OpenShift Container Platform 4.16

Jenkins

Jenkins

Last Updated: 2024-07-02

OpenShift Container Platform 4.16 Jenkins

Jenkins

法律通告

摘要

用于 OpenShift Container Platform 的 Jenkins

目录

第 1 章 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：

```
$ podman pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

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镜像流。然后，OpenShift Container Platform 资源便可引用镜像流。

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镜像流。

1.1. 配置和自定义

您可采用两种方式管理 Jenkins 身份验证：

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

1.1.1. 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 角色规定了用户拥有的特定 Jenkins 权限。默认使用的角色是预定义的 **admin**、**edit** 和 **view**。登录插件对 Jenkins 正在其中运行的项目或命名空间中的那些角色执行自身 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** 角色以及这些角色在 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** 的配置映射。

如果该插件找到并可以在该配置映射中读取，您可以定义到 Jenkins 权限映射的角色。具体来说：

- 登录插件将配置映射中的键值对视为 Jenkins 权限到 OpenShift Container Platform 角色映射。
- 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 角色添加 **Overall Jenkins Administer** 权限，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到的 OpenShift Container Platform 角色的列表，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** 角色以及您创建的新 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 Container Platform Login 插件轮询 OpenShift Container Platform API 服务器以获取权限，并使用从 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。

1.1.2. 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

- 输入以下命令来创建使用标准 Jenkins 身份验证的 Jenkins 应用程序：

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    ocp-tools-4/jenkins-rhel8
```

1.2. 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 Container Platform 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**
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**。
OPENSHIFT_PERMISSIONS_POLL_INTERVAL	指定 OpenShift Container Platform 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（PV）运行此镜像时，从镜像到 PV 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（PVC）时会分配 PV。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**

变量	定义	值和设置示例
OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS	当使用 Jenkins 配置目录的 OpenShift Container Platform PV 运行此镜像时，从镜像到 PV 的插件传输仅在镜像首次启动时执行，因为在创建 PVC 时会分配 PV。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 PVC 运行此镜像时，该环境变量允许在严重错误发生时，严重错误日志文件保留。严重错误文件保存在：**/var/lib/jenkins/logs**。	默认：**false**
AGENT_BASE_IMAGE	设置此值将覆盖使用此镜像提供的 Kubernetes 插件 pod 模板中的 **jnlp** 容器的镜像。否则，会使用 **openshift** 命名空间中的 **jenkins-agent-base-rhel8:latest** 镜像流标签中的镜像。	默认：**image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest**
JAVA_BUILDER_IMAGE	设置此值将覆盖使用此镜像提供的 **java-builder** 示例 Kubernetes 插件 Pod 模板中用于 **java-builder** 容器的镜像。否则，会使用 **openshift** 命名空间中的 **java:latest** 镜像流标签的镜像。	默认：**image-registry.openshift-image-registry.svc:5000/openshift/java:latest**
JAVA_FIPS_OPTIONS	设置此值控制 JVM 在 FIPS 节点上运行的运行时如何运行。如需更多信息，请参阅在 FIPS 模式中配置红帽构建的 OpenJDK 11。	默认：**-Dcom.redhat.fips=false**

1.3. 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. 输入以下命令识别具有适当权限访问 Jenkins 必须访问的项目的服务帐户的 secret：

```
$ oc describe serviceaccount jenkins
```

输出示例

```
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. 输入以下命令从 secret 检索令牌：

```
$ oc describe secret <secret name from above>
```

输出示例

```
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。

1.4. 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** 是 Jenkins 存储配置文件的数据目录，包括任务定义。

1.5. 通过 SOURCE-TO-IMAGE 自定义 JENKINS 镜像

要自定义官方 OpenShift Container Platform Jenkins 镜像，您可以使用该镜像作为 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**。

在 OpenShift Container Platform 中自定义 Jenkins 镜像的构建配置示例

```
apiVersion: build.openshift.io/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。

1.6. 配置 JENKINS KUBERNETES 插件

OpenShift Jenkins 镜像包含预安装的用于 Jenkins 的 Kubernetes 插件，以便使用 Kubernetes 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 提供了一个适合用作 Jenkins 代理的 OpenShift Agent Base 镜像。

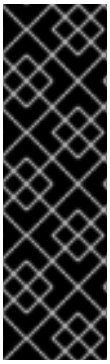
重要

OpenShift Container Platform 4.11 将 OpenShift Jenkins 和 OpenShift Agent Base 镜像移到 **registry.redhat.io** 的 **ocp-tools-4** 仓库中，以便红帽可以在 OpenShift Container Platform 生命周期外生成和更新镜像。在以前的版本中，这些镜像位于 OpenShift Container Platform 安装有效负载以及 **registry.redhat.io** 的 **openshift4** 存储库中。

OpenShift Jenkins Maven 和 NodeJS Agent 镜像已从 OpenShift Container Platform 4.11 有效负载中删除。红帽不再生成这些镜像，它们不能从 **registry.redhat.io** 的 **ocp-tools-4** 存储库中提供。根据 OpenShift Container Platform 生命周期政策，红帽会维护这些镜像的 4.10 及更早的版本，适用于任何重要的程序错误修复或安全 CVE。

如需更多信息，请参阅以下"添加资源"部分中的"重要更改 OpenShift Jenkins 镜像"链接。

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。该配置包含每个镜像的标签，您可以在 **Restrict where this project can run** 设置下应用到任何 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。



重要

在 OpenShift Container Platform 4.10 及更新的版本中，使用 Kubernetes 插件运行 Jenkins 代理的建议模式是使用带有 **jnlp** 和 **sidecar** 容器的 pod 模板。**jnlp** 容器使用 OpenShift Container Platform Jenkins Base 代理镜像来简化为构建启动一个单独的 pod。**sidecar** 容器镜像具有在启动的独立 pod 中的特定语言构建所需的工具。Red Hat Container Catalog 的许多容器镜像在 **openshift** 命名空间中的示例镜像流中引用。OpenShift Container Platform Jenkins 镜像有一个名为 **java-build** 的 pod 模板，它带有演示此方法的 sidecar 容器。此 pod 模板使用由 **openshift** 命名空间中的 **java** 镜像流提供的最新 Java 版本。

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。

在 OpenShift Container Platform 同步插件中，在 Jenkins 启动时，Jenkins 镜像会在其运行的项目中搜索，或者插件配置中列出的项目，用于以下项目：

- 将 **role** 标签设置为 **jenkins-agent** 的镜像流。
- 将 **role** 注解设置为 **jenkins-agent** 的镜像流标签。
- 将 **role** 标签设置为 **jenkins-agent** 的配置映射。

当 Jenkins 镜像找到具有适当标签的镜像流或带有适当注解的镜像流标签时，它会生成对应的 Kubernetes 插件配置。这样，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。

镜像流或镜像流标签的名称和镜像引用映射到 Kubernetes 插件 pod 模板中的名称和镜像字段。您可以通过使用 **agent-label** 键在镜像流或镜像流标签对象上设置注解来控制 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 控制台并更改 pod 模板配置。如果您在创建 pod 模板后这样做，且 OpenShift Container Platform Sync 插件检测到与镜像流或镜像流标签关联的镜像已更改，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。

如果您的配置需要更复杂的配置需求，请考虑配置映射方法。

当找到具有适当标签的配置映射时，Jenkins 镜像会假定配置映射的键值数据有效负载中的任何值都与 Jenkins 和 Kubernetes 插件 pod 模板的配置格式一致。配置映射与镜像流和镜像流标签的一个关键优点是，您可以控制所有 Kubernetes 插件 pod 模板参数。

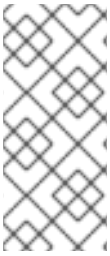
jenkins-agent 的配置映射示例

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
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>
```

以下示例显示了引用 **openshift** 命名空间中镜像流的两个容器。一个容器处理作为 Jenkins 代理启动 Pod 的 JNLP 合同。其他容器使用带有工具的镜像，以特定的编码语言构建代码：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
```

```
data:
  template2: |-
        <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
          <inheritFrom></inheritFrom>
          <name>template2</name>
          <instanceCap>2147483647</instanceCap>
          <idleMinutes>0</idleMinutes>
          <label>template2</label>
          <serviceAccount>jenkins</serviceAccount>
          <nodeSelector></nodeSelector>
          <volumes/>
          <containers>
            <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
              <name>jnlp</name>
              <image>image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-
rhel8:latest</image>
              <privileged>false</privileged>
              <alwaysPullImage>true</alwaysPullImage>
              <workingDir>/home/jenkins/agent</workingDir>
              <command></command>
              <args>\$(JENKINS_SECRET) \$(JENKINS_NAME)</args>
              <ttyEnabled>false</ttyEnabled>
              <resourceRequestCpu></resourceRequestCpu>
              <resourceRequestMemory></resourceRequestMemory>
              <resourceLimitCpu></resourceLimitCpu>
              <resourceLimitMemory></resourceLimitMemory>
              <envVars/>
            </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
            <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
              <name>java</name>
              <image>image-registry.openshift-image-registry.svc:5000/openshift/java:latest</image>
              <privileged>false</privileged>
              <alwaysPullImage>true</alwaysPullImage>
              <workingDir>/home/jenkins/agent</workingDir>
              <command>cat</command>
              <args></args>
              <ttyEnabled>true</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

不要登录到 Jenkins 控制台并更改 pod 模板配置。如果您在创建 pod 模板后这样做，且 OpenShift Container Platform Sync 插件检测到与镜像流或镜像流标签关联的镜像已更改，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。

如果您的配置需要更复杂的配置需求，请考虑配置映射方法。

安装后，OpenShift Container Platform Sync 插件会监控 OpenShift Container Platform 的 API 服务器，以获取对镜像流、镜像流标签和配置映射的更新，并调整 Kubernetes 插件的配置。

适用以下规则：

- 从配置映射、镜像流或镜像流标签中删除标签或注解，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。
- 如果您创建适当标记或注解的 **ConfigMap**、**ImageStream** 或 **ImageStreamTag** 对象，或者在初始创建后添加标签，这会导致在 Kubernetes-plugin 配置中创建 **PodTemplate**。
- 对于按照配置映射表单的 **PodTemplate**，对 **PodTemplate** 的配置映射数据的更改将应用到 Kubernetes 插件配置中的 **PodTemplate** 设置。更改还会覆盖通过 Jenkins UI 在配置映射更改之间对 **PodTemplate** 所做的任何更改。

要将容器镜像用作 Jenkins 代理，镜像必须运行该代理作为入口点。如需了解更多详细信息，请参阅官方 Jenkins 文档。

其他资源

- OpenShift Jenkins 镜像的重要变化

1.7. 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 Container Platform 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。

1.8. 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。模板在不同的存储策略中会有所不同，存储策略决定 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，所有数据都会丢失。该模板仅适用于开发或测试。
- **jenkins-persistent** 使用持久性卷（PV）存储。数据不会因 Pod 重启而丢失。

要使用 PV 存储，集群管理员必须在 OpenShift Container Platform 部署中定义一个 PV 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

- 使用以下任一方法新建 Jenkins 应用程序：
  - 一个 PV：

```
$ 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
```

使用这两个模板，您可以在它们上运行 **oc describe**，以查看可用于覆盖的所有参数。

例如：

```
$ oc describe jenkins-ephemeral
```

1.9. 使用 JENKINS KUBERNETES 插件

在以下示例中，**openshift-jee-sample BuildConfig** 对象会导致 Jenkins Maven 代理 pod 动态置备。pod 会克隆一些 Java 源代码，构建一个 WAR 文件，并导致第二个 **BuildConfig openshift-jee-sample-docker** 运行。第二个 **BuildConfig** 会将新的 WAR 文件分层到一个容器镜像中。

重要

OpenShift Container Platform 4.11 从其有效负载中删除 OpenShift Jenkins Maven 和 NodeJS Agent 镜像。红帽不再生成这些镜像，它们不能从 **registry.redhat.io** 的 **ocp-tools-4** 存储库中提供。根据 OpenShift Container Platform 生命周期政策，红帽会维护这些镜像的 4.10 及更早的版本，适用于任何重要的程序错误修复或安全 CVE。

如需更多信息，请参阅以下"添加资源"部分中的"重要更改 OpenShift Jenkins 镜像"链接。

使用 Jenkins Kubernetes 插件的 BuildConfig 示例

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: image.openshift.io/v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: build.openshift.io/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: build.openshift.io/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：

使用 Jenkins Kubernetes 插件的 BuildConfig 示例，指定内存限制和环境变量

```
kind: BuildConfig
apiVersion: build.openshift.io/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 模板可以从现有 pod 模板继承其配置。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. 节点片段引用定义的 pod 模板的名称。

构建完成后会默认删除 pod。可以使用插件或在 Jenkinsfile 管道中修改此行为。

上游 Jenkins 最近引入了一个 YAML 声明格式，用于定义带有您的管道的 **podTemplate** 管道 DSL。一个这种格式的示例，它使用 OpenShift Container Platform Jenkins 镜像中定义的 **java-builder** pod 模板示例：

```
def nodeLabel = 'java-buidler'

pipeline {
  agent {
    kubernetes {
      cloud 'openshift'
      label nodeLabel
      yaml """
apiVersion: v1
kind: Pod
metadata:
  labels:
    worker: ${nodeLabel}
spec:
  containers:
  - name: jnlp
    image: image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest
    args: ['\$(JENKINS_SECRET)', '\$(JENKINS_NAME)']
  - name: java
    image: image-registry.openshift-image-registry.svc:5000/openshift/java:latest
    command:
    - cat
    tty: true
"""
    }
  }

  options {
    timeout(time: 20, unit: 'MINUTES')
  }

  stages {
    stage('Build App') {
      steps {
        container("java") {
          sh "mvn --version"
        }
      }
    }
  }
}
```

其他资源

- OpenShift Jenkins 镜像的重要变化

1.10. 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。

建议您为 Jenkins Kubernetes 插件创建的代理容器指定内存请求和限制值。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。

1.11. 其他资源

- 有关 Red Hat Universal Base Images (UBI)的更多信息，请参阅基础镜像选项 。
- OpenShift Jenkins 镜像的重要变化

第 2 章 JENKINS 代理

OpenShift Container Platform 提供了一个基础镜像，用作 Jenkins 代理。

Jenkins 代理的 Base 镜像执行以下操作：

- 拉取(pull)所需工具、无头 Java、Jenkins JNLP 客户端以及有用的工具，包括 **git**、**tar**、**zip** 和 **nss** 等。
- 建立 JNLP 代理作为入口点。
- 包括 **oc** 客户端工具，用于从 Jenkins 任务调用命令行操作。
- 为 Red Hat Enterprise Linux(RHEL)和 **localdev** 镜像提供 Dockerfile。

重要

使用适合您的 OpenShift Container Platform 发行版本的代理镜像版本。嵌入了与 OpenShift Container Platform 版本不兼容的 **oc** 客户端版本可能会导致意外行为。

OpenShift Container Platform Jenkins 镜像还定义了以下示例 **java-builder** pod 模板，以说明如何将代理镜像用于 Jenkins Kubernetes 插件。

java-builder pod 模板采用两个容器：

- 使用 OpenShift Container Platform Base 代理镜像的 **jnlp** 容器，并处理 JNLP 合同来启动和停止 Jenkins 代理。
- 一个是 **java** 容器，它使用 **java** OpenShift Container Platform Sample ImageStream，其包含各种 Java 二进制文件，包括 Maven 二进制文件 **mvn**，用于构建代码。

2.1. 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/ocp-tools-4/jenkins-rhel8:<image_tag>
```

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-agent-base-rhel8:<image_tag>
```

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。

2.2. 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**
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**

变量	定义	值和设置示例
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**
USE_JAVA_VERSION	指定用来在容器中运行代理的 Java 版本版本。容器基础镜像安装了两个 java 版本： **java-11** 和 **java-1.8.0**。如果扩展容器基础镜像，您可以使用其关联的后缀指定 java 的任何替代版本。	默认值为 **java-11**。 示例设置： **java-1.8.0**

2.3. 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。

2.4. 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。

2.5. JENKINS 代理 POD 保留

构建完成后或停止后会默认删除 Jenkins 代理 pod。此行为可通过 Kubernetes 插件 pod 保留设置来更改。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，即仅限 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。

第 3 章 从 JENKINS 迁移到 OPENSHIFT PIPELINES 或 TEKTON

您可以将 CI/CD 工作流从 Jenkins 迁移到 Red Hat OpenShift Pipelines，这是基于 Tekton 项目的云原生 CI/CD 体验。

3.1. JENKINS 和 OPENSHIFT PIPELINES 概念的比较

您可以查看并比较 Jenkins 和 OpenShift Pipelines 中使用的以下等效术语。

3.1.1. Jenkins 术语

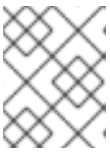
Jenkins 提供声明式和脚本化管道，它们可以使用共享库和插件扩展。Jenkins 中的一些基本术语如下：

- **管道（Pipeline）**：利用 Groovy 语法自动化构建、测试和部署应用的整个流程。
- **节点（Node）**：能够编配或执行脚本化管道的计算机。
- **阶段（Stage）**：在管道中执行的概念上不同的任务子集。插件或用户界面通常使用此块来显示任务的状态或进度。
- **步骤（Step）**：一项任务指定要执行的确切操作，可使用命令或脚本。

3.1.2. OpenShift Pipelines 术语

OpenShift Pipelines 使用 YAML 语法用于声明管道，由任务组成。OpenShift Pipelines 中的一些基本术语如下：

- **频道（Pipeline）**：一组串行或并行（或两者）任务。
- **任务（Task）**：作为命令、二进制文件或脚本的步骤序列。
- **管道运行（PipelineRun）**：执行包含一个或多个任务的管道。
- **任务运行（TaskRun）**：通过一个或多个步骤执行任务。



注意

您可以使用一组输入（如参数和工作区）启动 PipelineRun 或 TaskRun，执行会产生一组输出和工件。

- **workspace**：在 OpenShift Pipelines 中，工作区是概念块，用于以下目的：
  - 存储输入、输出和构建工件.
  - 在任务间共享数据的通用空间.
  - 在 secret 中保存的凭证挂载点、配置映射中保存的配置以及机构共享的通用工具。

注意

在 Jenkins 中，没有 OpenShift Pipelines 工作区直接对应的工作区。您可以将控制节点视为工作区，因为它存储克隆的代码存储库、构建历史记录和工件。当作业分配给其他节点时，克隆的代码和生成的工件会存储在该节点中，但控制节点维护构建历史记录。

3.1.3. 概念映射

Jenkins 和 OpenShift Pipelines 的构建块并不等同，特定比较并不提供技术准确的映射。Jenkins 和 OpenShift Pipelines 中的以下术语和概念一般关联：

表 3.1. Jenkins 和 OpenShift Pipelines - 基本比较

Jenkins	OpenShift Pipelines
Pipeline	Pipeline 和 PipelineRun
Stage	任务
Step	一个任务中的一个步骤

3.2. 将示例管道从 JENKINS 迁移到 OPENSHIFT PIPELINES

您可以使用以下等同的示例来帮助将构建、测试和部署管道从 Jenkins 迁移到 OpenShift Pipelines。

3.2.1. Jenkins 管道

考虑在 Groovy 中编写的 Jenkins 管道，用于构建、测试和部署：

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'make'
      }
    }
    stage('Test'){
      steps {
        sh 'make check'
        junit 'reports/**/*.xml'
      }
    }
    stage('Deploy') {
      steps {
        sh 'make publish'
      }
    }
  }
}
```

3.2.2. OpenShift Pipelines 管道

要在 OpenShift Pipelines 中创建相当于前面的 Jenkins 管道的管道，您需要创建以下三个任务：

build 任务 YAML 定义文件示例

```
apiVersion: tekton.dev/v1beta1
```

```
kind: Task
metadata:
  name: myproject-build
spec:
  workspaces:
  - name: source
  steps:
  - image: my-ci-image
    command: ["make"]
    workingDir: $(workspaces.source.path)
```

test 任务 YAML 定义文件示例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-test
spec:
  workspaces:
  - name: source
  steps:
  - image: my-ci-image
    command: ["make check"]
    workingDir: $(workspaces.source.path)
  - image: junit-report-image
    script: |
      #!/usr/bin/env bash
      junit-report reports/**/*.xml
    workingDir: $(workspaces.source.path)
```

deploy 任务 YAML 定义文件示例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myprojectd-deploy
spec:
  workspaces:
  - name: source
  steps:
  - image: my-deploy-image
    command: ["make deploy"]
    workingDir: $(workspaces.source.path)
```

您可以按顺序组合三个任务以在 OpenShift Pipelines 中组成管道：

示例：用于构建、测试和部署的 OpenShift Pipelines 管道

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: myproject-pipeline
spec:
  workspaces:
```

```
  - name: shared-dir
 tasks:
 - name: build
   taskRef:
     name: myproject-build
   workspaces:
   - name: source
     workspace: shared-dir
 - name: test
   taskRef:
     name: myproject-test
   workspaces:
   - name: source
     workspace: shared-dir
 - name: deploy
   taskRef:
     name: myproject-deploy
   workspaces:
   - name: source
     workspace: shared-dir
```

3.3. 从 JENKINS 插件迁移到 TEKTON HUB 任务

您可以使用 插件 来扩展 Jenkins 的功能。要在 OpenShift Pipelines 中实现类似的可扩展性，请使用 Tekton Hub 中提供的任何任务。

例如，考虑 Tekton Hub 中的 git-clone 任务，它对应于 Jenkins 的 git 插件。

示例：Tekton Hub 中的 git-clone 任务

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
 name: demo-pipeline
spec:
 params:
   - name: repo_url
   - name: revision
 workspaces:
   - name: source
 tasks:
   - name: fetch-from-git
     taskRef:
       name: git-clone
     params:
       - name: url
         value: $(params.repo_url)
       - name: revision
         value: $(params.revision)
     workspaces:
     - name: output
       workspace: source
```

3.4. 使用自定义任务和脚本扩展 OPENSHIFT PIPELINES 功能

在 OpenShift Pipelines 中，如果您在 Tekton Hub 中找不到正确的任务，或需要对任务进行更大的控制，您可以创建自定义任务和脚本来扩展 OpenShift Pipelines 的功能。

示例：运行 maven test 命令的自定义任务

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-test
spec:
  workspaces:
  - name: source
  steps:
  - image: my-maven-image
    command: ["mvn test"]
    workingDir: $(workspaces.source.path)
```

示例：通过提供自定义 shell 脚本的路径

```
...
steps:
  image: ubuntu
  script: |
      #!/usr/bin/env bash
      /workspace/my-script.sh
...
```

示例：在 YAML 文件中写入自定义 Python 脚本

```
...
steps:
  image: python
  script: |
      #!/usr/bin/env python3
      print(“hello from python!”)
...
```

3.5. JENKINS 和 OPENSHIFT PIPELINES 执行模型的比较

Jenkins 和 OpenShift Pipelines 提供了类似的功能，但在架构和执行方面有所不同。

表 3.2. Jenkins 和 OpenShift Pipelines 中的执行模型比较

Jenkins	OpenShift Pipelines
Jenkins 有一个控制器节点。Jenkins 集中运行管道和步骤，或编排其他节点上运行的作业。	OpenShift Pipelines 是无服务器且分布式的，它没有执行中央依赖项。
容器由 Jenkins 控制器节点通过管道启动。	OpenShift Pipelines 采用"容器先行"方法，其中的每个步骤都作为 pod 中的容器运行（等同于 Jenkins 中的节点）。

Jenkins	OpenShift Pipelines
使用插件可实现可扩展性。	使用 Tekton Hub 中的任务或创建自定义任务和脚本来实现可扩展性。

3.6. 常见使用案例示例

Jenkins 和 OpenShift Pipelines 都为常见 CI/CD 用例提供功能，例如：

- 使用 Apache Maven 编译、构建和部署镜像
- 使用插件扩展核心功能
- 重新使用可共享库和自定义脚本

3.6.1. 在 Jenkins 和 OpenShift Pipelines 中运行 Maven 管道

您可以在 Jenkins 和 OpenShift Pipelines 工作流中使用 Maven 来编译、构建和部署镜像。要将现有的 Jenkins 工作流映射到 OpenShift Pipelines，请考虑以下示例：

示例：完成并构建镜像，并使用 Jenkins 中的 Maven 部署到 OpenShift

```
#!/usr/bin/groovy
node('maven') {
   stage 'Checkout'
   checkout scm

   stage 'Build'
   sh 'cd helloworld && mvn clean'
   sh 'cd helloworld && mvn compile'

   stage 'Run Unit Tests'
   sh 'cd helloworld && mvn test'

   stage 'Package'
   sh 'cd helloworld && mvn package'

   stage 'Archive artifact'
   sh 'mkdir -p artifacts/deployments && cp helloworld/target/*.war artifacts/deployments'
   archive 'helloworld/target/*.war'

   stage 'Create Image'
   sh 'oc login https://kubernetes.default -u admin -p admin --insecure-skip-tls-verify=true'
   sh 'oc new-project helloworldproject'
   sh 'oc project helloworldproject'
   sh 'oc process -f helloworld/jboss-eap70-binary-build.json | oc create -f -'
   sh 'oc start-build eap-helloworld-app --from-dir=artifacts/'

   stage 'Deploy'
   sh 'oc new-app helloworld/jboss-eap70-deploy.json' }
```

示例：完成并构建镜像，并使用 OpenShift Pipelines 中的 Maven 部署到 OpenShift 中。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: kubeconfig-dir
      optional: true
  params:
    - name: repo-url
    - name: revision
    - name: context-path
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
        - name: revision
          value: $(params.revision)
    - name: mvn-build
      taskRef:
        name: maven
      runAfter:
        - fetch-repo
      workspaces:
        - name: source
          workspace: shared-workspace
        - name: maven-settings
          workspace: maven-settings
      params:
        - name: CONTEXT_DIR
          value: "$(params.context-path)"
        - name: GOALS
          value: ["-DskipTests", "clean", "compile"]
    - name: mvn-tests
      taskRef:
        name: maven
      runAfter:
        - mvn-build
      workspaces:
        - name: source
          workspace: shared-workspace
        - name: maven-settings
          workspace: maven-settings
      params:
```

```
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["test"]
  - name: mvn-package
    taskRef:
      name: maven
    runAfter:
      - mvn-tests
    workspaces:
      - name: source
        workspace: shared-workspace
      - name: maven-settings
        workspace: maven-settings
    params:
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["package"]
  - name: create-image-and-deploy
    taskRef:
      name: openshift-client
    runAfter:
      - mvn-package
    workspaces:
      - name: manifest-dir
        workspace: shared-workspace
      - name: kubeconfig-dir
        workspace: kubeconfig-dir
    params:
      - name: SCRIPT
        value: |
          cd "$(params.context-path)"
          mkdir -p ./artifacts/deployments && cp ./target/*.war ./artifacts/deployments
          oc new-project helloworldproject
          oc project helloworldproject
          oc process -f jboss-eap70-binary-build.json | oc create -f -
          oc start-build eap-helloworld-app --from-dir=artifacts/
          oc new-app jboss-eap70-deploy.json
```

3.6.2. 使用插件扩展 Jenkins 和 OpenShift Pipelines 的核心功能

Jenkins 利用了其广泛的用户群来多年开发的大量插件生态系统。您可以在 Jenkins 插件索引中搜索和浏览插件。

OpenShift Pipelines 还有许多任务由社区和企业用户开发并贡献。Tekton Hub 中提供了可重复使用的 OpenShift Pipelines 任务的公开目录。

另外，OpenShift Pipelines 包含在其核心功能中 Jenkins 生态系统的许多插件。例如，授权是 Jenkins 和 OpenShift Pipelines 中的关键功能。虽然 Jenkins 使用基于角色的访问控制插件来确保授权，OpenShift Pipelines 使用 OpenShift 的内置基于角色的访问控制系统。

3.6.3. 在 Jenkins 和 OpenShift Pipelines 中共享可重复使用的代码

Jenkins 共享库为 Jenkins 管道的各部分提供可重复使用的代码。该库在 Jenkinsfile 之间共享，以创建高度模块化的管道，而不重复代码。

虽然 OpenShift Pipelines 中没有直接等效的 Jenkins 共享库，但您可以使用 Tekton Hub 中的任务与自定义任务和脚本相结合来实现类似的工作流。

3.7. 其他资源

- 了解 OpenShift Pipelines
- 基于角色的访问控制

第 4 章 OPENSHIFT JENKINS 镜像的重要变化

OpenShift Container Platform 4.11 将 OpenShift Jenkins 和 OpenShift Agent Base 镜像移到 **registry.redhat.io** 的 **ocp-tools-4** 仓库中。它还从其有效负载中删除 OpenShift Jenkins Maven 和 NodeJS Agent 镜像：

- OpenShift Container Platform 4.11 将 OpenShift Jenkins 和 OpenShift Agent Base 镜像移到 **registry.redhat.io** 的 **ocp-tools-4** 仓库中，以便红帽可以在 OpenShift Container Platform 生命周期外生成和更新镜像。在以前的版本中，这些镜像位于 OpenShift Container Platform 安装有效负载以及 **registry.redhat.io** 的 **openshift4** 存储库中。

- OpenShift Container Platform 4.10 弃用了 OpenShift Jenkins Maven 和 NodeJS Agent 镜像。OpenShift Container Platform 4.11 从其有效负载中删除这些镜像。红帽不再生成这些镜像，它们不能从 **registry.redhat.io** 的 **ocp-tools-4** 存储库中提供。根据 OpenShift Container Platform 生命周期政策，红帽会维护这些镜像的 4.10 及更早的版本，适用于任何重要的程序错误修复或安全 CVE。

这些更改支持 OpenShift Container Platform 4.10 建议在 Jenkins Kubernetes 插件中使用多个容器 Pod 模板。

4.1. OPENSHIFT JENKINS 镜像重新定位

OpenShift Container Platform 4.11 对特定 OpenShift Jenkins 镜像的位置和可用性进行了大量更改。另外，您还可以配置何时以及如何更新这些镜像。

什么与 OpenShift Jenkins 镜像保持相同？

- Cluster Samples Operator 管理用于运行 OpenShift Jenkins 镜像的 **ImageStream** 和 **Template** 对象。

- 默认情况下，Jenkins pod 模板的 Jenkins **DeploymentConfig** 对象会在 Jenkins 镜像更改时触发重新部署。默认情况下，此镜像由 Samples Operator 有效负载中的 **ImageStream** YAML 文件中的 **openshift** 命名空间的 Jenkins 镜像流的 **jenkins:2** 镜像流标签引用。

- 如果您从 OpenShift Container Platform 4.10 及更早版本升级到 4.11，弃用的 **maven** 和 **nodejs** pod 模板仍处于默认镜像配置中。

- 如果您从 OpenShift Container Platform 4.10 及更新版本升级到 4.11，则 **jenkins-agent-maven** 和 **jenkins-agent-nodejs** 镜像流仍存在于集群中。要维护这些镜像流，请参见以下部分："**openshift** 命名空间中的 **jenkins-agent-maven** 和 **jenkins-agent-nodejs** 镜像流有什么变化？"

OpenShift Jenkins 镜像支持列表中的哪些变化？

registry.redhat.io registry 中的 **ocp-tools-4** 仓库中的每个新镜像都支持多个 OpenShift Container Platform 版本。当红帽更新其中一个新镜像时，所有版本都同时可用。当红帽更新镜像以响应安全公告时，此可用性是理想的选择。最初，这个更改适用于 OpenShift Container Platform 4.11 及更新的版本。计划此更改最终会应用到 OpenShift Container Platform 4.9 及更新的版本。

在以前的版本中，每个 Jenkins 镜像只支持一个 OpenShift Container Platform 版本，红帽可能会随时间顺序更新这些镜像。

OpenShift Jenkins 和 Jenkins Agent Base ImageStream 和 ImageStreamTag 对象增加了什么？

通过从 in-payload 镜像流移到引用非备份镜像的镜像流，OpenShift Container Platform 可以定义额外的镜像流标签。红帽创建了一系列新镜像流标签，并附带现有的 **"value": "jenkins:2"** 和 **"value": "image-**

registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest" 镜像流标签（位于 OpenShift Container Platform 4.10 及更早版本）。这些新镜像流标签处理了一些请求，以改进维护 Jenkins 相关镜像流的方式。

关于新镜像流标签：

ocp-upgrade-redeploy

要在升级 OpenShift Container Platform 时更新 Jenkins 镜像，请在 Jenkins 部署配置中使用此镜像流标签。此镜像流标签与 jenkins 镜像流现有的 **jenkins** 镜像流的 **2** 镜像流标签以及 **jenkins-agent-base-rhel8** 镜像流的 **latest** 镜像流标签对应。它仅使用特定于一个 SHA 或镜像摘要的镜像标签。当 **ocp-tools-4** 镜像更改时，如 Jenkins 安全公告，红帽工程团队会更新 Cluster Samples Operator 有效负载。

user-maintained-upgrade-redeploy

要在升级 OpenShift Container Platform 后手动重新部署 Jenkins，请在 Jenkins 部署配置中使用此镜像流标签。此镜像流标签使用至少可用的特定镜像版本指示符。重新部署 Jenkins 时，运行以下命令：**$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift**。当您发出此命令时，OpenShift Container Platform **ImageStream** 控制器可以访问 **registry.redhat.io** 镜像 registry，并将任何更新的镜像存储在 OpenShift 镜像 registry 的 Jenkins **ImageStreamTag** 对象的插槽中。否则，如果您没有运行此命令，您的 Jenkins 部署配置不会触发重新部署。

scheduled-upgrade-redeploy

要在释放后自动重新部署 Jenkins 镜像的最新版本，请在 Jenkins 部署配置中使用此镜像流标签。此镜像流标签使用 OpenShift Container Platform 镜像流控制器的定期导入镜像流标签功能，用于检查后备镜像中的更改。例如，如果镜像更改（例如由于最新的 Jenkins 安全公告），OpenShift Container Platform 会触发重新部署 Jenkins 部署配置。在以下"添加资源"中，请参阅"定期导入镜像流标签"。

openshift 命名空间中的 jenkins-agent-maven 和 jenkins-agent-nodejs 镜像流会发生什么？

OpenShift Container Platform 的 OpenShift Jenkins Maven 和 NodeJS Agent 镜像已在 4.10 中弃用，并在 4.11 中从 OpenShift Container Platform 安装有效负载中删除。它们没有在 **ocp-tools-4** 仓库中定义 alternatives。但是，您可以使用以下"Additional resources"部分中提到的"Jenkins 代理"主题描述的 sidecar 模式来解决这个问题。

但是，Cluster Samples Operator 不会删除之前版本创建的 **jenkins-agent-maven** 和 **jenkins-agent-nodejs** 镜像流，指向 **registry.redhat.io** 上对应 OpenShift Container Platform 有效负载镜像的标签。因此，您可以通过运行以下命令拉取 (pull) 镜像的更新：

```
$ oc import-image jenkins-agent-nodejs -n openshift
```

```
$ oc import-image jenkins-agent-maven -n openshift
```

4.2. 自定义 JENKINS 镜像流标签

要覆盖默认的升级行为，并控制 Jenkins 镜像的升级方式，您可以设置 Jenkins 部署配置使用的镜像流标签值。

默认升级行为是在 Jenkins 镜像是安装有效负载的一部分时存在的行为。**jenkins-rhel.json** 镜像流文件中的镜像流标签名称 **2** 和 **ocp-upgrade-redeploy** 使用 SHA 特定镜像引用。因此，当这些标签使用新的 SHA 更新时，OpenShift Container Platform 镜像更改控制器会自动从关联的模板重新部署 Jenkins 部署配置，如 **jenkins-ephemeral.json** 或 **jenkins-persistent.json**。

对于新部署，要覆盖该默认值，您可以在 **jenkins-ephemeral.json** Jenkins 模板中更改 **JENKINS_IMAGE_STREAM_TAG** 的值。例如，将 **"value": "jenkins:2"** 中的 **2** 替换为以下镜像流标签之一：

- 升级 OpenShift Container Platform 时，OCP **-upgrade-redeploy** （默认值）会更新 Jenkins 镜像。
- **user-maintained-upgrade-redeploy** 要求您在升级 OpenShift Container Platform 后运行 **$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** 来手动重新部署 Jenkins。
- **scheduled-upgrade-redeploy** 会定期检查给定的 **<image>:<tag>** 组合，以便在镜像更改时进行更改和升级。镜像更改控制器拉取更改的镜像，并重新部署由模板置备的 Jenkins 部署配置。有关此调度导入策略的更多信息，请参阅以下"Additional resources"中的"将标签添加到镜像流"。

注意

要覆盖现有部署的当前升级值，请更改与这些模板参数对应的环境变量值。

先决条件

- 您在 OpenShift Container Platform 4.16 上运行了 OpenShift Jenkins。
- 您知道部署 OpenShift Jenkins 的命名空间。

流程

- 设置镜像流标签值，将 **<namespace>** 替换为部署 OpenShift Jenkins 的命名空间，并将 **<image_stream_tag>** 替换为镜像流标签：

  Example

  ```
  $ oc patch dc jenkins -p '{"spec":{"triggers":[{"type":"ImageChange","imageChangeParams":
  {"automatic":true,"containerNames":["jenkins"],"from":
  {"kind":"ImageStreamTag","namespace":"<namespace>","name":"jenkins:
  <image_stream_tag>"}}}]}}'
  ```

  提示

  另外，要编辑 Jenkins 部署配置 YAML，请输入 **$ oc edit dc/jenkins -n <namespace>** 并更新 **value: 'jenkins:<image_stream_tag>'** 行。

4.3. 其他资源

- 向镜像流中添加标签
- 配置定期导入镜像流标签
- Jenkins 代理
- 认证的 **Jenkins** 镜像
- 认证的 **jenkins-agent-base** 镜像
- 认证的 **jenkins-agent-maven** 镜像
- 认证的 **jenkins-agent-nodejs** 镜像