OpenShift Container Platform 4.2

镜像

在 OpenShift Container Platform 4.2 中创建和管理镜像及镜像流

Last Updated: 2020-08-21

OpenShift Container Platform 4.2 镜像

在 OpenShift Container Platform 4.2 中创建和管理镜像及镜像流

法律通告

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本文档介绍在 OpenShift Container Platform 4.2 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 配置 SAMPLES OPERATOR 4
- 1.1. 了解 SAMPLES OPERATOR 4
- 1.2. SAMPLES OPERATOR 配置参数 4
- 1.3. 访问 SAMPLES OPERATOR 配置 6

第 2 章 使用带有一个备用 REGISTRY 的 SAMPLES OPERATOR 8
- 2.1. 关于镜像 REGISTRY 8
- 2.2. 创建镜像 REGISTRY 9
- 2.3. 在 PULL SECRET 中添加 REGISTRY 11
- 2.4. 镜像 OPENSHIFT CONTAINER PLATFORM 镜像存储库 13
- 2.5. 使用带有备用或经过镜像的 REGISTRY 的 SAMPLES OPERATOR 镜像流 14

第 3 章 了解容器、镜像和镜像流 16
- 3.1. 镜像 16
- 3.2. 容器 16
- 3.3. 镜像 REGISTRY 16
- 3.4. 镜像存储库 17
- 3.5. 镜像标签 17
- 3.6. 镜像 ID 17
- 3.7. 使用镜像流 17
- 3.8. 镜像流镜像 18
- 3.9. 镜像流触发器 18
- 3.10. 其它资源 18

第 4 章 创建镜像 19
- 4.1. 学习容器最佳实践 19
- 4.2. 包括镜像中的元数据 24
- 4.3. 测试 S2I 镜像 25

第 5 章 管理镜像 28
- 5.1. 管理镜像概述 28
- 5.2. 标记镜像 28
- 5.3. 镜像拉取（PULL）策略 31
- 5.4. 使用镜像 PULL SECRET 31

第 6 章 管理镜像流 35
- 6.1. 使用镜像流 35
- 6.2. 配置镜像流 35
- 6.3. 镜像流镜像 36
- 6.4. 镜像流标签（IMAGESTREAMTAGS） 37
- 6.5. 镜像流更改触发器 38
- 6.6. 镜像流映射 38
- 6.7. 使用镜像流 40

第 7 章 镜像配置资源 45
- 7.1. 镜像控制器配置参数 45
- 7.2. 配置镜像设置 46

第 8 章 使用模板 52
- 8.1. 了解模板 52
- 8.2. 上传模板 52
- 8.3. 使用 WEB 控制台创建应用程序 52
- 8.4. 使用 CLI 从模板创建对象 53

8.5. 修改所上传的模板 55
8.6. 使用 INSTANT APP 和 QUICKSTART 模板 55
8.7. 编写模板 56

第 9 章 使用 RUBY ON RAILS 66
9.1. 设置数据库 66
9.2. 编写应用程序 67
9.3. 将应用程序部署至 OPENSHIFT CONTAINER PLATFORM 69

第 10 章 使用镜像 73
10.1. 使用镜像概述 73
10.2. 配置 JENKINS 镜像 73
10.3. JENKINS 代理 85

第 1 章 配置 SAMPLES OPERATOR

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。

先决条件

- 部署一个 OpenShift Container Platform 集群。

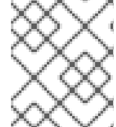
1.1. 了解 SAMPLES OPERATOR

在安装过程中，Operator 会为自己创建默认配置对象，然后再创建示例（sample）镜像流和模板，包括 Quickstart 模板。

Samples Operator 将安装程序捕获的 pull secret 复制到 OpenShift 命名空间中，并将该 secret 命名为**samples-registry-credentials**，以便从 **registry.redhat.io** 导入镜像流。此外，为便于从其他需要凭证的 registry 导入镜像流，集群管理员可在处理镜像导入的 OpenShift 命名空间中创建包含 Docker **config.json** 文件内容的额外 Secret。

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。

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，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。

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。
- **samples-registry-credentials** secret。

删除示例资源后，Samples Operator 会使用默认配置重新创建资源。

1.2. SAMPLES OPERATOR 配置参数

示例资源提供以下配置字段：

参数	描述
managementState	**Managed**：Samples Operator 根据配置要求更新示例。 **Unmanaged**：Samples Operator 忽略对 OpenShift 命名空间中配置资源对象及任何镜像流或模板的更新。 **Removed**：Samples Operator 移除 OpenShift 命名空间中的**Managed** 镜像流和模板组。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。完成移除后，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。 **注意** 在镜像流导入仍在进行时，删除操作或将 **Management State** 设置为 **Removed** 均未完成。进程完成后，无论成功还是出错，都将开始删除操作。 开始删除后，secret、镜像流和模板监控事件都会被忽略。
samplesRegistry	覆盖从中导入镜像的 registry。 **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。目前仅支持 x86。
skippedImagestreams	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	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，Samples Operator 会检测到这些事件并对事件队列重新排序。

1.2.1. 配置限制

当 Samples Operator 开始支持多个构架时，处于 **Managed** 状态下的架构列表将不可更改。

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**。

在 **Removed** 状态下，Samples Operator 仍可处理 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（不过如果您在切换到 **Managed** 后创建 secret，创建示例会延迟至处理完 secret 事件）。如果您选择在切换前移除所有示例（虽然这不是必须的）以确保获得一个清洁的状态，则会有助于更改 registry。

1.2.2. 条件

示例资源将在所处状态下保持以下条件：

条件	描述
SamplesExists	表明 OpenShift 命名空间中已创建示例。
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。
ImportCredentialsExist	**samples-registry-credentials** secret 复制到 OpenShift 命名空间中。
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	表明存在待处理的 **Management State: Removed** 设置，但将等到进行中的镜像流完成后再处理。
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 Samples Operator 检测到对应版本与安装当前示例集的 Samples Operator 版本不同时，显示为 **True**。

1.3. 访问 SAMPLES OPERATOR 配置

您可使用所提供的参数来编辑文件，以此配置 Samples Operator。

先决条件

- 安装 OpenShift 命令行界面 (CLI)，通常称为 **oc**。

流程

- 访问 Samples Operator 配置：

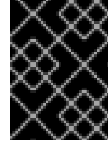
  ```
  $ oc get configs.samples.operator.openshift.io/cluster -o yaml
  ```

  Samples Operator 配置类似以下示例：

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
projectName: cluster-samples-operator
...
```

第 2 章 使用带有一个备用 REGISTRY 的 SAMPLES OPERATOR

您可以通过创建镜像 registry 来将 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。

2.1. 关于镜像 REGISTRY

您可以镜像 OpenShift Container Platform registry 的内容，也可以镜像生成安装程序所需的镜像。

镜像 registry 是一个关键组件，有了它才能在受限网络中完成安装。您可以在堡垒主机上创建此镜像，该主机可同时访问互联网和您的封闭网络，也可以使用满足您的限制条件的其他方法。

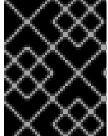
由于 OpenShift Container Platform 验证发行版本有效负载完整性的方式，您的本地 registry 中的镜像引用与红帽在 Quay.io 上托管的镜像相同。在安装的 bootstrap 过程中，不论镜像是从哪个存储库提取的，镜像都必须有相同的摘要。要确保发行版有效负载一致，请将镜像镜像到您的本地存储库。

2.1.1. 准备镜像主机

在创建镜像 registry 前，您必须准备镜像（mirror）主机。

2.1.2. 安装 CLI

为了可以使用命令行界面与 OpenShift Container Platform 进行交互，您需要安装 CLI。

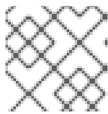


重要

如果安装了旧版本的 **oc**，则无法使用 OpenShift Container Platform 4.2 中的所有命令。下载并安装新版本的 **oc**。

流程

1. 在 Red Hat OpenShift Cluster Manager 站点的 Infrastructure Provider 页面中导航至您的安装类型页面，并点击 **Download Command-line Tools**。
2. 点您的操作系统和系统架构的文件夹，然后点压缩文件。

   注意

   您可在 Linux 、Windows 或 macOS 上安装 **oc**。

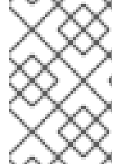
3. 将文件保存到文件系统。
4. 展开压缩文件。
5. 把它放到 **PATH** 中的一个目录下。

安装 CLI 后，就可以使用**oc**命令：

```
$ oc <command>
```

2.2. 创建镜像 REGISTRY

创建 registry 来托管安装 OpenShift Container Platform 所需的镜像内容。



注意

以下流程创建一个简单的 registry，它可在 **/opt/registry** 文件夹中保存数据并在 **podman** 容器中运行。您可以使用不同的 registry 解决方案，例如 Red Hat Quay。检查以下流程以确保 registry 可以正常工作。

先决条件

- 网络上有一个 Red Hat Enterprise Linux (RHEL) 服务器充当 registry 主机。
- registry 主机可以访问互联网。

流程

1. 安装所需的软件包：

```
# yum -y install podman httpd-tools
```

podman 软件包提供容器软件包，用于运行 registry。**httpd-tools** 软件包提供 **htpasswd** 实用程序，用于创建用户。

2. 为 registry 创建文件夹：

```
# mkdir -p /opt/registry/{auth,certs,data}
```

这些文件夹挂载到 registry 容器中。

3. 为 registry 提供证书。如果您没有现有的可信证书颁发机构，您可以生成自签名证书：

```
$ cd /opt/registry/certs
# openssl req -newkey rsa:4096 -nodes -sha256 -keyout domain.key -x509 -days 365 -out
domain.crt
```

在提示符处，为证书提供所需的值：

国家/地区名称（双字母代码）	指定您所在位置的双字母 ISO 国家/地区代码。请参见 ISO 3166 国家/地区代码标准。
州或省名称（完整名称）	输入您的州或省的完整名称。
本地名称（例如，城市）	输入您的城市名称。

机构名称（例如，公司）	输入公司的名称。
组织单元名称（例如，部门）	输入您的部门名称。
通用名称（例如，您的名字或服务器主机名）	输入 registry 主机的主机名。确保您的主机名在 DNS 中，并且解析为预期的 IP 地址。
电子邮件地址	输入您的电子邮件地址。如需了解更多信息，请参阅 OpenSSL 文档中的 req 说明。

4. 为 registry 生成使用 **bcrpt** 格式的用户名和密码：

```
# htpasswd -bBc /opt/registry/auth/htpasswd <user_name> <password> 1
```

1 将 **<user_name>** 和 **<password>** 替换为用户名和密码。

5. 创建 **mirror-registry** 容器以托管 registry：

```
# podman run --name mirror-registry -p <local_registry_host_port>:5000 \ 1
     -v /opt/registry/data:/var/lib/registry:z \
     -v /opt/registry/auth:/auth:z \
     -e "REGISTRY_AUTH=htpasswd" \
     -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
     -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
     -v /opt/registry/certs:/certs:z \
     -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
     -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
     -e REGISTRY_COMPATIBILITY_SCHEMA1_ENABLED=true \
     -d docker.io/library/registry:2
```

1 对于 **<local_registry_host_port>**，请指定您的镜像 registry 用于提供内容的端口。

6. 为 registry 打开所需的端口：

```
# firewall-cmd --add-port=<local_registry_host_port>/tcp --zone=internal --permanent 1
# firewall-cmd --add-port=<local_registry_host_port>/tcp --zone=public   --permanent 2
# firewall-cmd --reload
```

1 2 对于 **<local_registry_host_port>**，请指定您的镜像 registry 用于提供内容的端口。

7. 将自签名证书添加到您的可信证书列表中：

```
# cp /opt/registry/certs/domain.crt /etc/pki/ca-trust/source/anchors/
# update-ca-trust
```

您必须信任您的证书，才能在镜像过程中登录到 registry。

8. 确认 registry 可用：

```
$ curl -u <user_name>:<password> -k https://<local_registry_host_name>:<local_registry_host_port>/v2/_catalog 1

{"repositories":[]}
```

1 对于 **<user_name>** 和 **<password>**，指定 registry 的用户名和密码。对于 **<local_registry_host_name>**，请指定在您的证书中指定的 registry 域名，如 **registry.example.com**。对于 **<local_registry_host_port>**，请指定您的镜像 registry 用于提供内容的端口。

如果命令输出显示一个空存储库，则您的 registry 已经可用。

2.3. 在 PULL SECRET 中添加 REGISTRY

在受限网络中安装 OpenShift Container Platform 集群前，需要为 OpenShift Container Platform 集群修改 pull secret 来使用本地 registry。

先决条件

- 配置了一个镜像（mirror） registry 在受限网络中使用。

流程

在堡垒主机上完成以下步骤：

1. 从 Red Hat OpenShift Cluster Manager 站点的 Pull Secret 页面下载 **registry.redhat.io** 的 pull secret。

2. 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。

3. 以 JSON 格式创建您的 pull secret 副本：

```
$ cat ./pull-secret.text | jq .  > <path>/<pull-secret-file> 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"
    }
  }
}
```

4. 编辑新文件并添加描述 registry 的部分：

```
  "auths": {
...
    "<local_registry_host_name>:<local_registry_host_port>": { (1)
      "auth": "<credentials>", (2)
      "email": "you@example.com"
  },
...
```

(1) 使用 **<local_registry_host_name>** 指定您证书中指定的 registry 域名，使用 **<local_registry_host_port>** 指定镜像 registry 用来提供内容的端口。

(2) 使用 **<credentials>** 为您生成的镜像 registry 指定 base64 编码的用户名和密码。

该文件类似于以下示例：

```
{
  "auths": {
    "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"
    },
    "<local_registry_host_name>:<local_registry_host_port>": {
      "auth": "<credentials>",
```

```
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

2.4. 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 镜像存储库。

先决条件

- 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 站点的 Pull Secret页面下载了 pull secret，并已修改为包含镜像存储库身份验证信息。

流程

在堡垒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。
2. 设置所需的环境变量：

```
$ export OCP_RELEASE=<release_version> (1)
$ export LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>' (2)
$ export LOCAL_REPOSITORY='<repository_name>' (3)
$ export PRODUCT_REPO='openshift-release-dev' (4)
$ export LOCAL_SECRET_JSON='<path_to_pull_secret>' (5)
$ export RELEASE_NAME="ocp-release" (6)
```

(1) 对于 **<release_version>**，请指定要安装的 OpenShift Container Platform 版本号，如 **4.2.0**。

(2) 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。

(3) 对于 **<repository_name>**，请指定要在 registry 中创建的存储库名称，如 **ocp4/openshift4**。

(4) 要镜像的存储库。对于生产环境版本，必须指定 **openshift-release-dev**。

(5) 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。

(6) 发行版本镜像。对于生产环境版本，您必须指定 **ocp-release**。

3. 镜像存储库：

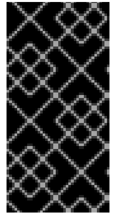
```
$ oc adm -a ${LOCAL_SECRET_JSON} release mirror \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}
```

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。

4. 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。

5. 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：

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



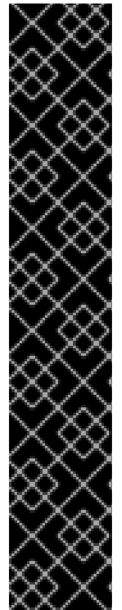
重要

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。

2.5. 使用带有备用或经过镜像的 REGISTRY 的 SAMPLES OPERATOR 镜像流

OpenShift 命名空间中大多数由 Samples Operator 管理的镜像流指向位于 registry.redhat.io 上红帽 registry 中的镜像。



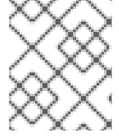
重要

jenkins、**jenkins-agent-maven** 和 **jenkins-agent-nodejs** 镜像流来自安装有效负载，并由 Samples Operator 管理。

将 Sample Operator 配置文件中的 **samplesRegistry** 字段设置为 registry.redhat.io 是多余的，因为它已将除 Jenkins 镜像和镜像流以外的所有内容都定向到 registry.redhat.io。它还会破坏 Jenkins 镜像流的安装有效负载。

Samples Operator 禁止将以下 registry 用于 Jenkins 镜像流：

- docker.io
- registry.redhat.io
- registry.access.redhat.com
- *.quay.io



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 Samples Operator 管理。此流程中不涉及这些镜像流。

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. 来自 registry.redhat.io 的被镜像（mirror）的镜像与您需要的任何镜像流相关联

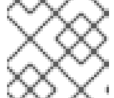
```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

3. 在集群的镜像配置对象中，为镜像添加所需的可信 CA：

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

4. 更新 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：

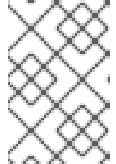
```
$ oc get configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```



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。

5. 将所有未镜像的镜像流添加到 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，请在 Samples Operator 配置对象中将 Samples Operator 设置为 **Removed**。



注意

镜像流导入失败两小时后，任何没有跳过的未镜像的镜像流，或者如果 Samples Operator 没有更改为 **Removed**，都会导致 Samples Operator 报告 **Degraded** 状态。

OpenShift 命名空间中的多个模板都引用镜像流。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。

第 3 章 了解容器、镜像和镜像流

开始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。

3.1. 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。每个不同的镜像都可以通过其唯一的哈希值识别（很长的十六进制数，如**fd44297e2ddb050ec4f...**），通常可缩短为 12 个字符（如：**fd44297e2ddb**）。

3.2. 容器

OpenShift Container Platform 应用程序的基本单元称为*容器*。Linux 容器技术是一个用于隔离不同运行进程的轻量机制。它可以把运行的进程限制为只与相应的资源进行交互。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。

3.3. 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 还可提供其内部 registry，用于管理自定义容器镜像。

3.4. 镜像存储库

镜像存储库是一个相关容器镜像和标识它们的标签（tag）的集合。例如，OpenShift Jenkins 镜像位于以下存储库中：

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

3.5. 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，这里的 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。

3.6. 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
```

3.7. 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。
- 外部 registry，如 **registry.redhat.io** 或 **hub.docker.com**。
- OpenShift Container Platform 集群中的其他镜像流。

您在定义引用 imagestreamtag 的对象时（如构建或部署配置），您指向了 imagestreamtag，而非 Docker 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。

镜像流元数据会连同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。
- 您可通过设置镜像流的访问权限来配置安全性，以控制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。

3.7.1. 镜像流标签（Imagestreamtags）

镜像流标签是指向镜像流中镜像的一个指针。镜像流标签与容器镜像标签类似。

3.8. 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。

3.9. 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。

3.10. 其它资源

- 有关使用镜像流的更多信息，请参阅管理镜像流。

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。创建完镜像后，您可将其推送到内部 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
建议您尽可能使用 **FROM** 语句将您的镜像基于适用的上游镜像。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 *foo* 的镜像，当前包含 1.0 版，则可使用 *foo:v1* 标签。当您更新了镜像时，只要仍与原始镜像兼容，就可继续使用 *foo:v1* 做为新镜像的标签。而使用这个标签的下游用户就可获得更新，而不会出现问题。

如果后续发布了不兼容的更新，则需要使用新标签，例如 *foo: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 *foo:latest*，则可能要承担引入不兼容更改的风险。

避免多进程
我们不建议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。这并不符合预期，解释如下：

假设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。现在假设您要通过 **CTRL+C** 终止容器。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，一切都将按照您的预期运作。如果没有在 wrapper 脚本中使用 exec，则 **podman** 会将 SIGINT 发送至 wrapper 脚本的进程，而您的进程将毫不受影响地继续运行。

另请注意，您的进程在容器中运行时，将作为 PID 1 运行。这表示，如果主进程终止，则整个容器都会停止，继而终止您已从 PID 1 进程启动的所有子进程。

如需了解更多影响，请参阅“Docker 和 PID 1 僵尸进程收割问题”博客文章。如需深入了解 PID 1 和 **init** 系统，请参阅“揭秘 init 系统 (PID 1)”博客文章。

清理临时文件
构建过程中创建的所有临时文件均应删除。这也包括通过 **ADD** 命令添加的任何文件。例如，我们强烈建议您在执行 **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。务必要将极少变化的指令放置在 **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：

- 公开端口将显示在 **podman ps** 下，与从您的镜像创建的容器关联
- 公开端口还存在于通过 **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
最好避免设置默认密码。很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。应该使用环境变量来使密码可以被配置。

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。

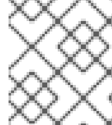
避免 sshd
最好避免在您的镜像中运行 sshd。您可使用 **podman exec** 或 **docker exec** 命令访问本地主机上运行的容器。另外，也可使用 **oc exec** 命令或 **oc rsh** 命令访问 OpenShift Container Platform 集群上运行的容器。在镜像中安装并运行 sshd 会为安全攻击打开额外通道，因而需要安装安全补丁。

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。

另外，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。

其它资源

- Docker 文档 - 编写 Dockerfile 的最佳实践
- Project Atomic 文档 - 容器镜像创建者指南

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。

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。

例如，该 Python 镜像定义了构建各个版本的 Python 应用程序的 S2I 脚本。

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
```

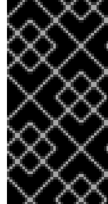
因为容器用户始终是 root 组的成员，所以容器用户可以读写这些文件。

警告

在修改容器敏感区域的目录和文件权限时，必须小心。

对于敏感区域，如 **/etc/passwd**，用户意外地对这些文件进行修改，可能会导致容器或主机被暴露。CRI-O 支持将随机用户 ID 插入容器的 **/etc/passwd** 中，因此不需要更改其权限。

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。为了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，您可向 **特权安全上下文约束** (SCC) 添加项目的构建程序服务帐户（**system:serviceaccount:<your-project>:builder**)。此外，您还可允许所有镜像以任何用户身份运行。

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。

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。

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。这表示，运行时配置应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。

有关如何在容器中管理 **cgroup** 配额的更多信息，请参阅以下参考资料：

- 博客文章 - Docker 中的资源管理
- Docker 文档 - 运行时指标
- 博客文章 - Linux 容器内存

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。

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。

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。

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。

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。

其它资源

- Docker 基础知识

- Dockerfile 参考
- 容器镜像创建者 Project Atomic 指南

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。

标签名称一般应具有命名空间。命名空间应进行相应设置，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`

变量	描述
io.openshift.non-scalable	镜像可能会使用此变量来表明它不支持扩展。UI 随后会将这一信息传达给该镜像的用户。不可扩展基本上意味着 **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 8Gi`
	`LABEL io.openshift.min-cpu     4`

4.3. 测试 S2I 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。

4.3.1. 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。

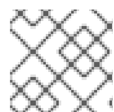
S2I 结合了应用程序源代码与构建程序镜像，因此为了对其进行测试，您需要一个示例应用程序源来验证该源是否成功转换成了可运行的容器镜像。示例应用程序应简单，但也应执行 **assemble** 和 **run** 脚本的关键步骤。

4.3.2. 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。

4.3.3. 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
```

4.3.4. 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。

4.3.5. 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。简单定义一个指向您的存储库的 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，这里的 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*持久性*标签，指向一段时间内的特定镜像。如果正在使用 _permanent_tag 并且源更改，则目的地的标签不会更改。

*跟踪*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>
  ```

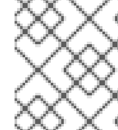
  例如：要将 **ruby** 镜像流 **static-2.0**标签配置为始终引用 **ruby**镜像流**2.0** 标签的当前镜像：

  ```
  $ 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 imagestreamtag。运行 **oc tag** 时，新标签会直接引用 **ruby:2.0** imagestreamtag 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*标签创建持久性别名，例如：**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 imagestreamtag。该标签持久指向源位置。
  如果您希望指示 OpenShift 始终从集成的 registry 中获取标记的镜像，则请使用 **--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
```

或者：

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

5.2.5. 引用镜像流中的镜像

您可通过以下引用类型，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或标签的镜像。
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>
  ```

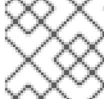
- 要引用给定镜像流的镜像和镜像 **sha** ID，请使用 **ImageStreamImage**：

  ```
  <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
```

5.2.6. 其他信息

- CentOS 镜像流的镜像流定义示例。

5.3. 镜像拉取（PULL）策略

Pod 中的每个容器均有容器镜像。您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

如果您在使用 OpenShift Container Platform 的内部 registry，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。

5.4.1. 允许 Pod 在项目间引用镜像

使用内部 registry 时，为允许 **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。

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

Docker 客户端的 **.dockercfg $HOME/.docker/config.json** 文件是一个 Docker 凭证文件，如果您之前已登录安全或不安全的 registry，则该文件会保存您的身份验证信息。

要拉取（pull）并非来自 OpenShift Container Platform 内部 registry 的安全容器镜像，您必须从 Docker 凭证创建一个 pull secret，并将其添加到您的服务帐户。

流程

- 如果您已有该安全 registry 的 **.dockercfg** 文件，则可运行以下命令从该文件中创建一个 secret：

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

- 或者，如果您已有 **$HOME/.docker/config.json** 文件，则可运行以下命令：

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

- 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（pull）镜像，您必须将该 secret 添加到您的服务帐户。本例中服务帐户的名称应与 Pod 所用服务帐户的名称匹配。**default** 是默认服务帐户：

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

- 要使用 secret 来推送（push）和拉取（pull）构建镜像，该 secret 必须可挂载至 Pod 中。您可通过运行以下命令实现这一目的：

```
$ oc secrets link builder <pull_secret_name>
```

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

您可为集群更新全局 pull secret。

警告

集群资源必须调整为新的 pull secret，这样可暂时限制集群的可用性。

先决条件

- 您有新的或修改的 pull secret 文件可上传。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

- 运行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。在这段时间中，节点会排空，Pod 将在剩余节点上重新调度。

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. 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。
- 外部 registry，如 **registry.redhat.io** 或 **hub.docker.com**。
- OpenShift Container Platform 集群中的其他镜像流。

您在定义引用 imagestreamtag 的对象时（如构建或部署配置），您指向了 imagestreamtag，而非 Docker 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。

镜像流元数据会连同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。
- 您可通过设置镜像流的访问权限来配置安全性，以控制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

镜像流对象文件包含以下元素。

镜像流对象定义

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
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: 2017-09-29T13:40:11Z
      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/更新镜像。
3. 此 imagestreamtag 当前引用的 SHA 标识符。引用此 imagestreamtag 的资源使用此标识符。
4. 此 imagestreamtag 之前引用的 SHA 标识符。可用于回滚至旧镜像。
5. imagestreamtag 名称。

6.3. 镜像流镜像

镜像流镜像从镜像流内部指向特定镜像 ID。

镜像流镜像用于从标记了镜像的特定镜像流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>
```

要引用镜像流对象示例中的镜像，则镜像流镜如下所示：

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

6.4. 镜像流标签（IMAGESTREAMTAGS）

imagestreamtag 是指向 *imagestream* 中镜像的命名指针，通常缩写为 *istag*。imagestreamtag 用于引用或检索给定镜像流或标签的镜像。

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。之前占据第一位置的镜像将移至第二位置，以此类推。这样便于回滚，从而让标签再次指向历史镜像。

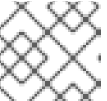
以下 imagestreamtag 来自镜像流对象：

历史记录中含有两个镜像的镜像流标签

```
  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: 2017-09-29T13:40:11Z
      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。

- *持久性标签*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，跟在另一 imagestreamtag 的后面，以后可进行更新以更改它们跟随的镜像，类似于符号链接。请注意，这些新等级无法保证向后兼容。
  例如，OpenShift Container Platform 附带的 **latest** imagestreamtag 是跟踪标签。这表示，当有新级别可用时，**latest** imagestreamtag 的用户将会更新到镜像提供的最新框架级别。指向 **v3.10** 的 **latest** imagestreamtag 可随时更改为指向 **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 imagestreamtag，可能并非镜像的最新版本。例如，如果 **latest** imagestreamtag 指向 **v3.10** 镜像，则当发布了 **3.11** 版时，**latest** 标签不会自动更新到 **v3.11**，仍会保持 **v3.10**，直到手动更新为指向 **v3.11** 镜像流。



注意

跟踪标签仅限于单个镜像流，无法引用其他镜像流。

您可根据自己的需求创建自己的 imagestreamtag。

imagestreamtag 由一个镜像流名称和一个标签组成，中间用冒号隔开：

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

例如：为引用前面镜像流对象示例中的 **sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d** 镜像，imagestreamtag 将是：

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

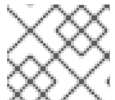
6.5. 镜像流更改触发器

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: 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

下面章节介绍如何使用镜像流和 imagestreamtag。

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

- 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：

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

  例如：

```
$ oc describe is/python

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
```

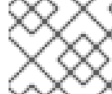
- 获取有关特定 imagestreamtag 的所有可用信息：

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

例如：

```
$ oc describe istag/python:latest

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。

6.7.2. 为镜像流添加标签

您可为镜像流添加额外标签。

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

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

- 确认镜像流有两个标签，一个 (**3.5**) 指向外部容器镜像，另一个标签 (**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

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
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 imagestreamtag

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

6.7.5. 删除 imagestreamtag

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:3.5

  Deleted tag default/python:3.5.
  ```

6.7.6. 配置定期导入 imagestreamtag

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 <repositiory/image> <image-name:tag> --scheduled
   ```

   例如：

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

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>
```

第 7 章 镜像配置资源

使用以下流程配置镜像 registry。

7.1. 镜像控制器配置参数

image.config.openshift.io/cluster 资源提供以下配置参数。

参数	描述
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**。 **Spec**：包含用户可设置的配置值。您可以编辑 **spec** 子章节。 **status**：包含从集群中观察到的值。
ImageSpec	**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。 **additionalTrustedCA**：引用包含 **ImageStream import**、**pod image pull**、**openshift-image-registry pullthrough** 和构建期间应受信任的额外 CA 的 ConfigMap。 该 ConfigMap 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。 **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。
ImageStatus	**internalRegistryHostname**：由控制 **internalRegistryHostname** 的 Image Registry Operator 设置。它为默认内部镜像 registry 设置主机名。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**：为默认的外部镜像 registry 提供主机名。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 ImageStream 中的 **publicDockerImageRepository** 字段。该值必须采用 **hostname[:port]** 格式。
RegistryLocation	包含由 registry 域名指定的 registry 的位置。域名可能包含通配符。 **domainname**：指定 registry 的域名。如果 registry 使用了非标准（80 或 443）端口，则该端口还应包含在域名中。 **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。

参数	描述
RegistrySources	包含有关如何处理 registry 配置的集群范围信息。 **insecureRegistries**：无有效 TLS 证书或仅支持 HTTP 连接的 registry。 **blockedRegistries**：列入镜像拉取（pull）和推送（push）操作黑名单。允许所有其他 registry。 **allowedRegistries**：列入镜像拉取（pull）和推送（push）操作白名单。阻止所有其他 registry。 仅可设置 **blockedRegistries** 或 **allowedRegistries** 中的一项

警告

当定义 **allowedRegistries** 参数时，所有 registry（包括 **registry.redhat.io** 和 **quay.io**)都会被阻断，除非明确列出。如果使用该参数，请根据环境中有效负载镜像的要求声明明源 registry **registry.redhat.io** 和 **quay.io**，以防止 Pod 失败。对于断开连接的集群，还应添加镜像的 registry。

7.2. 配置镜像设置

您可通过编辑 **image.config.openshift.io/cluster** 资源来配置镜像 registry 设置。Machine Config Operator (MCO) 会监控 **image.config.openshift.io/cluster** 以了解对 registry 的任何更改，并在检测到更改时重启节点。

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** 资源示例：

```
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
   insecureRegistries: 5
   - insecure.com
   blockedRegistries: 6
   - untrusted.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。
3. **additionalTrustedCA**：引用包含 **ImageStream import**、**pod image pull**、**openshift-image-registry pullthrough** 和构建期间应受信任的额外 CA 的 ConfigMap。该 ConfigMap 的命名空间为 **openshift-config**。ConfigMap 的格式是使用 registry 主机名作为键，使用 base64 编码证书作为值，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。
5. **insecureRegistries**：无有效 TLS 证书或仅支持 HTTP 连接的 registry。
6. **blockedRegistries**：列入镜像拉取（pull）和推送（push）操作黑名单。允许所有其他 registry。

7.2.1. 导入不安全的 registry 和阻止 registry

您可通过编辑 **image.config.openshift.io/cluster** 自定义资源 (CR) 来添加不安全的 registry 或阻止任何 registry。OpenShift Container Platform 会将对此 CR 的更改应用到集群中的所有节点。

应避免不安全的外部 registry，如无有效 TLS 证书或仅支持 HTTP 连接的 registry。

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** 资源示例：

```
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:
    insecureRegistries: 1
    - insecure.com
    blockedRegistries: 2
    - untrusted.com
    allowedRegistries:
    - quay.io 3
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 指定不安全的 registry。
2. 指定应列入镜像拉取（pull）和推送（push）操作黑名单的 registry。允许所有其他 registry。可以设置 **blockedRegistries** 或 **allowedRegistries**， 但不能同时都被设置。
3. 指定允许进行镜像拉取（pull）和推送（push）操作的 registry。所有其他 registry 都将被阻止。可以设置 **blockedRegistries** 或 **allowedRegistries**， 但不能同时都被设置。

Machine Config Operator (MCO) 会监控 **image.config.openshift.io/cluster** 以了解对 registry 的任何更改，并在检测到更改时重启节点。对 registry 的更改将出现在每个节点上的 **/host/etc/containers/registries.conf** 文件中。

```
cat /host/etc/containers/registries.conf
[registries]
  [registries.search]
    registries = ["registry.access.redhat.com", "docker.io"]
  [registries.insecure]
    registries = ["insecure.com"]
  [registries.block]
    registries = ["untrusted.com"]
```

7.2.2. 配置镜像 registry 存储库镜像

通过设置容器 registry 存储库镜像，您可以：

- 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。

以下是 OpenShift Container Platform 中存储库镜像的一些属性：

- 镜像拉取（pull）可应对 registry 停机时间

- 受限网络中的集群可请求从关键位置（如 quay.io）拉取（pull）镜像，并让公司防火墙后的 registry 提供所请求的镜像。
- 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，镜像则会被拉取（pull）至节点中。

可通过以下方式设置存储库镜像：

- 安装 OpenShift Container Platform 时：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。详情请参阅镜像 OpenShift Container Platform 的镜像存储库。
- 安装 OpenShift Container Platform 后：即使没有在 OpenShift Container Platform 安装期间配置镜像 (mirror)，之后您仍可使用 **ImageContentSourcePolicy** 对象进行配置。

以下流程提供安装后镜像配置，您可在此处创建 **ImageContentSourcePolicy** 对象来识别：

- 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。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 配置已镜像的存储库。为此，您可以：
    - 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目录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/ubi8/ubi-
      minimal@sha256:c505667389712dc337986e29ffcb65116879ef27629dc3ce6e1b17727c06
      e78f \
        docker://example.io/ubi8/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，您希望将 **ubi8/ubi-minimal** 镜像从 **registry.access.redhat.com** 复制到此镜像存储库。创建该 registry 后，您可将 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. 创建 **ImageContentSourcePolicy** 文件（如：**registryrepomirror.yaml**），将源和镜像 (mirror) 替换为您自己的 registry、存储库对和镜像中的源和镜像：

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
  repositoryDigestMirrors:
  - mirrors:
    - example.io/example/ubi-minimal (1)
    source: registry.access.redhat.com/ubi8/ubi-minimal (2)
  - mirrors:
    - example.com/example/ubi-minimal
    source: registry.access.redhat.com/ubi8/ubi-minimal
~
```

1. 指明镜像 registry 和存储库的名称
2. 指明包含所镜像内容的 registry 和存储库

4. 创建新的 **ImageContentSourcePolicy**：

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

创建 **ImageContentSourcePolicy** 之后，新的设置会部署到每个节点，随即使用已镜像的存储库向源存储库发起请求。

5. 要检查已镜像的配置是否正常工作，请转至其中一个节点。例如：

a. 列出您的节点：

```
$ oc get node
NAME                      STATUS                 ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                    worker   7m   v1.14.6+90fadebfa
ip-10-0-138-148.ec2.internal   Ready                    master   11m  v1.14.6+90fadebfa
ip-10-0-139-122.ec2.internal   Ready                    master   11m  v1.14.6+90fadebfa
ip-10-0-147-35.ec2.internal    Ready,SchedulingDisabled   worker   7m
v1.14.6+90fadebfa
ip-10-0-153-12.ec2.internal    Ready                    worker   7m   v1.14.6+90fadebfa
ip-10-0-154-10.ec2.internal    Ready                    master   11m  v1.14.6+90fadebfa
```

您可以发现，在应用更改时每个 worker 节点上的调度都会被禁用。

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

```
$ oc debug node/ip-10-0-147-35.ec2.internal
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# chroot /host
sh-4.2# cat /etc/containers/registries
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
[[registry]]
  location = "registry.access.redhat.com/ubi8/"
```

```
insecure = false
blocked = false
mirror-by-digest-only = true
prefix = ""

[[registry.mirror]]
  location = "example.io/example/ubi8-minimal"
  insecure = false

[[registry.mirror]]
  location = "example.com/example/ubi8-minimal"
  insecure = false
```

c. 将镜像从源拉取（pull）到节点，并检查是否真正通过镜像解析。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-minimal
```

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** 文件的格式。现在是第 2 版，采用 TOML 格式。*

第 8 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。

8.1. 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 DeploymentConfig。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。

8.2. 上传模板

如果您有可定义模板的 JSON 或 YAML 文件，如本例中所示，则可以使用 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。本主题后面会介绍如何编写自己的模板。

流程

- 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>
```

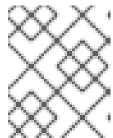
现在可使用 web 控制台或 CLI 选择该模板。

8.3. 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. 在所需项目中，点击 **Add to Project**。
2. 从项目中的镜像列表或从服务目录中选择构建程序镜像。



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:
```

```
dockerImageRepository: "registry.redhat.io/openshift3/ruby-20-rhel7"
tags:
  -
    name: "2.0"
    annotations:
      description: "Build and run Ruby 2.0 applications"
      iconClass: "icon-ruby"
      tags: "builder,ruby" 1
      supports: "ruby:2.0,ruby"
      version: "2.0"
```

1 此处包含 **builder** 可确保该 **ImageStreamTag** 作为构建程序出现在 web 控制台中。

3. 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。

8.4. 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。

8.4.1. 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
```

8.4.2. 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 Quickstart 模板的参数时的输出：

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

8.4.3. 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=-
```

8.5. 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>
```

8.6. 使用 INSTANT APP 和 QUICKSTART 模板

OpenShift Container Platform 提供很多默认的 Instant App 和 Quickstart 模板，有助于快速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。您的集群管理员应已在默认的全局 **openshift** 项目中创建了这些模板，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

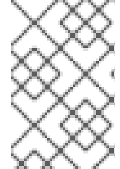
1. 您可通过以下命令列出可用的默认 Instant App 和 Quickstart 模板：

```
$ 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。

8.6.1. Quickstart 模板

Quickstart 是 OpenShift Container Platform 上运行的应用程序的一个基本示例。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 DeploymentConfig 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 Quickstart，请从模板创建应用程序。您的管理员可能已在您的 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。

Quickstarts 是指包含应用程序源代码的源存储库。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。

8.6.1.1. Web 框架 Quickstart 模板

这些 Quickstart 模板提供了指定框架和语言的基本应用程序：

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

8.7. 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: 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
```

8.7.1. 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: 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。此外，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。尽可能从现有徽标图标中选择。也可使用 FontAwesome 中的图标。此外，还可通过 CSS 自定义提供图标，将其添加到使用您的模板的 OpenShift Container Platform 集群中。您必须指定一个存在的图标类，否则它将阻止回退到通用图标。
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。

8.7.2. 编写模板标签

模板可包括一组标签。这些标签将添加至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，将应用于从该模板创建的所有对象。
2. 参数化标签，也将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。

8.7.3. 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 **${PARAMETER_NAME}** 的值放在模板的任意字符串字段中。
- 作为 json/yaml 值，将格式为 **${PARAMETER_NAME}** 的值放在模板的任意字段中。

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，则结果值将加引号并被视为标准字符串。

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，如果用户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}"
```

在前面的示例中，处理将生成一个全部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**和 **\a** 修饰符：

- **[\w]{10}** 生成 10 个字母字符、数字和下划线。这遵循 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。等同于 **[a-zA-Z]{10}**。

下面是附带参数定义和参考的完整模板示例：

```
kind: Template
apiVersion: v1
metadata:
  name: my-template
objects:
  - kind: BuildConfig
    apiVersion: 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: "${CONTEXT_DIR}"
- kind: DeploymentConfig
  apiVersion: v1
  metadata:
    name: frontend
  spec:
    replicas: "${{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。这将对用户显示。
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. 如果用户实例化该模板时未覆盖该值，则将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. 指示此参数为必填项，表示用户无法用空白值覆盖它。如果该参数未提供默认值或生成值，则用户必须提供一个值。
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。此项告知用户生成的值。

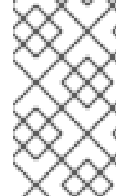
8.7.4. 编写模板对象列表

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 **BuildConfig**、**DeploymentConfig**、**Service** 等。该对象将完全按照此处定义创建，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) 将由该模板创建的 **Service** 的定义。



注意

如果对象定义的元数据包含固定的 **namespace** 字段值，则在模板实例化过程中，该字段将从定义中剥离出来。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。

8.7.5. 将模板标记为可绑定

Template Service Broker 会在目录中为其知晓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。

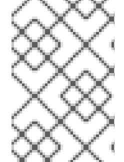
8.7.6. 公开模板对象字段

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 上的公开字段，并在用户绑定代理所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。

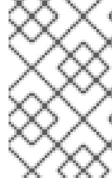
每个移除前缀的注解键均会被传递成为 **bind** 响应中的一个键。

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: 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: bar
- 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: 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。

8.7.7. 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	对象报告新的 ReplicaSet 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错误

类型	就绪	失败
DeploymentConfig	对象报告新的 ReplicationController 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错误
Job	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 quickstart 模板中找到。

```
kind: Template
apiVersion: v1
metadata:
  name: my-template
objects:
- kind: BuildConfig
  apiVersion: 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: 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。

8.7.8. 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

1. 以 YAML 格式导出项目中的对象：

```
$ oc get -o yaml --export 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 --export all 中包括的对象类型是：

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

第 9 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采取的步骤，以确保正确执行了所有步骤。

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。

9.1. 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
```

该命令将创建 **/var/lib/pgsql/data** 目录，数据将存储在其中。

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。

9.2. 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
Successfully installed rails-4.2.0
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:
```

7. 创建应用程序的开发和测试数据库：

```
$ rake db:create
```

这将在您的 PostgreSQL 服务器中创建 **development** 和 **test** 数据库。

9.2.1. 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：

- 使用索引操作创建 controller
- 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 **route**

Rails 提供了一个生成器，可为您执行所有必要步骤。

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。

9.2.2. 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：

```
<% 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"] %>
```

9.2.3. 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
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。

1. 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
```

2. 将应用程序推送（push）到远程 git 存储库。

```
$ git push
```

9.3. 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的项目：

```
$ oc new-project rails-app --description="My Rails application" --display-name="Rails Application"
```

9.3.1. 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。对于该服务，请使用 PostgeSQL 数据库镜像。

要创建数据库服务，可使用 **oc new-app** 命令。您必须将一些要在数据库容器内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
```

9.3.2. 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** DeploymentConfig 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：

```
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，在 OpenShift Container Platform 中查看正在运行的 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。

9.3.3. 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 10 章 使用镜像

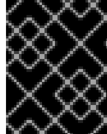
10.1. 使用镜像概述

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。

本部分涵盖的红帽支持的所有镜像均在 Red Hat Container Catalog 中进行说明。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。

10.2. 配置 JENKINS 镜像

OpenShift Container Platform 为运行 Jenkins 提供容器镜像。此镜像提供 Jenkins 服务器实例，可用于为连续测试、集成和交付设置基本流程。

该镜像基于 Red Hat Universal Base Images（UBI)。

OpenShift Container Platform 遵从 Jenkins 的 LTS 的发行版本。OpenShift Container Platform 提供一个包含 Jenkins 2.x 的镜像。

OpenShift Container Platform Jenkins 镜像在 **quay.io** 或 **registry.redhat.io** 上提供。

例如：

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins:<v4.2.0>
```

要使用这些镜像，您可直接从这些 registry 访问镜像或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。另外，您还可在容器镜像 registry 或外部位置创建一个指向镜像的 ImageStream。然后，OpenShift Container Platform 资源便可引用 ImageStream。

但为方便起见，OpenShift Container Platform 会在 **openshift** 命名空间中为核心 Jenkins 镜像以及针对 OpenShift Container Platform 与 Jenkins 集成提供的示例代理镜像提供 ImageStream。

10.2.1. 配置和自定义

您可采用两种方式管理 Jenkins 身份验证：

- 由 OpenShift Login 插件提供的 OpenShift Container Platform Oauth 身份验证。
- 由 Jenkins 提供的标准身份验证。

10.2.1.1. OpenShift Container Platform OAuth 身份验证

OAUTH 身份验证激活方法：配置 Jenkins UI 中 **Configure Global Security** 面板上的选项，或者将 Jenkins **Deployment configuration** 上的 **OPENSHIFT_ENABLE_OAUTH** 环境变量设置为非 **false**。这会激活 OpenShift Container Platform Login 插件，该插件从 Pod 数据或通过与 OpenShift Container Platform API 服务器交互来检索配置信息。

有效凭证由 OpenShift Container Platform 身份提供程序控制。

Jenkins 支持浏览器和非浏览器访问。

登录时，有效用户会自动添加到 Jenkins 授权列表中，其中的 OpenShift Container Platform **Role** 规定了用户拥有的特定 Jenkins 权限。默认使用的 **Role** 是预定义的 **admin**、**edit** 和 **view**。登录插件对 Jenkins 正在其中运行的 **Project** 或命名空间中的那些 **Role** 执行自身 SAR 请求。

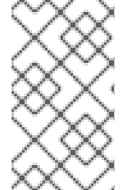
具有 **Admin** 角色的用户拥有传统 Jenkins 管理用户权限，而具有 **edit** 或 **view** 角色的用户的权限逐渐减少。

默认的 OpenShift Container Platform **admin**、**edit** 和 **view Role** 以及这些 **Role** 在 Jenkins 实例中分配的 Jenkins 权限均可配置。

在 OpenShift Container Platform **Pod** 中运行 Jenkins 时，登录插件会在 Jenkins 正在其中运行的命名空间中查找名为 **openshift-jenkins-login-plugin-config** 的 **ConfigMap**。

如果该插件找到 **ConfigMap** 且可读取，您就可以定义 **Role** 到 Jenkins 权限的映射。具体来说：

- 登录插件会将 **ConfigMap** 中的键值对视为 Jenkins 权限到 OpenShift 角色的映射。
- 其中，键是 Jenkins 权限组短 ID 和 Jenkins 权限短 ID，两者之间用连字符隔开。
- 如果要向 OpenShift Container Platform **Role** 中添加 **Overall Jenkins Administer** 权限，该键应为 **Overall-Administer**。
- 要了解有哪些权限组和权限 ID 可用，请转至 Jenkins 控制台中的列表授权页，并在它们提供的表中查找组 ID 和个别权限。
- 键值对的值是权限应当应用于的 OpenShift Container Platform **Roles** 的列表，各个角色之间用逗号隔开。
- 如果要将 **Overall Jenkins Administer** 权限添加到默认的 **admin** 和 **edit Role** 以及您创建的新 Jenkins 角色，则 **Overall-Administer** 键的值将为 **admin,edit,jenkins**。



注意

使用 OpenShift Container Platform OAuth 时，OpenShift Container Platform Jenkins 镜像中预填充了管理特权的 **admin** 用户不会被授予这些特权。要授予这些权限，OpenShift Container Platform 集群管理员必须在 OpenShift Container Platform 身份提供程序中显式定义该用户，并为该用户分配 **admin** 角色。

最初建立用户后，可对存储的 Jenkins 用户权限进行更改。OpenShift Login 插件轮询 OpenShift Container Platform API 服务器以获取权限，并利用从 OpenShift Container Platform 检索的权限更新存储在 Jenkins 中的每个用户的权限。如果 Jenkins UI 用于为 Jenkins 用户更新权限，则权限更改将在插件下次轮询 OpenShift Container Platform 时被覆盖。

您可通过 **OPENSHIFT_permissions_poll_interval** 环境变量来控制轮询频率。默认轮询间隔为五分钟。

使用 Oauth 身份验证创建新的 Jenkins 服务的最简单方式是借助模板。

10.2.1.2. Jenkins 身份验证

如果镜像未使用模板直接运行，则默认使用 Jenkins 身份验证。

Jenkins 首次启动时，配置与管理员用户和密码一同创建。默认用户凭证为 **admin** 和 **password**。在使用标准 Jenkins 身份验证时，且仅这种情况下，通过设置 **JENKINS_PASSWORD** 环境变量来配置默认密码。

流程

- 创建使用标准 Jenkins 身份验证的 Jenkins 应用程序：

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    openshift4/ose-jenkins
```

10.2.2. Jenkins 环境变量

Jenkins 服务器可通过以下环境变量进行配置：

变量	定义	值和设置示例
OPENSHIFT_ENABLE_OAUTH	决定在登录 Jenkins 时，OpenShift Login 插件可否管理身份验证。要启用，请设为 **true**。	默认：**false**
JENKINS_PASSWORD	使用标准 Jenkins 身份验证时 **admin** 用户的密码。**OPENSHIFT_ENABLE_OAUTH** 设置为 **true** 时不适用。	默认：**password**
JAVA_MAX_HEAP_PARAM、**CONTAINER_HEAP_PERCENT**、**JENKINS_MAX_HEAP_UPPER_BOUND_MB**	这些值控制 Jenkins JVM 的最大堆大小。如果设置了 **JAVA_MAX_heap_PARAM**，则优先使用其值。否则，最大堆大小将动态计算为容器内存限值的 **CONTAINER_HEAP_PERCENT**，可选上限为 **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB。 默认情况下，Jenkins JVM 的最大堆大小设置为容器内存限值的 50%，且无上限。	**Java_MAX_heap_PARAM** 示例设置：**-Xmx512m** **container_heap_percent** 默认：**0.5** 或 50% **Jenkins_MAX_heap_UPPER_BOUND_MB** 示例设置：**512 MiB**

变量	定义	值和设置示例
JAVA_INITIAL_HEAP_PARAM、CONTAINER_INITIAL_PERCENT	这些值控制 Jenkins JVM 的初始堆大小。如果设置了 **JAVA_INITIAL_heap_PARAM**，则优先使用其值。否则，初始堆大小将动态计算为动态计算的最大堆大小的 **CONTAINER_INITIAL_PERCENT**。 默认情况下，JVM 设置初始堆大小。	**java_INITIAL_heap_PARAM** 示例设置：**-Xms32m** **container_INITIAL_percent** 示例设置：**0.1**或 10%
CONTAINER_CORE_LIMIT	如果设置，请将用于调整内部 JVM 线程数的内核数指定为整数。	示例设置：**2**
JAVA_TOOL_OPTIONS	指定应用于该容器中运行的所有 JVM 的选项。不建议覆盖该值。	默认：**-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**
JAVA_GC_OPTS	指定 Jenkins JVM 垃圾回收参数。不建议覆盖该值。	默认：**-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**
JENKINS_JAVA_OVERRIDES	指定适用于 Jenkins JVM 的附加选项。这些选项附加到所有其他选项中，包括上面的 Java 选项，必要时可用于覆盖其中任何一个选项。用空格分开各个附加选项；如有任意选项包含空格字符，请使用反斜杠转义。	示例设置：**-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**。
JENKINS_OPTS	为 Jenkins 指定参数。	
INSTALL_PLUGINS	指定在容器首次运行或 **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** 设置为 **true** 时需要安装的 Jenkins 附加插件。插件被指定为用逗号分隔的“名称:版本”对列表。	示例设置：**git:3.7.0,subversion:2.10.2**。

变量	定义	值和设置示例
OPENSHIFT_PERMISSIONS_POLL_INTERVAL	指定 OpenShift Login 插件轮询 OpenShift Container Platform 的时间间隔（以毫秒为单位），以获取与 Jenkins 中定义的每个用户关联的权限。	默认：**300000** - 5 分钟
OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG	当使用 Jenkins 配置目录的 OpenShift Container Platform 持久性卷运行此镜像时，从镜像到持久性卷的配置传输仅在镜像首次启动时执行，因为在创建持久性卷声明时会分配持久性卷。如果您在初始启动后创建自定义镜像来扩展此镜像并更新自定义镜像中的配置，则不会复制该配置，除非将该环境变量设置为 **true**。	默认：**false**
OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS	当使用 Jenkins 配置目录的 OpenShift Container Platform 持久性卷运行此镜像时，从镜像到持久性卷的插件传输仅在镜像首次启动时执行，因为在创建持久性卷声明时会分配持久性卷。如果您在初始启动后创建自定义镜像来扩展此镜像并更新自定义镜像中的插件，则不会复制该插件，除非将该环境变量设置为 **true**。	默认：**false**
ENABLE_FATAL_ERROR_LOG_FILE	当使用 Jenkins 配置目录的 OpenShift Container Platform 持久性卷声明运行此镜像时，该环境变量允许在发生严重错误时，保留严重错误日志文件。严重错误文件保存在：**/var/lib/jenkins/logs**。	默认：**false**
NODEJS_SLAVE_IMAGE	设置此值将覆盖用于 NodeJS 代理 Pod 默认配置的镜像。项目中有一个名为 **jenkins-agent-nodejs** 的相关 imagestreamtag。该变量必须在 Jenkins 首次启动前进行设置，以便其生效。	Jenkins 服务器中的默认 NodeJS 代理镜像：**image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-nodejs:latest**
MAVEN_SLAVE_IMAGE	设置此值将覆盖用于默认 Maven 代理 Pod 配置的镜像。项目中有一个名为 **jenkins-agent-maven** 的相关 imagestreamtag。该变量必须在 Jenkins 首次启动前进行设置，以便其生效。	Jenkins 服务器中的默认 Maven 代理镜像：**image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-maven:latest**

10.2.3. 向 Jenkins 提供跨项目访问权限

如果要在与您的项目不同的其他位置运行 Jenkins，则必须向 Jenkins 提供访问令牌来访问您的项目。

流程

1. 识别对 Jenkins 必须访问的项目具有适当访问权限的服务帐户的 secret：

```
$ oc describe serviceaccount jenkins
Name:       default
Labels:     <none>
Secrets:    {  jenkins-token-uyswp    }
            {  jenkins-dockercfg-xcr3d    }
Tokens:     jenkins-token-izv1u
            jenkins-token-uyswp
```

   这种情况下，secret 被命名为 **jenkins-token-extensionswp**。

2. 从 secret 中检索令牌：

```
$ oc describe secret <secret name from above>
Name:       jenkins-token-uyswp
Labels:     <none>
Annotations:    kubernetes.io/service-account.name=jenkins,kubernetes.io/service-
account.uid=32f5b661-2a8f-11e5-9528-3c970e3bf0b7
Type:   kubernetes.io/service-account-token
Data
====
ca.crt: 1066 bytes
token:  eyJhbGc..<content cut>....wRA
```

   令牌参数包含 Jenkins 访问项目所需的令牌值。

10.2.4. Jenkins 跨卷挂载点

可使用挂载卷运行 Jenkins 镜像，以便为配置启用持久性存储：

- **/Var/lib/jenkins** - 这是 Jenkins 存储配置文件的数据目录，包含任务定义。

10.2.5. 通过 Source-To-Image 自定义 Jenkins 镜像

要自定义官方 OpenShift Container Platform Jenkins 镜像，可使用该镜像作为 Source-To-Image (S2I) 构建程序。

您可使用 S2I 来复制自定义 Jenkins 任务定义，添加其它插件，或使用您自己的自定义配置来替换所提供的 **config.xml** 文件。

要在 Jenkins 镜像中包括您的修改，必须要有采用以下目录结构的 Git 存储库：

plugins

该目录包含要复制到 Jenkins 中的二进制 Jenkins 插件。

plugins.txt

该文件使用以下语法列出要安装的插件:

```
pluginId:pluginVersion
```

configuration/jobs
该目录包含 Jenkins 任务定义。

configuration/config.xml
该文件包含您的自定义 Jenkins 配置。

configuration/ 目录的内容会被复制到 **/var/lib/jenkins/** 目录中，以便也可以包括其他文件，如 **credentials.xml**。

以下构建配置示例对 OpenShift Container Platform 中的 Jenkins 镜像进行了自定义：

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                       (1)
    git:
      uri: https://github.com/custom/repository
    type: Git
  strategy:                     (2)
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:2
        namespace: openshift
    type: Source
  output:                       (3)
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

(1) **source** 参数使用上述布局定义源 Git 存储库。

(2) **strategy** 参数定义用作构建的源镜像的原始 Jenkins 镜像。

(3) **output** 参数定义可用于部署配置的生成自定义 Jenkins 镜像，而非官方 Jenkins 镜像。

10.2.6. 配置 Jenkins Kubernetes 插件

OpenShift Container Platform Jenkins 镜像包含预装的 Kubernetes 插件，支持使用 Kubernes 和 OpenShift Container Platform 在多个容器主机上动态置备 Jenkins 代理。

为了使用 Kubernetes 插件，OpenShift Container Platform 提供了适合用作 Jenkins 代理的镜像：**Base**、**Maven** 和 **Node.js** 镜像。

Maven 和 Node.js 代理镜像均会在 OpenShift Container Platform Jenkins 镜像的 Kubernetes 插件配置中自动配置为 Kubernetes Pod 模板镜像。该配置包含各个镜像的标签，可应用于 **Restrict where this project can be run** 设置下的任何 Jenkins 任务。如果应用了标签，任务将在运行相应代理镜像的 OpenShift Container Platform Pod 下运行。

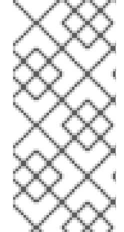
Jenkins 镜像还为 Kubernetes 插件提供附加代理镜像的自动发现和自动配置。

使用 OpenShift Container Platform Sync 插件，Jenkins 启动上的 Jenkins 镜像会在其运行的项目中或插件配置中特别列出的项目中搜索以下内容：

- 将标签 **role** 设置为 **jenkins-slave** 的镜像流。
- 将注解 **role** 设置为 **jenkins-slave** 的镜像流标签。
- 将标签 **role** 设置为 **jenkins-slave** 的 ConfigMaps。

当该镜像找到具有适当标签的镜像流或具有适当注解的 imagestreamtag 时，它会生成对应的 Kubernetes 插件配置，以便您可以将 Jenkins 任务分配到运行镜像流提供的容器镜像的 Pod 中运行。

镜像流或 imagestreamtag 的名称和镜像引用被映射到 Kubernetes 插件 Pod 模板中的名称和镜像字段。您可利用 **slave-label** 键在镜像流或 imagestreamtag 对象上设置注解，从而控制 Kubernetes 插件 Pod 模板的标签字段。否则，名称将用作标签。



注意

不要登录到 Jenkins 控制台，也不要修改 Pod 模板配置。如果您在创建 Pod 模板后这样做，且 OpenShift Sync 插件检测到与 ImageStream 或 ImageStreamTag 关联的镜像已更改，则该插件会替换 Pod 模板并覆盖这些配置更改。您无法将新配置与现有配置合并。

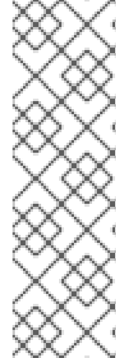
如有更为复杂的配置需求，请考虑 ConfigMap 方法。

找到具有适当标签的 ConfigMap 时，它会假定 ConfigMap 的键值数据有效负载中的任何值均包含与 Jenkins 和 Kubernetes 插件 Pod 模板的配置格式一致的 XML。使用 ConfigMaps 而非镜像流或 imagestreamtag 时，需要注意的一个关键区别是您可以控制 Kubernetes 插件 Pod 模板的所有参数。

jenkins-agent 的 ConfigMap 示例：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-slave
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
      <inheritFrom></inheritFrom>
      <name>template1</name>
      <instanceCap>2147483647</instanceCap>
      <idleMinutes>0</idleMinutes>
      <label>template1</label>
      <serviceAccount>jenkins</serviceAccount>
      <nodeSelector></nodeSelector>
      <volumes/>
      <containers>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
          <privileged>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/tmp</workingDir>
          <command></command>
          <args>${computer.jnlpmac} ${computer.name}</args>
```

```
      <ttyEnabled>false</ttyEnabled>
      <resourceRequestCpu></resourceRequestCpu>
      <resourceRequestMemory></resourceRequestMemory>
      <resourceLimitCpu></resourceLimitCpu>
      <resourceLimitMemory></resourceLimitMemory>
      <envVars/>
    </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
  </containers>
  <envVars/>
  <annotations/>
  <imagePullSecrets/>
  <nodeProperties/>
</org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```



注意

如果您登录到 Jenkins 控制台并在 Pod 模板创建后对 Pod 模板配置进行进一步更改，且 OpenShift Sync 插件检测到 ConfigMap 已更改，则该插件会替换 Pod 模板并覆盖这些配置更改。您无法将新配置与现有配置合并。

不要登录到 Jenkins 控制台，也不要修改 Pod 模板配置。如果您在创建 Pod 模板后这样做，且 OpenShift Sync 插件检测到与 ImageStream 或 ImageStreamTag 关联的镜像已更改，则该插件会替换 Pod 模板并覆盖这些配置更改。您无法将新配置与现有配置合并。

如有更为复杂的配置需求，请考虑 ConfigMap 方法。

安装后，OpenShift Sync 插件会监控 OpenShift Container Platform 的 API 服务器，以获取对 **ImageStreams**、**ImageStreamTags** 和 **ConfigMaps** 的更新，并调整 Kubernetes 插件的配置。

适用以下规则：

- 从 **ConfigMap**、**ImageStream** 或 **ImageStreamTag** 中删除标签或注解会导致从 Kubernetes 插件配置中删除任何现有 **PodTemplate**。
- 如果删除了这些对象，相应配置也会从 Kubernetes 插件中删除。
- 无论是创建适当标记或注解的 **ConfigMap**、**ImageStream** 或 **ImageStreamTag** 对象，还是在最初创建后添加标签，都会导致在 Kubernetes-plugin 配置中创建 **PodTemplate**。
- 对于按照 **ConfigMap** 表单排布的 **PodTemplate**，对 **PodTemplate** 的 **ConfigMap** 数据的更改将应用于 Kubernetes 插件配置中的 **PodTemplate** 设置，并在 **ConfigMap** 更改之间覆盖通过 Jenkins UI 对 **PodTemplate** 所做的任何更改。

要将容器镜像用作 Jenkins 代理，该镜像必须运行 slave 代理，作为入口点。有关此方面的更多详情，请参阅官方 Jenkins 文档。

10.2.7. Jenkins 权限

在 ConfigMap 中，如果 Pod 模板 XML 的 **<serviceAccount>** 元素是用于生成的 Pod 的 OpenShift Container Platform 服务帐户，则服务帐户凭证将挂载到 Pod 中。权限与服务帐户关联，并控制允许从 Pod 对 OpenShift Container Platform master 执行的操作。

考虑以下场景，服务帐户用于 OpenShift Container Platform Jenkins 镜像中运行的 Kubernetes 插件启动的 Pod：

如果您使用 OpenShift Container Platform 提供的 Jenkins 示例模板，则 **jenkins** 服务帐户将由运行 Jenkins 的项目的 **edit** 角色定义，且 master Jenkins Pod 已挂载了该服务帐户。

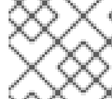
注入 Jenkins 配置的两个默认 Maven 和 NodeJS Pod 模板也将设置为使用与 Jenkins master 相同的服务帐户。

- 由于镜像流或 imagestreamtag 具有所需的标签或注解而被 OpenShift Sync 插件自动发现的任何 Pod 模板均会被配置为使用 Jenkins master 的服务帐户作为其服务帐户。
- 对于其他方法，您可在 Jenkins 和 Kubernetes 插件中提供 Pod 模板定义，但必须明确指定要使用的服务帐户。其它方法包括 Jenkins 控制台、由 Kubernetes 插件提供的 **podTemplate** 管道 DSL，或标记其数据为 Pod 模板的 XML 配置的 ConfigMap。
- 如果没有为服务帐户指定值，则将使用 **default** 服务帐户。
- 确保所使用的任何服务帐户均具有 OpenShift Container Platform 中定义的必要权限、角色等，以操作您选择从 Pod 中操作的任何项目。

10.2.8. 从模板创建 Jenkins 服务

模板提供参数字段来定义具有预定义默认值的所有环境变量。OpenShift Container Platform 提供模板，以简化 Jenkins 服务的新建操作。在初始集群设置期间，您的集群管理员应在默认 **openshift** 项目中注册 Jenkins 模板。

提供的两个模板均定义了部署配置和服务。两个模板的不同之处在于存储策略，这会影响在 Pod 重启后是否保留 Jenkins 内容。



注意

当 Pod 移到另一节点，或当对部署配置的更新触发了重新部署时，Pod 可能会重启。

- **jenkins-ephemeral** 使用临时存储。Pod 重启时，所有数据均会丢失。该模板仅适用于开发或测试。
- **jenkins-persistent** 使用持久性卷存储。数据不会因 Pod 重启而丢失。

要使用持久卷存储，集群管理员必须在 OpenShift Container Platform 部署中定义一个持久性卷池。

选好所需模板后，您还必须对模板进行实例化，才能使用 Jenkins。

流程

1. 使用以下任一方法新建 Jenkins 应用程序：
   - 持久性卷：
     ```
     $ oc new-app jenkins-persistent
     ```
   - 或 **emptyDir** 类型卷，其配置在 Pod 重启后不保留：
     ```
     $ oc new-app jenkins-ephemeral
     ```

10.2.9. 使用 Jenkins Kubernetes 插件

在以下示例中，**openshift-jee-sample** BuildConfig 会致使 Jenkins Maven 代理 Pod 进行动态置备。该 Pod 会克隆一些 Java 源代码，构建一个 WAR 文件，并致使第二 BuildConfig“**openshift-jee-sample-docker**”运行。第二 BuildConfig 会将新的 WAR 文件分层到一个容器镜像中。

以下示例是使用 Jenkins Kubernetes 插件的 BuildConfig。

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample-docker
  spec:
    strategy:
      type: Docker
    source:
      type: Docker
      dockerfile: |-
        FROM openshift/wildfly-101-centos7:latest
        COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
        CMD $STI_SCRIPTS_PATH/run
      binary:
        asFile: ROOT.war
    output:
      to:
        kind: ImageStreamTag
        name: openshift-jee-sample:latest
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
        jenkinsfile: |-
          node("maven") {
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
    triggers:
    - type: ConfigChange
```

它还可覆盖动态创建的 Jenkins 代理 Pod 的规范。下面是对前一示例的修改，可覆盖容器内存并指定环境变量：

以下示例是使用 Jenkins Kubernetes 插件的 BuildConfig，指定内存限值和环境变量。

```
kind: BuildConfig
apiVersion: v1
```

```
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", (1)
                    cloud: "openshift", (2)
                    inheritFrom: "maven", (3)
                    containers: [
          containerTemplate(name: "jnlp", (4)
                            image: "openshift/jenkins-agent-maven-35-centos7:v3.10", (5)
                            resourceRequestMemory: "512Mi", (6)
                            resourceLimitMemory: "512Mi", (7)
                            envVars: [
            envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") (8)
          ])
        ]) {
          node("mypod") { (9)
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
        }
  triggers:
  - type: ConfigChange
```

1. 动态定义的名为 **mypod** 的新 Pod 模板。新 Pod 模板名称在节点片段中引用。
2. **cloud** 值必须设置为 **openshift**。
3. 新 Pod 模板可继承现有 Pod 模板的配置。在本例中，继承自 OpenShift Container Platform 预定义的 Maven Pod 模板。
4. 本例覆盖了预先存在容器中的值，且必须按名称指定。OpenShift Container Platform 附带的所有 Jenkins 代理镜像均使用容器名称 **jnlp**。
5. 再次指定容器镜像名称。这是个已知问题。
6. 指定了 **512 Mi** 的内存请求。
7. 指定了 **512 Mi** 的内存限值。
8. **CONTAINER_HEAP_PERCENT** 环境变量，其值指定为 **0.25**。
9. 节点片段引用所定义的 Pod 模板的名称。

构建完成后会默认删除 pod。该行为可通过插件或在 Jenkinsfile 管道中修改。

10.2.10. Jenkins 内存要求

使用所提供的 Jenkins Ephemeral 或 Jenkins Persistent 模板部署时，默认内存限值为 **1 Gi**。

默认情况下，Jenkins 容器中运行的所有其他进程使用的内存总量不超过 **512 MiB**。如果这些进程需要更多内存，容器将停止。因此，我们强烈建议管道尽可能在代理容器中运行外部命令。

如果 **Project** 配额允许，请参阅 Jenkins 文档，了解 Jenkins master 应具有多少内存的建议。这些建议禁止为 Jenkins master 分配更多内存。

建议在由 Jenkins Kubernetes 插件创建的代理容器上指定内存请求和限值。管理员用户可通过 Jenkins 配置基于每个代理镜像设置默认值。内存请求和限值参数也可基于每个容器覆盖。

在实例化 Jenkins Ephemeral 或 Jenkins Persistent 模板时，您可通过覆盖 **MEMORY_LIMIT** 参数来增加 Jenkins 的可用内存量。

10.2.11. 其它资源

- 有关 Red Hat Universal Base Images (UBI) 的更多信息，请参阅 基础镜像选项。

10.3. JENKINS 代理

OpenShift Container Platform 提供了适合用作 Jenkins 代理的三种镜像：***Base***、***Maven*** 和 ***Node.js*** 镜像。

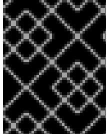
第一个是适用于 Jenkins 代理的基础镜像：

- 它会拉取（pull）所需工具、无头 Java、Jenkins JNLP 客户端以及一些实用工具，其中包括 **git**、**tar**、**zip** 和 **nss** 等。
- 它将 JNLP 代理设立为入口点。
- 它包含 **oc** 客户端工具，用于从 Jenkins 任务调用命令行操作。
- 它为 Red Hat Enterprise Linux (RHEL) 和 **localdev** 镜像提供 Dockerfile。

另外还提供了扩展基础镜像的两个镜像：

- Maven v3.5 镜像
- Node.js v8 镜像

Maven 和 Node.js Jenkins 代理镜像为通用基础镜像 (UBI) 提供 Dockerfile，您可在构建新代理镜像时引用。另请注意 **contrib** 和 **contrib/bin** 子目录。这些子目录可用于为您的镜像插入配置文件和可执行脚本。



重要

为 OpenShift Container Platform 使用并扩展适当的代理镜像版本。如果嵌入至该代理镜像的 **oc** 客户端版本与 OpenShift Container Platform 版本不兼容，则可能引发意外行为。

10.3.1. Jenkins 代理镜像

OpenShift Container Platform Jenkins 代理镜像在 **quay.io** 或 **registry.redhat.io** 上提供。

Jenkins 镜像通过 Red Hat Registry 提供：

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins:<v4.2.0>
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-nodejs:<v4.2.0>
```

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-maven:<v4.2.0>
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-base:<v4.2.0>
```

要使用这些镜像，您可直接从 **quay.io** 或 **registry.redhat.io** 访问或将其推送（push）到 OpenShift Container Platform 容器镜像 registry 中。

10.3.2. Jenkins 代理环境变量

每个 Jenkins 代理容器均可通过以下环境变量进行配置。

变量	定义	值和设置示例
JAVA_MAX_HEAP_PARAM、**CONTAINER_HEAP_PERCENT**、**JENKINS_MAX_HEAP_UPPER_BOUND_MB**	这些值控制 Jenkins JVM 的最大堆大小。如果设置了 **JAVA_MAX_heap_PARAM**，则优先使用其值。否则，最大堆大小将动态计算为容器内存限值的 **CONTAINER_HEAP_PERCENT**，可选上限为 **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB。 默认情况下，Jenkins JVM 的最大堆大小设置为容器内存限值的 50%，且无上限。	**Java_MAX_heap_PARAM** 示例设置：**-Xmx512m** **container_heap_percent** 默认：**0.5** 或 50% **Jenkins_MAX_heap_UPPER_BOUND_MB** 示例设置：**512 MiB**
JAVA_INITIAL_HEAP_PARAM、**CONTAINER_INITIAL_PERCENT**	这些值控制 Jenkins JVM 的初始堆大小。如果设置了 **JAVA_INITIAL_heap_PARAM**，则优先使用其值。否则，初始堆大小将动态计算为动态计算的最大堆大小的 **CONTAINER_INITIAL_PERCENT**。 默认情况下，JVM 设置初始堆大小。	**java_INITIAL_heap_PARAM** 示例设置：**-Xms32m** **container_INITIAL_percent** 示例设置：**0.1**或 10%
CONTAINER_CORE_LIMIT	如果设置，请将用于调整内部 JVM 线程数的内核数指定为整数。	示例设置：**2**
JAVA_TOOL_OPTIONS	指定应用于该容器中运行的所有 JVM 的选项。不建议覆盖该值。	默认：**-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**

变量	定义	值和设置示例
JAVA_GC_OPTS	指定 Jenkins JVM 垃圾回收参数。不建议覆盖该值。	默认：**-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**
JENKINS_JAVA_OVERRIDES	指定适用于 Jenkins JVM 的附加选项。这些选项附加至所有其他选项中，包括上面的 Java 选项，必要时可用于覆盖其中任何一个选项。用空格分开各个附加选项；如有任意选项包含空格字符，请使用反斜杠转义。	示例设置：**-Dfoo -Dbar; -Dfoo=first\ value -Dbar=second\ value**

10.3.3. Jenkins 代理内存要求

所有 Jenkins 代理均使用 JVM 来托管 Jenkins JNLP 代理和运行任何 Java 应用程序，如 **javac**、Maven 或 Gradle。

默认情况下，Jenkins JNLP 代理 JVM 会将容器内存限值的 50% 用于其堆。该值可通过 **CONTAINER_HEAP_PERCENT** 环境变量修改，还可设置上限或整个覆盖。

默认情况下，Jenkins 代理容器中运行的其它进程（如 shell 脚本或从管道运行的 **oc** 命令）在不引发 OOM 终止的情况下，所用内存均不得超过剩余的 50% 内存限值。

默认情况下，Jenkins 代理容器中运行的每个其他 JVM 进程最多可将 25% 的容器内存限值用于其堆。对于很多构建工作负载，可能还需调整此限值。

10.3.4. Jenkins 代理 Gradle 构建

在 OpenShift Container Platform 上的 Jenkins 代理中托管 Gradle 构建会出现其他复杂情况，因为除了 Jenkins JNLP 代理和 Gradle JVM 外，Gradle 还会生成第三个 JVM 来运行测试（若已指定）。

建议将以下设置作为起始点，在 OpenShift Container Platform 上内存受限的 Jenkins 代理中运行 Gradle 构建。您还可按需修改这些设置。

- 通过将 **org.gradle.daemon=false** 添加到 **gradle.properties** 文件中来确保禁用长期 Gradle 守护进程。
- 通过确保 **gradle.properties** 文件中未设置 **org.gradle.parallel=true** 且 **--parallel** 未设置为命令行参数来禁用并行构建执行。
- 要防止 Java 编译超出进程范围，请在 **build.gradle** 文件中设置 **java { options.fork = false }**。
- 通过确保在 **build.gradle** 文件中设置 **test { maxParallelForks = 1 }** 来禁用多个附加测试进程。
- 使用 **GRADLE_OPTS**、**JAVA_OPTS** 或 **JAVA_TOOL_OPTIONS** 环境变量覆盖 Gradle JVM 内存参数。

- 通过在 **build.gradle** 中定义 **maxHeapSize** 和 **jvmArgs** 设置，或通过 **-Dorg.gradle.jvmargs** 命令行参数来为任何 Gradle 测试 JVM 设置最大堆大小和 JVM 参数。

10.3.5. Jenkins 代理 pod 保留

构建完成或停止后，默认删除 Jenkins 代理 pod，也称 slave pod。该行为可通过设置 Kubernetes 插件的*Pod 保留*设置来修改。Pod 保留可针对所有 Jenkins 构建设置，并覆盖每个 pod 模板。支持以下行为：

- **Always** 保留构建 pod，不受构建结果的限制。
- **Default** 使用插件值（仅限 pod 模板）。
- **Never** 始终删除 pod。
- **On Failure** 如果构建过程中失败，则保留 pod。

您可覆盖管道 Jenkinsfile 中的 pod 保留：

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(), (1)
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

(1) **podRetention** 允许的值为 **never()**、**onFailure()**、**always()** 和 **default()**。

警告

保留的 Pod 可能会根据资源配额继续运行和计数。