshift-install
     "${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
     ${ARCHITECTURE}"
     ```

   - 如果本地容器 registry 连接到镜像主机，请运行以下命令：

     ```
     $ oc adm release extract -a ${LOCAL_SECRET_JSON} --command=openshift-install
     "${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
     ${ARCHITECTURE}"
     ```



重要

要确保将正确的镜像用于您选择的 OpenShift Container Platform 版本，您必须从镜像内容中提取安装程序。

您必须在有活跃互联网连接的机器上执行这个步骤。

5. 对于使用安装程序置备的基础架构的集群，运行以下命令：

   ```
   $ openshift-install
   ```

3.4. 使用带有备用或镜像 REGISTRY 的 CLUSTER SAMPLES OPERATOR 镜像流

openshift 命名空间中大多数由 Cluster Samples Operator 管理的镜像流指向位于 registry.redhat.io 上红帽容器镜像仓库中的镜像。

注意

cli、**installer**、**must-gather** 和 **test** 镜像流虽然属于安装有效负载的一部分，但不由 Cluster Samples Operator 管理。此流程中不涉及这些镜像流。

重要

Cluster Samples Operator 必须在断开连接的环境中设置为 **Managed**。要安装镜像流，您需要有一个镜像的 registry。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 为您的镜像 registry 创建 pull secret。

流程

1. 访问被镜像（mirror）的特定镜像流的镜像，例如：

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep
registry.redhat.io
```

2. 镜像 registry.redhat.io 中与您需要的任何镜像流关联的镜像

```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

3. 创建集群的镜像配置对象：

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. 在集群的镜像配置对象中，为镜像添加所需的可信 CA：

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

5. 更新 Cluster Samples Operator 配置对象中的 **samplesRegistry** 字段，使其包含镜像配置中定义的镜像位置的 **hostname** 部分：

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

注意

这是必要的，因为镜像流导入过程在此刻不使用镜像（mirro）或搜索机制。

6. 将所有未镜像的镜像流添加到 Cluster Samples Operator 配置对象的 **skippedImagestreams** 字段。或者，如果您不想支持任何示例镜像流，在 Cluster Samples Operator 配置对象中将 Cluster Samples Operator 设置为 **Removed**。

注意

如果镜像流导入失败，Cluster Samples Operator 会发出警告，但 Cluster Samples Operator 会定期重试，或看起来并没有重试它们。

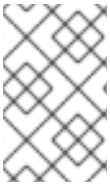
openshift 命名空间中的许多模板都引用镜像流。因此，使用 **Removed** 清除镜像流和模板，将避免在因为缺少镜像流而导致镜像流和模板无法正常工作时使用它们。

3.4.1. 协助镜像的 Cluster Samples Operator

在安装过程中，OpenShift Container Platform 在 **openshift-cluster-samples-operator** 命名空间中创建一个名为 **imagestreamtag-to-image** 的配置映射。**imagestreamtag-to-image** 配置映射包含每个镜像流标签的条目（填充镜像）。

配置映射中 data 字段中每个条目的键格式为 **<image_stream_name>_<image_stream_tag_name>**。

在断开连接的 OpenShift Container Platform 安装过程中，Cluster Samples Operator 的状态被设置为 **Removed**。如果您将其改为 **Managed**，它会安装示例。



注意

在网络限制或断开连接的环境中使用示例可能需要通过网络访问服务。某些示例服务包括：Github、Maven Central、npm、RubyGems、PyPi 等。这可能需要执行额外的步骤，让集群 samples operator 对象能够访问它们所需的服务。

您可以使用此配置映射作为导入镜像流所需的镜像的引用。

- 在 Cluster Samples Operator 被设置为 **Removed** 时，您可以创建镜像的 registry，或决定您要使用哪些现有镜像 registry。
- 使用新的配置映射作为指南来镜像您要镜像的 registry 的示例。
- 将没有镜像的任何镜像流添加到 Cluster Samples Operator 配置对象的 **skippedImagestreams** 列表中。
- 将 Cluster Samples Operator 配置对象的 **samplesRegistry** 设置为已镜像的 registry。
- 然后，将 Cluster Samples Operator 设置为 **Managed** 来安装您已镜像的镜像流。

如需了解详细信息，请参阅使用带有备用或镜像 registry 的 Cluster Samples Operator 镜像流 。

第 4 章 创建镜像

了解如何基于就绪可用的预构建镜像来创建自己的容器镜像。这一过程包括学习编写镜像、定义镜像元数据、测试镜像以及使用自定义构建程序工作流创建可用于 OpenShift Container Platform 的镜像的最佳实践。创建镜像后，您可将其推送到 OpenShift 镜像 registry。

4.1. 学习容器最佳实践

在创建 OpenShift Container Platform 上运行的容器镜像时，镜像创建者需考虑诸多最佳实践，以确保为镜像的使用者提供良好体验。镜像原则上不可变且应按原样使用，所以请遵守以下准则，以确保您的镜像高度可用，且易于在 OpenShift Container Platform 上使用。

4.1.1. 常规容器镜像准则

无论容器镜像是否在 OpenShift Container Platform 中使用，在创建容器镜像时都需要遵循以下指导信息。

重复利用镜像
您的镜像尽可能使用 **FROM** 语句基于适当的上游镜像。这可确保，在上游镜像更新时您的镜像也可轻松从中获取安全修复，而不必再直接更新依赖项。

此外，请使用 **FROM** 指令中的标签（如 **rhel:rhel7**），方便用户准确了解您的镜像基于哪个版本的镜像。使用除 **latest** 以外的标签可确保您的镜像不受 **latest** 版上游镜像重大更改的影响。

在标签内维持兼任性
在为自己的镜像添加标签时，建议尽量在标签内保持向后兼容性。例如，如果您提供名为 **image** 的镜像，且当前包含 **1.0** 版本，您可以提供一个 **image:v1** 标签。当您更新镜像时，只要它仍然与原始镜像兼容，您可以继续标记新镜像 **image:v1**，并且此标签的下游用户就可以获得更新，而不会出现问题。

如果后续发布了不兼容的更新，则切换到新标签，如 **image:v2**。这样，下游用户就可以根据需要选择是否升级到新版本，而不会因为不兼容的新镜像造成问题。下游用户如果使用 **image:latest**，则可能要承担引入不兼容更改的风险。

避免多进程
不要在同一容器中启动多个服务，如数据库和 **SSHD**。因为容器是轻量级的，可轻松链接到一起以编排多个进程，所以没有在同一个容器中启动多个服务的必要。您可以利用 OpenShift Container Platform 将相关镜像分组到一个 pod 中来轻松并置和共同管理镜像。

这种并置可确保容器共享一个网络命名空间和存储进行通信。因为对每个镜像的更新频率较低且可以独立进行，所以更新所可能带来的破坏风险也较小，单一进程的信号处理流程也更加清晰，因为无需管理将信号路由到多个分散进程的操作。

在 wrapper 脚本中使用 exec
很多镜像在启动正在运行的软件的进程前，会先使用 wrapper 脚本进行一些设置。如果您的镜像使用这类脚本，则该脚本应使用 **exec**，以便使您的软件可以替代脚本的进程。如果不使用 **exec**，则容器运行时发送的信号将进入 wrapper 脚本，而非软件的进程。这不是您想要的。

如果您有一个为某些服务器启动进程的 wrapper 脚本。您启动了容器（例如使用 **podman run -i**），该容器运行 wrapper 脚本，继而启动您的进程。如果要使用 **CTRL+C** 关闭容器。如果您的 wrapper 脚本使用了 **exec** 来启动服务器进程，则 **podman** 会将 SIGINT 发送至服务器进程，一切都可以正常工作。如果您没有在 wrapper 脚本中使用 **exec**，则 **podman** 会将 SIGINT 发送至 wrapper 脚本的进程，并且您的进程不会象任何情况一样继续运行。

另请注意，您的进程在容器中运行时，将作为 **PID 1** 运行。这意味着，如果主进程终止，则整个容器都会停止，取消您从 **PID 1** 进程启动的所有子进程。

清理临时文件
删除构建过程中创建的所有临时文件。这也包括通过 **ADD** 命令添加的任何文件。例如，在执行 **yum install** 操作后运行 **yum clean** 命令。

您可按照如下所示创建 **RUN** 语句来防止 **yum** 缓存最终留在镜像层：

```
RUN yum -y install mypackage && yum -y install myotherpackage && yum clean all -y
```

请注意，如果您改写为：

```
RUN yum -y install mypackage
RUN yum -y install myotherpackage && yum clean all -y
```

则首次 **yum** 调用会将额外文件留在该层，后续运行 **yum clean** 操作时无法删除这些文件。最终镜像中看不到这些额外文件，但它们存在于底层中。

如果在较早层中删除了一些内容时，当前容器构建进程不允许在较晚的层中运行一个命令来缩减镜像使用的空间。但是，这个行为可能会在以后的版本中有所改变。这表示，如果在较晚层中执行 **rm** 命令，虽然被这个命令删除的文件不会出现在镜像中，但它不会使下载的镜像变小。因此，与 **yum clean** 示例一样，最好尽可能使用创建文件的同一命令删除文件，以免文件最终写入层中。

另外，在单个 **RUN** 语句中执行多个命令可减少镜像中的层数，缩短下载和提取时间。

按正确顺序放置指令
容器构建程序读取 **Dockerfile**，并自上而下运行指令。成功执行的每个指令都会创建一个层，可在下次构建该镜像或其他镜像时重复使用。务必要将很少更改的指令放置在 **Dockerfile** 的顶部。这样做可确保下次构建相同镜像会非常迅速，因为缓存不会因为上层变化而失效。

例如：如果您正在使用 **Dockerfile**，它包含一个用于安装正在迭代的文件的 **ADD** 命令，以及一个用于 **yum install** 软件包的 **RUN** 命令，则最好将 **ADD** 命令放在最后：

```
FROM foo
RUN yum -y install mypackage && yum clean all -y
ADD myfile /test/myfile
```

这样，您每次编辑 **myfile** 和重新运行 **podman build** 或 **docker build** 时，系统都可重复利用 **yum** 命令的缓存层，仅为 **ADD** 操作生成新层。

如果您将 **Dockerfile** 改写为：

```
FROM foo
ADD myfile /test/myfile
RUN yum -y install mypackage && yum clean all -y
```

则您每次更改 **myfile** 和重新运行 **podman build** 或 **docker build** 时，**ADD** 操作都会导致 **RUN** 层缓存无效，因此 **yum** 操作也必须要重新运行。

标记重要端口
EXPOSE 指令使主机系统和其它容器可使用容器中的端口。尽管可以指定应当通过 **podman run** 调用来公开端口，但在 **Dockerfile** 中使用 EXPOSE 指令可显式声明您的软件需要运行的端口，让用户和软件更易于使用您的镜像：

- 公开端口显示在 **podman ps** 下，与从您的镜像创建的容器关联。
- 公开端口存在于 **podman inspect** 返回的镜像元数据中。

- 当将一个容器链接到另一容器时，公开端口也会链接到一起

设置环境变量
设置环境变量的最佳做法是使用 **ENV** 指令设置环境变量。一个例子是设置项目版本。这让人可以无需通过查看 **Dockerfile** 便可轻松找到版本。另一示例是在系统上公告可供其他进程使用的路径，如 **JAVA_HOME**。

避免默认密码
避免设置默认密码。许多人扩展镜像时会忘记删除或更改默认密码。如果在生产环境中的用户被分配了众所周知的密码，则这可能引发安全问题。可以使用环境变量来配置密码。

如果您的确要选择设置默认密码，请确保在容器启动时显示适当的警告消息。消息中应告知用户默认密码的值并说明如何修改密码，例如要设置什么环境变量。

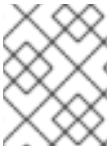
避免 sshd
最好避免在您的镜像中运行 **sshd**。您可使用 **podman exec** 或 **docker exec** 命令访问本地主机上运行的容器。另外，也可使用 **oc exec** 命令或 **oc rsh** 命令访问 OpenShift Container Platform 集群上运行的容器。在镜像中安装并运行 **sshd** 会为安全攻击打开额外通道，因而需要安装安全补丁。

对持久性数据使用卷
镜像应对持久性数据使用卷。这样，OpenShift Container Platform 便可将网络存储挂载至运行容器的节点，如果容器移至新节点，存储也将重新连接至该节点。通过使用卷来满足所有持久性存储需求，即使容器重启或移动，其内容也会保留下来。如果您的镜像将数据写入容器中的任意位置，则其内容可能无法保留。

所有在容器销毁后仍需要保留的数据都必须写入卷中。容器引擎支持容器的 **readonly** 标记，可用于严格执行不将数据写入容器临时存储的良好做法。现在围绕该功能设计您的镜像，将更便于以后利用。

在 **Dockerfile** 中显式定义卷可方便镜像用户轻松了解在运行您的镜像时必须要定义的卷。

有关如何在 OpenShift Container Platform 中使用卷的更多信息，请参阅 Kubernetes 文档。



注意

即使具有持久性卷，您的镜像的每个实例也都有自己的卷，且文件系统不会在实例之间共享。这意味着卷无法用于共享集群中的状态。

4.1.2. OpenShift Container Platform 特定准则

以下是创建 OpenShift Container Platform 上专用的容器镜像时适用的准则。

4.1.2.1. 启用 Source-to-Image (S2I) 的镜像

对于计划运行由第三方提供的应用程序代码的镜像，例如专为运行由开发人员提供的 Ruby 代码而设计的 Ruby 镜像，您可以让镜像与 Source-to-Image (S2I) 构建工具协同工作。S2I 是一个框架，便于编写以应用程序源代码为输入的镜像和生成以运行汇编应用程序为输出的新镜像。

4.1.2.2. 支持任意用户 id

默认情况下，OpenShift Container Platform 使用任意分配的用户 ID 来运行容器。这对因容器引擎漏洞而逸出容器的进程提供了额外的安全防护，从而避免在主机节点上出现未授权的权限升级的问题。

对于支持以任意用户身份运行的镜像，由镜像中进程写入的目录和文件应归 root 组所有，并可由该组读/写。待执行文件还应具有组执行权限。

向 Dockerfile 中添加以下内容可将目录和文件权限设置为允许 root 组中的用户在构建镜像中访问它们：

```
RUN chgrp -R 0 /some/directory && \
    chmod -R g=u /some/directory
```

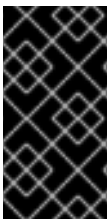
因为容器用户始终是 root 组的成员，所以容器用户可以读写这些文件。

警告

在更改容器敏感区域的目录和文件权限时，需要格外小心。在对敏感区域（如 **/etc/passwd**）应用更改时，可能会造成这些文件被意料外的用户修改，并可能会导致容器或主机被暴露。CRI-O 支持将任意用户 ID 插入容器的 **/etc/passwd** 文件中。因此，不需要更改权限。

此外，**/etc/passwd** 文件不应存在于任何容器镜像中。如果这样做，CRI-O 容器运行时将无法将随机 UID 注入 **/etc/passwd** 文件。在这种情况下，容器可能会在解决活跃 UID 时面临挑战。无法满足这些要求可能会影响某些容器化应用的功能。

此外，容器中运行的进程不是以特权用户身份运行，因此不得监听特权端口（低于 1024 的端口）。



重要

如果您的 S2I 镜像不含带有用户 id 的 **USER** 声明，则您的构建将默认失败。要允许使用指定用户或 root **0** 用户的镜像在 OpenShift Container Platform 中进行构建，您可以将项目的构建器服务帐户 **system:serviceaccount:<your-project>:builder** 添加至 **anyuid** 安全性上下文约束(SCC)。此外，您还可允许所有镜像以任何用户身份运行。

4.1.2.3. 使用服务进行镜像间通信

对于您的镜像需要与另一镜像提供的服务通信的情况，例如需要访问数据库镜像来存储和检索数据的 web 前端镜像，则您的镜像应使用一个 OpenShift Container Platform 服务。服务为访问提供静态端点，该端点不会随着容器的停止、启动或移动而改变。此外，服务还会为请求提供负载均衡。

4.1.2.4. 提供通用库

对于要运行由第三方提供的应用程序代码的镜像，请确保您的镜像包含适用于您的平台的通用库。特别要为平台使用的通用数据库提供数据库驱动程序。例如，在创建 Java 框架镜像时，要为 MySQL 和 PostgreSQL 提供 JDBC 驱动程序。这样做可避免在应用程序汇编期间下载通用依赖项，从而加快应用程序镜像构建。此外还简化了应用程序开发人员为确保满足所有依赖项而需要做的工作。

4.1.2.5. 使用环境变量进行配置

您的镜像用户应在无需基于您的镜像创建下游镜像的情况下也可进行配置。这意味着运行时配置使用环境变量进行处理。对于简单的配置，运行中的进程可直接使用环境变量。对于更为复杂的配置或对于不支持此操作的运行时，可通过定义启动过程中处理的模板配置文件来配置运行时。在此处理过程中，可将使用环境变量提供的值替换到配置文件中，或用于决定要在配置文件中设置哪些选项。

此外，也可以使用环境变量将证书和密钥等 secret 传递到容器中，这是建议操作。这样可确保 secret 值最终不会提交到镜像中，也不会泄漏到容器镜像 registry 中。

提供环境变量可方便您的镜像用户自定义行为，如数据库设置、密码和性能调优，而无需在镜像顶部引入新层。相反，用户可在定义 pod 时简单定义环境变量值，且无需重新构建镜像也可更改这些设置。

对于极其复杂的场景，还可使用在运行时挂载到容器中的卷来提供配置。但是，如果选择这种配置方式时，您必须确保当不存在必要卷或配置时，您的镜像可在启动时提供清晰的错误消息。

本主题与"使用服务进行镜像间通信"主题之间的相关之处在于，数据源等配置应当根据提供服务端点信息的环境变量来定义。这使得应用程序在不修改应用程序镜像的情况下即可动态使用 OpenShift Container Platform 环境中定义的数据源服务。

另外，调整应通过检查容器的 **cgroups** 设置来实现。这使得镜像可根据可用内存、CPU 和其他资源自行调整。例如，基于 Java 的镜像应根据 **cgroup** 最大内存参数调整其堆大小，以确保不超过限值且不出现内存不足错误。

4.1.2.6. 设置镜像元数据

定义镜像元数据有助于 OpenShift Container Platform 更好地使用您的容器镜像，允许 OpenShift Container Platform 使用您的镜像为开发人员创造更好的体验。例如，您可以添加元数据以提供有用的镜像描述，或针对可能也需要的其他镜像提供建议。

4.1.2.7. 集群

您必须充分了解运行镜像的多个实例的意义。在最简单的情况下，服务的负载均衡功能会处理将流量路由到镜像的所有实例。但是，许多框架必须共享信息才能执行领导选举机制或故障转移状态，例如在会话复制中。

设想您的实例在 OpenShift Container Platform 中运行时如何完成这一通信。尽管 pod 之间可直接相互通信，但其 IP 地址会随着 pod 的启动、停止和移动而变化。因此，集群方案必须是动态的。

4.1.2.8. 日志记录

最好将所有日志记录发送至标准输出。OpenShift Container Platform 从容器收集标准输出，然后将其发送至集中式日志记录服务，以供查看。如果必须将日志内容区分开来，请在输出前添加适当关键字，这样便可过滤消息。

如果您的镜像日志记录到文件，则用户必须通过手动操作进入运行中的容器，并检索或查看日志文件。

4.1.2.9. 存活 (liveness) 和就绪 (readiness) 探针

记录可用于您的镜像的示例存活和就绪探针。有了这些探针，用户便可放心部署您的镜像，确保在容器准备好处理流量之前，流量不会路由到容器，并且如果进程进入不健康状态，容器将重启。

4.1.2.10. 模板

考虑为您的镜像提供一个示例模板。用户借助模板可轻松利用有效的配置快速部署您的镜像。模板应包括与镜像一同记录的存活和就绪探针，以保证完整性。

4.2. 包括镜像中的元数据

定义镜像元数据有助于 OpenShift Container Platform 更好地使用您的容器镜像，允许 OpenShift Container Platform 使用您的镜像为开发人员创造更好的体验。例如，您可以添加元数据以提供有用的镜像描述，或针对可能也需要的其他镜像提供建议。

本主题仅定义当前用例集所需的元数据。以后可能还会添加其他元数据或用例。

4.2.1. 定义镜像元数据

您可使用 **Dockerfile** 中的 **LABEL** 指令来定义镜像元数据。标签与环境变量的相似之处在于标签是附加到镜像或容器中的键值对。标签与环境变量的不同之处在于标签对运行中的应用程序不可见，可用于快速查找镜像和容器。

有关 **LABEL** 指令的更多信息，请参阅 Docker 文档。

标签名称通常具有命名空间。命名空间会进行相应设置，以反映将要提取和使用标签的项目。对于 OpenShift Container Platform，命名空间应设置为 **io.openshift**；而对于 Kubernet，命名空间应设置为 **io.k8s**。

有关格式的详细信息，请参阅 Docker 自定义元数据文档。

表 4.1. 所支持的元数据

变量	描述
io.openshift.tags	该标签包含一个标签列表，以逗号分隔的字符串值的列表表示。标签是将容器镜像归类到广泛功能区域的方法。标签有助于 UI 和生成工具在应用程序创建过程中建议相关容器镜像。 `LABEL io.openshift.tags mongodb,mongodb24,nosql`
io.openshift.wants	指定标签列表，如果您未向容器镜像附带给定标签，则生成工具和 UI 可使用该列表提供相关建议。例如，如果容器镜像需要 **mysql** 和 **redis**，而您未向容器镜像附带 **redis** 标签，则 UI 可能会建议您将此镜像添加到部署中。 `LABEL io.openshift.wants mongodb,redis`
io.k8s.description	该标签可用于向容器镜像用户提供有关此镜像所提供服务或功能的更详细信息。然后 UI 可结合使用此描述与容器镜像名称，为最终用户提供更人性化的信息。 `LABEL io.k8s.description The MySQL 5.5 Server with master-slave replication support`
io.openshift.non-scalable	镜像可以使用此变量来表明它不支持扩展。UI 然后将这一信息传达给该镜像的用户。不可扩展表示 **replicas** 值的最初设置不应超过 **1**。 `LABEL io.openshift.non-scalable true`

io.openshift.min-memory 和 **io.openshift.min-cpu**	该标签建议容器镜像正常工作可能需要的资源量。UI 可能会警告用户：部署此容器镜像可能会超出其用户配额。值必须与 Kubernetes 数量兼容。
	`LABEL io.openshift.min-memory 16Gi`
	`LABEL io.openshift.min-cpu 4`

4.3. 使用 SOURCE-TO-IMAGE 从源代码创建镜像

Source-to-Image (S2I) 是一种框架，它可以轻松地将应用程序源代码作为输入，生成可运行编译的应用程序的新镜像。

使用 S2I 构建可重复生成的容器镜像的主要优点是便于开发人员使用。作为构建器镜像作者，您必须理解两个基本概念，构建过程和 S2I 脚本，才能让您的镜像提供最佳的 S2I 性能。

4.3.1. 了解 source-to-image 构建过程

构建过程包含以下三个基本元素，这些元素组合成最终的容器镜像：

- 源
- Source-to-image(S2I)脚本
- 构建器镜像

S2I 生成带有构建器镜像的 Dockerfile 作为第一个 **FROM** 指令。然后，由 S2I 生成的 Dockerfile 会被传递给 Buildah。

4.3.2. 如何编写 Source-to-image 脚本

您可以使用任何编程语言编写 S2I 脚本，只要脚本可在构建器镜像中执行。S2I 支持多种提供 **assemble/run/save-artifacts** 脚本的选项。每次构建时按以下顺序检查所有这些位置：

1. 构建配置中指定的脚本。
2. 在应用程序源 **.s2i/bin** 目录中找到的脚本。
3. 在默认镜像 URL 中找到的带有 **io.openshift.s2i.scripts-url** 标签的脚本。

镜像中指定的 **io.openshift.s2i.scripts-url** 标签和构建配置中指定的脚本都可以采用以下形式之一：

- **image:///path_to_scripts_dir**：镜像中 S2I 脚本所处目录的绝对路径
- **file:///path_to_scripts_dir**：主机上 S2I 脚本所处目录的相对或绝对路径
- **http(s)://path_to_scripts_dir**：S2I 脚本所处目录的 URL

表 4.2. S2I 脚本

脚本	描述
assemble	**assemble** 用来从源代码构建应用程序工件，并将其放置在镜像内部的适当目录中的脚本。这个脚本是必需的。此脚本的工作流为： 1. 可选：恢复构建工件。如果要支持增量构建，确保同时定义了 **save-artifacts**。 2. 将应用程序源放在所需的位置。 3. 构建应用程序工件。 4. 将工件安装到适合它们运行的位置。
run	**run** 脚本将执行您的应用程序。这个脚本是必需的。

脚本	描述
save-artifacts	**save-artifacts** 脚本将收集所有可加快后续构建过程的依赖项。这个脚本是可选的。例如： • 对于 Ruby，由 Bundler 安装的 **gem**。 • 对于 Java，**.m2** 内容。 这些依赖项会收集到一个 **tar** 文件中，并传输到标准输出。
usage	借助 **usage** 脚本，可以告知用户如何正确使用您的镜像。这个脚本是可选的。
test/run	借助 **test/run** 脚本，可以创建一个进程来检查镜像是否正常工作。这个脚本是可选的。该流程的建议工作流是： 1. 构建镜像。 2. 运行镜像以验证 **usage** 脚本。 3. 运行 **s2i build** 以验证 **assemble** 脚本。 4. 可选：再次运行 **s2i build**，以验证 **save-artifacts** 和 **assemble** 脚本的保存和恢复工件功能。 5. 运行镜像，以验证测试应用程序是否正常工作。  **注意** 建议将 **test/run** 脚本构建的测试应用程序放置到镜像存储库中的 **test/test-app** 目录。

S2I 脚本示例

以下示例 S2I 脚本采用 Bash 编写。每个示例都假定其 **tar** 内容解包到 **/tmp/s2i** 目录中。

assemble 脚本：

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
    mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all
```

```
# install the artifacts
make install
popd
```

run 脚本：

```
#!/bin/bash

# run the application
/opt/application/run.sh
```

save-artifacts 脚本：

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
    # all deps contents to tar stream
    tar cf - deps
fi
popd
```

usage 脚本：

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

其他资源

- S2I 镜像创建教程

4.4. 关于测试 SOURCE-TO-IMAGE 镜像

作为 Source-to-Image (S2I) 构建程序镜像创建者，您可在本地测试 S2I 镜像，并使用 OpenShift Container Platform 构建系统进行自动化测试和连续集成。

为了成功运行 S2I 构建，S2I 需要存在 **assemble** 和 **run** 脚本。提供 **save-artifacts** 脚本可重复利用构建工件，而提供 **usage** 脚本则可确保有人在 S2I 以外运行容器镜像时，使用情况信息能够打印到控制台上。

测试 S2I 镜像的目的在于确保所有这些描述命令均能正常工作，即使基本容器镜像已改变或命令所用工具已更新也不受影响。

4.4.1. 了解测试要求

test 脚本的标准位置为 **test/run**。该脚本由 OpenShift Container Platform S2I 镜像构建程序调用，可以是简单的 Bash 脚本，也可以是静态的 Go 二进制文件。

test/run 脚本会执行 S2I 构建，因此您的 **$PATH** 中必须有 S2I 二进制文件。必要情况下，请遵循 S2I README 中的安装说明。

S2I 结合了应用程序源代码与构建程序镜像，因此为了对其进行测试，您需要一个示例应用程序源来验证该源是否成功转换成了可运行的容器镜像。示例应用程序应简单，但也应执行 **assemble** 和 **run** 脚本的关键步骤。

4.4.2. 生成脚本和工具

S2I 工具随附功能强大的生成工具，可加快新 S2I 镜像的创建过程。**s2i create** 命令生成所有必要的 S2I 脚本和测试工具以及 **Makefile**：

```
$ s2i create <image_name> <destination_directory>
```

所生成的 **test/run** 脚本必须经过调整才可使用，但它为开始开发提供了一个良好起点。

注意

由 **s2i create** 命令生成的 **test/run** 脚本要求示例应用程序源位于 **test/test-app** 目录中。

4.4.3. 本地测试

本地运行 S2I 镜像测试的最简单方法是使用所生成的 **Makefile**。

如果未使用 **s2i create** 命令，则可复制以下 **Makefile** 模板，并将 **IMAGE_NAME** 参数替换为您的镜像名称。

Makefile 示例

```
IMAGE_NAME = openshift/ruby-20-centos7
CONTAINER_ENGINE := $(shell command -v podman 2> /dev/null | echo docker)

build:
 ${CONTAINER_ENGINE} build -t $(IMAGE_NAME) .

.PHONY: test
test:
 ${CONTAINER_ENGINE} build -t $(IMAGE_NAME)-candidate .
 IMAGE_NAME=$(IMAGE_NAME)-candidate test/run
```

4.4.4. 基本测试工作流

test 脚本会假定您已构建要测试的镜像。如果需要，请先构建 S2I 镜像。运行以下任一命令：

- 如果使用 Podman，请运行以下命令：

  ```
  $ podman build -t <builder_image_name>
  ```

- 如果使用 Docker，请运行以下命令：

  ```
  $ docker build -t <builder_image_name>
  ```

以下步骤描述测试 S2I 镜像构建程序的默认工作流：

1. 验证 **usage** 脚本是否正在工作：
   - 如果使用 Podman，请运行以下命令：

     ```
     $ podman run <builder_image_name> .
     ```

   - 如果使用 Docker，请运行以下命令：

     ```
     $ docker run <builder_image_name> .
     ```

2. 构建镜像：

   ```
   $ s2i build file:///path-to-sample-app _<BUILDER_IMAGE_NAME>_
   _<OUTPUT_APPLICATION_IMAGE_NAME>_
   ```

3. 可选：如果支持 **save-artifacts**，请再次运行第 2 步，验证保存和恢复工件是否正常工作。
4. 运行容器：
   - 如果使用 Podman，请运行以下命令：

     ```
     $ podman run <output_application_image_name>
     ```

   - 如果使用 Docker，请运行以下命令：

     ```
     $ docker run <output_application_image_name>
     ```

5. 验证容器是否正在运行，应用程序是否有所反应。

通常，运行这些步骤便足以说明构建程序镜像是否按预期工作。

4.4.5. 使用 OpenShift Container Platform 构建镜像

有了 **Dockerfile** 和组成新 S2I 构建程序镜像的其他工件后，您可以将它们放入 git 存储库中，并使用 OpenShift Container Platform 来构建和推送（push）镜像。定义指向您的存储库的 Docker 构建。

如果 OpenShift Container Platform 实例托管在一个公共 IP 地址上，则每次推送（push）到您的 S2I 构建程序镜像 GitHub 存储库时，均可触发构建。

您还可使用 **ImageChangeTrigger** 来基于您所更新的 S2I 构建程序镜像来触发应用程序的重新构建。

第 5 章 管理镜像

5.1. 管理镜像概述

您可通过 OpenShift Container Platform 与镜像交互并设置镜像流，具体取决于镜像 registry 所处的位置、这些 registry 的任何身份验证要求以及您预期的构建和部署性能。

5.1.1. 镜像概述

镜像流包含由标签识别的任意数量的容器镜像。提供相关镜像的单一虚拟视图，类似于容器镜存储库。

通过监控镜像流，构建和部署可在添加或修改新镜像时收到通知，并通过分别执行构建或部署来作出反应。

5.2. 标记镜像

以下章节提供在容器镜像上下文中使用镜像标签（tag）的概述和说明，以便使用 OpenShift Container Platform 镜像流及其标签。

5.2.1. 镜像标签

镜像标签（tag）是应用于存储库中容器镜像的标签，用于将特定镜像与镜像流中的其他镜像区分开来。标签通常代表某种版本号。例如，这里 **:v3.11.59-2** 是标签：

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

您可以向镜像添加其他标签。例如，可为镜像分配 **:v3.11.59-2** 和 **:latest** 标签。

OpenShift Container Platform 提供 **oc tag** 命令，该命令类似于 **docker tag** 命令，但是在镜像流上运行，而非直接在镜像上运行。

5.2.2. 镜像标签惯例

镜像随时间不断发展，其标签反应了这一点。一般来说，镜像标签会始终指向最新镜像构建。

如果标签名称中嵌入太多信息，如 **v2.0.1- May-2019**，则标签仅指向镜像的一个版本，不会更新。使用默认镜像修剪选项，此类镜像不会被删除。在庞大集群中，为每个修改后的镜像创建新标签这种模式最终可能会使用早已过期的镜像的多余标签元数据来填充 etcd 数据存储。

如果标签命名为 **v2.0**，则镜像修改的可能性更大。这会导致标签历史记录更长，镜像修剪器 (pruner) 也更有可能删除旧的和未使用的镜像。

您可自行决定标签命名惯例，下面提供了一些 **<image_name>:<image_tag>** 格式的示例：

表 5.1. 镜像标签命名惯例

描述	示例
修订	**myimage:v2.0.1**
架构	**myimage:v2.0-x86_64**

描述	示例
基础镜像	**myimage:v1.2-centos7**
最新（可能不稳定）	**myimage:latest**
最新稳定	**myimage:stable**

如果标签名称中需要日期，请定期检查旧的和不受支持的镜像以及 **istags**，并予以删除。否则，您可能遇到保留的旧镜像导致资源使用量增加的情况。

5.2.3. 向镜像流中添加标签

OpenShift Container Platform 中的镜像流包含 0 个或更多由标签标识的容器镜像。

有各种不同类型的标签可用。默认行为使用 **permanent** 标签，指向一段时间内的特定镜像。如果正在使用 **permanent** 标签并且源更改，则目的地的标签不会更改。

tracking 标签表示，在导入源标签期间对目的地标签的元数据进行了更新。

流程

- 您可使用 **oc tag** 命令向镜像流中添加标签：

```
$ oc tag <source> <destination>
```

例如：要将 **ruby** 镜像流 **static-2.0**标签配置为始终引用 **ruby**镜像流**2.0** 标签的当前镜像：

```
$ oc tag ruby:2.0 ruby:static-2.0
```

这会在 **ruby** 镜像流中创建名为 **static-2.0** 的新镜像流标签。运行 **oc tag** 时，新标签会直接引用 **ruby:2.0** 镜像流标签所指向的镜像 id，而所指向的镜像不会改变。

- 为确保目标标签在源标签更改时进行更新，请使用 **--alias=true** 标志：

```
$ oc tag --alias=true <source> <destination>
```

注意

使用跟踪标签创建持久别名，如 **latest** 或 **stable**。该标签只在单一镜像流中正常工作。试图创建跨镜像流别名会出错。

- 您还可添加 **--scheduled=true** 标志来定期刷新或重新导入目的地标签。周期在系统级别进行全局配置。

- **--reference** 标志会创建一个非导入的镜像流标签。该标签持久指向源位置。
如果您想指示 OpenShift Container Platform 始终从集成的 registry 中获取标记的镜像，请使用 **--reference-policy=local**。registry 使用 pull-through 功能为客户端提供镜像。默认情况下，镜像 Blob 由 registry 在本地进行镜像。因此，下次需要时便可更快拉取（pull）。只要镜像流具有不安全的注解，或者标签具有不安全的导入策略，该标志也允许从不安全的 registry 拉取（pull），无需向容器运行时提供 **--insecure-registry**。

5.2.4. 从镜像流中删除标签

您可以从镜像流中删除标签。

流程

- 要从镜像流运行中完全删除标签：

```
$ oc delete istag/ruby:latest
```

或：

```
$ oc tag -d ruby:latest
```

5.2.5. 引用镜像流中的镜像

您可以通过以下引用类型，使用标签来引用镜像流中的镜像。

表 5.2. 镜像流引用类型

引用类型	描述
ImageStreamTag	**ImageStreamTag** 用于引用或检索给定镜像流和标签的镜像。
ImageStreamImage	**ImageStreamImage** 用于引用或检索给定镜像流和镜像 **sha** ID 的镜像。
DockerImage	**DockerImage** 用于引用或检索给定外部 registry 的镜像。其名称使用标准 Docker **拉取（pull）规格**。

查看镜像流定义示例时，您可能会发现它们包含 **ImageStreamTag** 的定义以及对 **DockerImage** 的引用，但与 **ImageStreamImage** 无关。

这是因为当您将镜像导入或标记到镜像流时，OpenShift Container Platform 中会自动创建 **ImageStreamImage** 对象。您不必在用于创建镜像流的任何镜像流定义中显式定义 **ImageStreamImage** 对象。

流程

- 要引用给定镜像流和标签的镜像，请使用 **ImageStreamTag**：

```
<image_stream_name>:<tag>
```

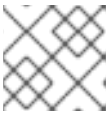
- 要引用给定镜像流的镜像和镜像 **sha** ID，请使用 **ImageStreamImage**：

```
<image_stream_name>@<id>
```

<id> 是针对特定镜像的不可变标识符，也称摘要。

- 要引用或检索给定外部 registry 的镜像，请使用 **DockerImage**：

```
openshift/ruby-20-centos7:2.0
```



注意

如果未指定标签，则会假定使用 **latest**标签。

此外，您还可引用第三方 registry：

```
registry.redhat.io/rhel7:latest
```

或者带有摘要的镜像：

```
centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b2
8e
```

5.3. 镜像拉取（PULL）策略

Pod 中的每个容器均有容器镜像。在创建了镜像并将其推送（push）到 registry 后，即可在 Pod 中引用它。

5.3.1. 镜像拉取（pull）策略概述

OpenShift Container Platform 在创建容器时，会使用容器的 **imagePullPolicy** 来决定是否应在启动容器前拉取（pull）镜像。**imagePullPolicy** 有三个可能的值：

表 5.3. imagePullPolicy 值

值	描述
Always	始终拉取（pull）镜像。
IfNotPresent	仅拉取（pull）节点上不存在的镜像。
Never	永不拉取（pull）镜像。

如果没有指定容器的 **imagePullPolicy** 参数，OpenShift Container Platform 会根据镜像标签来设置。

1. 如果标签为 **latest**，OpenShift Container Platform 会将 **imagePullPolicy** 默认设置为 **Always**。
2. 否则，OpenShift Container Platform 会将 **imagePullPolicy** 默认设置为 **IfNotPresent**。

5.4. 使用镜像 PULL SECRET

如果您使用 OpenShift 镜像 registry，并且从位于同一项目中的镜像流拉取 (pull)，则您的 Pod 服务帐户应已具备正确的权限，且无需额外操作。

然而，对于其他场景，例如在 OpenShift Container Platform 项目间或从安全 registry 引用镜像，则还需其他配置步骤。

您可以从 Red Hat OpenShift Cluster Manager 获取镜像 pull secret 。此 pull secret 名为 **pullSecret**。

您可以使用此 pull secret 来使用所含授权机构提供的服务进行身份验证，这些服务包括为 OpenShift Container Platform 组件提供容器镜像的 Quay.io 和 registry.redhat.io。

5.4.1. 允许 pod 在项目间引用镜像

使用 OpenShift 镜像 registry 时，要允许 **project-a** 中的 pod 引用 **project-b** 中的镜像，**project-a** 中的服务帐户必须绑定到 **project-b** 中的 **system:image-puller** 角色：

注意

在创建 pod 服务帐户或命名空间时，请等待服务帐户置备了 docker pull secret；如果在其服务帐户被完全置备前创建 pod，则 pod 无法访问 OpenShift 镜像 registry。

流程

1. 要允许 **project-a** 中的 pod 引用 **project-b** 中的镜像，请将 **project-a** 中的服务帐户绑定到 **project-b** 中的 **system:image-puller** 角色。

```
$ oc policy add-role-to-user \
    system:image-puller system:serviceaccount:project-a:default \
    --namespace=project-b
```

   添加该角色后，**project-a** 中引用默认服务帐户的 pod 能够从 **project-b** 拉取（pull）镜像。

2. 要允许访问 **project-a** 中的任意服务帐户，请使用组：

```
$ oc policy add-role-to-group \
    system:image-puller system:serviceaccounts:project-a \
    --namespace=project-b
```

5.4.2. 允许 Pod 引用其他安全 registry 中的镜像

要从其他私有或安全 registry 中拉取安全容器，您必须从容器客户端凭证（如 Docker 或 Podman）创建一个 pull secret，并将其添加到您的服务帐户中。

Docker 和 Podman 都使用配置文件来存储身份验证详情，以登录到安全或不安全的 registry：

- docker ：默认情况下，**Docker** 使用 **$HOME/.docker/config.json**。
- **Podman**: 默认情况下，Podman 使用 **$HOME/.config/containers/auth.json**。

如果您之前已登录到安全或不安全的 registry，则这些文件会存储您的身份验证信息。

注意

如果 Docker 和 Podman 凭证文件和关联的 pull secret 都包含对同一 registry 的多个引用，如 **quay.io** 和 **quay.io/<example_repository>**。但是，Docker 和 Podman 都不支持完全相同的 registry 路径的多个条目。

config.json 文件示例

```
{
  "auths":{
    "cloud.openshift.com":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io/repository-main":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    }
  }
}
```

pull secret 示例

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

流程

- 从现有身份验证文件创建 secret：
  - 对于使用 **.docker/config.json** 的 Docker 客户端，请输入以下命令：

    ```
    $ oc create secret generic <pull_secret_name> \
        --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
        --type=kubernetes.io/dockerconfigjson
    ```

  - 对于使用 **.config/containers/auth.json** 的 Podman 客户端，请输入以下命令：

    ```
    $ oc create secret generic <pull_secret_name> \
        --from-file=<path/to/.config/containers/auth.json> \
        --type=kubernetes.io/podmanconfigjson
    ```

- 如果您还没有安全 registry 的 Docker 凭证文件，则可运行以下命令创建一个 secret：

  ```
  $ oc create secret docker-registry <pull_secret_name> \
      --docker-server=<registry_server> \
      --docker-username=<user_name> \
  ```

```
    --docker-password=<password> \
    --docker-email=<email>
```

- 要使用 secret 为 Pod 拉取镜像，您必须将 secret 添加到您的服务帐户中。本例中服务帐户的名称应与 Pod 使用的服务帐户的名称匹配。默认服务帐户是 **default**：

```
$ oc secrets link default <pull_secret_name> --for=pull
```

5.4.2.1. 通过委托身份验证从私有 registry 拉取（pull）

私有 registry 可将身份验证委托给单独服务。这种情况下，必须为身份验证和 registry 端点定义镜像 pull secret。

流程

1. 为委托的身份验证服务器创建 secret：

```
$ oc create secret docker-registry \
    --docker-server=sso.redhat.com \
    --docker-username=developer@example.com \
    --docker-password=******** \
    --docker-email=unused \
    redhat-connect-sso

secret/redhat-connect-sso
```

2. 为私有 registry 创建 secret：

```
$ oc create secret docker-registry \
    --docker-server=privateregistry.example.com \
    --docker-username=developer@example.com \
    --docker-password=******** \
    --docker-email=unused \
    private-registry

secret/private-registry
```

5.4.3. 更新全局集群 pull secret

您可以通过替换当前的 pull secret 或附加新的 pull secret 来更新集群的全局 pull secret。

重要

要将集群转让给另一所有者，您必须首先在 OpenShift Cluster Manager 中启动转让，然后更新集群中的 pull secret。在不启动 OpenShift Cluster Manager 中的传输的情况下更新集群的 pull secret 会导致集群停止在 OpenShift Cluster Manager 中报告 Telemetry 指标。

有关 传输集群所有权的更多信息，请参阅 Red Hat OpenShift Cluster Manager 文档中的 "Transferring cluster ownership"。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 可选：要将新的 pull secret 附加到现有 pull secret 中，请完成以下步骤：

   a. 输入以下命令下载 pull secret：

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' ><pull_secret_location> 1
```

   1 提供 pull secret 文件的路径。

   b. 输入以下命令来添加新 pull secret：

```
$ oc registry login --registry="<registry>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

   1 提供新的 registry。您可以在同一个 registry 中包含多个软件仓库，例如：**--registry="<registry/my-namespace/my-repository>"**。

   2 提供新 registry 的凭据。

   3 提供 pull secret 文件的路径。

   另外，您可以对 pull secret 文件执行手动更新。

2. 输入以下命令为您的集群更新全局 pull secret：

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> 1
```

   1 提供新 pull secret 文件的路径。

   该更新将推广至所有节点，可能需要一些时间，具体取决于集群大小。

注意

从 OpenShift Container Platform 4.7.4 开始，对全局 pull secret 的更改不再触发节点排空或重启。

第 6 章 管理镜像流

镜像流提供了一种方式来持续创建和更新容器镜像。随着镜像改进，标签可用于分配新版本号并跟踪变化。本文档描述了对镜像流的管理方式。

6.1. 为什么使用镜像流

镜像流及其关联标签提供了一个用于从 OpenShift Container Platform 中引用容器镜像的抽象集。镜像流及其标签用于查看可用镜像，确保您使用所需的特定镜像，即使存储库中的镜像发生变化也是如此。

镜像流不含实际镜像数据，它提供了相关镜像的一个单独的虚拟视图，类似于镜像存储库。

您可配置构建（Build）和部署（Depolyment）来监测一个镜像流的通知。当新的镜像被添加时，执行相应的构建或部署。

例如，如果部署正在使用某个镜像并且创建了该镜像的新版本，则会自动执行部署以获取镜像的新版本。

但是，如果部署或构建所用的 imagestreamtag 没有更新，则即使更新了容器镜像 registry 中的容器镜像，构建或部署仍会继续使用之前的，已知良好的镜像。

源镜像可存储在以下任一位置：

- OpenShift Container Platform 集成的 registry。
- 一个外部 registry，如 registry.redhat.io 或 quay.io。
- OpenShift Container Platform 集群中的其他镜像流。

当您定义引用镜像流标签的对象时，如构建或部署配置，您将指向镜像流标签而不是存储库。您在构建或部署应用程序时，OpenShift Container Platform 会使用 imagestreamtag 来查询 Docker 存储库，以找到相应的镜像 ID，并使用正确的镜像。

镜像流元数据会与其他集群信息一起存储在 etcd 实例中。

使用镜像流有以下几大优势：

- 您可以添加标签、回滚标签和快速处理镜像，而无需使用命令行重新执行 push 操作。
- 当一个新镜像被推送（push）到 registry 时，可触发构建和部署。另外，OpenShift Container Platform 还针对 Kubernetes 对象等其他资源提供了通用触发器。
- 您可以为定期重新导入标记标签。如果源镜像已更改，则这个更改会被发现并反应在镜像流中。取决于构建或部署的具体配置，这可能会触发构建和/或部署流程。
- 您可使用细粒度访问控制来共享镜像，快速向整个团队分发镜像。
- 如果源更改，imagestreamtag 仍将指向已知良好的镜像版本，以确保您的应用程序不会意外中断。
- 您可以通过镜像流对象的权限配置安全性，以了解谁可以查看和使用镜像。
- 在集群级别上缺少读取或列出镜像权限的用户仍可使用镜像流来检索项目中标记的镜像。

6.2. 配置镜像流

ImageStream 对象文件包含以下元素。

镜像流对象定义

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample (2)
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d (3)
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 (4)
    - created: 2017-09-01T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest (5)
```

(1) 镜像流的名称。

(2) Docker 存储库路径，在此处可推送（push）新镜像，以在此镜像流中添加或更新镜像。

(3) 此镜像流标签当前引用的 SHA 标识符。引用此镜像流标签的资源使用此标识符。

(4) 此镜像流标签之前引用的 SHA 标识符。可用于回滚至旧镜像。

(5) 镜像流标签名称。

6.3. 镜像流镜像

从镜像流内部到特定镜像 ID 的镜像流镜像点。

镜像流镜像允许您从标记了镜像的特定镜像流中检索有关镜像的元数据。

每当您将镜像导入或标记到镜像流时，OpenShift Container Platform 中会自动创建镜像流镜像对象。您不必在用于创建镜像流的任何镜像流定义中显式定义镜像流镜像对象。

镜像流镜像包含来自存储库的镜像流名称和镜像 ID，用 @ 符号分隔：

```
<image-stream-name>@<image-id>
```

要引用 **ImageStream** 对象示例中的镜像，镜像流镜像如下所示：

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.4. 镜像流标签

镜像流标签是指向镜像流中镜像的命名指针。缩写为 **istag**。镜像流标签用于引用或检索给定镜像流和标签的镜像。

镜像流标签可引用任何本地管理或外部管理的镜像。它包含镜像历史记录，表示为标签曾指向的所有镜像的堆栈。每当特定镜像流标签下标记了新的或现有镜像时，该镜像将置于历史记录堆栈的第一位置。以前，顶层位置位于第二个位置。这样便于回滚，从而让标签再次指向历史镜像。

以下镜像流标签来自 **ImageStream** 对象：

历史记录中有两个镜像的镜像流标签

```
kind: ImageStream
apiVersion: image.openshift.io/v1
metadata:
  name: my-image-stream
# ...
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
    - created: 2017-09-01T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest
# ...
```

镜像流标签可以是持久性标签，也可以是跟踪标签。

- 本文标签是特定于版本的标签，指向镜像的特定版本，如 Python 3.5。
- 跟踪标签是引用标签，跟随另一个镜像流标签，并可更新以更改它们跟随的镜像，如符号链接。这些新等级无法保证向后兼容。
  例如，OpenShift Container Platform 附带的 **latest** 镜像流标签是跟踪标签。这意味着，当有新级别可用时，使用 **latest** 镜像流标签的用户会更新为镜像提供的框架的最新级别。指向 **v3.10** 的 **latest** 镜像流标签可以随时更改为 **v3.11**。请务必注意，这些 **latest** 镜像流标签的行为与 Docker **latest** 标签不同。在本例中，**latest** 镜像流标签不指向 Docker 存储库中的最新镜像。它指向另一个镜像流标签，可能并非镜像的最新版本。例如，如果 **latest** 镜像流标签指向 **v3.10** 镜像，则当发布了 **3.11** 版时，**latest** 标签不会自动更新至 **v3.11**，并保持 **v3.10**，直到手动更新为指向 **v3.11** 镜像流标签。

注意

跟踪标签仅限于单个镜像流，无法引用其他镜像流。

您可以根据自己的需要创建自己的镜像流标签。

镜像流标签由镜像流名称和一个标签组成，用冒号隔开：

```
<imagestream name>:<tag>
```

例如：为引用前面 **ImageStream** 对象示例中的 **sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d** 镜像，镜像流标签将是：

```
origin-ruby-sample:latest
```

6.5. 镜像流更改触发器

当有新版本上游镜像时，镜像流触发器支持自动调用构建和部署。

例如，修改镜像流标签时，构建和部署可以自动启动。实现方法是通过监控特定镜像流标签并在检测到变化时通知构建或部署。

6.6. 镜像流映射

集成的 registry 收到新镜像时，便会创建镜像流映射并将其发送至 OpenShift Container Platform，从而提供镜像的项目、名称、标签和镜像元数据。

注意

配置镜像流映射是高级功能。

该信息用于创建新镜像（如果尚未存在）并将此镜像标记到镜像流中。OpenShift Container Platform 存储每个镜像的完整元数据，如命令、入口点和环境变量。OpenShift Container Platform 中的镜像不可变，名称最长 63 个字符。

以下镜像流映射示例结果是将镜像标记为 **test/origin-ruby-sample:latest**：

镜像流映射对象定义

```
apiVersion: image.openshift.io/v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
```

```
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Labels:
        build-date: 2015-12-23
        io.k8s.description: Platform for building and running Ruby 2.2 applications
        io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-sample:latest
        io.openshift.build.commit.author: Ben Parees <bparees@users.noreply.github.com>
        io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
        io.openshift.build.commit.id: 00cadc392d39d5ef9117cbc8a31db0889eedd442
        io.openshift.build.commit.message: 'Merge pull request #51 from php-coder/fix_url_and_sti'
        io.openshift.build.commit.ref: master
        io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
        io.openshift.build.source-location: https://github.com/openshift/ruby-hello-world.git
        io.openshift.builder-base-version: 8d95148
        io.openshift.builder-version: 8847438ba06307f86ac877465eadc835201241df
        io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
        io.openshift.tags: builder,ruby,ruby22
        io.s2i.scripts-url: image:///usr/libexec/s2i
        license: GPLv2
        name: CentOS Base Image
        vendor: CentOS
      User: "1001"
      WorkingDir: /opt/app-root/src
    Container: 86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
    ContainerConfig:
      AttachStdout: true
      Cmd:
```

```
      - /bin/sh
      - -c
      - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Hostname: ruby-sample-build-1-build
      Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
      OpenStdin: true
      StdinOnce: true
      User: "1001"
      WorkingDir: /opt/app-root/src
    Created: 2016-01-29T13:40:00Z
    DockerVersion: 1.8.2.fc21
    Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
    Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
    Size: 441976279
    apiVersion: "1.0"
    kind: DockerImage
  dockerImageMetadataVersion: "1.0"
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.7. 使用镜像流

以下小节介绍了如何使用镜像流和镜像流标签。

重要

不要在默认项目中运行工作负载或共享对默认项目的访问权限。为运行核心集群组件保留默认项目。

以下默认项目被视为具有高度特权：**default**, **kube-public**, **kube-system**, **openshift**, **openshift-infra**, **openshift-node**, 其他系统创建的项目的标签 **openshift.io/run-level** 被设置为 **0** 或 **1**。依赖于准入插件（如 pod 安全准入、安全性上下文约束、集群资源配额和镜像引用解析）的功能无法在高特权项目中工作。

6.7.1. 获取有关镜像流的信息

您可获取有关镜像流的常规信息及其指向的所有标签的详细信息。

流程

- 要获取有关镜像流的常规信息及其指向的所有标签的详细信息，请输入以下命令：

```
$ oc describe is/<image-name>
```

例如：

```
$ oc describe is/python
```

输**出示例**

```
Name:   python
Namespace:  default
Created:  About a minute ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   1

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago
```

- 要获取有关特定镜像流标签的所有信息，请输入以下命令：

```
$ oc describe istag/<image-stream>:<tag-name>
```

例如：

```
$ oc describe istag/python:latest
```

输**出示例**

```
Image Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image: centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name:  sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created: 2 minutes ago
Image Size: 251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created: 2 weeks ago
Author:  <none>
Arch:  amd64
Entrypoint: container-entrypoint
```

```
Command: /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir: /opt/app-root/src
User:  1001
Exposes Ports: 8080/tcp
Docker Labels: build-date=20170801
```

注意

输出的信息多于显示的信息。

- 输入以下命令发现镜像流标签支持的构架或操作系统：

```
$ oc get istag <image-stream-tag> -ojsonpath="{range .image.dockerImageManifests[*]}
{.os}/{.architecture}{'\n'}{end}"
```

例如：

```
$ oc get istag busybox:latest -ojsonpath="{range .image.dockerImageManifests[*]}
{.os}/{.architecture}{'\n'}{end}"
```

输**出示例**

```
linux/amd64
linux/arm
linux/arm64
linux/386
linux/mips64le
linux/ppc64le
linux/riscv64
linux/s390x
```

6.7.2. 为镜像流添加标签

您可以向镜像流添加其他标签。

流程

- 使用 `oc tag` 命令添加指向其中一个现有标签的标签：

```
$ oc tag <image-name:tag1> <image-name:tag2>
```

例如：

```
$ oc tag python:3.5 python:latest
```

输**出示例**

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

- 确认镜像流有两个标签，**3.5** 指向外部容器镜像，**latest** 指向同一镜像，因为它基于第一个标签创建而成。

```
$ oc describe is/python
```

输**出示例**

```
Name:   python
Namespace:  default
Created:  5 minutes ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   2

latest
  tagged from
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      5 minutes ago
```

6.7.3. 为外部镜像添加标签

您可为外部镜像添加标签。

流程

- 通过使用 **oc tag** 命令执行所有标签相关操作，添加指向内部或外部镜像的标签：

```
$ oc tag <repository/image> <image-name:tag>
```

例如，该命令可将 **docker.io/python:3.6.0** 镜像映射到 **python** 镜像流中的 **3.6** 标签。

```
$ oc tag docker.io/python:3.6.0 python:3.6
```

输**出示例**

```
Tag python:3.6 set to docker.io/python:3.6.0.
```

如果外部镜像安全，则您必须创建带有凭证的 secret 以访问该 registry。

6.7.4. 更新镜像流标签

您可以更新标签以反映镜像流中的另一标签。

流程

- 更新标签：

```
$ oc tag <image-name:tag> <image-name:latest>
```

例如，以下命令更新了 **latest** 标签，以反映镜像流中的 **3.6** 标签：

```
$ oc tag python:3.6 python:latest
```

输**出示例**

```
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

6.7.5. 删除镜像流标签

您可以从镜像流中删除旧标签。

流程

- 从镜像流中删除旧标签：

```
$ oc tag -d <image-name:tag>
```

例如：

```
$ oc tag -d python:3.6
```

输**出示例**

```
Deleted tag default/python:3.6
```

如需有关 Cluster Samples Operator 如何处理已弃用的镜像流标签的更多信息，请参阅从 Cluster Samples Operator 中删除已弃用的镜像流标签。

6.7.6. 配置定期导入镜像流标签

使用外部容器镜像 registry 时，如需定期重新导入镜像（例如为了获取最新安全更新），可使用 **--scheduled** 标志。

流程

1. 调度导入镜像：

```
$ oc tag <repository/image> <image-name:tag> --scheduled
```

例如：

```
$ oc tag docker.io/python:3.6.0 python:3.6 --scheduled
```

输出示例

```
Tag python:3.6 set to import docker.io/python:3.6.0 periodically.
```

该命令可使 OpenShift Container Platform 定期更新该特定镜像流标签。此周期是集群范围的设置，默认设为 15 分钟。

2. 删除定期检查，重新运行上述命令，但忽略 **--scheduled** 标志。这会将其行为重置为默认值。

```
$ oc tag <repositiory/image> <image-name:tag>
```

6.8. 导入和使用镜像和镜像流

以下小节介绍了如何导入和使用镜像流。

6.8.1. 从私有容器镜像仓库（registry）导入镜像和镜像流

镜像流可以被配置为从需要身份验证的私有镜像仓库中导入标签和镜像元数据。如果您将 Cluster Samples Operator 用来拉取内容的位置改为 registry.redhat.io 以外的位置，则适用这个过程。

注意

从不安全或安全容器镜像仓库导入时，secret 中定义的容器镜像仓库 URL 必须包含 **:80** 端口后缀，或在尝试从容器仓库仓库导入时不使用 secret。

流程

1. 您必须通过输入以下命令来创建一个用于存储凭证的 **secret** 对象：

```
$ oc create secret generic <secret_name> --from-file=.dockerconfigjson=
<file_absolute_path> --type=kubernetes.io/dockerconfigjson
```

2. 配置 secret 后，请创建新镜像流或输入 **oc import-image** 命令：

```
$ oc import-image <imagestreamtag> --from=<image> --confirm
```

在导入过程中，OpenShift Container Platform 会提取 secret，并将其提供给远程方。

6.8.1.1. 允许 Pod 引用其他安全 registry 中的镜像

要从其他私有或安全 registry 中拉取安全容器，您必须从容器客户端凭证（如 Docker 或 Podman）创建一个 pull secret，并将其添加到您的服务帐户中。

Docker 和 Podman 都使用配置文件来存储身份验证详情，以登录到安全或不安全的 registry：

- docker ：默认情况下，**Docker** 使用 **$HOME/.docker/config.json**。
- **Podman**: 默认情况下，Podman 使用 **$HOME/.config/containers/auth.json**。

如果您之前已登录到安全或不安全的 registry，则这些文件会存储您的身份验证信息。

注意

如果 Docker 和 Podman 凭证文件和关联的 pull secret 都包含对同一 registry 的多个引用，如 **quay.io** 和 **quay.io/<example_repository>**。但是，Docker 和 Podman 都不支持完全相同的 registry 路径的多个条目。

config.json 文件示例

```
{
  "auths":{
    "cloud.openshift.com":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io/repository-main":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    }
  }
}
```

pull secret 示例

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

流程

- 从现有身份验证文件创建 secret：
  - 对于使用 **.docker/config.json** 的 Docker 客户端，请输入以下命令：

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

- 对于使用 **.config/containers/auth.json** 的 Podman 客户端，请输入以下命令：

```
$ oc create secret generic <pull_secret_name> \
    --from-file=<path/to/.config/containers/auth.json> \
    --type=kubernetes.io/podmanconfigjson
```

- 如果您还没有安全 registry 的 Docker 凭证文件，则可运行以下命令创建一个 secret：

```
$ oc create secret docker-registry <pull_secret_name> \
  --docker-server=<registry_server> \
  --docker-username=<user_name> \
  --docker-password=<password> \
  --docker-email=<email>
```

- 要使用 secret 为 Pod 拉取镜像，您必须将 secret 添加到您的服务帐户中。本例中服务帐户的名称应与 Pod 使用的服务帐户的名称匹配。默认服务帐户是 **default**：

```
$ oc secrets link default <pull_secret_name> --for=pull
```

6.8.2. 使用清单列表

使用 **oc import-image** 或 **oc tag** CLI 命令时，您可以通过添加 **--import-mode** 标志来导入单个子清单或清单列表的所有清单。

请参考以下命令，以创建包含单个子清单或多架构镜像的镜像流。

流程

- 输入以下命令来创建包含多架构镜像的镜像流，并将导入模式设置为 **PreserveOriginal** ：

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal' --reference-policy=local --confirm
```

输出**示例**

```
---
Arch:           <none>
Manifests:      linux/amd64
sha256:6e325b86566fafd3c4683a05a219c30c421fbccbf8d87ab9d20d4ec1131c3451
                linux/arm64
sha256:d8fad562ffa75b96212c4a6dc81faf327d67714ed85475bf642729703a2b5bf6
                linux/ppc64le
sha256:7b7e25338e40d8bdeb1b28e37fef5e64f0afd412530b257f5b02b30851f416e1
---
```

- 或者，输入以下命令使用 **Legacy** 导入模式导入镜像，这会丢弃清单列表并导入单个子清单：

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='Legacy' --confirm
```

注意

--import-mode= 默认值为 **Legacy**。排除这个值，或者没有指定 **Legacy** 或 **PreserveOriginal**，会导入单个子清单。无效的导入模式会返回以下错误：**error: valid ImportMode values are Legacy or PreserveOriginal**。

限制
使用清单列表有以下限制：

- 在某些情况下，用户可能想直接使用子清单。当运行 **oc adm prune images** 或 **CronJob** 修剪器运行时，它们无法检测何时使用子清单列表。因此，使用 **oc adm prune images** 或 **CronJob** 修剪器的管理员可能会删除整个清单列表，包括子清单列表。
  为避免这种限制，您可以根据标签或摘要使用清单列表。

6.8.2.1. 配置定期导入清单列表

要定期重新导入清单列表，您可以使用 **--scheduled** 标志。

流程

- 输入以下命令将镜像流设置为定期更新清单列表：

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal' --scheduled=true
```

6.8.2.2. 导入清单列表时配置 SSL/TSL

要在导入清单列表时配置 SSL/TSL，您可以使用 **--insecure** 标志。

流程

- 设置 **--insecure=true**，以便导入清单列表跳过 SSL/TSL 验证。例如：

```
$ oc import-image <multiarch-image-stream-tag> --from=<registry>/<project_name>/<image-
name> \
--import-mode='PreserveOriginal' --insecure=true
```

6.8.3. 为 --import-mode 指定架构

您可以通过排除或包含 **--import-mode=** 标志来在多架构和单一构架之间交换导入的镜像流

流程

- 运行以下命令，通过排除 **--import-mode=** 标志，将镜像流从多架构更新至单一架构：

```
$ oc import-image <multiarch-image-stream-tag> --from=<registry>/<project_name>/<image-
name>
```

- 运行以下命令，将镜像流从单架构更新至多架构：

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal'
```

6.8.4. --import-mode 的配置字段

下表描述了 **--import-mode=** 标志可用的选项：

参数	描述
Legacy	**--import-mode** 的默认选项。指定后，清单列表将被丢弃，并导入单个子清单。平台会按照优先级顺序进行选择： 1. 标签注解 2. control plane 架构 3. Linux/AMD64 4. 列表中的第一个清单
PreserveOriginal	指定后，会保留原始清单。对于清单列表，将导入清单列表及其所有子清单。

第 7 章 KUBERNETES 资源使用镜像流

作为 OpenShift Container Platform 的原生资源，镜像流可用于 OpenShift Container Platform 中的所有原生资源，如 **Build** 或 **DeploymentConfigs** 资源。也可以将它们用于原生 Kubernetes 资源，如 **Job**、**ReplicationController**、**ReplicaSet** 或 Kubernetes **Deployment** 资源。

7.1. 使用 KUBERNETES 资源启用镜像流

在 Kubernetes 资源中使用镜像流时，您只能引用位于与资源相同的项目中的镜像流。镜像流引用必须包含单个片段值，如 **ruby:2.5**，其中 **ruby** 是镜像流的名称，它具有名为 **2.5** 的标签，并位于与进行引用的资源相同的项目中。

重要

不要在默认项目中运行工作负载或共享对默认项目的访问权限。为运行核心集群组件保留默认项目。

以下默认项目被视为具有高度特权：**default**, **kube-public**, **kube-system**, **openshift**, **openshift-infra**, **openshift-node**，其他系统创建的项目的标签 **openshift.io/run-level** 被设置为 **0** 或 **1**。依赖于准入插件（如 pod 安全准入、安全性上下文约束、集群资源配额和镜像引用解析）的功能无法在高特权项目中工作。

使用 Kubernetes 资源启用镜像流的方法有两种：

- 启用针对特定资源的镜像流解析。这只允许此资源使用 image 字段中的镜像流名称。
- 在镜像流上启用镜像流解析。这允许指向此镜像流的所有资源在 image 字段中使用它。

流程

您可以使用 **oc set image-lookup** 对镜像流上的特定资源或镜像流解析启用镜像流解析功能。

1. 要允许所有资源引用名为 **mysql** 的镜像流，请输入以下命令：

```
$ oc set image-lookup mysql
```

这会将 **Imagestream.spec.lookupPolicy.local** 字段设置为 true。

启用镜像查询的镜像流

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/display-name: mysql
  name: mysql
  namespace: myproject
spec:
  lookupPolicy:
    local: true
```

启用后，会为镜像流中的所有标签启用此行为。

2. 然后，您可以查询镜像流并查看是否设置了选项：

```
-
```

```
$ oc set image-lookup imagestream --list
```

您可以在特定资源上启用镜像查找。

- 要允许名为 **mysql** 的 Kubernetes 部署使用镜像流，请运行以下命令：

  ```
  $ oc set image-lookup deploy/mysql
  ```

  这会在部署上设置 **alpha.image.policy.openshift.io/resolve-names** 注解。

  启**用**镜**像**查询**部署**

  ```
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: mysql
    namespace: myproject
  spec:
    replicas: 1
    template:
      metadata:
        annotations:
          alpha.image.policy.openshift.io/resolve-names: '*'
      spec:
        containers:
        - image: mysql:latest
          imagePullPolicy: Always
          name: mysql
  ```

您可以禁用镜像查找。

- 要禁用镜像查找，使用 **--enabled=false**:

  ```
  $ oc set image-lookup deploy/mysql --enabled=false
  ```

第 8 章 在镜像流更改时触发更新

当更新镜像流标签以指向新镜像时，OpenShift Container Platform 可以自动采取行动将新镜像推出到使用旧镜像的资源。您可以根据引用镜像流标签的资源类型以不同方式配置此行为。

8.1. OPENSHIFT CONTAINER PLATFORM 资源

OpenShift Container Platform Deployment 配置和构建配置可通过更改镜像流标签自动触发。可使用更新的镜像流标签引用的镜像的新值运行触发的操作。

8.2. 触发 KUBERNETES 资源

Kubernetes 资源没有用于触发的字段，这与部署和构建配置不同（在部署和构建配置中包括作为 API 定义的一组控制触发器字段）。您可以使用 OpenShift Container Platform 中的注解请求触发器。

该注解定义如下：

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    image.openshift.io/triggers:
      [
       {
         "from": {
           "kind": "ImageStreamTag", 1
           "name": "example:latest", 2
           "namespace": "myapp" 3
         },
         "fieldPath": "spec.template.spec.containers[?(@.name==\"web\")].image", 4
         "paused": false 5
       },
      # ...
      ]
# ...
```

1. 必需： **kind** 是要从中触发的资源，必须是 **ImageStreamTag**。
2. 必需： **name** 必须是镜像流标签的名称。
3. 可选： **namespace** 默认为对象的命名空间。
4. 必需： **fieldPath** 是到更改的 JSON 路径。此字段受限制，仅接受通过 ID 或索引完全匹配容器的 JSON 路径表达式。对于 pod，JSON 路径为 **spec.containers[? (@.name='web')].image**。
5. 可选： **paused** 代表触发器是否暂停，默认值为 **false**。将 **paused** 设置为 **true** 以临时禁用这个触发器。

当其中一个核心 Kubernetes 资源同时包含 pod 模板和此注解时，OpenShift Container Platform 会尝试使用当前与触发器引用的镜像流标签关联的镜像来更新对象。更新针对指定的 **fieldPath** 进行。

可以包含 pod 模板和注解的核心 Kubernetes 资源示例包括：

- **CronJobs**
- **部署**
- **StatefulSets**
- **DaemonSets**
- **Jobs**
- **ReplicationController**
- **Pods**

8.3. 在 KUBERNETES 资源上设置镜像触发器

在部署中添加镜像触发器时，您可以使用 **oc set triggers** 命令。例如，此流程中的示例命令将镜像更改触发器添加到名为 **example** 的部署中，以便在更新 **example:latest** 镜像流标签时，部署中的 **web** 容器使用新的镜像值。此命令在部署资源上设置正确的 **image.openshift.io/triggers** 注解。

流程

- 输入 **oc set triggers** 命令来触发 Kubernetes 资源：

```
$ oc set triggers deploy/example --from-image=example:latest -c web
```

使用触发器注解的部署示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    image.openshift.io/triggers: '[{"from":
{"kind":"ImageStreamTag","name":"example:latest"},"fieldPath":"spec.template.spec.containers[
?(@.name==\"container\")].image"}]'
# ...
```

除非部署暂停，否则此 pod 模板更新自动导致使用新镜像值进行部署。

第 9 章 镜像配置资源

使用以下流程配置镜像 registry。

9.1. 镜像控制器配置参数

Image.config.openshift.io/cluster 资源包含有关如何处理镜像的集群范围信息。规范且唯一有效的名称是 **cluster**。它的 **spec** 提供以下配置参数。

注意

参数，如 **DisableScheduledImport**, **MaxImagesBulkImportedPerRepository**, **MaxScheduledImportsPerMinute**, **ScheduledImageImportMinimumIntervalSeconds**, **InternalRegistryHostname** 不可配置。

参数	描述
allowedRegistriesForImport	限制普通用户可从中导入镜像的容器镜像 registry。将此列表设置为您信任包含有效镜像且希望应用程序能够从中导入的 registry。有权从 API 创建镜像或 **ImageStreamMappings** 的用户不受此策略的影响。通常只有集群管理员具有适当权限。 这个列表中的每个项包含由 registry 域名指定的 registry 的位置。 **domainname**：指定 registry 的域名。如果 registry 使用非标准的**80** 或 **443** 端口，则该端口也应包含在域名中。 **insecure**：不安全指示 registry 是否安全。默认情况下，如果未另行指定，registry 假定为安全。
additionalTrustedCA	对包含 **image stream import**、**pod image pull**、**openshift-image-registry pullthrough** 和构建期间应受信任的额外 CA 的配置映射的引用。 此配置映射的命名空间为 **openshift-config**。ConfigMap 的格式是使用 registry 主机名作为键，使用 PEM 编码证书作为值，用于每个要信任的额外 registry CA。
externalRegistryHostnames	提供默认外部镜像 registry 的主机名。只有在镜像 registry 对外公开时才应设置外部主机名。第一个值用于镜像流中的 **publicDockerImageRepository** 字段。该值必须采用 **hostname[:port]** 格式。

参数	描述
registrySources	包含用于决定容器运行时在访问构建和 pod 的镜像时应如何处理个别 registry 的配置。例如，是否允许不安全的访问。它不包含内部集群 registry 的配置。 **insecureRegistries**：无有效 TLS 证书或仅支持 HTTP 连接的 registry。要指定所有子域，请在域名中添加星号 (*) 通配符字符作为前缀。例如：***.example.com**。您可以在 registry 中指定单独的软件仓库。例如：**reg1.io/myrepo/myapp:latest**。 **blockedRegistries**：拒绝镜像拉取（pull）和推送（push）操作的 registry。要指定所有子域，请在域名中添加星号 (*) 通配符字符作为前缀。例如：***.example.com**。您可以在 registry 中指定单独的软件仓库。例如：**reg1.io/myrepo/myapp:latest**。允许所有其他 registry。 **allowedRegistries**：允许镜像拉取（pull）和推送（push）操作的 registry。要指定所有子域，请在域名中添加星号 (*) 通配符字符作为前缀。例如：***.example.com**。您可以在 registry 中指定单独的软件仓库。例如：**reg1.io/myrepo/myapp:latest**。阻止所有其他 registry。 **containerRuntimeSearchRegistries**：允许使用镜像短名称的镜像拉取（pull）和推送（push）操作的 registry。阻止所有其他 registry。 可以设置 **blockedRegistries** 或 **allowedRegistries**，但不能同时都被设置。

警告

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 **registry.redhat.io** 和 **quay.io** registry 和默认的 OpenShift 镜像 registry）都会被阻断。当使用参数时，为了避免 pod 失败，将所有 registry（包括 **registry.redhat.io** 和 **quay.io** registry）和 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。对于断开连接的集群，还应添加镜像的 registry。

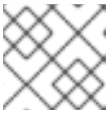
image.config.openshift.io/cluster 资源的 **status** 项包括了从集群观察到的值。

参数	描述
internalRegistryHostname	由控制 **internalRegistryHostname** 的 Image Registry Operator 设置。它设置默认 OpenShift 镜像 registry 的主机名。该值必须采用 **hostname[:port]** 格式。为实现向后兼容，您仍可使用 **OPENSHIFT_DEFAULT_REGISTRY** 环境变量，但该设置会覆盖环境变量。
externalRegistryHostnames	由 Image Registry Operator 设置，在镜像 registry 通过外部公开时为它提供外部主机名。第一个值用于镜像流中的 **publicDockerImageRepository** 字段。该值必须采用 **hostname[:port]** 格式。

9.2. 配置镜像 REGISTRY 设置

您可以通过编辑 **image.config.openshift.io/cluster** 自定义资源（CR）来配置镜像 registry 设置。当对 registry 的更改应用到 **image.config.openshift.io/cluster** CR 时，Machine Config Operator (MCO) 执行以下顺序操作：

1. 对节点进行 cordon 操作
2. 通过重启 CRI-O 应用更改
3. 取消记录节点



注意

MCO 在检测到更改时不会重启节点。

流程

1. 编辑 **image.config.openshift.io/cluster** 自定义资源：

```
$ oc edit image.config.openshift.io/cluster
```

以下是 **image.config.openshift.io/cluster** CR 示例：

```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: (2)
    - domainName: quay.io
      insecure: false
  additionalTrustedCA: (3)
    name: myconfigmap
  registrySources: (4)
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

(1) **Image**：包含有关如何处理镜像的集群范围信息。规范且唯一有效的名称是 **cluster**。

2 **allowedRegistriesForImport**：限制普通用户可从中导入镜像的容器镜像 registry。将此列表设置为您信任包含有效镜像且希望应用程序能够从中导入的 registry。有权从 API 创建镜

3 **additionalTrustedCA**：引用包含镜像流导入、Pod 镜像拉取、**openshift-image-registry** pullthrough 和构建期间受信任的额外证书颁发机构（CA）的配置映射。此配置映射的命名空间为 **openshift-config**。ConfigMap 的格式是使用 registry 主机名作为键，使用 PEM 证书作为值，用于每个要信任的额外 registry CA。

4 **registrySources** ：包含用于决定容器运行时在访问构建和 pod 的镜像时是否允许或阻止个别 registry 的配置。可以设置 **allowedRegistries** 参数或 **blockedRegistries** 参数，但不能同时设置这两个参数。您还可以定义是否允许访问允许使用镜像短名称的不安全的 registry。本例使用 **allowedRegistries** 参数，该参数定义允许使用的 registry。不安全 registry **insecure.com** 也被允许。**registrySources** 参数不包含内部集群 registry 的配置。

注意

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 registry.redhat.io 和 quay.io registry 和默认的 OpenShift 镜像 registry）都会被阻断。如果使用参数，为了避免 pod 失败，您必须将 **registry.redhat.io** 和 **quay.io** registry 以及 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。不要将 **registry.redhat.io** 和 **quay.io** registry 添加到 **blockedRegistries** 列表中。

使用 **allowedRegistries**、**blockedRegistries** 或 **insecureRegistries** 参数时，您可以在 registry 中指定单独的存储库。例如： **reg1.io/myrepo/myapp:latest**。

应避免使用不安全的外部 registry，以减少可能的安全性风险。

2. 要检查是否应用了更改，请列出您的节点：

```
$ oc get nodes
```

输出示例

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker                 65m   v1.27.3
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane          74m   v1.27.3
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane          75m   v1.27.3
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker                 65m   v1.27.3
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker                 63m   v1.27.3
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane          73m   v1.27.3
```

9.2.1. 添加特定的 registry

您可以通过编辑 **image.config.openshift.io/cluster** 自定义资源（CR）在 registry 中添加允许进行镜像拉取（pull）和推送（push）操作的 registry 列表（可选）。OpenShift Container Platform 会将对此 CR 的更改应用到集群中的所有节点。

在拉取或推送镜像时，容器运行时会搜索 **image.config.openshift.io/cluster** CR 的 **registrySources** 参数中列出的 registry。如果您在 **allowedRegistries** 参数下创建了 registry 列表，则容器运行时仅搜索这些 registry。不在列表中的 registry 会被阻断。

警告

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 **registry.redhat.io** 和 **quay.io** registry 和默认的 OpenShift 镜像 registry）都会被阻断。如果使用参数，为了避免 pod 失败，您必须将 **registry.redhat.io** 和 **quay.io** registry 以及 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。对于断开连接的集群，还应添加镜像的 registry。

流程

- 编辑 **image.config.openshift.io/cluster** 自定义资源：

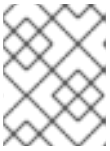
```
$ oc edit image.config.openshift.io/cluster
```

以下是一个带有允许列表的 **image.config.openshift.io/cluster** CR 示例：

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: 1
    allowedRegistries: 2
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 包含用于决定容器运行时在访问构建和 pod 的镜像时应如何处理个别 registry 的配置。它不包含内部集群 registry 的配置。

2. 指定 registry 以及该 registry 中的存储库（可选）用于镜像拉取（pull）和推送（push）操作。阻止所有其他 registry。



注意

可以设置 **allowedRegistries** 参数或 **blockedRegistries** 参数，但不能同时设置这两个参数。

Machine Config Operator（MCO）会监控 **image.config.openshift.io/cluster** 资源以了解对 registry 的任何更改。当 MCO 检测到更改时，它会在机器配置池(MCP)的节点中触发推出部署。允许的 registry 列表用于更新每个节点上的 **/etc/containers/policy.json** 文件中的镜像签名策略。对 **/etc/containers/policy.json** 文件的更改不需要节点排空。

验证

- 输入以下命令获取节点列表：

```
$ oc get nodes
```

输出示例

```
NAME              STATUS   ROLES                AGE   VERSION
<node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
```

1. 运行以下命令在节点上进入 debug 模式：

```
$ oc debug node/<node_name>
```

2. 出现提示时，在终端中输入 **chroot /host**：

```
sh-4.4# chroot /host
```

3. 输入以下命令检查 registry 是否已添加到策略文件中：

```
sh-5.1# cat /etc/containers/policy.json | jq '.'
```

以下策略表示，仅允许来自 example.com、quay.io 和 registry.redhat.io registry 中的镜像拉取和推送镜像：

例 9.1. 镜像签名策略文件示例

```
{
  "default":[
    {
      "type":"reject"
    }
  ],
  "transports":{
    "atomic":{
      "example.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "image-registry.openshift-image-registry.svc:5000":[
        {
```

```
            "type":"insecureAcceptAnything"
          }
        ],
        "insecure.com":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "quay.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "reg4.io/myrepo/myapp:latest":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "registry.redhat.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      },
      "docker":{
        "example.com":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "image-registry.openshift-image-registry.svc:5000":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "insecure.com":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "quay.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "reg4.io/myrepo/myapp:latest":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "registry.redhat.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      },
```

```
    "docker-daemon":{
      "":[
        {
          "type":"insecureAcceptAnything"
        }
      ]
    }
  }
}
```

注意

如果您的集群使用 **registrySources.insecureRegistries** 参数，请确保将任何不安全的 registry 包含在允许的列表中。

例如：

```
spec:
  registrySources:
    insecureRegistries:
    - insecure.com
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com
    - image-registry.openshift-image-registry.svc:5000
```

9.2.2. 阻塞特定的 registry

您可以通过编辑 **image.config.openshift.io/cluster** 自定义资源（CR）来阻止任何 registry 以及 registry 中的单独存储库。OpenShift Container Platform 会将对此 CR 的更改应用到集群中的所有节点。

在拉取或推送镜像时，容器运行时会搜索 **image.config.openshift.io/cluster** CR 的 **registrySources** 参数中列出的 registry。如果您在 **blockedRegistries** 参数下创建了 registry 列表，则容器运行时不会搜索这些 registry。允许所有其他 registry。

警告

为防止 pod 失败，请不要将 **registry.redhat.io** 和 **quay.io** registry 添加到 **blockedRegistries** 列表中，因为环境中有效负载镜像需要它们。

流程

- 编辑 **image.config.openshift.io/cluster** 自定义资源：

```
$ oc edit image.config.openshift.io/cluster
```

以下是一个带有块列表的 **image.config.openshift.io/cluster** CR 示例：

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: 1
    blockedRegistries: 2
    - untrusted.com
    - reg1.io/myrepo/myapp:latest
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 包含用于决定容器运行时在访问构建和 pod 的镜像时应如何处理个别 registry 的配置。它不包含内部集群 registry 的配置。

2. 指定 registry，并可选择性地指定 registry 中的存储库，以使它们不应该用于镜像拉取（pull）和推送（push）操作。允许所有其他 registry。

注意

可以设置 **blockedRegistries** registry 或 **allowedRegistries** registry，但不能同时设置这两个 registry。

Machine Config Operator（MCO）会监控 **image.config.openshift.io/cluster** 资源以了解对 registry 的任何更改。当 MCO 检测到更改时，它会排空节点，应用更改，并对节点进行 uncordon 处理。节点返回 **Ready** 状态后，在每个节点上的 **/etc/containers/registries.conf** 文件中会显示对被阻断的 registry 的更改。

验证

- 输入以下命令获取节点列表：

```
$ oc get nodes
```

输出示例

```
NAME              STATUS   ROLES                AGE   VERSION
<node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
```

1. 运行以下命令在节点上进入 debug 模式：

```
$ oc debug node/<node_name>
```

2. 出现提示时，在终端中输入 **chroot /host**：

```
sh-4.4# chroot /host
```

3. 输入以下命令检查 registry 是否已添加到策略文件中：

```
sh-5.1# cat etc/containers/registries.conf
```

以下示例显示，在进行镜像拉取和推送时，不使用 **untrusted.com**：

输**出示例**

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "untrusted.com"
  blocked = true
```

9.2.2.1. 阻塞一个 payload registry

在镜像配置中，您可以使用 **ImageContentSourcePolicy** (ICSP) 对象在断开连接的环境中阻断上游 payload registry。以下示例步骤演示了如何阻止 **quay.io/openshift-payload** payload registry。

流程

1. 使用 **ImageContentSourcePolicy** (ICSP) 对象创建镜像配置，以便将 payload 镜像到您的实例中 registry。以下示例 ICSP 文件对 payload **internal-mirror.io/openshift-payload** 进行了镜像：

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: my-icsp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - internal-mirror.io/openshift-payload
    source: quay.io/openshift-payload
```

2. 对象部署到节点上后，通过检查 **/etc/containers/registries.conf** 文件来验证镜像配置是否已设置：

输**出示例**

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

3. 使用以下命令来编辑 **image.config.openshift.io** 自定义资源文件：

```
$ oc edit image.config.openshift.io cluster
```

4. 要阻断 payload registry，请在 **image.config.openshift.io** 自定义资源文件中添加以下配置：

```
spec:
  registrySources:
    blockedRegistries:
     - quay.io/openshift-payload
```

验证

- 通过检查节点上的 **/etc/containers/registries.conf** 文件，验证上游 payload registry 是否被阻止。

  输**出示例**

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  blocked = true
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

9.2.3. 允许不安全的 registry

您可以通过编辑 **image.config.openshift.io/cluster** 自定义资源（CR）来添加不安全的 registry 及 registry 中的特定存储库（可选）。OpenShift Container Platform 会将对此 CR 的更改应用到集群中的所有节点。

没有使用有效 SSL 证书或不需要 HTTPS 连接的 registry 被视为是不安全的 registry。

警告

应避免使用不安全的外部 registry，以减少可能的安全性风险。

流程

- 编辑 **image.config.openshift.io/cluster** 自定义资源：

```
$ oc edit image.config.openshift.io/cluster
```

以下是一个带有不安全 registry 列表的 **image.config.openshift.io/cluster** CR 示例：

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
```

```
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: 1
    insecureRegistries: 2
    - insecure.com
    - reg4.io/myrepo/myapp:latest
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com 3
    - reg4.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1 包含用于决定容器运行时在访问构建和 pod 的镜像时应如何处理个别 registry 的配置。它不包含内部集群 registry 的配置。

2 指定不安全的 registry。您可以在该 registry 中指定存储库。

3 确保将任何不安全的 registry 包含在 **allowedRegistries** 列表中。

注意

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 registry.redhat.io 和 quay.io registry 和默认的 OpenShift 镜像 registry）都会被阻断。如果使用参数，为了避免 pod 失败，将所有 registry（包括 **registry.redhat.io** 和 **quay.io** registry）和 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。对于断开连接的集群，还应添加镜像的 registry。

Machine Config Operator（MCO）会监控 **image.config.openshift.io/cluster** CR 是否有对 registry 的更改，然后在检测到更改时排空并取消记录节点。节点返回 **Ready** 状态后，改为在每个节点的 **/etc/containers/registries.conf** 文件中列出的不安全和受阻 registry。

验证

- 要检查 registry 是否已添加到策略文件中，请在节点上使用以下命令：

```
$ cat /etc/containers/registries.conf
```

以下示例表示来自 **insecure.com** registry 的镜像是不安全的，并允许进行镜像拉取和推送。

输出示例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "insecure.com"
  insecure = true
```

9.2.4. 添加允许镜像短名称的 registry

您可以通过编辑 **image.config.openshift.io/cluster** 自定义资源（CR）来添加 registry 来搜索镜像短名称。OpenShift Container Platform 会将对此 CR 的更改应用到集群中的所有节点。

镜像简短名称允许您搜索镜像，而无需在 pull spec 中包含完全限定域名。例如： 您可以使用 **rhel7/etcd** 而不是 **registry.access.redhat.com/rhe7/etcd**。

在无法使用完整路径的情况下，您可以使用简短名称。例如，如果您的集群引用了 DNS 频繁变化的多个内部 registry，则需要更新 pull spec 中的完全限定域名并进行每次更改。在这种情况下，使用镜像简短名称可能很有用。

在拉取或推送镜像时，容器运行时会搜索 **image.config.openshift.io/cluster** CR 的 **registrySources** 参数中列出的 registry。如果您在 **containerRuntimeSearchRegistries** 参数下创建了 registry 列表，则容器运行时会搜索这些 registry。

警告

强烈建议不要将镜像短名称与公共 registry 搭配使用，因为如果公共 registry 需要身份验证，则镜像可能无法部署。将完全限定镜像名称与公共 registry 搭配使用。

红帽内部或私有 registry 通常支持使用镜像短名称。

如果您在 **containerRuntimeSearchRegistries** 参数下列出公共 registry （包括 **registry.redhat.io**、**docker.io** 和 **quay.io** registry），您可以将凭证公开给列表上的所有 registry，并存在对网络和 registry 进行攻击的风险。因为您只能有一个 pull secret 用于拉取镜像，所以由全局 pull secret 定义，该 secret 用于对该列表中的每个 registry 进行身份验证。因此，如果您在列表中包含公共 registry，则会出现安全风险。

如果每个公共 registry 需要不同的凭证，且集群不会在全局 pull secret 中列出公共 registry，则无法在 **containerRuntimeSearchRegistries** 参数下列出多个公共 registry。

对于需要身份验证的公共 registry，只有在 registry 具有其凭证存储在全局 pull secret 中时，才能使用镜像短名称。

Machine Config Operator（MCO）会监控 **image.config.openshift.io/cluster** 资源以了解对 registry 的任何更改。当 MCO 检测到更改时，它会排空节点，应用更改，并对节点进行 uncordon 处理。节点返回 **Ready** 状态后，如果添加了 **containerRuntimeSearchRegistries** 参数，MCO 会在每个带有列出 registry 的节点的 **/etc/containers/registries.conf.d** 目录中创建一个文件。该文件覆盖 **/etc/containers/registries.conf** 文件中的非全限定搜索 registry 的默认列表。没有办法回退到非全限定搜索 registry 的默认列表。

containerRuntimeSearchRegistries 参数只适用于 Podman 和 CRI-O 容器引擎。列表中的 registry 只能用于 pod 规格，不能用于构建和镜像流。

流程

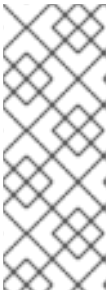
- 编辑 **image.config.openshift.io/cluster** 自定义资源：

```
$ oc edit image.config.openshift.io/cluster
```

以下是 **image.config.openshift.io/cluster** CR 示例：

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport:
    - domainName: quay.io
      insecure: false
  additionalTrustedCA:
    name: myconfigmap
  registrySources:
    containerRuntimeSearchRegistries: 1
    - reg1.io
    - reg2.io
    - reg3.io
    allowedRegistries: 2
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io
    - reg2.io
    - reg3.io
    - image-registry.openshift-image-registry.svc:5000
...
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 指定用于镜像简短名称的 registry。您应该只使用带有内部或私有 registry 的镜像短名称，以减少可能的安全问题。
2. 确保在 **containerRuntimeSearchRegistries** 下列出的任何 registry 都包含在 **allowedRegistries** 列表中。



注意

当定义 **allowedRegistries** 参数时，除非明确列出，否则所有 registry （包括 **registry.redhat.io** 和 **quay.io** registry 和默认的 OpenShift 镜像 registry）都会被阻断。如果使用此参数，为了避免 pod 失败，请将所有 registry（包括 **registry.redhat.io** 和 **quay.io** registry）和 **internalRegistryHostname** 添加到 **allowedRegistries** 列表中，因为环境中有效负载镜像需要它们。对于断开连接的集群，还应添加镜像的 registry。

验证

- 输入以下命令获取节点列表：

```
$ oc get nodes
```

输出示例

```
NAME              STATUS   ROLES                  AGE   VERSION
<node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
```

1. 运行以下命令在节点上进入 debug 模式：

```
$ oc debug node/<node_name>
```

2. 出现提示时，在终端中输入 **chroot /host**：

```
sh-4.4# chroot /host
```

3. 输入以下命令检查 registry 是否已添加到策略文件中：

```
sh-5.1# cat /etc/containers/registries.conf.d/01-image-searchRegistries.conf
```

输**出示例**

```
unqualified-search-registries = ['reg1.io', 'reg2.io', 'reg3.io']
```

9.2.5. 为镜像 registry 访问配置额外的信任存储

Image.config.openshift.io/cluster 自定资源可包含对配置映射的引用，该配置映射包含要在镜像 registry 访问期间被信任的额外证书颁发机构。

先决条件

- 证书颁发机构（CA）必须经过 PEM 编码。

流程

您可以在**openshift-config**命名空间中创建配置映射，并在 **image.config.openshift.io** 子定义资源中的 **AdditionalTrustedCA** 中使用其名称，以提供与外部 registry 联系时可以被信任的额外CA。

对于每个要信任的额外 registry CA，配置映射键是带有要信任此 CA 的端口的 registry 的主机名，而 PEM 证书内容是要信任的每个额外 registry CA。

镜像 registry CA 配置映射示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: | 1
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

1 如果 registry 带有端口，如 **registry-with-port.example.com:5000**，**:** 需要被 **..** 替换。

您可以按照以下过程配置其他CA。

- 配置其他CA：

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n
openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

9.3. 了解镜像 REGISTRY 仓库镜像

通过设置容器 registry 存储库镜像，您可以执行以下任务：

- 配置 OpenShift Container Platform 集群，以便重定向从源镜像 registry 上的存储库拉取（pull）镜像的请求，并通过已镜像 (mirror) 的镜像 registry 上的存储库来解决该请求。
- 为每个目标存储库识别多个已镜像 (mirror)的存储库，以确保如果一个镜像停止运作，仍可使用其他镜像。

OpenShift Container Platform 中的存储库镜像包括以下属性：

- 镜像拉取（pull）可应对 registry 停机的问题。
- 在断开连接的环境中的集群可以从关键位置（如 quay.io）拉取镜像，并让公司防火墙后面的 registry 提供请求的镜像。
- 发出镜像拉取（pull）请求时尝试特定 registry 顺序，通常最后才会尝试持久性 registry。
- 您所输入的镜像信息会添加到 OpenShift Container Platform 集群中每个节点上的 **/etc/containers/registries.conf** 文件中。

- 当节点从源存储库中请求镜像时，它会依次尝试每个已镜像的存储库，直到找到所请求的内容。如果所有镜像均失败，集群则会尝试源存储库。如果成功，则镜像拉取至节点中。

可通过以下方式设置存储库镜像：

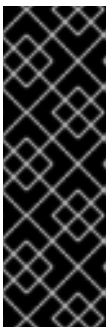
- 在 OpenShift Container Platform 安装中：
  通过拉取（pull） OpenShift Container Platform 所需的容器镜像，然后将这些镜像放至公司防火墙后，即可将 OpenShift Container Platform 安装到受限网络中的数据中心。

- 安装 OpenShift Container Platform 后：
  如果您没有在 OpenShift Container Platform 安装过程中配置镜像，您可以在安装后使用以下自定义资源 (CR) 对象之一进行配置：

  - **ImageDigestMirrorSet** (IDMS).此对象允许您使用摘要规格从镜像 registry 中拉取镜像。IDMS CR 可让您设置回退策略，在镜像拉取失败时继续尝试从源 registry 中拉取。

  - **ImageTagMirrorSet** (ITMS)。此对象允许您使用镜像标签从已镜像的 registry 中拉取镜像。ITMS CR 可让您设置回退策略，在镜像拉取失败时继续尝试从源 registry 中拉取。

  - **ImageContentSourcePolicy** (ICSP)。此对象允许您使用摘要规格从镜像 registry 中拉取镜像。如果镜像无法正常工作，ICSP 始终回退到源 registry。



重要

使用 **ImageContentSourcePolicy** (ICSP)对象配置存储库镜像是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。如果您有用于创建 **ImageContentSourcePolicy** 对象的 YAML 文件，您可以使用 **oc adm migrate icsp** 命令将这些文件转换为 **ImageDigestMirrorSet** YAML 文件。如需更多信息，请参阅以下部分"协调 ImageContentSourcePolicy (ICSP)文件以进行镜像 registry 存储库镜像"。

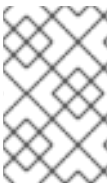
每个自定义资源对象都标识以下信息：

- 您希望镜像 (mirror) 的容器镜像存储库的源。

- 您希望为其提供从源存储库请求的内容的每个镜像存储库的单独条目。

对于新集群，您可以根据需要使用 IDMS、ITMS 和 ICSP CR 对象。但是，建议使用 IDMS 和 ITMS。

如果您升级了集群，则任何现有 ICSP 对象都会保持稳定，并且支持 IDMS 和 ICSP 对象。使用 ICSP 对象的工作负载可以按预期工作。但是，如果要利用 IDMS CR 中引入的回退策略，您可以使用 **oc adm migrate icsp** 命令将当前工作负载迁移到 IDMS 对象，如后面的**镜像 registry 存储库镜像**部分所示。迁移到 IDMS 对象不需要重启集群。



注意

如果您的集群使用 **ImageDigestMirrorSet**、**ImageTagMirrorSet** 或 **ImageContentSourcePolicy** 对象来配置存储库镜像，则只能使用镜像的 registry 的全局 pull secret。您不能在项目中添加 pull secret。

其他资源

- 如需有关全局 pull secret 的更多信息，请参阅更新全局集群 pull secret。

9.3.1. 配置镜像 registry 存储库镜像

您可以创建安装后镜像配置自定义资源 (CR)，将源镜像 registry 中的镜像拉取请求重定向到镜像 registry。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 通过以下方法配置已镜像的存储库：
   - 按照 Red Hat Quay 存储库镜像 中所述，使用 Red Hat Quay 来设置已镜像的存储库。使用 Red Hat Quay 有助于您将镜像从一个存储库复制到另一存储库，并可随着时间的推移重复自动同步这些存储库。
   - 使用 **skopeo** 等工具手动将镜像从源存储库复制到已镜像的存储库。
     例如：在 Red Hat Enterprise Linux（RHEL 7 或 RHEL 8）系统上安装 skopeo RPM 软件包后，使用 **skopeo** 命令，如下例所示：

     ```
     $ skopeo copy \
     docker://registry.access.redhat.com/ubi9/ubi-minimal:latest@sha256:5cf... \
     docker://example.io/example/ubi-minimal
     ```

     在本例中，您有一个名为 **example.io** 的容器镜像 registry，其中包含一个名为 **example** 的镜像存储库，您要将 **ubi9/ubi-minimal** 镜像从 **registry.access.redhat.com** 复制到其中。创建已镜像的 registry 后，您可以将 OpenShift Container Platform 集群配置为将源存储库的请求重定向到已镜像的存储库。
2. 登录您的 OpenShift Container Platform 集群。
3. 使用以下示例之一创建安装后镜像配置 CR：
   - 根据需要，创建一个 **ImageDigestMirrorSet** 或 **ImageTagMirrorSet** CR，将源和镜像 (mirror)替换为您自己的 registry、存储库对和镜像：

     ```
     apiVersion: config.openshift.io/v1 (1)
     kind: ImageDigestMirrorSet (2)
     metadata:
       name: ubi9repo
     spec:
       imageDigestMirrors: (3)
       - mirrors:
         - example.io/example/ubi-minimal (4)
         - example.com/example/ubi-minimal (5)
         source: registry.access.redhat.com/ubi9/ubi-minimal (6)
         mirrorSourcePolicy: AllowContactingSource (7)
       - mirrors:
         - mirror.example.com/redhat
         source: registry.redhat.io/openshift4 (8)
         mirrorSourcePolicy: AllowContactingSource
       - mirrors:
         - mirror.example.com
         source: registry.redhat.io (9)
     ```

```
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage (10)
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example (11)
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com (12)
    mirrorSourcePolicy: AllowContactingSource
```

1. 指明此 CR 要使用的 API。这必须是 **config.openshift.io/v1**。
2. 根据 pull 类型指示对象类型：
   - **ImageDigestMirrorSet** ：提取摘要引用镜像。
   - **ImageTagMirrorSet** ：提取标签引用镜像。
3. 表示镜像拉取方法的类型，请执行以下任一方法：
   - **imageDigestMirrors** ：用于 **ImageDigestMirrorSet** CR。
   - **imageTagMirrors** ：用于 **ImageTagMirrorSet** CR。
4. 指明镜像 registry 和存储库的名称。
5. 可选：指定每个目标仓库的二级镜像存储库。如果一个镜像停机，则目标仓库可以使用另一个镜像。
6. 指明 registry 和存储库源，这是在镜像拉取规格中引用的存储库。
7. 可选：如果镜像拉取失败，则指示回退策略：
   - **AllowContactingSource** ：允许继续尝试从源存储库拉取镜像。这是默认值。
   - **NeverContactSource**: 防止继续尝试从源存储库拉取镜像。
8. 可选：指示 registry 中的命名空间，它允许您使用该命名空间中的任何镜像。如果您使用 registry 域作为源，则对象将应用到 registry 中的所有存储库。
9. 可选：指示一个 registry，它允许您使用该 registry 中的任何镜像。如果指定了 registry 名称，对象将应用到源 registry 中的所有存储库到镜像 registry。
10. 从 mirror **mirror.example.net/image@sha256:..** 拉取镜像 **registry.example.com/example/myimage@sha256:...**。
11. 从 mirror **mirror.example.net/image@sha256:...** 的源 registry 命名空间中拉取镜像 **registry.example.com/example/image@sha256:...**。
12. 从 mirror registry **example.net/registry-example-com/myimage@sha256:...** 中拉取镜像 **registry.example.com/myimage@sha256**。

- 创建 **ImageContentSourcePolicy** 自定义资源，将源和镜像替换为您自己的 registry、存储库对和镜像：

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: mirror-ocp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - mirror.registry.com:443/ocp/release 1
    source: quay.io/openshift-release-dev/ocp-release 2
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

1. 指定镜像 registry 和存储库的名称。
2. 指定包含所镜像内容的在线 registry 和存储库。

4. 创建新对象：

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

创建对象后，Machine Config Operator (MCO) 只会排空 **ImageTagMirrorSet** 对象的节点。MCO 不会排空 **ImageDigestMirrorSet** 和 **ImageContentSourcePolicy** 对象的节点。

5. 要检查是否应用了镜像的配置设置，请在其中一个节点上执行以下操作。

a. 列出您的节点：

```
$ oc get node
```

输出示例

```
NAME                         STATUS                     ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                        worker   7m   v1.28.5
ip-10-0-138-148.ec2.internal   Ready                         master   11m  v1.28.5
ip-10-0-139-122.ec2.internal   Ready                         master   11m  v1.28.5
ip-10-0-147-35.ec2.internal    Ready                        worker   7m   v1.28.5
ip-10-0-153-12.ec2.internal    Ready                        worker   7m   v1.28.5
ip-10-0-154-10.ec2.internal    Ready                        master   11m  v1.28.5
```

b. 启动调试过程以访问节点：

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

输出示例

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

c. 将您的根目录改为 **/host** ：

```
sh-4.2# chroot /host
```

d. 检查 **/etc/containers/registries.conf** 文件，确保已完成更改：

```
sh-4.2# cat /etc/containers/registries.conf
```

以下输出代表了应用安装后镜像配置 CR 的 **registry.conf** 文件。最后的两个条目分别标记为 **digest-only** 和 **tag-only**。

输出示例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
short-name-mode = ""

[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal" (1)

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal" (2)
    pull-from-mirror = "digest-only" (3)

  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"

  [[registry.mirror]]
    location = "mirror.example.net"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example/myimage"

  [[registry.mirror]]
    location = "mirror.example.net/image"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.redhat.io"

  [[registry.mirror]]
```

```
    location = "mirror.example.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.redhat.io/openshift4"

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
    pull-from-mirror = "digest-only"
[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal"
  blocked = true 4

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal-tag"
    pull-from-mirror = "tag-only" 5
```

1. 指明在 pull spec 中引用的存储库。
2. 指明该存储库的镜像。
3. 表示从 mirror 的镜像拉取是一个摘要引用镜像。
4. 表示为此存储库设置了 **NeverContactSource** 参数。
5. 表示从 mirror 的镜像拉取是一个标签引用镜像。

e. 从源拉取镜像到节点，并检查是否通过 mirror 解析。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi9/ubi-
minimal@sha256:5cf...
```

存储库镜像故障排除

如果存储库镜像流程未按规定工作，请使用以下有关存储库镜像如何工作的信息协助排查问题。

- 首个工作镜像用于提供拉取（pull）的镜像。
- 只有在无其他镜像工作时，才会使用主 registry。
- 从系统上下文，**Insecure** 标志用作回退。
- 最近更改了 **/etc/containers/registries.conf** 文件的格式。现在它是第 2 版，采用 TOML 格式。

9.3.2. 为镜像 registry 存储库镜像转换 ImageContentSourcePolicy (ICSP) 文件

使用 **ImageContentSourcePolicy** (ICSP)对象配置存储库镜像是一个已弃用的功能。此功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

ICSP 对象被 **ImageDigestMirrorSet** 和 **ImageTagMirrorSet** 对象替代，以配置存储库镜像。如果您有用于创建 **ImageContentSourcePolicy** 对象的 YAML 文件，您可以使用 **oc adm migrate icsp** 命令将这些文件转换为 **ImageDigestMirrorSet** YAML 文件。命令将 API 更新至当前版本，将 **kind** 值更改为

ImageDigestMirrorSet，并将 **spec.repositoryDigestMirrors** 更改为 **spec.imageDigestMirrors**。文件的其余部分不会改变。

因为迁移不会更改 **registry.conf** 文件，所以集群不需要重启。

有关 **ImageDigestMirrorSet** 或 **ImageTagMirrorSet** 对象的更多信息，请参阅上一节中的"配置镜像 registry 存储库镜像"。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 确保集群中具有 **ImageContentSourcePolicy** 对象。

流程

1. 使用以下命令，将一个或多个 **ImageContentSourcePolicy** YAML 文件转换为 **ImageDigestMirrorSet** YAML 文件：

```
$ oc adm migrate icsp <file_name>.yaml <file_name>.yaml <file_name>.yaml --dest-dir
<path_to_the_directory>
```

其中：

<file_name>

指定源 **ImageContentSourcePolicy** YAML 的名称。您可以列出多个文件名。

--dest-dir

可选：指定输出 **ImageDigestMirrorSet** YAML 的目录。如果未设置，则会将该文件写入当前目录中。

例如，以下命令可将 **icsp.yaml** 和 **icsp-2.yaml** 文件转换，并将新的 YAML 文件保存到 **idms-files** 目录中。

```
$ oc adm migrate icsp icsp.yaml icsp-2.yaml --dest-dir idms-files
```

输**出示例**

```
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi8repo.5911620242173376087.yaml
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi9repo.6456931852378115011.yaml
```

2. 运行以下命令来创建 CR 对象：

```
$ oc create -f <path_to_the_directory>/<file-name>.yaml
```

其中：

<path_to_the_directory>

如果使用 **--dest-dir** 标志，请指定目录的路径。

<file_name>

指定 **ImageDigestMirrorSet** YAML 的名称。

3. 在推出 IDMS 对象后，删除 ICSP 对象。

第 10 章 使用模板

下面章节介绍模板概述以及模板的创建和使用方法。

10.1. 了解模板

模板描述了一组可参数化和处理的对象，用于生成对象列表，供 OpenShift Container Platform 用于创建。可对模板进行处理，以创建您有权在项目中创建的任何内容，如服务、构建配置和部署配置。模板还可定义一系列标签（label），以应用到该模板中定义的每个对象。

您可以使用 CLI 从模板创建对象列表，或者如果模板已上传至项目或全局模板库，则可使用 web 控制台来创建。

10.2. 上传模板

如果您有定义模板的 JSON 或 YAML 文件，您可以使用 CLI 将模板上传到项目。此操作将模板保存到项目，供任何有适当权限访问该项目的用户重复使用。本主题后面会提供有关编写自己的模板的说明。

流程

- 使用以下方法之一上传模板：
  - 将模板上传到当前项目的模板库，并使用以下命令传递 JSON 或 YAML 文件：

    ```
    $ oc create -f <filename>
    ```

  - 使用 **-n** 选项与项目名称将模板上传到不同项目：

    ```
    $ oc create -f <filename> -n <project>
    ```

现在可使用 web 控制台或 CLI 选择该模板。

10.3. 使用 WEB 控制台创建应用程序

您可使用 web 控制台从模板创建应用程序。

流程

1. 在 Web 控制台导航菜单顶部的上下文选择器中，选择 **Developer**。
2. 在相关的项目中点 **+Add**
3. 点 **Developer Catalog** 标题中的 **All services**。
4. 点 **Type** 下的 **Builder Images** 以查看可用的构建器镜像。

注意

只有其注解中列出 **builder** 标签的 imagestreamtag 才会出现在此列表中，如下所示：

```
kind: "ImageStream"
```

```
apiVersion: "v1"
metadata:
  name: "ruby"
  creationTimestamp: null
spec:
# ...
  tags:
    - name: "2.6"
      annotations:
        description: "Build and run Ruby 2.6 applications"
        iconClass: "icon-ruby"
        tags: "builder,ruby" (1)
        supports: "ruby:2.6,ruby"
        version: "2.6"
# ...
```

(1) 此处包含 **builder** 可确保该镜像流标签作为构建程序出现在 web 控制台中。

5. 修改新应用程序屏幕中的设置，以配置对象来支持您的应用程序。

10.4. 使用 CLI 从模板创建对象

您可以使用 CLI 来处理模板，并使用所生成的配置来创建对象。

10.4.1. 添加标签

标签 (label) 用于管理和组织所生成的对象，如 pod。模板中指定的标签应用于从模板生成的每个对象。

流程

- 从以下命令行在模板中添加标签：

```
$ oc process -f <filename> -l name=otherLabel
```

10.4.2. 列出参数

模板的 **parameter** 部分列出了可覆盖的参数列表。

流程

1. 您可使用以下命令并指定要用的文件通过 CLI 列出参数：

```
$ oc process --parameters -f <filename>
```

或者，如果模板已上传：

```
$ oc process --parameters -n <project> <template_name>
```

例如，下面显示了在默认 **openshift** 项目中列出其中一个快速启动模板的参数时的输出：

```
$ oc process --parameters -n openshift rails-postgresql-example
```

输**出示例**

```
NAME                     DESCRIPTION
GENERATOR          VALUE
SOURCE_REPOSITORY_URL       The URL of the repository with your application source
code                                                  https://github.com/sclorg/rails-ex.git
SOURCE_REPOSITORY_REF       Set this to a branch name, tag or other ref of your
repository if you are not using the default branch
CONTEXT_DIR                 Set this to the relative path to your project if it is not in the root of
your repository
APPLICATION_DOMAIN          The exposed hostname that will route to the Rails service
rails-postgresql-example.openshiftapps.com
GITHUB_WEBHOOK_SECRET       A secret string used to configure the GitHub webhook
expression          [a-zA-Z0-9]{40}
SECRET_KEY_BASE             Your secret key for verifying the integrity of signed cookies
expression          [a-z0-9]{127}
APPLICATION_USER            The application user that is used within the sample application
to authorize access on pages                          openshift
APPLICATION_PASSWORD        The application password that is used within the sample
application to authorize access on pages                          secret
DATABASE_SERVICE_NAME       Database service name
postgresql
POSTGRESQL_USER             database username
expression          user[A-Z0-9]{3}
POSTGRESQL_PASSWORD         database password
expression          [a-zA-Z0-9]{8}
POSTGRESQL_DATABASE         database name
root
POSTGRESQL_MAX_CONNECTIONS  database max connections
10
POSTGRESQL_SHARED_BUFFERS   database shared buffers
12MB
```

该输出标识了在处理模板时使用类似正则表达式的生成器生成的几个参数。

10.4.3. 生成对象列表

您可以使用 CLI 来处理定义模板的文件，以便将对象列表返回到标准输出。

流程

1. 处理定义模板的文件进以将对象列表返回到标准输出：

```
$ oc process -f <filename>
```

或者，如果模板已上传到当前项目：

```
$ oc process <template_name>
```

2. 通过处理模板并将输出传送至 **oc create** 来从模板创建对象：

```
$ oc process -f <filename> | oc create -f -
```

或者，如果模板已上传到当前项目：

```
$ oc process <template> | oc create -f -
```

3. 您可以为每个要覆盖的 **<name>=<value>** 对添加 **-p** 选项，以覆盖文件中定义的任何参数值。参数引用可能会出现在模板项目内的任何文本字段中。
例如，在以下部分中，模板的 **POSTGRESQL_USER** 和 **POSTGRESQL_DATABASE** 参数被覆盖，以输出带有自定义环境变量的配置：

   a. 从模板创建对象列表

```
$ oc process -f my-rails-postgresql \
    -p POSTGRESQL_USER=bob \
    -p POSTGRESQL_DATABASE=mydatabase
```

   b. JSON 文件可重定向到文件，也可直接应用，而无需通过将已处理的输出传送到 **oc create** 命令来上传模板：

```
$ oc process -f my-rails-postgresql \
    -p POSTGRESQL_USER=bob \
    -p POSTGRESQL_DATABASE=mydatabase \
    | oc create -f -
```

   c. 如有大量参数，可将其保存到文件中，然后将此文件传递到 **oc process**：

```
$ cat postgres.env
POSTGRESQL_USER=bob
POSTGRESQL_DATABASE=mydatabase
```

```
$ oc process -f my-rails-postgresql --param-file=postgres.env
```

   d. 此外，您还可使用 **"-"** 作为 **--param-file** 的参数，从标准输入中读取环境：

```
$ sed s/bob/alice/ postgres.env | oc process -f my-rails-postgresql --param-file=-
```

10.5. 修改所上传的模板

您可编辑已上传至项目中的模板。

流程

- 修改已上传的模板：

```
$ oc edit template <template>
```

10.6. 使用即时应用程序和快速启动模板

OpenShift Container Platform 提供了很多默认的即时应用程序和快速启动模板，以便您轻松开始为不同语言创建新应用程序。提供了适用于 Rails (Ruby)、Django (Python)、Node.js、CakePHP (PHP) 和 Dancer (Perl) 的模板。您的集群管理员必须在默认的全局 **openshift** 项目中创建这些模板，以便您访问。

默认情况下，模板会使用 GitHub 上包含必要应用程序代码的公共源存储库进行构建。

流程

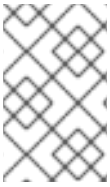
1. 您可以通过以下命令列出可用的默认即时应用程序和快速启动模板：

```
$ oc get templates -n openshift
```

2. 要修改源并构建您自己的应用程序版本：

   a. 对模板默认的 **SOURCE_REPOSITORY_URL** 参数引用的存储库进行分叉。

   b. 在从模板创建时，覆盖 **SOURCE_REPOSITORY_URL** 参数的值，从而指定您的分叉而非默认值。
   这样，模板创建的构建配置将指向应用程序代码的分叉，您可随意修改代码和重新构建应用程序。



注意

某些 Instant App 和 Quickstart 模板会定义一个数据库部署配置。它们定义的配置对数据库内容使用临时存储。这些模板仅限于演示目的，因为如果数据库 pod 因任何原因重启，所有数据库数据都将丢失。

10.6.1. 快速启动模板

快速启动模板是 OpenShift Container Platform 上运行的应用程序的基本示例。Quickstarts 提供多种语言和框架，并在模板中定义，模板由一组服务、构建配置和部署配置组成。该模板引用了构建和部署应用程序所需的镜像和源存储库。

要探索快速启动，请从模板创建应用程序。您的管理员必须已在 OpenShift Container Platform 集群中安装了这些模板，在这种情况下，您只需从 web 控制台中选择即可。

快速开始引用包含应用源代码的源存储库。要自定义 Quickstart，请分叉存储库，并在从模板创建应用程序时，用分叉的存储库替换默认的源存储库名称。这将导致使用您的源代码而非所提供的示例源来执行构建。然后，您可以更新源存储库中的代码，并启动新的构建来查看反映在所部署的应用程序中的更改。

10.6.1.1. Web 框架快速启动模板

这些快速启动模板提供了指定框架和语言的基本应用程序：

- Cakephp:包含 MySQL 数据库的 PHP web 框架
- Dancer:包含 MySQL 数据库的 Perl Web 框架
- Django:包含 PostgreSQL 数据库的 Python web 框架
- NodeJS：包含 MongoDB 数据库的 NodeJS web 应用程序
- Rails：Ruby web 框架（包括 PostgreSQL 数据库）

10.7. 编写模板

您可以定义新模板，以便轻松重新创建应用程序的所有对象。该模板将定义由其创建的对象以及一些元数据，以指导创建这些对象。

以下是简单模板对象定义 (YAML) 的示例：

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: redis-template
  annotations:
    description: "Description"
    iconClass: "icon-redis"
    tags: "database,nosql"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
      - name: REDIS_PASSWORD
        value: ${REDIS_PASSWORD}
      image: dockerfile/redis
      name: master
      ports:
      - containerPort: 6379
        protocol: TCP
parameters:
- description: Password used for Redis authentication
  from: '[A-Z0-9]{8}'
  generate: expression
  name: REDIS_PASSWORD
labels:
  redis: master
```

10.7.1. 编写模板描述

模板描述向用户介绍模板的作用，有助于用户在 web 控制台中搜索查找模板。除模板名称以外的其他元数据均为可选，但若有则会非常有用。除常规描述性信息外，元数据还应包含一组标签。实用标签包括与模板相关的语言名称，如 Java、PHP、Ruby 等。

以下是模板描述性元数据的示例：

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: cakephp-mysql-example 1
  annotations:
    openshift.io/display-name: "CakePHP MySQL Example (Ephemeral)" 2
    description: >-
      An example CakePHP application with a MySQL database. For more information
      about using this template, including OpenShift considerations, see
      https://github.com/sclorg/cakephp-ex/blob/master/README.md.


      WARNING: Any data stored will be lost upon pod destruction. Only use this
      template for testing." 3
    openshift.io/long-description: >-
```

```
    This template defines resources needed to develop a CakePHP application,
    including a build configuration, application DeploymentConfig, and
    database DeploymentConfig.  The database is stored in
    non-persistent storage, so this configuration should be used for
    experimental purposes only. 4
  tags: "quickstart,php,cakephp" 5
  iconClass: icon-php 6
  openshift.io/provider-display-name: "Red Hat, Inc." 7
  openshift.io/documentation-url: "https://github.com/sclorg/cakephp-ex" 8
  openshift.io/support-url: "https://access.redhat.com" 9
message: "Your admin credentials are ${ADMIN_USERNAME}:${ADMIN_PASSWORD}" 10
```

1. 模板的唯一名称。
2. 可由用户界面使用的简单、用户友好型名称。
3. 模板的描述。包含充足的详细信息，方便用户了解所部署的内容以及部署前须知的注意事项。还应提供其他信息链接，如 README 文件。可包括换行符来创建段落。
4. 其他模板描述。例如，这可按照服务目录显示。
5. 要与模板关联的标签，用于搜索和分组。添加将包含在其中一个提供的目录类别中的标签。请参见控制台常量文件中 **CATALOG_CATEGORIES** 中的 **id** 和 **categoryAliases**。此外，还可为整个集群自定义类别。
6. 在 web 控制台中与模板一同显示的图标。

例 10.1. 可用图标

- **icon-3scale**
- **icon-aerogear**
- **icon-amq**
- **icon-angularjs**
- **icon-ansible**
- **icon-apache**
- **icon-beaker**
- **icon-camel**
- **icon-capedwarf**
- **icon-cassandra**
- **icon-catalog-icon**
- **icon-clojure**
- **icon-codeigniter**
- **icon-cordova**

- **icon-datagrid**
- **icon-datavirt**
- **icon-debian**
- **icon-decisionserver**
- **icon-django**
- **icon-dotnet**
- **icon-drupal**
- **icon-eap**
- **icon-elastic**
- **icon-erlang**
- **icon-fedora**
- **icon-freebsd**
- **icon-git**
- **icon-github**
- **icon-gitlab**
- **icon-glassfish**
- **icon-go-gopher**
- **icon-golang**
- **icon-grails**
- **icon-hadoop**
- **icon-haproxy**
- **icon-helm**
- **icon-infinispan**
- **icon-jboss**
- **icon-jenkins**
- **icon-jetty**
- **icon-joomla**
- **icon-jruby**
- **icon-js**

- **icon-knative**
- **icon-kubevirt**
- **icon-laravel**
- **icon-load-balancer**
- **icon-mariadb**
- **icon-mediawiki**
- **icon-memcached**
- **icon-mongodb**
- **icon-mssql**
- **icon-mysql-database**
- **icon-nginx**
- **icon-nodejs**
- **icon-openjdk**
- **icon-openliberty**
- **icon-openshift**
- **icon-openstack**
- **icon-other-linux**
- **icon-other-unknown**
- **icon-perl**
- **icon-phalcon**
- **icon-php**
- **icon-play**
- **iconpostgresql**
- **icon-processserver**
- **icon-python**
- **icon-quarkus**
- **icon-rabbitmq**
- **icon-rails**
- **icon-redhat**

- **icon-redis**
- **icon-rh-integration**
- **icon-rh-spring-boot**
- **icon-rh-tomcat**
- **icon-ruby**
- **icon-scala**
- **icon-serverlessfx**
- **icon-shadowman**
- **icon-spring-boot**
- **icon-spring**
- **icon-sso**
- **icon-stackoverflow**
- **icon-suse**
- **icon-symfony**
- **icon-tomcat**
- **icon-ubuntu**
- **icon-vertx**
- **icon-wildfly**
- **icon-windows**
- **icon-wordpress**
- **icon-xamarin**
- **icon-zend**

7 提供模板的个人或组织的名称。

8 用于参考更多模板文档的 URL。

9 用于获取模板支持的 URL。

10 模板实例化时显示的说明消息。该字段应向用户介绍如何使用新建资源。显示消息前，对消息进行参数替换，以便输出中包含所生成的凭据和其他参数。其中包括用户应遵守的所有后续步骤文档链接。

10.7.2. 编写模板标签

模板可包括一组标签。这些标签添加到模板实例化时创建的各个对象中。采用这种方式定义标签可方便用户查找和管理从特定模板创建的所有对象。

以下是模板对象标签的示例：

```
kind: "Template"
apiVersion: "v1"
...
labels:
  template: "cakephp-mysql-example" 1
  app: "${NAME}" 2
```

1 标签（label）应用于从该模板创建的所有对象。

2 参数化标签也应用于从该模板创建的所有对象。对标签键和值均执行参数扩展。

10.7.3. 编写模板参数

允许用户提供一个值或在实例化模板时生成一个值作为参数。然后，该值将在引用参数的任意位置上被替换。可在对象列表字段中的任意字段中定义引用。这有助于生成随机密码，或允许用户提供自定义模板时所需的主机名或其他用户特定值。可通过以下两种方式引用参数：

- 作为字符串值，将格式为 **${PARAMETER_NAME}** 的值放在模板的任意字符串字段中。
- 作为 JSON 或 YAML 值，将格式为 **${{PARAMETER_NAME}}** 的值放在模板中的任意字段中。

使用 **${PARAMETER_NAME}** 语法时，可将多个参数引用合并到一个字段中，并可将引用嵌入到固定数据中，如 **"http://${PARAMETER_1}${PARAMETER_2}"**。两个参数值均将被替换，结果值将是一个带引号的字符串。

使用 **${{PARAMETER_NAME}}** 语法时，仅允许单个参数引用，不允许使用前导/尾随字符。执行替换后，结果值将不加引号，除非结果不是有效的 JSON 对象。如果结果不是有效的 JSON 值，则结果值会被添加引号并视为标准字符串。

单个参数可在模板中多次引用，且可在单个模板中使用两种替换语法来引用。

可提供默认值，如果您未提供其他值则使用默认值：

以下是将确切值设置为默认值的示例：

```
parameters:
  - name: USERNAME
    description: "The user name for Joe"
    value: joe
```

还可根据参数定义中指定的规则生成参数值，例如：

```
parameters:
  - name: PASSWORD
    description: "The random user password"
    generate: expression
    from: "[a-zA-Z0-9]{12}"
```

在上例中，处理生成一个由大小写字母和数字组成的 12 个字符长的随机密码。

可用语法并非完整的正则表达式语法。但是，您可以使用 **\w**、**\d**、**\a** 和 **\A** 修饰符：

- **[\w]{10}** 生成 10 个字母字符、数字和下划线。它遵循 PCRE 标准，等同于 **[a-zA-Z0-9_]{10}**。
- **[\d]{10}** 生成 10 个数字。等同于 **[0-9]{10}**。
- **[\a]{10}** 生成 10 个字母字符。这等同于 **[a-zA-Z]{10}**。
- **[\a]{10}** 生成 10 个标点或符号字符。这等同于 **[~!@#$%\^&*()\-_+={}\[\]\\|<,>.?/"';:`]{10}**。

注意

取决于模板是以 YAML 还是以 JSON 编写，以及其中的修饰符的字符串类型，您可能需要用第二个反斜杠转义反斜杠。以下示例等同于：

带有修饰程序的 YAML 模板示例

```
parameters:
- name: singlequoted_example
  generate: expression
  from: '[\A]{10}'
- name: doublequoted_example
  generate: expression
  from: "[\\A]{10}"
```

带有修饰符的 JSON 模板示例

```
{
  "parameters": [
    {
      "name": "json_example",
      "generate": "expression",
      "from": "[\\A]{10}"
    }
  ]
}
```

下面是附带参数定义和参考的完整模板示例：

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
  - kind: BuildConfig
    apiVersion: build.openshift.io/v1
    metadata:
      name: cakephp-mysql-example
      annotations:
        description: Defines how to build the application
    spec:
      source:
        type: Git
        git:
```

```
        uri: "${SOURCE_REPOSITORY_URL}" (1)
        ref: "${SOURCE_REPOSITORY_REF}"
      contextDir: "${CONTEXT_DIR}"
  - kind: DeploymentConfig
    apiVersion: apps.openshift.io/v1
    metadata:
      name: frontend
    spec:
      replicas: "${{REPLICA_COUNT}}" (2)
parameters:
  - name: SOURCE_REPOSITORY_URL (3)
    displayName: Source Repository URL (4)
    description: The URL of the repository with your application source code (5)
    value: https://github.com/sclorg/cakephp-ex.git (6)
    required: true (7)
  - name: GITHUB_WEBHOOK_SECRET
    description: A secret string used to configure the GitHub webhook
    generate: expression (8)
    from: "[a-zA-Z0-9]{40}" (9)
  - name: REPLICA_COUNT
    description: Number of replicas to run
    value: "2"
    required: true
message: "... The GitHub webhook secret is ${GITHUB_WEBHOOK_SECRET} ..." (10)
```

1. 模板实例化时，该值将被替换为 **SOURCE_REPOSITORY_URL** 参数的值。
2. 模板实例化时，该值将被替换为 **REPLICA_COUNT** 参数的不加引号值。
3. 参数的名称。该值用于引用模板中的参数。
4. 参数的用户友好型名称。这会为用户显示。
5. 参数的描述。出于参数目的提供更详细的信息，包括对预期值的任何限制。描述应当按照控制台的文本标准使用完整句子。不可与显示名称重复。
6. 如果您实例化该模板时不覆盖该值，则使用该参数的默认值。密码之类避免使用默认值，而应结合使用生成的参数与 secret。
7. 指示此参数是必需的，这意味着您无法使用空值覆盖它。如果参数未提供默认值或生成值，您必须提供一个值。
8. 生成其值的参数。
9. 生成器的输入。这种情况下，生成器会生成一个 40 个字符的字母数字值，其中包括大写和小写字符。
10. 参数可包含在模板消息中。这将告知您生成的值。

10.7.4. 编写模板对象列表

模板主要部分为对象列表，将在模板实例化时创建。这可以是任何有效的 API 对象，如构建配置、部署配置或服务。该对象按照此处定义创建，创建前替换任意参数值。这些对象的定义可引用前面定义的参数。

以下是对象列表的示例：

```
kind: "Template"
apiVersion: "v1"
metadata:
  name: my-template
objects:
  - kind: "Service" 1
    apiVersion: "v1"
    metadata:
      name: "cakephp-mysql-example"
      annotations:
        description: "Exposes and load balances the application pods"
    spec:
      ports:
        - name: "web"
          port: 8080
          targetPort: 8080
      selector:
        name: "cakephp-mysql-example"
```

1 服务的定义，由该模板创建。

注意

如果对象定义元数据包含固定的 **namespace** 字段值，则会在模板实例化过程中从定义中分离。如果 **namespace** 字段包含参数引用，则将执行正常的参数替换，并参数替换将值解析到的任何命名空间中创建对象，假定用户有权在该命名空间中创建对象。

10.7.5. 将模板标记为可绑定

Template Service Broker 会在目录中为其了解的每个模板对象公告一个服务。默认情况下，每个服务均会公告为“可绑定”，表示允许最终用户绑定制备的服务。

流程

模板创建者可以防止最终用户绑定从给定模板制备的服务。

- 通过将注解 **template.openshift.io/bindable: "false"** 添加至模板中，防止最终用户绑定从给定模板制备的服务。

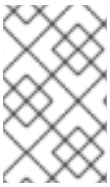
10.7.6. 公开模板对象字段

模板创建者可指定模板中的特定对象字段应公开。Template Service Broker 会识别 **ConfigMap**、**Secret**、**Service** 和 **Route** 上公开的字段，并在用户绑定代理支持的服务时返回公开字段的值。

要公开对象的一个或多个字段，请在模板中为对象添加以 **template.openshift.io/expose-** 或 **template.openshift.io/base64-expose-** 为前缀的注解。

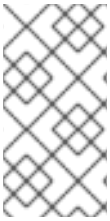
每个移除前缀的注解键均会被传递成为 **bind** 响应中的一个键。

每个注解值是一个 Kubernetes JSONPath 表达式，该表达式将在绑定时解析，以指示应在 **bind** 响应中返回值的对象字段。



注意

Bind 响应键/值对可在系统其他部分用作环境变量。因此，建议删除前缀的每个注解键均应为有效的环境变量名称，以字符 **A-Z**、**a-z** 或 _ 开头，后跟 0 或 **A-Z**、**a-z**、**0-9** 或 _ 等更多字符。



注意

除非用反斜杠转义，否则 Kubernetes 的 JSONPath 实现会将 **.**、**@** 字符等解析为元字符，而无关其在表达式中的位置。因此，例如要引用名为 **my.key** 的 **ConfigMap** 数据，所需 JSONPath 表达式应为 **{.data['my\.key']}**。根据 JSONPath 表达式在 YAML 中的编写方式，可能需要额外增加反斜杠，如 **"{.data['my\\.key']}"**。

以下是被公开的不同对象字段的示例：

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
- kind: ConfigMap
  apiVersion: v1
  metadata:
    name: my-template-config
    annotations:
      template.openshift.io/expose-username: "{.data['my\\.username']}"
  data:
    my.username: foo
- kind: Secret
  apiVersion: v1
  metadata:
    name: my-template-config-secret
    annotations:
      template.openshift.io/base64-expose-password: "{.data['password']}"
  stringData:
    password: <password>
- kind: Service
  apiVersion: v1
  metadata:
    name: my-template-service
    annotations:
      template.openshift.io/expose-service_ip_port: "{.spec.clusterIP}:{.spec.ports[?
(.name==\"web\")].port}"
  spec:
    ports:
    - name: "web"
      port: 8080
- kind: Route
  apiVersion: route.openshift.io/v1
  metadata:
    name: my-template-route
    annotations:
      template.openshift.io/expose-uri: "http://{.spec.host}{.spec.path}"
  spec:
    path: mypath
```

下面是在遵守上述部分模板情况下，对 **bind** 操作的一个响应示例：

```
{
  "credentials": {
    "username": "foo",
    "password": "YmFy",
    "service_ip_port": "172.30.12.34:8080",
    "uri": "http://route-test.router.default.svc.cluster.local/mypath"
  }
}
```

流程

- 使用 **template.openshift.io/expose-** 注解来以字符串形式返回字段值。这样很方便，尽管没有处理任意二进制数据。
- 如果要返回二进制数据，请在返回前使用 **template.openshift.io/base64-expose-** 注解对数据进行 base64 编码。

10.7.7. 等待模板就绪

模板创建者可指定：在服务目录、Template Service Broker 或 **TemplateInstance** API 进行的模板实例化被视为完成之前，应等待模板中的某些对象。

要使用该功能，请使用以下注解在模板中标记一个或多个**Build**、**BuildConfig**、**Deployment**、**DeploymentConfig**、**Job** 或 **StatefulSet** 类型的对象：

```
"template.alpha.openshift.io/wait-for-ready": "true"
```

直到标有注解的所有对象报告就绪时，模板实例化才算完成。同样，如果任何注解的对象报告失败，或者模板未能在一小时的固定超时内就绪，则模板实例化将失败。

就实例化而言，各种对象类型的就绪和失败定义如下：

类型	就绪	失败
Build	对象报告阶段完成。	对象报告阶段取消、错误或失败
BuildConfig	最新关联构建对象报告阶段完成	最新关联构建对象报告阶段取消、错误或失败
Deployment	对象报告新副本集和部署可用。这遵循对象上定义的就绪探针。	对象报告进度状况为 false。
DeploymentConfig	对象报告新的复制控制器和部署可用。这遵循对象上定义的就绪探针。	对象报告进度状况为 false。
作业	对象报告完成。	对象报告出现一个或多个故障。
StatefulSet	对象报告所有副本就绪。这遵循对象上定义的就绪探针。	不适用。

以下是使用 **wait-for-ready** 注解的模板提取示例。更多示例可在 OpenShift Container Platform 快速启动模板中找到。

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: ...
    annotations:
      # wait-for-ready used on BuildConfig ensures that template instantiation
      # will fail immediately if build fails
      template.alpha.openshift.io/wait-for-ready: "true"
  spec:
    ...
- kind: DeploymentConfig
  apiVersion: apps.openshift.io/v1
  metadata:
    name: ...
    annotations:
      template.alpha.openshift.io/wait-for-ready: "true"
  spec:
    ...
- kind: Service
  apiVersion: v1
  metadata:
    name: ...
  spec:
    ...
```

其他建议

- 设置内存、CPU 和存储的默认大小，以确保您的应用程序获得足够资源使其平稳运行。
- 如果要在主版本中使用该标签，请避免引用来自镜的 **latest** 标签。当新镜像被推送（push）到该标签时，这可能会导致运行中的应用程序中断。
- 良好的模板可整洁地构建和部署，无需在部署模板后进行修改。

10.7.8. 从现有对象创建模板

您可以 YAML 格式从项目中导出现有对象，然后通过添加参数和其他自定义作为模板表单来修改 YAML，而无需从头开始编写整个模板。

流程

- 以 YAML 格式导出项目中的对象：

  ```
  $ oc get -o yaml all > <yaml_filename>
  ```

  您还可替换特定资源类型或多个资源，而非 **all** 资源。运行 **oc get -h** 获取更多示例。

oc get -o yaml all 中包括的对象类型是：

- **BuildConfig**
- **Build**
- **DeploymentConfig**
- **ImageStream**
- **Pod**
- **ReplicationController**
- **Route**
- **Service**

注意

不建议使用 **all** 别名，因为内容在不同的集群和版本中可能有所不同。相反，明确指定所有需要的资源。

第 11 章 使用 RUBY ON RAILS

Ruby on Rails 是采用 Ruby 编写的 web 框架。本指南介绍在 OpenShift Container Platform 上使用 Rails 4。

警告

浏览整个教程，了解在 OpenShift Container Platform 上运行应用程序的所有步骤。如果遇到问题，请尝试通读整个教程，然后再回看问题。该教程还可用于审查您之前的步骤，以确保正确运行所有步骤。

11.1. 先决条件

- 具备 Ruby 和 Rails 基础知识。
- 本地已安装 Ruby 2.0.0+ 版、RubyGems、Bundler。
- 具备 Git 基础知识。
- OpenShift Container Platform 4 的运行实例。
- 确保 OpenShift Container Platform 实例正在运行且可用。另外还需确保已安装 **oc** CLI 客户端，且可从命令 shell 访问命令，以便您可以使用您的电子邮件地址和密码通过客户端登录。

11.2. 设置数据库

Rails 应用程序几乎总是与数据库一同使用。对于本地开发，请使用 PostgreSQL 数据库。

流程

1. 安装数据库：

```
$ sudo yum install -y postgresql postgresql-server postgresql-devel
```

2. 初始化数据库：

```
$ sudo postgresql-setup initdb
```

这个命令会创建 **/var/lib/pgsql/data** 目录，数据存储在其中。

3. 启动数据库：

```
$ sudo systemctl start postgresql.service
```

4. 数据库运行时，创建 **rails** 用户：

```
$ sudo -u postgres createuser -s rails
```

注意，所创建的用户无密码。

11.3. 编写应用程序

如果要从头开始启动 Rails 应用程序，必须先安装 Rails gem，然后才可编写应用程序。

流程

1. 安装 Rails gem：

```
$ gem install rails
```

输**出示例**

```
Successfully installed rails-4.3.0
1 gem installed
```

2. 安装完 Rails gem 后，使用 PostgreSQL 创建一个新应用程序，作为数据库：

```
$ rails new rails-app --database=postgresql
```

3. 更改至新应用程序目录：

```
$ cd rails-app
```

4. 如果您已有应用程序，请确保 **Gemfile** 中存在 **pg** (postgresql) gem。如果尚无应用程序，则通过添加 gem 来编辑 **Gemfile**：

```
gem 'pg'
```

5. 使用所有依赖项生成新的 **Gemfile.lock**：

```
$ bundle install
```

6. 除了将 **postgresql** 数据库与 **pg** gem 结合使用外，您还必须确保 **config/database.yml** 正在使用 **postgresql** 适配器。
请确保更新了 **config/database.yml** 文件中的 **default** 部分，如下所示：

```
default: &default
  adapter: postgresql
  encoding: unicode
  pool: 5
  host: localhost
  username: rails
  password: <password>
```

7. 创建应用程序的开发和测试数据库：

```
$ rake db:create
```

这会在您的 PostgreSQL 服务器中创建 **development** 和 **test** 数据库。

11.3.1. 创建欢迎页面

由于 Rails 4 在生产中不再提供静态 **public/index.html** 页面，您必须创建一个新的 root 页面。

要想具有自定义欢迎页面，必须执行以下步骤：

- 使用 index 操作创建控制器。
- 为 welcome 控制器 index 操作创建 view 页面。
- 使用所创建的 controller 和 view 创建一个提供应用程序 root 页面的路由。

Rails 提供了一个生成器，用于完成您所有必要的步骤。

流程

1. 运行 Rails 生成器：

```
$ rails generate controller welcome index
```

已创建所有必要文件。

2. 按如下方式编辑 **config/routes.rb** 文件中第 2 行：

```
root 'welcome#index'
```

3. 运行 rails 服务器以验证页面是否可用：

```
$ rails server
```

在浏览器中访问 http://localhost:3000 即可查看您的页面。如果没有看到该页面，请检查输出至服务器的日志进行调试。

11.3.2. 为 OpenShift Container Platform 配置应用程序

要让您的应用程序与 OpenShift Container Platform 中运行的 PostgreSQL 数据库服务通信，必须编辑 **config/database.yml** 中的 **default** 部分，以便在创建数据库服务时使用环境变量，稍后会对这些变量进行定义。

流程

- 使用预定义的变量按照以下方式编辑 **config/database.yml** 中的 **default** 部分：

config/database YAML 文件示例

```
<% user = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ? "root" :
ENV["POSTGRESQL_USER"] %>
<% password = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ?
ENV["POSTGRESQL_ADMIN_PASSWORD"] : ENV["POSTGRESQL_PASSWORD"] %>
<% db_service = ENV.fetch("DATABASE_SERVICE_NAME","").upcase %>

default: &default
  adapter: postgresql
  encoding: unicode
```

```
# For details on connection pooling, see rails configuration guide
# http://guides.rubyonrails.org/configuring.html#database-pooling
pool: <%= ENV["POSTGRESQL_MAX_CONNECTIONS"] || 5 %>
username: <%= user %>
password: <%= password %>
host: <%= ENV["#{db_service}_SERVICE_HOST"] %>
port: <%= ENV["#{db_service}_SERVICE_PORT"] %>
database: <%= ENV["POSTGRESQL_DATABASE"] %>
```

11.3.3. 将应用程序存储在 Git 中

在 OpenShift Container Platform 中构建应用程序通常需要将源代码存储在 git 存储库中，因此如果还没有 **git**，必须要安装。

先决条件

- 安装 git。

流程

1. 运行 **ls -1** 命令，确保已在 Rails 应用程序目录中。命令输出应类似于：

```
$ ls -1
```

输**出示例**

```
app
bin
config
config.ru
db
Gemfile
Gemfile.lock
lib
log
public
Rakefile
README.rdoc
test
tmp
vendor
```

2. 在 Rails 应用程序目录中运行以下命令，以便初始化代码并将其提交给 git：

```
$ git init
```

```
$ git add .
```

```
$ git commit -m "initial commit"
```

提交应用程序后，必须将其推送（push）到远程存储库。Github 帐户，您可使用它创建新的存储库。

3. 设置指向 **git** 存储库的远程存储库：

```
$ git remote add origin git@github.com:<namespace/repository-name>.git
```

4. 将应用程序推送（push）到远程 git 存储库。

```
$ git push
```

11.4. 将应用程序部署至 OPENSHIFT CONTAINER PLATFORM

您可将您的应用程序部署至 OpenShift Container Platform。

创建 **rails-app** 项目后，您将自动切换到新的项目命名空间。

在 OpenShift Container Platform 中部署应用程序涉及三个步骤：

- 从 OpenShift Container Platform 的 PostgreSQL 镜像创建数据库服务。
- 从 OpenShift Container Platform 的 Ruby 2.0 构建程序镜像和 Ruby on Rails 源代码创建前端服务，这些服务将与数据库服务相连接。
- 为应用程序创建路由。

流程

- 要部署 Ruby on Rails 应用程序，请为应用程序创建一个新项目：

```
$ oc new-project rails-app --description="My Rails application" --display-name="Rails
Application"
```

11.4.1. 创建数据库服务

您的 Rails 应用程序需要一个正在运行的数据库服务。对于此服务，请使用 PostgreSQL 数据库镜像。

要创建数据库服务，使用 **oc new-app** 命令。您必须将一些在数据库容器中使用的必要环境变量传递给此命令。设置用户名、密码和数据库名称需要这些环境变量。您可随意更改这些环境变量的值。变量如下：

- POSTGRESQL_DATABASE
- POSTGRESQL_USER
- POSTGRESQL_PASSWORD

设置这些变量可确保：

- 存在具有指定名称的数据库。
- 存在具有指定名称的用户。
- 用户可使用指定密码访问指定数据库。

流程

1. 创建数据库服务：

```
$ oc new-app postgresql -e POSTGRESQL_DATABASE=db_name -e
POSTGRESQL_USER=username -e POSTGRESQL_PASSWORD=password
```

若也要为数据库管理员设置密码，请将以下内容附加至上一命令中：

```
-e POSTGRESQL_ADMIN_PASSWORD=admin_pw
```

2. 监控进度：

```
$ oc get pods --watch
```

11.4.2. 创建前端服务

要将应用程序添加到 OpenShift Container Platform 中，您必须指定应用程序所在存储库。

流程

1. 创建前端服务，并指定创建数据库服务时设置的数据库相关环境变量：

```
$ oc new-app path/to/source/code --name=rails-app -e POSTGRESQL_USER=username -e
POSTGRESQL_PASSWORD=password -e POSTGRESQL_DATABASE=db_name -e
DATABASE_SERVICE_NAME=postgresql
```

通过此命令，OpenShift Container Platform 可获取源代码，设置构建程序来构建应用程序镜像，并与指定的环境变量一同来部署新创建的镜像。该应用程序命名为 **rails-app**。

2. 通过查看 **rails-app** 部署配置的 JSON 文档来验证环境变量是否已添加：

```
$ oc get dc rails-app -o json
```

您应看到以下部分：

输**出示例**

```
env": [
    {
        "name": "POSTGRESQL_USER",
        "value": "username"
    },
    {
        "name": "POSTGRESQL_PASSWORD",
        "value": "password"
    },
    {
        "name": "POSTGRESQL_DATABASE",
        "value": "db_name"
    },
    {
        "name": "DATABASE_SERVICE_NAME",
        "value": "postgresql"
    }

],
```

3. 检查构建流程：

```
$ oc logs -f build/rails-app-1
```

4. 构建完成后，查看 OpenShift Container Platform 中运行的 pod：

```
$ oc get pods
```

您应看到其中一行命令以 **myapp-<number>-<hash>** 开头，这是您在 OpenShift Container Platform 中运行的应用程序。

5. 在应用程序正常工作前，您必须运行数据库迁移脚本来初始化数据库。具体可通过两种方式实现：

- 从正在运行的前端容器手动实现：
  - 使用 **rsh** 命令执行到前端容器中：

```
$ oc rsh <frontend_pod_id>
```

  - 从容器内部运行迁移：

```
$ RAILS_ENV=production bundle exec rake db:migrate
```

如果在 **development** 或 **test** 环境中运行 Rails 应用程序，则不必指定 **RAILS_ENV** 环境变量。

- 通过在模板中添加部署前生命周期 hook。

11.4.3. 为您的应用程序创建路由

您可公开服务来为您的应用程序创建路由。

流程

- 要通过向服务提供外部可访问的主机名（如 **www.example.com**）来公开服务，请使用 OpenShift Container Platform 路由。对于您的情况，需要通过键入以下命令来公开前端服务：

```
$ oc expose service rails-app --hostname=www.example.com
```

警告

确保您指定的主机名解析为路由器的 IP 地址。

第 12 章 使用镜像

12.1. 使用镜像概述

以下介绍了不同的 Source-to-Image (S2I)、数据库以及其他可供 OpenShift Container Platform 用户使用的容器镜像。

红帽官方容器镜像在 registry.redhat.io 上的 Red Hat Registry 中提供。OpenShift Container Platform 支持的 S2I、数据库和 Jenkins 镜像在 Red Hat Quay Registry 的 **openshift4** 存储库中提供。例如：**quay.io/openshift-release-dev/ocp-v4.0-<address>** 是 OpenShift Application Platform 镜像的名称。

xPaaS 中间件镜像在 Red Hat Registry 上的相应产品存储库中提供，后缀为 **-openshift**。例如：**registry.redhat.io/jboss-eap-6/eap64-openshift** 是 JBoss EAP 镜像的名称。

本节涵盖的红帽支持的所有镜像均在红帽生态系统目录的容器镜像部分进行描述。如需了解每个镜像的各种版本，请查看其内容和使用方法详情。浏览或搜索您感兴趣的镜像。

重要

较新版容器镜像与较早版 OpenShift Container Platform 不兼容。根据您的 OpenShift Container Platform 版本，验证并使用正确的容器镜像版本。

12.2. SOURCE-TO-IMAGE

您可以将 Red Hat Software Collections 镜像用作依赖特定运行时环境（如 Node.js、Perl 或 Python）的应用程序的基础。您可以使用 Red Hat Java Source-to-Image for OpenShift 文档作为使用 Java 的运行时环境的参考。其中一些运行时基础镜像的特殊版本称为 Source-to-Image (S2I) 镜像。使用 S2I 镜像时，您可以将代码插入到可随时运行该代码的基础镜像环境中。

S2I 镜像包括：

- .NET
- Java
- Go
- Node.js
- Perl
- PHP
- Python
- Ruby

您可以按照以下流程直接从 OpenShift Container Platform Web 控制台使用 S2I 镜像：

1. 使用您的登录凭证登录到 OpenShift Container Platform web 控制台。OpenShift Container Platform Web 控制台的默认视图是 **Administrator** 视角。
2. 使用视角切换功能把它切换到 **Developer** 视角。

3. 在 **+Add** 视图中，使用 **Project** 下拉列表选择现有项目或创建新项目。
4. 点 **Developer Catalog** 标题中的 **All services**。
5. 点 **Type** 下的 **Builder Images** 以查看可用的 S2I 镜像。

S2I 镜像也可以通过 Cluster Samples Operator 获得。

12.2.1. Source-to-image 构建过程概述

Source-to-Image（S2I）通过将源代码注入准备要运行的源代码的容器来生成可随时运行的镜像。它执行以下步骤：

1. 运行 **FROM <builder image>** 命令
2. 将源代码复制到构建器镜像中定义的位置
3. 在构建器镜像中运行 assemble 脚本
4. 将构建器镜像中的 run 脚本设置为默认命令

然后，Buildah 会创建容器镜像。

12.2.2. 其他资源

- 配置 Cluster Samples Operator
- 使用构建策略
- 对 Source-to-Image 进行故障排除
- 使用 Source-to-image 从源代码创建镜像
- 关于测试 source-to-image 镜像
- 使用 Source-to-image 从源代码创建镜像

12.3. 自定义 SOURCE-TO-IMAGE 镜像

Source-to-Image（S2I）构建器镜像包含 assemble 和 run 脚本，但这些脚本的默认行为并不适用于所有用户。您可以自定义包含默认脚本的 S2I 构建器的行为。

12.3.1. 调用嵌入在镜像中的脚本

构建器镜像提供自己的 Source-to-image（S2I）脚本版本，它适用于最常用的用例。如果这些脚本无法满足您的需要，S2I 提供了在 **.s2i/bin** 目录中添加自定义脚本覆盖它们的方法。但是，这样做代表完全替换了标准脚本。在某些情况下，替换脚本是可以接受的，但在其他情况下，您可以在脚本之前或之后运行一些命令，同时保留镜像中提供的脚本逻辑。要重复使用标准脚本，您可以创建一个运行自定义逻辑的 wrapper 脚本，并将它进一步分配给镜像中的默认脚本。

流程

1. 查看 **io.openshift.s2i.scripts-url** 标签的值，以确定构建器镜像中的脚本位置：

```
$ podman inspect --format='{{ index .Config.Labels "io.openshift.s2i.scripts-url" }}'
wildfly/wildfly-centos7
```

输**出示例**

```
image:///usr/libexec/s2i
```

您检查了 **wildfly/wildfly-centos7** 构建器镜像，并发现脚本位于 **/usr/libexec/s2i** 目录中。

2. 创建一个包含其它命令中嵌套的标准脚本之一的脚本：

.s2i/bin/assemble 脚本

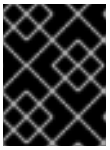
```
#!/bin/bash
echo "Before assembling"

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
    echo "After successful assembling"
else
    echo "After failed assembling"
fi

exit $rc
```

这个示例显示了一个自定义 assemble 脚本，它输出信息，从镜像中运行标准 assemble 脚本，并根据 assemble 脚本的退出代码输出另一个信息。



重要

当嵌套 run 脚本时，您必须使用 **exec** 来调用它来确保正确处理信号。使用 **exec** 也无法在调用默认镜像运行脚本后运行附加命令。

.s2i/bin/run 脚本

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```