Red Hat OpenShift Pipelines 1.15

创建 CI/CD 管道

在 OpenShift Pipelines 中创建并运行任务和管道入门

Last Updated: 2024-06-28

Red Hat OpenShift Pipelines 1.15 创建 CI/CD 管道

在 OpenShift Pipelines 中创建并运行任务和管道入门

法律通告

摘要

本文档提供有关在 OpenShift Pipelines 中创建和运行任务和管道的信息。

目录

第 1 章 为使用 OPENSHIFT PIPELINES 的应用程序创建 CI/CD 解决方案

使用 Red Hat OpenShift Pipelines，您可以创建一个自定义的 CI/CD 解决方案来构建、测试和部署应用程序。

要为应用程序创建一个完整的自助 CI/CD 管道，请执行以下任务：

- 创建自定义任务，或安装现有的可重复使用的任务。
- 为应用程序创建并定义交付管道。
- 使用以下方法之一提供附加到管道执行的工作区中的存储卷或文件系统：
  - 指定创建持久性卷声明的卷声明模板
  - 指定一个持久性卷声明
- 创建一个 **PipelineRun** 对象来实例化并调用管道。
- 添加触发器以捕获源仓库中的事件。

本节使用 **pipelines-tutorial** 示例来演示前面的任务。这个示例使用一个简单的应用程序，它由以下部分组成：

- 一个前端接口，**pipelines-vote-ui**，它的源代码在 **pipelines-vote-ui** Git 存储库中。
- 一个后端接口 **pipelines-vote-api**，它的源代码在 **pipelines-vote-api** Git 存储库中。
- **apply-manifests** 和 **update-deployment** 任务在 **pipelines-tutorial** Git 存储库中。

1.1. 先决条件

- 有访问 OpenShift Container Platform 集群的权限。
- 已使用在 OpenShift OperatorHub 中列出的 Red Hat OpenShift Pipelines Operator 安装了 OpenShift Pipelines。在安装后，它可用于整个集群。
- 已安装 OpenShift Pipelines CLI。
- 使用您的 GitHub ID fork 前端 **pipelines-vote-ui** 和后端 **pipelines-vote-api** Git 存储库，并具有对这些存储库的管理员访问权限。
- 可选：已克隆了 **pipelines-tutorial** Git 存储库。

1.2. 创建项目并检查管道服务帐户

流程

1. 登录您的 OpenShift Container Platform 集群：

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. 为示例应用程序创建一个项目。在本例中，创建 **pipelines-tutorial** 项目：

```
$ oc new-project pipelines-tutorial
```

注意

如果您使用其他名称创建项目，请确定使用您的项目名称更新示例中使用的资源 URL。

3. 查看 **pipeline** 服务帐户：
Red Hat OpenShift Pipelines Operator 添加并配置一个名为 **pipeline** 的服务帐户，该帐户有足够的权限来构建和推送镜像。**PipelineRun** 对象使用此服务帐户。

```
$ oc get serviceaccount pipeline
```

1.3. 创建管道任务

流程

1. 从 **pipelines-tutorial** 存储库安装 **apply-manifests** 和 **update-deployment** 任务资源，其中包含可为管道重复使用的任务列表：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/01_pipeline/02_update_deployment_task.yaml
```

2. 使用 **tkn task list** 命令列出您创建的任务：

```
$ tkn task list
```

输出会确认创建了 **apply-manifests** 和 **update-deployment** 任务：

```
NAME                DESCRIPTION   AGE
apply-manifests                   1 minute ago
update-deployment                 48 seconds ago
```

3. 使用 **tkn clustertasks list** 命令列出由 Operator 安装的额外集群任务，如 **buildah** 和 **s2i-python**：

注意

要在受限环境中使用 **buildah** 集群任务，您必须确保 Dockerfile 使用内部镜像流作为基础镜像。

```
$ tkn clustertasks list
```

输出列出了 Operator 安装的 **ClusterTask** 资源：

```
NAME                DESCRIPTION   AGE
buildah                           1 day ago
git-clone                         1 day ago
```

```
s2i-python                  1 day ago
tkn                         1 day ago
```

重要

在 Red Hat OpenShift Pipelines 1.10 中，**ClusterTask** 功能已弃用，计划在以后的发行版本中删除。

其他资源

- 管理未指定版本的和版本化的集群任务

1.4. 组装管道

管道（pipeline）代表一个 CI/CD 流，由要执行的任务定义。它被设计为在多个应用程序和环境中通用且可重复使用。

管道指定任务如何使用 **from** 和 **runAfter** 参数相互交互以及它们执行的顺序。它使用 **workspaces** 字段指定管道中每个任务在执行过程中所需的一个或多个卷。

在本小节中，您将创建一个管道，从 GitHub 获取应用程序的源代码，然后在 OpenShift Container Platform 上构建和部署应用程序。

管道为后端应用程序 **pipelines-vote-api** 和前端应用程序 **pipelines-vote-ui** 执行以下任务：

- 通过引用 **git-url** 和 **git-revision** 参数，从 Git 存储库中克隆应用程序的源代码。
- 使用 **openshift-pipelines** 命名空间中提供的 **buildah** 任务构建容器镜像。
- 通过引用 **image** 参数将镜像推送到 OpenShift 镜像 registry。
- 通过使用 **apply-manifests** 和 **update-deployment** 任务在 OpenShift Container Platform 上部署新镜像。

流程

1. 复制以下管道 YAML 文件示例内容并保存：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
  - name: shared-workspace
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
```

```
    default: "pipelines-1.15"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: URL
      value: $(params.git-url)
    - name: SUBDIRECTORY
      value: ""
    - name: DELETE_EXISTING
      value: "true"
    - name: REVISION
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: buildah
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
    params:
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
```

```
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

Pipeline 定义提取 Git 源存储库和镜像 registry 的特定内容。当一个管道被触发并执行时，这些详细信息会作为 **params** 添加。

2. 创建管道：

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

或者，还可以从 Git 存储库直接执行 YAML 文件：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/01_pipeline/04_pipeline.yaml
```

3. 使用 **tkn pipeline list** 命令来验证管道是否已添加到应用程序中：

```
$ tkn pipeline list
```

检查输出来验证创建了 **build-and-deploy** pipeline：

```
NAME              AGE            LAST RUN   STARTED   DURATION   STATUS
build-and-deploy  1 minute ago   ---        ---       ---        ---
```

1.5. 镜像以在受限环境中运行管道

要在断开连接的集群或受限环境中置备的集群中运行 OpenShift Pipelines，请确保为受限网络配置了 Samples Operator，或者集群管理员创建了带有镜像 registry 的集群。

以下流程使用 **pipelines-tutorial** 示例，使用带有镜像 registry 的集群在受限环境中为应用程序创建管道。为确保 **pipelines-tutorial** 示例在受限环境中工作，您必须为前端接口（**pipelines-vote-ui**）后端接口（**pipelines-vote-api**）和 **cli** 从 mirror registry 中镜像相应的构建器镜像。

流程

1. 为前端接口 **pipelines-vote-ui** 从 mirror registry 中镜像构建器镜像。

   a. 验证所需镜像标签没有导入：

```
$ oc describe imagestream python -n openshift
```

输出示例

```
Name:   python
Namespace:  openshift
[...]

3.8-ubi9 (latest)
```

```
  tagged from registry.redhat.io/ubi9/python-38:latest
    prefer registry pullthrough when referencing this tag

  Build and run Python 3.8 applications on UBI 8. For more information about using this
builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-
python-container/blob/master/3.8/README.md.
  Tags: builder, python
  Supports: python:3.8, python
  Example Repo: https://github.com/sclorg/django-ex.git

[...]
```

b. 将支持的镜像标签镜像到私有 registry：

```
$ oc image mirror registry.redhat.io/ubi9/python-39:latest <mirror-registry>:
<port>/ubi9/python-39
```

c. 导入镜像：

```
$ oc tag <mirror-registry>:<port>/ubi9/python-39 python:latest --scheduled -n openshift
```

您必须定期重新导入镜像。**--scheduled** 标志启用镜像自动重新导入。

d. 验证带有指定标签的镜像已被导入：

```
$ oc describe imagestream python -n openshift
```

输出示例

```
Name:   python
Namespace:  openshift
[...]

latest
  updates automatically from registry  <mirror-registry>:<port>/ubi9/python-39

  *  <mirror-registry>:<port>/ubi9/python-39@sha256:3ee...

[...]
```

2. 为后端接口 **pipelines-vote-api** 从 mirror registry 中镜像构建器镜像。

a. 验证所需镜像标签没有导入：

```
$ oc describe imagestream golang -n openshift
```

输出示例

```
Name:   golang
Namespace:  openshift
[...]

1.14.7-ubi8 (latest)
  tagged from registry.redhat.io/ubi8/go-toolset:1.14.7
```

```
    prefer registry pullthrough when referencing this tag

  Build and run Go applications on UBI 8. For more information about using this builder
image, including OpenShift considerations, see https://github.com/sclorg/golang-
container/blob/master/README.md.
  Tags: builder, golang, go
  Supports: golang
  Example Repo: https://github.com/sclorg/golang-ex.git

[...]
```

b. 将支持的镜像标签镜像到私有 registry：

```
$ oc image mirror registry.redhat.io/ubi9/go-toolset:latest <mirror-registry>:
<port>/ubi9/go-toolset
```

c. 导入镜像：

```
$ oc tag <mirror-registry>:<port>/ubi9/go-toolset golang:latest --scheduled -n openshift
```

您必须定期重新导入镜像。**--scheduled** 标志启用镜像自动重新导入。

d. 验证带有指定标签的镜像已被导入：

```
$ oc describe imagestream golang -n openshift
```

输出示例

```
Name:   golang
Namespace:  openshift
[...]

latest
  updates automatically from registry <mirror-registry>:<port>/ubi9/go-toolset

  * <mirror-registry>:<port>/ubi9/go-
toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421
b37

[...]
```

3. 从 **cli** 的镜像 registry 中镜像构建器镜像。

a. 验证所需镜像标签没有导入：

```
$ oc describe imagestream cli -n openshift
```

输出示例

```
Name:                cli
Namespace:           openshift
[...]

latest
```

```
  updates automatically from registry quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


  * quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


[...]
```

b. 将支持的镜像标签镜像到私有 registry：

```
$ oc image mirror quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551
<mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev:latest
```

c. 导入镜像：

```
$ oc tag <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev cli:latest --
scheduled -n openshift
```

您必须定期重新导入镜像。**--scheduled** 标志启用镜像自动重新导入。

d. 验证带有指定标签的镜像已被导入：

```
$ oc describe imagestream cli -n openshift
```

输出示例

```
Name:                   cli
Namespace:              openshift
[...]

latest
  updates automatically from registry <mirror-registry>:<port>/openshift-release-dev/ocp-
v4.0-art-dev

  * <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


[...]
```

其他资源

- 为受限集群配置 Samples Operator
- 创建带有镜像 registry 的集群

1.6. 运行管道

PipelineRun 资源启动管道，并将其与 Git 和用于特定调用的镜像资源相关联。它为管道中的每个任务自动创建并启动 **TaskRun** 资源。

流程

1. 启动后端应用程序的管道：

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.15/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-api \
    -p git-url=https://github.com/openshift/pipelines-vote-api.git \
    -p IMAGE='image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-api' \
    --use-param-defaults
```

上一命令使用卷声明模板，该模板为管道执行创建持久性卷声明。

2. 要跟踪管道运行的进度，请输入以下命令：

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

上述命令中的 <pipelinerun_id> 是上一命令输出返回的 **PipelineRun** 的 ID。

3. 启动前端应用程序的管道：

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.15/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-ui \
    -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
    -p IMAGE='image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-ui' \
    --use-param-defaults
```

4. 要跟踪管道运行的进度，请输入以下命令：

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

上述命令中的 <pipelinerun_id> 是上一命令输出返回的 **PipelineRun** 的 ID。

5. 几分钟后，使用 **tkn pipelinerun list** 命令列出所有管道运行来验证管道是否成功运行：

```
$ tkn pipelinerun list
```

输出列出了管道运行：

```
NAME                     STARTED      DURATION     STATUS
build-and-deploy-run-xy7rw   1 hour ago   2 minutes    Succeeded
build-and-deploy-run-z2rz8   1 hour ago   19 minutes   Succeeded
```

6. 获取应用程序路由：

```
$ oc get route pipelines-vote-ui --template='http://{{.spec.host}}'
```

记录上一个命令的输出。您可以使用此路由来访问应用程序。

7. 要重新运行最后的管道运行,请使用上一管道的管道资源和服务帐户运行：

```
$ tkn pipeline start build-and-deploy --last
```

其他资源

- 使用 secret 使用存储库验证管道

1.7. 在管道中添加触发器

触发器（Trigger）使 Pipelines 可以响应外部 GitHub 事件，如推送事件和拉取请求。在为应用程序组装并启动管道后，添加 **TriggerBinding**、**TriggerTemplate**、**Trigger** 和 **EventListener** 资源来捕获 GitHub 事件。

流程

1. 复制以下 **TriggerBinding** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: vote-app
spec:
  params:
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

2. 创建 **TriggerBinding** 资源：

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

或者，您可以直接从 **pipelines-tutorial** Git 仓库创建 **TriggerBinding** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/01_binding.yaml
```

3. 复制以下 **TriggerTemplate** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: vote-app
spec:
  params:
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
```

```
    description: The git revision
    default: pipelines-1.15
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1
    kind: PipelineRun
    metadata:
      generateName: build-deploy-$(tt.params.git-repo-name)-
    spec:
      taskRunTemplate:
        serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
        value: $(tt.params.git-revision)
      - name: IMAGE
        value: image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/$(tt.params.git-repo-name)
      workspaces:
      - name: shared-workspace
        volumeClaimTemplate:
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 500Mi
```

模板指定一个卷声明模板，用于创建用于为工作空间定义存储卷的持久性卷声明。因此，您不需要创建持久性卷声明来提供数据存储。

4. 创建 **TriggerTemplate** 资源：

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

另外，您还可以从 **pipelines-tutorial** Git 仓库直接创建 **TriggerTemplate** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-
1.15/03_triggers/02_template.yaml
```

5. 复制以下 **Trigger** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: vote-trigger
spec:
  taskRunTemplate:
```

```
    serviceAccountName: pipeline
  bindings:
    - ref: vote-app
  template:
    ref: vote-app
```

6. 创建 **Trigger** 资源：

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

另外，您还可以直接从 **pipelines-tutorial** Git 仓库创建 **Trigger** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/03_trigger.yaml
```

7. 复制以下 **EventListener** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  triggers:
    - triggerRef: vote-trigger
```

或者，如果您还没有定义触发器自定义资源，将绑定和模板规格添加到 **EventListener** YAML 文件中，而不是引用触发器的名称：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  triggers:
  - bindings:
    - ref: vote-app
    template:
      ref: vote-app
```

8. 通过执行以下步骤来创建 **EventListener** 资源：

- 使用安全 HTTPS 连接创建 **EventListener** 资源：

  a. 添加一个标签，在 Eventlistener 资源中启用安全 **HTTPS** 连接：

  ```
  $ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
  ```

  b. 创建 **EventListener** 资源：

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

或者，您可以直接从 **pipelines-tutorial** Git 仓库创建 **EvenListener** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/04_event_listener.yaml
```

c. 使用重新加密 TLS 终止创建路由：

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

另外，您可以创建一个重新加密 TLS 终止 YAML 文件，以创建安全路由。

安全路由重新加密 TLS 终止 YAML 示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend (2)
  tls:
    termination: reencrypt (3)
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- (4)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1) (2) 对象的名称，长度限于 63 个字符。

(3) **termination** 字段设置为 **reencrypt**。这是唯一需要 **tls** 的字段。

(4) 重新加密需要。**destinationCACertificate** 指定用来验证端点证书的 CA 证书，保护从路由器到目标 pod 的连接。如果服务使用服务签名证书，或者管理员为路由器指定默认 CA 证书，且服务有由该 CA 签名的证书，则可以省略此字段。

如需了解更多选项，请参阅 **oc create route reencrypt --help**。

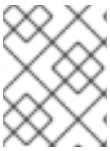
- 使用不安全的 HTTP 连接创建 **EventListener** 资源：

a. 创建 **EventListener** 资源。

b. 将 **EventListener** 服务公开为 OpenShift Container Platform 路由，使其可以被公开访问：

```
$ oc expose svc el-vote-app
```

1.8. 配置事件监听程序为多个命名空间提供服务



注意

如果要创建一个基本的 CI/CD 管道，您可以跳过此部分。但是，如果您的部署策略涉及多个命名空间，您可以将事件监听程序配置为为多个命名空间提供服务。

为了提高 **EvenListener** 对象的可重用性，集群管理员可将它们配置为为多个命名空间的多租户事件监听程序进行配置和部署。

流程

1. 为事件监听程序配置集群范围的获取权限。

   a. 设置在 **ClusterRoleBinding** 和 **EventListener** 对象中使用的服务帐户名称。例如，**el-sa**。

   ServiceAccount.yaml 示例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: el-sa
---
```

   b. 在 **ClusterRole.yaml** 文件的 **rules** 部分，为每个事件监听器部署设置适当的权限，以便正常工作集群范围的。

   ClusterRole.yaml 示例

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: el-sel-clusterrole
rules:
- apiGroups: ["triggers.tekton.dev"]
  resources: ["eventlisteners", "clustertriggerbindings", "clusterinterceptors",
"triggerbindings", "triggertemplates", "triggers"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["configmaps", "secrets"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["impersonate"]
...
```

   c. 使用适当的服务帐户名称和集群角色名称配置集群角色绑定。

   ClusterRoleBinding.yaml 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: el-mul-clusterrolebinding
subjects:
- kind: ServiceAccount
```

```
  name: el-sa
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: el-sel-clusterrole
...
```

2. 在事件监听器的 **spec** 参数中，添加服务帐户名称，如 **el-sa**。使用事件监听程序要服务的命名空间名称填充 **namespaceSelector** 参数。

EventListener.yaml 示例

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: namespace-selector-listener
spec:
  taskRunTemplate:
    serviceAccountName: el-sa
  namespaceSelector:
    matchNames:
    - default
    - foo
...
```

3. 创建具有必要权限的服务帐户，如 **foo-trigger-sa**。使用它来绑定触发器。

ServiceAccount.yaml 示例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: foo-trigger-sa
  namespace: foo
...
```

RoleBinding.yaml 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggercr-rolebinding
  namespace: foo
subjects:
- kind: ServiceAccount
  name: foo-trigger-sa
  namespace: foo
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
...
```

4. 使用适当的触发器模板、触发器绑定和服务帐户名称创建触发器。

Trigger.yaml 示例

```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: trigger
  namespace: foo
spec:
  taskRunTemplate:
    serviceAccountName: foo-trigger-sa
  interceptors:
    - ref:
        name: "github"
      params:
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["push"]
  bindings:
    - ref: vote-app
  template:
    ref: vote-app
...
```

1.9. 创建 WEBHOOK

Webhook 是事件监听程序在存储库中配置事件时接收到的 HTTP POST 信息。然后，事件有效负载映射到触发器绑定，并由触发器模板处理。触发器模板最终启动一个或多个管道运行，从而创建并部署 Kubernetes 资源。

在本小节中，您将在 Git 存储库 **pipelines-vote-ui** 和 **pipelines-vote-api** 的副本中配置 webhook URL。这个 URL 指向公开访问的 **EventListener** 服务路由。

注意

添加 Webhook 需要对该存储库有管理特权。如果您没有对库的管理权限，请联络您的系统管理员来添加 webhook。

流程

1. 获取 Webhook URL：

- 对于安全 HTTPS 连接：

```
$ echo "URL: $(oc  get route el-vote-app --template='https://{{.spec.host}}')"
```

- 对于 HTTP（不安全）连接：

```
$ echo "URL: $(oc  get route el-vote-app --template='http://{{.spec.host}}')"
```

记录下输出中的 URL。

2. 在前端存储库中手动配置 Webhook：
   a. 在浏览器中打开前端 Git 存储库 **pipelines-vote-ui**。
   b. 点 **Settings → Webhooks → Add Webhook**
   c. 在 **Webhooks/Add Webhook** 页面中：
      i. 在 **Payload URL** 字段中输入第 1 步中的 webhook URL
      ii. 为 **Content type** 选择 **application/json**
      iii. 在 **Secret** 字段中指定 secret
      iv. 确定选择了 **Just the push event**
      v. 选择 **Active**
      vi. 点击 **Add webhook**。
3. 重复步骤 2 来使用后端存储库 **pipelines-vote-api**。

1.10. 触发一个管道运行

每当 Git 仓库中发生 **push** 事件时，配置的 Webhook 会将事件有效负载发送到公开的 **EventListener** 服务路由。应用程序的 **EventListener** 服务处理有效负载，并将其传递给相关的 **TriggerBinding** 和 **TriggerTemplate** 资源对。**TriggerBinding** 资源提取参数，**TriggerTemplate** 资源使用这些参数并指定必须创建资源的方式。这可能会重建并重新部署应用程序。

在本小节中，您将把一个空的提交推送到前端 **pipelines-vote-ui** 存储库，该存储库将触发管道运行。

流程

1. 在终端中，克隆 fork 的 Git 存储库 **pipelines-vote-ui**：

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.15
```

2. 推送空提交：

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.15
```

3. 检查管道运行是否已触发：

```
$ tkn pipelinerun list
```

请注意，一个新的管道运行被启动。

1.11. 为用户定义的项目启用触发器监控事件监听程序

作为集群管理员，在用户定义的项目中收集 **Triggers** 服务的事件监听程序指标，并在 OpenShift Container Platform Web 控制台中显示它们，您可以为每个事件监听程序创建服务监控器。在收到 HTTP 请求时，**Triggers** 服务的事件监听程序返回三个指标数据 - **eventlistener_http_duration_seconds**, **eventlistener_event_count**, 和 **eventlistener_triggered_resources**。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- 已安装 Red Hat OpenShift Pipelines Operator。
- 您已为用户定义的项目启用了监控。

流程

1. 对于每个事件侦听器，创建一个服务监控器。例如，要查看 **test** 命名空间中的 **github-listener** 事件监听程序的指标，请创建以下服务监控器：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/managed-by: EventListener
    app.kubernetes.io/part-of: Triggers
    eventlistener: github-listener
  annotations:
    networkoperator.openshift.io/ignore-errors: ""
  name: el-monitor
  namespace: test
spec:
  endpoints:
    - interval: 10s
      port: http-metrics
  jobLabel: name
  namespaceSelector:
    matchNames:
      - test
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: EventListener
      app.kubernetes.io/part-of: Triggers
      eventlistener: github-listener
...
```

2. 通过将请求发送到事件监听程序来测试服务监控器。例如，推送空提交：

```
$ git commit -m "empty-commit" --allow-empty && git push origin main
```

3. 在 OpenShift Container Platform web 控制台中进入 **Administrator → Observe → Metrics**。
4. 要查看指标，请按名称搜索。例如，若要查看 **github-listener** 事件监听器的 **eventlistener_http_resources** 指标的详细信息，请使用 **eventlistener_http_resources** 关键字搜索。

其他资源

- 为用户定义的项目启用监控

1.12. 在 GITHUB 拦截器中配置拉取请求功能

使用 GitHub Interceptor，您可以创建验证和过滤 GitHub Webhook 的逻辑。例如，您可以验证 webhook 的源并根据指定条件过滤传入的事件。当使用 GitHub Interceptor 过滤事件数据时，您可以在字段中指定拦截器可以接受的事件类型。在 Red Hat OpenShift Pipelines 中，您可以使用 GitHub Interceptor 的以下功能：

- 根据已更改的文件过滤拉取请求事件
- 根据配置的 GitHub 所有者验证拉取请求

1.12.1. 使用 GitHub Interceptor 过滤拉取请求

您可以根据为推送和拉取事件更改的文件过滤 GitHub 事件。这有助于您只对 Git 仓库中的相关更改执行管道。GitHub Interceptor 添加以逗号分隔的所有更改的文件列表，并使用 CEL Interceptor 根据更改的文件过滤传入的事件。更改的文件列表添加到顶层 **extensions** 字段中事件有效负载的 **changed_files** 属性中。

先决条件

- 已安装 Red Hat OpenShift Pipelines Operator。

流程

1. 执行以下步骤之一：
   - 对于公共 GitHub 存储库，在 YAML 配置文件中将 **addChangedFiles** 参数的值设置为 **true**，如下所示：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
          - name: "secretRef"
            value:
              secretName: github-secret
              secretKey: secretToken
          - name: "eventTypes"
            value: ["pull_request", "push"]
          - name: "addChangedFiles"
            value:
              enabled: true
        - ref:
            name: cel
          params:
          - name: filter
            value: extensions.changed_files.matches('controllers/')
...
```

- 对于私有 GitHub 存储库，将 **addChangedFiles** 参数的值设置为 **true**，并在以下所示的 YAML 配置文件中提供访问令牌详情、**secretName** 和 **secretKey** ：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
          - name: "secretRef"
            value:
              secretName: github-secret
              secretKey: secretToken
          - name: "eventTypes"
            value: ["pull_request", "push"]
          - name: "addChangedFiles"
            value:
              enabled: true
              personalAccessToken:
                secretName: github-pat
                secretKey: token
        - ref:
            name: cel
          params:
          - name: filter
            value: extensions.changed_files.matches('controllers/')
...
```

2. 保存配置文件。

1.12.2. 使用 GitHub Interceptors 验证拉取请求

您可以使用 GitHub Interceptor 根据为存储库配置的 GitHub 所有者验证拉取请求处理。此验证可帮助您防止对 **PipelineRun** 或 **TaskRun** 对象进行不必要的执行。只有在用户名列为拥有者或者由存储库的所有者发出可配置注释时，GitHub Interceptor 才会处理拉取请求。例如，当您以所有者身份在拉取请求上注释掉 **/ok-to-test** 时，会触发 **PipelineRun** 或 **TaskRun**。

注意

所有者在存储库根目录的 **OWNERS** 文件中配置。

先决条件

- 已安装 Red Hat OpenShift Pipelines Operator。

流程

1. 创建 secret 字符串值。

2. 使用该值配置 GitHub Webhook。
3. 创建名为 **secretRef** 的 Kubernetes secret，其中包含您的 secret 值。
4. 将 Kubernetes secret 作为对 GitHub Interceptor 的引用。
5. 创建一个 **owners** 文件，并将批准者列表添加到 **approvers** 部分。
6. 执行以下步骤之一：

- 对于公共 GitHub 存储库，在 YAML 配置文件中将 **githubOwners** 参数的值设置为 **true**，如下所示：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-owners-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
            - name: "secretRef"
              value:
                secretName: github-secret
                secretKey: secretToken
            - name: "eventTypes"
              value: ["pull_request", "issue_comment"]
            - name: "githubOwners"
              value:
                enabled: true
                checkType: none
...
```

- 对于私有 GitHub 存储库，将 **githubOwners** 参数的值设置为 **true**，并在以下所示的 YAML 配置文件中提供访问令牌详情、**secretName** 和 **secretKey** ：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-owners-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
            - name: "secretRef"
              value:
```

```
      secretName: github-secret
      secretKey: secretToken
  - name: "eventTypes"
    value: ["pull_request", "issue_comment"]
  - name: "githubOwners"
    value:
      enabled: true
      personalAccessToken:
        secretName: github-token
        secretKey: secretToken
      checkType: all
...
```

注意

checkType 参数用于指定需要身份验证的 GitHub 所有者。您可以将其值设为 **orgMembers**、**repoMembers** 或 **all**。

7. 保存配置文件。

1.13. 其他资源

- 要将 Pipelines as Code 和应用程序源代码包含在同一存储库中，请参阅关于 Pipelines as Code。
- 如需有关 **Developer** 视角中的管道的更多信息，请参阅 web 控制台中的使用 OpenShift Pipelines 部分。
- 要了解更多有关安全性上下文约束（SCC）的信息，请参阅 管理安全性上下文约束部分。
- 如需有关可重复使用的任务的更多示例，请参阅 OpenShift Catalog 仓库。另外，您还可以在 Tekton 项目中看到 Tekton Catalog。
- 要安装并部署 Tekton Hub 的自定义实例，以了解可重复使用的任务和管道，请参阅使用带有 Red Hat OpenShift Pipelines 的 Tekton Hub。
- 有关重新加密 TLS 终止的详情，请参阅重新加密终止。
- 有关安全路由的详情，请参阅安全路由部分。

第 2 章 在 WEB 控制台中使用 RED HAT OPENSHIFT PIPELINES

您可以使用 **Administrator** 或 **Developer** 视角，从 OpenShift Container Platform web 控制台的 **Pipelines** 页面创建和修改 **Pipeline**、**PipelineRun** 和 **Repository** 对象。您还可以使用 web 控制台的 **Developer** 视角中的 **+Add** 页面为软件交付过程创建 CI/CD 管道。

2.1. 在 DEVELOPER 视角中使用 RED HAT OPENSHIFT PIPELINES

在 **Developer** 视角中，您可以从 **+Add** 页面访问创建管道的以下选项：

- 使用 **+Add → Pipelines → Pipeline builder** 选项为您的应用程序创建自定义管道。
- 在创建应用程序时，使用 **+Add → From Git** 选项使用管道模板和资源创建管道。

在为应用程序创建管道后，可以在 **Pipelines** 视图中查看并以视觉化的形式与部署进行交互。您还可以使用 **Topology** 视图与使用 **From Git** 选项创建的管道交互。您需要将自定义标识应用到使用 **Pipeline builder** 创建的管道，以便在 **Topology** 视图中查看它。

先决条件

- 您可以访问 OpenShift Container Platform 集群并切换到 **Developer** 视角。
- 已在集群中安装了 OpenShift Pipelines Operator。
- 您是集群管理员，或是有创建和编辑权限的用户。
- 您已创建了一个项目。

2.1.1. 使用 Pipeline 构建器构建管道

在控制台的 **Developer** 视角中，您可以使用 **+Add → Pipeline → Pipeline Builder** 选项：

- 使用 **Pipeline 构建器** 或 **YAML 视图** 配置管道。
- 使用现有任务和集群任务构建管道流。安装 OpenShift Pipelines Operator 时，它会在集群中添加可重复使用的管道集群任务。

重要

在 Red Hat OpenShift Pipelines 1.10 中，**ClusterTask** 功能已弃用，计划在以后的发行版本中删除。

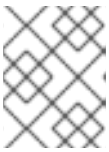
- 指定管道运行所需的资源类型，如有必要，将额外参数添加到管道。
- 引用管道中的每个任务中的这些管道资源作为输入和输出资源。
- 如果需要，引用任务中添加至管道的任何额外参数。任务的参数会根据任务的规格预先填充。
- 使用 Operator 安装的、可重复使用的片断和示例来创建详细的管道。
- 搜索并添加您配置的本地 Tekton Hub 实例的任务。

重要

在 Developer 视角中，您可以使用自己的一组策展的任务创建自定义管道。要从开发人员控制台直接搜索、安装和升级任务，集群管理员需要安装和部署本地 Tekton Hub 实例，并将 hub 链接到 OpenShift Container Platform 集群。如需了解更多详细信息，请参阅*附加资源*部分中的*将 Tekton Hub 与 OpenShift Pipelines 搭配使用*。如果您没有部署任何本地 Tekton Hub 实例，则默认只能访问集群任务、命名空间任务和公共 Tekton Hub 任务。

流程

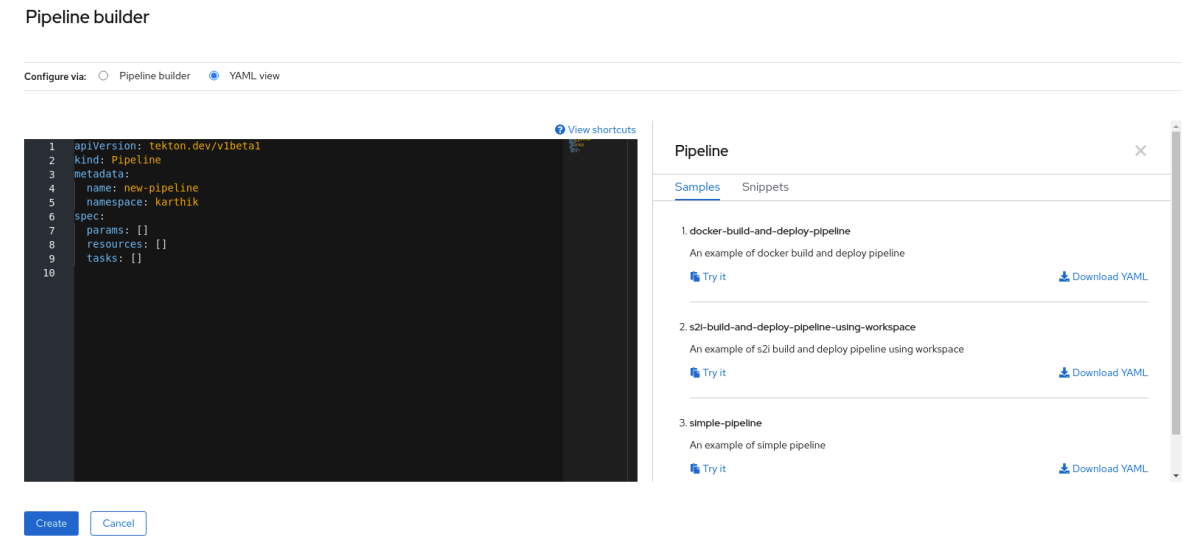
1. 在 **Developer** 视角的 **Add** 视图中，点 **Pipeline** 标题查看 **Pipeline Builder** 页面。
2. 使用 **Pipeline 构建器** 视图或 **YAML 视图**配置管道。



注意

Pipeline 构建器 视图支持有限的字段，而 **YAML 视图**支持所有可用字段。另外，您还可以使用 Operator 安装的、可重复使用的片断和样本来创建详细的管道。

图 2.1. YAML 视图



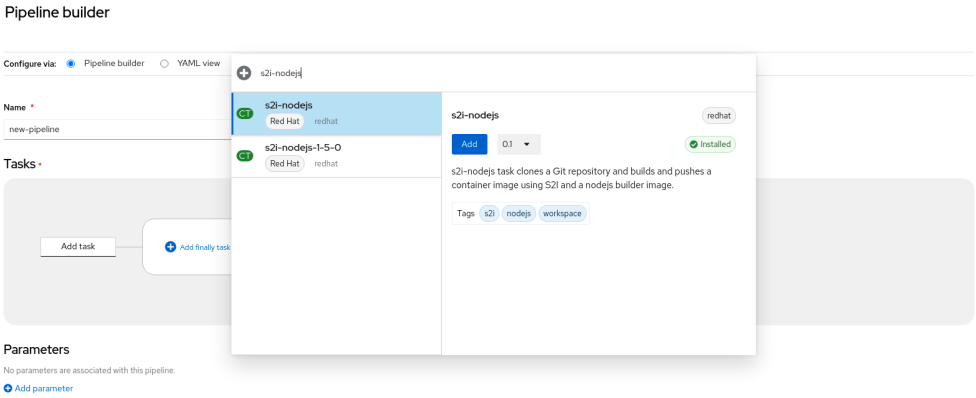
3. 使用 **Pipeline 构建器**配置管道：
   a. 在 **Name** 字段中输入管道的唯一名称。
   b. 在 **Tasks** 部分：
      i. 单击 **Add task**。
      ii. 使用快速搜索字段搜索任务，然后从显示的列表中选择所需的任务。
      iii. 点击 **Add** 或 **Install and add**。在本例中,使用 **s2i-nodejs** 任务。

注意

搜索列表包含集群中可用的所有 Tekton Hub 任务和任务。此外，如果某一任务已经安装，它将显示 **Add** 添加该任务，而它将显示 **Install and add** 以安装和添加该任务。当使用更新的版本添加相同的任务时，它将显示 **Update and add**。

- 在管道中添加后续任务：
  - 点击任务右侧或左侧的加号图标 → 点 **Add task**。
  - 使用快速搜索字段搜索任务，然后从显示的列表中选择所需的任务。
  - 点击 **Add** 或 **Install and add**。

  图 2.2. Pipeline 构建器



- 添加最后一项任务：
  - 点 **Add finally task** → Click **Add task**。
  - 使用快速搜索字段搜索任务，然后从显示的列表中选择所需的任务。
  - 点击 **Add** 或 **Install and add**。

c. 在 **Resources** 部分，点 **Add Resources** 指定管道运行的资源的名称和类型。这些资源然后会被管道中的任务使用作为输入和输出。在此例中：

   i. 添加一个输入资源。在 **Name** 字段中输入 **Source**，从 **Resource Type** 下拉列表中选择 **Git**。

   ii. 添加一个输出资源。在 **Name** 字段中输入 **img**，从 **Resource Type** 下拉列表中选择 **Image**。

注意

如果缺少资源，任务旁边会出现一个红色图标。

d. 可选：任务的 **Parameters** 部分会根据任务的规格预先填充。如果需要，使用 **Parameters** 部分中的 **Add Parameters** 链接来添加额外的参数。

e. 在 **Workspaces** 部分，点 **Add workspace**，并在 **Name** 字段中输入唯一工作区名称。您可以在管道中添加多个工作区。

f. 在 **Tasks** 部分中，点 **s2i-nodejs** 任务，以查看带任务详情的侧面板。在任务侧面板中，为 **s2i-nodejs** 任务指定资源和参数：

   i. 如果需要，在 **Parameters** 部分，使用 $(params.<param-name>)语法为默认参数添加更多参数。

   ii. 在 **Image** 部分中，按在 **Resources** 部分中的指定输入 **Img**。

iii. 从 **Workspaces** 部分下的 **source** 下拉菜单中选择一个工作区。

g. 将资源、参数和工作空间添加到 **openshift-client** 任务。

4. 点 **Create** 在 **Pipeline Details** 页面中创建并查看管道。

5. 点 **Actions** 下拉菜单，然后点 **Start**，查看 **Start Pipeline** 页面。

6. **Workspaces** 部分列出了您之前创建的工作区。使用对应的下拉菜单为您的工作区指定卷源。您有以下选项： **mpty Directory**、**Config Map**、**Secret**、**PersistentVolumeClaim** 或 **VolumeClaimTemplate**。

2.1.2. 创建 OpenShift Pipelines 和应用程序

要与应用程序一同创建管道，使用 **Developer** 视角的 **Add+** 视图中的 **From Git** 选项。您可以查看所有可用的管道，并在导入代码或部署镜像时选择用来创建应用程序的管道。

Tekton Hub 集成默认是启用的，您可以看到集群支持的 Tekton Hub 中的任务。管理员可以选择不使用 Tekton Hub 集成，将不会显示 Tekton Hub 任务。您还可以检查是否为生成的管道存在 webhook URL。为使用 **+Add** 流创建的管道添加默认 Webhook，URL 在 Topology 视图中所选资源侧面面板中可见。

如需更多信息，请参阅使用 Developer 视角创建应用程序。

2.1.3. 添加包含管道的 GitHub 仓库

在 **Developer** 视角中，您可以将包含管道的 GitHub 存储库添加到 OpenShift Container Platform 集群中。当相关 Git 事件（如推送或拉取请求）被触发时，可以在集群中的 GitHub 存储库中运行管道和任务。

注意

您可以同时添加公共和私有 GitHub 仓库。

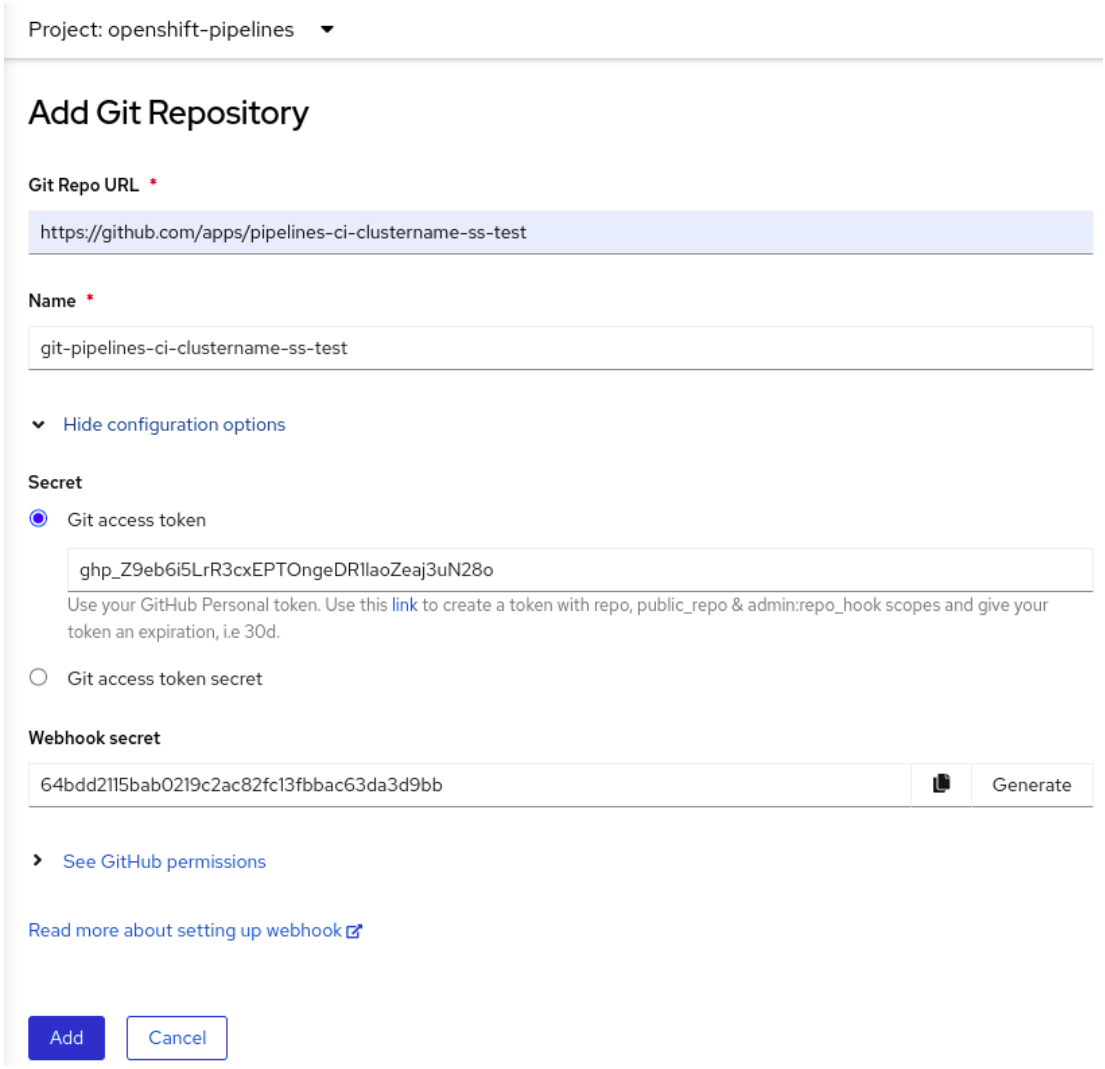
先决条件

- 确保集群管理员已在管理员视角中配置了所需的 GitHub 应用程序。

流程

1. 在 **Developer** 视角中，选择要在其中添加 GitHub 仓库的命名空间或项目。
2. 使用左侧导航窗格进入到 **Pipelines**。
3. 在 **Pipelines** 页面右侧点 **Create → Repository**。
4. 输入 **Git Repo URL**，控制台会自动获取仓库名称。
5. 点 **Show configuration options**。默认情况下，您只看到一个选项 **Setup a webhook**。如果您配置了 GitHub 应用程序，您会看到两个选项：
   - **Use GitHub App**: 选择这个选项在仓库中安装 GitHub 应用程序。
   - **Setup a webhook** ：选择此选项，将 webhook 添加到您的 GitHub 应用程序。
6. 使用 **Secret** 部分中的以下选项之一设置 webhook：

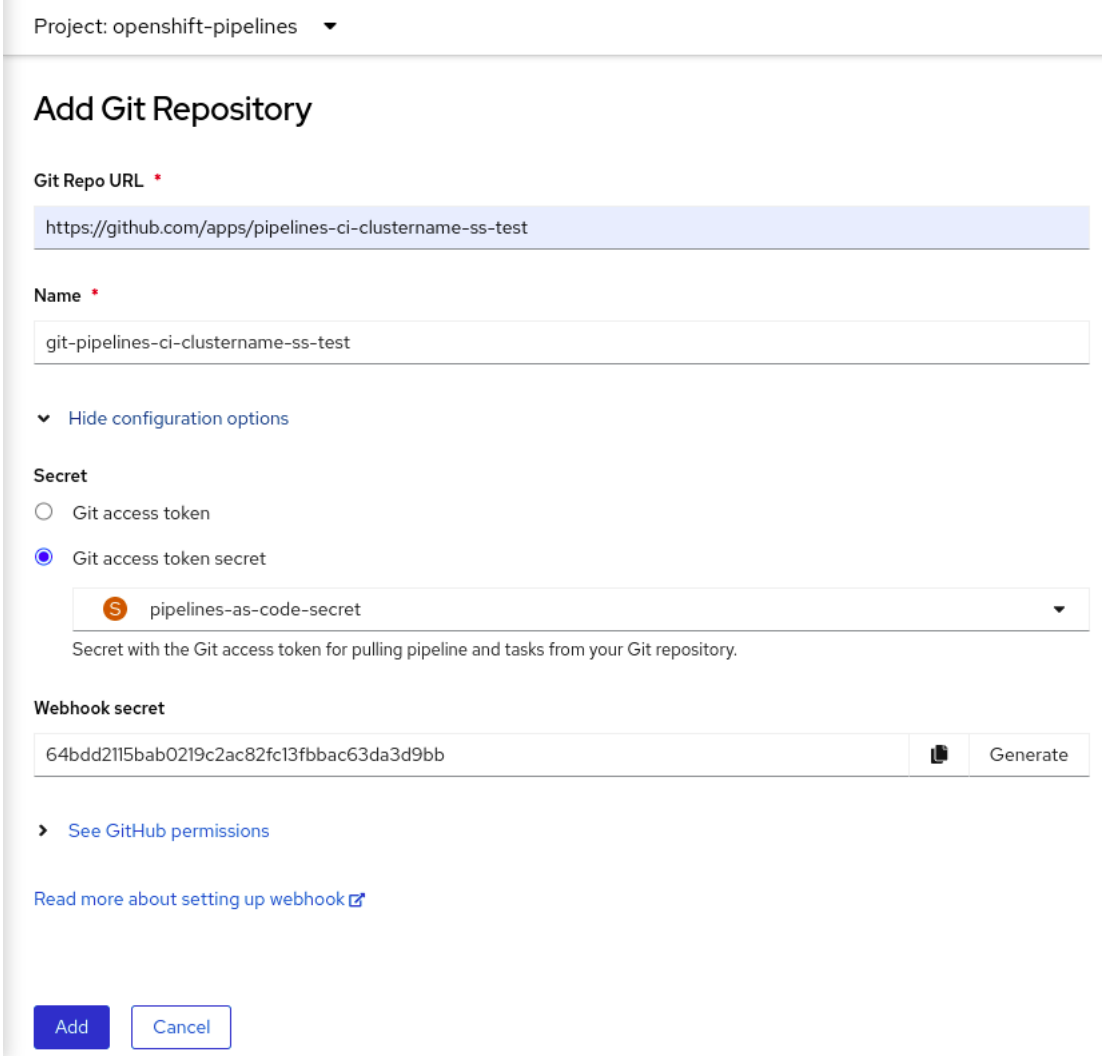
- 使用 **Git 访问令牌** 设置 webhook：
  a. 输入您的个人访问令牌。
  b. 点与 **Webhook secret** 字段对应的 **Generate** 以生成新的 webhook secret。



注意

如果您没有个人访问令牌并且想要创建新令牌，可以点 **Git access token** 字段下的链接。

- 使用 **Git 访问令牌 secret** 设置 webhook：
  - 从下拉列表中选择命名空间中的 secret。根据您选择的 secret，会自动生成 Webhook secret。



7. 在您的 GitHub 仓库中添加 webhook secret 详情：
   a. 复制 **Webhook URL** 并前往 GitHub 仓库设置。
   b. 点 **Webhooks → Add webhook**。
   c. 从开发人员控制台复制 **Webhook URL**，并将它粘贴到 GitHub 仓库设置的 **Payload URL** 字段中。
   d. 选择**内容类型**。
   e. 从开发人员控制台复制 **Webhook secret**，并将它粘贴到 GitHub 仓库设置的 **Secret** 字段中。
   f. 选择 **SSL 验证**选项之一。
   g. 选择要触发此 webhook 的事件。
   h. 点击 **Add webhook**。
8. 返回到开发人员控制台，然后点 **Add**。
9. 阅读执行步骤的详情，并点 **Close**。
10. 查看您刚才创建的仓库的详细信息。

注意

当使用 **Import from Git** 导入应用程序时，Git 存储库有一个 **.tekton** 目录，您可以为应用程序配置 **pipelines-as-code**。

2.1.4. 使用 Developer 视角与管道交互

开发者视角中的 **Pipelines** 视图列出了项目中的所有管道，以及以下详细信息：

- 创建管道的命名空间
- 最后一次管道运行
- 管道运行中的任务状态
- 管道运行的状态
- 最后一次管道运行的创建时间

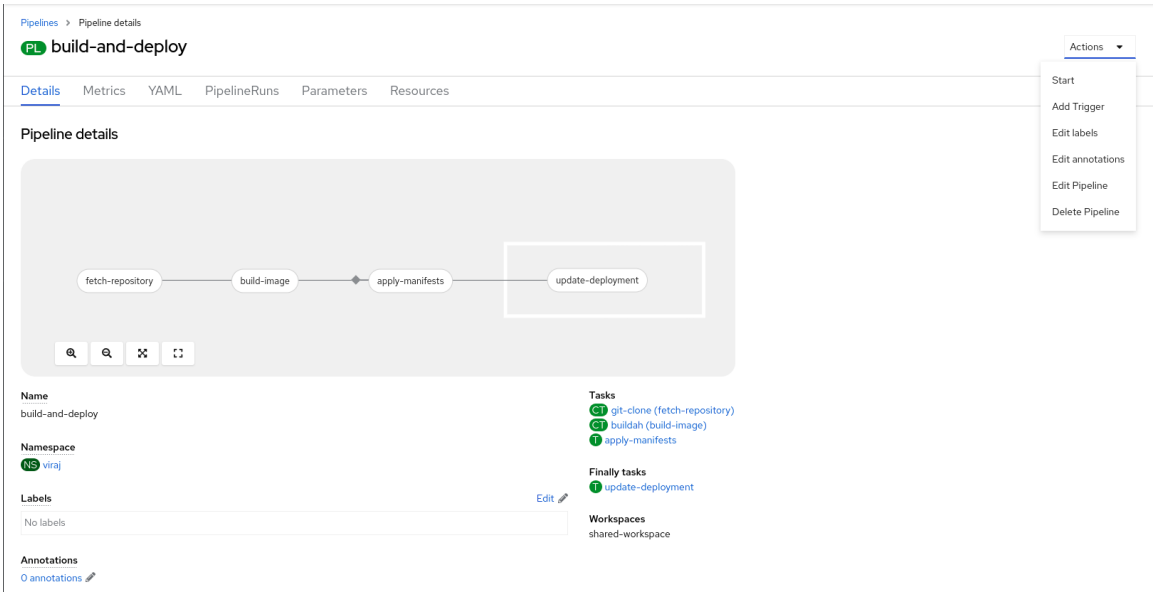
流程

1. 在 **Developer** 视角的 **Pipelines** 视图中，从 **Project** 下拉列表中选择一个项目，以查看该项目中的管道。
2. 点击所需管道查看 **Pipeline 详情**页面。
   默认情况下，**Details** 选项卡显示所有 **serial** 任务，**parallel** 任务、**finally** 任务以及管道中的 **when** 表达式的可视化表示。这些任务和 **finally** 的任务列在页面的右下角。

   要查看任务详情，请点列出的 **Tasks** 和 **Finally** 任务。另外，您可以执行以下操作：

   - 使用 **Pipeline 详情**视觉化左下角显示的标准图标，使用 zoom in、zoom out 和 reset view 功能。
   - 使用鼠标轮改管道视觉化的缩放因素。
   - 将鼠标悬停在任务上，再查看任务详情。

   图 2.3. Pipeline 详情



3. 可选： 在 **Pipeline 详情**页面中，点 **Metrics** 选项卡查看有关管道的以下信息：
   - **Pipeline 成功率**
   - **Pipeline 运行数量**
   - **管道运行持续时间**
   - **任务运行持续时间**
     您可以使用这些信息来改进管道工作流，并在管道生命周期的早期解决问题。
4. 可选：点 **YAML** 选项卡编辑管道的 YAML 文件。
5. 可选：点 **Pipeline Runs** 选项卡查看已完成、正在运行或运行失败的管道。
   Pipeline Runs 选项卡提供有关管道运行、任务状态以及调试失败管道运行的链接。您可以使用 Options 菜单 来停止正在运行的管道，使用与之前的管道执行相同的参数和资源重新运行管道，或删除管道运行。
   - 点所需的管道运行查看 **Pipeline Run details** 页面。默认情况下，**Details** 选项卡显示所有串行任务、并行任务、**finally** 任务以及管道中运行的 when 表达式的可视化表示。成功运行的结果会显示在页面底部的 **Pipeline Run 结果**窗格下。另外，您只能查看由集群支持的 Tekton Hub 中的任务。在查看某个任务时，您可以点击相关链接来跳至任务文档。

     注意

     Pipeline Run Details 页面的 **Details** 部分显示失败管道运行的**日志片段**。**日志片断** 提供一般错误信息和日志片断。**Logs** 部分的链接可让您快速访问失败运行的详细信息。
   - 在 **Pipeline Run details** 页面中，点 **Task Runs** 选项卡查看已完成、正在运行和运行失败的任务。
     Task Runs 选项卡提供有关任务运行的信息，以及任务和 pod 的链接，以及任务运行的状态和持续时间。使用 Options 菜单 来删除任务运行。

     注意

     TaskRuns 列表页面带有一个 **Manage column** 按钮，您还可以使用它来添加 **Duration** 列。
   - 点所需的任务运行查看 **Task Run details** 页面。成功运行的结果显示在页面底部的 **Task Run 结果**窗格下。

     注意

     Task Run details 页面的 **Details** 部分显示失败任务运行的**日志片段**。**日志片断** 提供一般错误信息和日志片断。**Logs** 部分的链接可让您快速访问失败的任务运行的详细信息。
6. 点 **Parameters** 标签，查看管道中定义的参数。您还可以根据需要添加或者编辑附加参数。
7. 点 **Resources** 标签页，查看管道中定义的资源。您还可以根据需要添加或编辑附加资源。

2.1.5. 从 Pipelines 视图启动管道

创建管道后，您需要启动它以在定义的顺序中执行包含的任务。您可从 Pipelines 视图、Pipeline Details 页面或 Topology 视图启动管道。

流程

使用 Pipelines 视图启动管道：

1. 在 Developer 视角 的 Pipelines 视图中，点附加到 Pipeline 的 Options  菜单，然后选择 Start。
2. Start Pipeline对话框显示 Git Resources 以及基于管道定义的 Image Resources。

注意

对于使用 From Git选项创建的管道，Start Pipeline对话框也会在 Parameters 部分显示 APP_NAME 字段，对话框中的所有字段都由管道模板预先填充。

   a. 如果您在命名空间中有资源，Git Resources 和 Image Resources 字段会预先填充这些资源。如果需要，使用下拉菜单选择或创建所需资源并自定义管道运行实例。

3. 可选：修改 Advanced Options 以添加验证指定私有 Git 服务器或镜像 registry 的凭证。
   a. 在 Advanced Options 下，点 Show Credentials Options并选择 Add Secret。
   b. 在 Create Source Secret部分，指定以下内容：
      i. secret 的唯一 Secret Name。
      ii. 在 要被验证的指定供应商 部分，在 Access to 字段中指定要验证的供应商，以及基本 服务器 URL。
      iii. 选择 Authentication Type 并提供凭证：
         - 对于 Authentication Type Image Registry Credentials，请指定您要进行身份验证的 Registry 服务器地址，并通过Username、Password 和 Email 项中提供您的凭证。
           如果要指定额外的 Registry 服务器地址，选择 Add Credentials。
         - 如果 Authentication Type 为 Basic Authentication，在 UserName 和 Password or Token 项中指定相关的值。
         - 如果 Authentication Type 为 SSH Keys 时，在 SSH Private Key 字段中指定相关的值。

注意

对于基本身份验证和 SSH 身份验证，您可以使用注解，例如：

- **tekton.dev/git-0: https://github.com**
- **tekton.dev/git-1: https://gitlab.com**.

      iv. 选择要添加 secret 的检查标记。

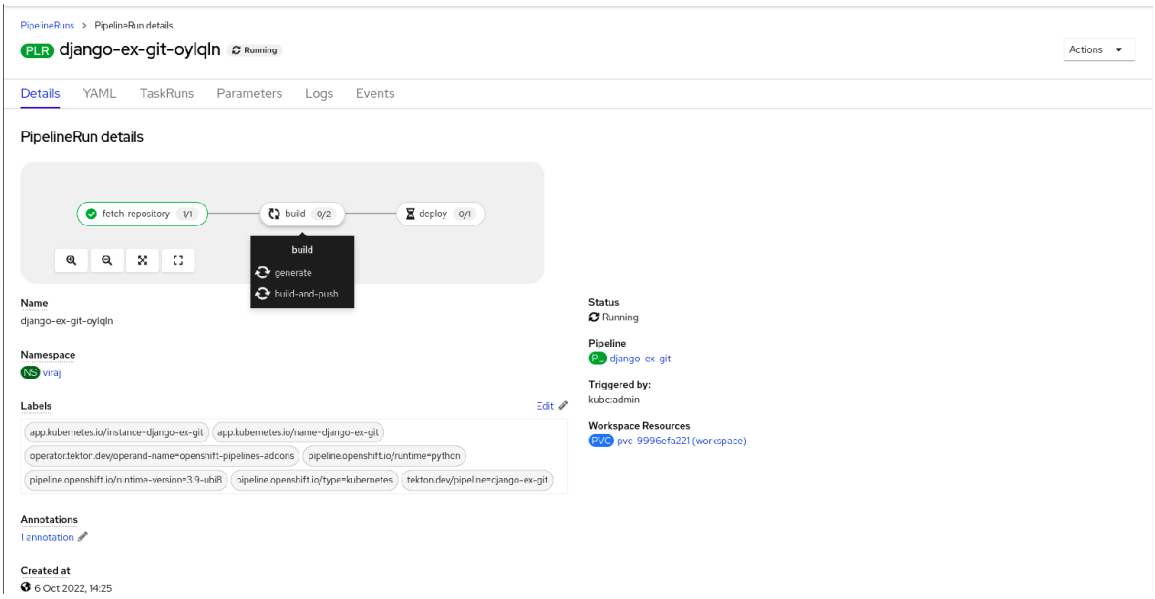
您可以根据频道中的资源数量添加多个 secret。

4. 点 **Start** 启动管道。

5. **PipelineRun 详情**页面显示正在执行的管道。管道启动后，每个任务中的任务和步骤都会被执行。您可以：

   - 使用 PipelineRun 详情页面的左下角，使用 zoom in、zoom out、适用于屏幕和重置视图功能，它们位于 **PipelineRun 详情**页面的左下角。
   - 使用鼠标轮改管道运行视觉化的缩放因素。在特定的缩放条件中，任务更改的背景颜色表示错误或警告状态。
   - 将鼠标悬停在任务上，以查看详细信息，例如执行每个步骤、任务名称和任务状态所需的时间。
   - 将鼠标悬停在任务徽标上，以查看已完成的任务和任务总数。
   - 点一个任务来查看任务中每一步骤的日志。
   - 点 **Logs** 选项卡查看与任务执行顺序相关的日志。您还可以使用相关按钮扩展窗格，单独或批量下载日志。
   - 点 **Events** 选项卡查看管道运行生成的事件流。
     您可以使用 **Task Runs**、**Logs** 和 **Events** 选项卡来帮助调试失败的管道运行或失败的任务运行。

     图 2.4. Pipeline 运行详情



2.1.6. 从 Topology 视图启动管道

对于使用 **From Git** 选项创建的管道，您可以在启动后使用 **Topology** 视图来与管道进行交互：

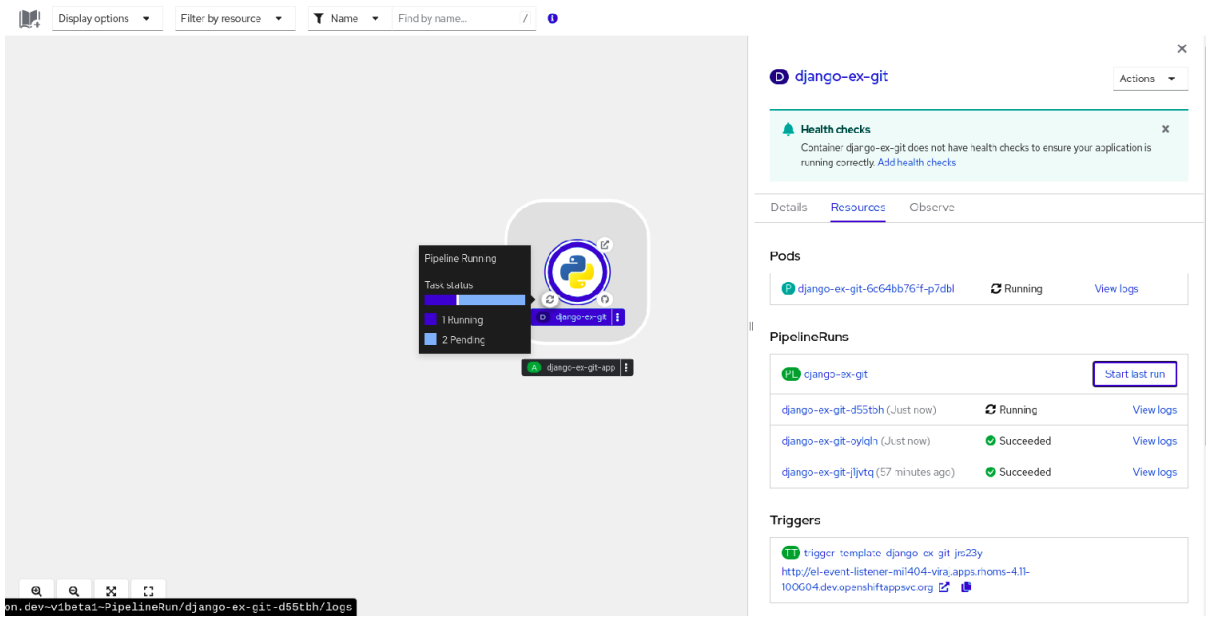
注意

要使用 **Pipeline Builder** 在 **Topology** 视图中查看创建的管道，自定义管道标识来把 Pipeline 与应用程序负载相连接。

流程

1. 单击左侧导航面板中的 **Topology**。
2. 点应用程序在侧面面板中显示 **Pipeline Runs**。
3. 在 **Pipeline Runs** 中，点 **Start Last Run** 来启动一个新的管道运行，使用与前一个相同的参数和资源。如果没有启动管道运行，这个选项会被禁用。您还可以在创建管道时启动管道运行。

图 2.5. Topology 视图中的管道



在 **Topology** 页面中，把鼠标移到应用程序的左侧，以查看其管道运行的状态。添加管道后，底部左图标表示存在关联的管道。

2.1.7. 从 Topology 视图与管道交互

Topology 页面中的应用程序节点的侧面板显示管道运行的状态，并可与之交互。

- 如果管道运行没有自动启动，则侧面板会显示管道无法自动启动的消息，因此需要手动启动。
- 如果创建一个管道，但用户没有启动管道，则其状态不会启动。当用户点击 **Not started** status 图标时，**Topology** 视图中会打开启动对话框。
- 如果管道没有构建或构建配置，则无法看到 **Builds** 部分。如果存在管道和构建配置，则 **Builds 部分** 可见。
- 当管道运行在特定任务运行时，侧边面板会显示一个 **日志片段**。您可以在 **Pipeline Runs** 部分的 **Resources** 选项卡中查看**日志片断**。它提供常规错误消息和日志片断。**Logs** 部分的链接可让您快速访问失败运行的详细信息。

2.1.8. 编辑管道

您可以使用 web 控制台的 **Developer** 视角编辑集群中的管道：

流程

1. 在 **Developer** 视角的 **Pipelines** 视图中，选择您要编辑的管道来查看管道的详情。在 **Pipeline Details** 页中，点 **Actions** 并选择 **Edit Pipeline**。
2. 在 **Pipeline builder** 页面中，您可以执行以下任务：

- 在管道中添加其他任务、参数或资源。
- 点要修改的任务来查看侧面面板中的任务详情，并修改所需的任务详情，如显示名称、参数和资源。
- 或者，要删除任务，点任务，在侧面面板中点 **Actions** 并选择 **Remove Task**。

3. 点 **Save** 保存修改后的管道。

2.1.9. 删除管道

您可以使用 web 控制台的 **Developer** 视角删除集群中的管道。

流程

1. 在 **Developer 视角** 的 **Pipelines** 视图中，点 Pipeline 旁的 **Options** 菜单，然后选择 **Delete Pipeline**。
2. 在 **Delete Pipeline** 确认提示下，点 **Delete** 以确认删除。

2.2. 其他资源

- 在 OpenShift Pipelines 中使用 Tekton Hub

2.3. 在 ADMINISTRATOR 视角中创建管道模板

作为集群管理员，您可以创建管道模板，供开发人员在集群上创建管道时重复使用。

先决条件

- 您可以使用集群管理员权限访问 OpenShift Container Platform 集群，并切换到 **Administrator** 视角。
- 已在集群中安装了 OpenShift Pipelines Operator。

流程

1. 进入 **Pipelines** 页面查看现有的管道模板。
2. 点 图标进入 **Import YAML** 页面。
3. 为管道模板添加 YAML。模板必须包括以下信息：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
# ...
  namespace: openshift (1)
  labels:
```

```
    pipeline.openshift.io/runtime: <runtime> 2
    pipeline.openshift.io/type: <pipeline-type> 3
# ...
```

1. 模板必须在 **openshift** 命名空间中创建。
2. 模板必须包含 **pipeline.openshift.io/runtime** 标签。此标签接受的运行时值有 **nodejs**,**golang**,**dotnet**,**java**,**php**,**ruby**,**perl**,**python**,**nginx**, 和 **httpd**。
3. 模板必须包含 **pipeline.openshift.io/type:** 标签。此标签接受的类型值为 **openshift**,**knative**, 和 **kubernetes**。

4. 点 **Create**。创建管道后，您将进入 **Pipeline 详情页面**，您可以在其中查看或编辑管道的信息。

2.4. WEB 控制台中的管道执行统计

您可以在 web 控制台中查看与执行管道相关的统计信息。

要查看统计信息，您必须完成以下步骤：

- 安装 Tekton 结果。有关安装 Tekton 结果的更多信息，*请参阅附加资源部分中的 Tekton Results for OpenShift Pipelines observability*。
- 启用 OpenShift Pipelines 控制台插件。

统计信息适用于所有管道以及每个管道。

重要

OpenShift Pipelines Pipelines 控制台插件只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

附加资源

- 为 OpenShift Pipelines 可观察性使用 Tekton 结果

2.4.1. 启用 OpenShift Pipelines 控制台插件

要查看统计信息，您必须首先启用 OpenShift Pipelines 控制台插件。

先决条件

- 已在集群中安装了 Red Hat OpenShift Pipelines Operator。
- 使用集群管理员权限登录到 web 控制台。

重要

OpenShift Pipelines 控制台插件需要 OpenShift Container Platform 版本 4.15 或更高版本。

流程

1. 在 Web 控制台的 **Administrator** 视角中，选择 **Operators → Installed Operators**。
2. 在 Operator 表中点 **Red Hat OpenShift Pipelines**。
3. 在屏幕上的右侧窗格中，检查 **Console 插件** 下的 status 标签。标签可以是 **Enabled** 或 **Disabled**。
4. 如果标签是 **Disabled**，点此标签。在显示的窗口中，选择 **Enable**，然后单击 **Save**。

2.4.2. 一起查看所有管道的统计信息

您可以查看与系统中所有管道相关的整合统计信息。

先决条件

- 已在集群中安装了 Red Hat OpenShift Pipelines Operator。
- 已安装 Tekton 结果。
- 已安装 OpenShift Pipelines web 控制台插件。

流程

1. 在 web 控制台的 **Administrator** 视角中，选择 **Pipelines → Overview**。
   显示统计数据概览。此概述包括以下信息： **一个图形反映在同一时间段内运行管道的总和状态、平均和最长持续时间。| 在同一周期内运行的管道总数。**

   也会显示管道表。此表列出了在时间段内运行的所有管道，显示它们的持续时间和成功率。
2. 可选：根据需要更改统计的设置：
   - **项目** ：用于显示统计信息的项目或命名空间。
   - **时间范围** ：显示统计信息的时间段。
   - **刷新间隔** ：当您查看时，Red Hat OpenShift Pipelines 必须更新窗口中的数据的频率。

2.4.3. 查看特定管道的统计信息

您可以查看与特定管道相关的统计信息。

先决条件

- 已在集群中安装了 Red Hat OpenShift Pipelines Operator。
- 已安装 Tekton 结果。
- 已安装 OpenShift Pipelines web 控制台插件。

流程

1. 在 web 控制台的 **Administrator** 视角中，选择 **Pipelines → Pipelines**。
2. 点管道列表中的管道。**Pipeline 详情视图** 会显示。
3. 点 **Metrics** 选项卡。
   显示统计数据概览。此概述包括以下信息： **一个图形反映在同一时间段内运行管道的总和状态、平均和最长持续时间。| 在同一周期内运行的管道总数。**
4. 可选：根据需要更改统计的设置：
   - **项目** ：用于显示统计信息的项目或命名空间。
   - **时间范围** ：显示统计信息的时间段。
   - **刷新间隔** ：当您查看时，Red Hat OpenShift Pipelines 必须更新窗口中的数据的频率。

第 3 章 使用解析器指定远程管道和任务

管道和任务是 CI/CD 进程的可重复使用的块。您可以重复使用之前开发的管道或任务，或者由其他人开发的任务，而无需复制并粘贴其定义。这些管道或任务可从集群中的其他命名空间中的几种源到公共目录提供。

在管道运行资源中，您可以从现有源指定管道。在管道资源或任务运行资源中，您可以从现有源指定任务。

在这些情况下，Red Hat OpenShift Pipelines 中的 *解析器* 在运行时从指定源中检索管道或任务定义。

以下解析器包括在 Red Hat OpenShift Pipelines 的默认 installaton 中：

hub 解析器

从 Artifact Hub 或 Tekton Hub 上的 Pipelines Catalog 检索任务或管道。

捆绑包解析器

从 Tekton 捆绑包检索任务或管道，该捆绑包是可从任何 OCI 存储库（如 OpenShift 容器存储库）提供的 OCI 镜像。

集群解析器

检索已在特定命名空间中在同一 OpenShift Container Platform 集群中创建的任务或管道。

Git 解析器

从 Git 存储库检索任务或管道绑定。您必须指定存储库、分支和路径。

OpenShift Pipelines 安装包括一组可以在管道中使用的标准任务。这些任务位于 OpenShift Pipelines 安装命名空间中，通常是 **openshift-pipelines** 命名空间。您可以使用集群解析器访问任务。

3.1. 从 TEKTON 目录指定远程管道或任务

您可以使用 hub 解析器指定在 Artifact Hub 的公共 Tekton 目录或 Tekton Hub 实例中定义的远程管道或任务。

重要

Red Hat OpenShift Pipelines 不支持 Artifact Hub 项目。仅支持 Artifact Hub 的配置。

3.1.1. 配置 hub 解析器

您可以通过配置 hub 解析器来更改拉取资源的默认 hub 和默认目录设置。

流程

1. 要编辑 **TektonConfig** 自定义资源，请输入以下命令：

```
$ oc edit TektonConfig config
```

2. 在 **TektonConfig** 自定义资源中，编辑 **pipeline.hub-resolver-config** spec：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
```

```
pipeline:
  hub-resolver-config:
    default-tekton-hub-catalog: Tekton (1)
    default-artifact-hub-task-catalog: tekton-catalog-tasks (2)
    default-artifact-hub-pipeline-catalog: tekton-catalog-pipelines (3)
    defailt-kind: pipeline (4)
    default-type: tekton (5)
    tekton-hub-api: "https://my-custom-tekton-hub.example.com" (6)
    artifact-hub-api: "https://my-custom-artifact-hub.example.com" (7)
```

1. 用于拉取资源的默认 Tekton Hub 目录。
2. 用于拉取任务资源的默认 Artifact Hub 目录。
3. 用于拉取管道资源的默认 Artifact Hub 目录。
4. 用于引用的默认对象类型。
5. 用于拉取资源的默认 hub，对于 Artifact Hub 是**工件（artifact）**，对于 **tekton** 是 Tekton Hub。
6. 使用的 Tekton Hub API，如果 **default-type** 选项被设置为 **tekton**。
7. 可选：如果 **default-type** 选项被设置为 **artifact**，则使用 Artifact Hub API。

重要

如果将 **default-type** 选项设置为 **tekton**，则必须通过设置 **tekton-hub-api** 值来配置您自己的 Tekton Hub 实例。

如果将 **default-type** 选项设置为 **artifact**，则解析器默认使用位于 https://artifacthub.io/ 的公共 hub API。您可以通过设置 **artifact-hub-api** 值来配置您自己的 Artifact Hub API。

3.1.2. 使用 hub 解析器指定远程管道或任务

在创建管道运行时，您可以指定来自 Artifact Hub 或 Tekton Hub 的远程管道。在创建管道或任务运行时，您可以从 Artifact Hub 或 Tekton Hub 指定远程任务。

流程

- 要指定来自 Artifact Hub 或 Tekton Hub 的远程管道或任务，请在 **pipelineRef** 或 **taskRef** spec 中使用以下引用格式：

```
# ...
  resolver: hub
  params:
  - name: catalog
    value: <catalog>
  - name: type
    value: <catalog_type>
  - name: kind
    value: [pipeline|task]
  - name: name
```

```
    value: <resource_name>
  - name: version
    value: <resource_version>
# ...
```

表 3.1. hub 解析器支持的参数

参数	描述	示例值
catalog	用于拉取资源的目录。	默认：**tekton-catalog-tasks**（用于 **task** 类型）；**tekton-catalog-pipelines**（用于 **pipeline** 类型）。
type	用于拉取资源的目录类型。**artifact**（Artifact Hub）或 **tekton**（Tekton Hub）。	默认: **artifact**
kind	输入 **task** 或 **pipeline**。	默认: **task**
name	从 hub 获取的任务或管道的名称。	**golang-build**
version	要从 hub 获取的任务或管道版本。您必须使用引号(**''**)来括起数据。	**"0.5.0"**

如果管道或任务需要额外的参数，在管道、管道运行或任务运行规格的 **params** 部分中指定这些参数的值。**pipelineRef** 或 **taskRef** 规格的 **params** 部分必须包含解析器支持的参数。

以下示例管道运行引用目录中的远程管道：

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: hub-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: hub
    params:
    - name: catalog
      value: tekton-catalog-pipelines
    - name: type
      value: artifact
    - name: kind
      value: pipeline
    - name: name
      value: example-pipeline
    - name: version
      value: "0.1"
  params:
  - name: sample-pipeline-parameter
    value: test
```

以下示例管道从目录中引用远程任务：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-cluster-task-reference-demo
spec:
  tasks:
  - name: "cluster-task-reference-demo"
    taskRef:
      resolver: hub
      params:
      - name: catalog
        value: tekton-catalog-tasks
      - name: type
        value: artifact
      - name: kind
        value: task
      - name: name
        value: example-task
      - name: version
        value: "0.6"
    params:
    - name: sample-task-parameter
      value: test
```

以下示例任务运行来引用目录中的远程任务：

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: cluster-task-reference-demo
spec:
  taskRef:
    resolver: hub
    params:
    - name: catalog
      value: tekton-catalog-tasks
    - name: type
      value: artifact
    - name: kind
      value: task
    - name: name
      value: example-task
    - name: version
      value: "0.6"
  params:
  - name: sample-task-parameter
    value: test
```

3.2. 从 TEKTON 捆绑包指定远程管道或任务

您可以使用捆绑包解析器指定 Tekton 捆绑包中的远程管道或任务。Tekton 捆绑包是任何 OCI 存储库（如 OpenShift 容器存储库）提供的 OCI 镜像。

3.2.1. 配置捆绑包解析器

您可以通过配置捆绑包解析器来更改默认服务帐户名称和从 Tekton 捆绑包中拉取资源的默认 kind。

流程

1. 要编辑 **TektonConfig** 自定义资源，请输入以下命令：

```
$ oc edit TektonConfig config
```

2. 在 **TektonConfig** 自定义资源中，编辑 **pipeline.bundles-resolver-config** spec:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    bundles-resolver-config:
      default-service-account: pipelines 1
      default-kind: task 2
```

1. 用于捆绑包请求的默认服务帐户名称。
2. 捆绑包镜像中的默认层类型。

3.2.2. 使用捆绑包解析器指定远程管道或任务

在创建管道运行时，您可以从 Tekton 捆绑包中指定远程管道。在创建管道或任务运行时，您可以从 Tekton 捆绑包中指定远程任务。

流程

- 要从 Tekton 捆绑包中指定远程管道或任务，请在 **pipelineRef** 或 **taskRef** spec 中使用以下引用格式：

```
# ...
  resolver: bundles
  params:
  - name: bundle
    value: <fully_qualified_image_name>
  - name: name
    value: <resource_name>
  - name: kind
    value: [pipeline|task]
# ...
```

表 3.2. 捆绑包解析器支持的参数

参数	描述	示例值

参数	描述	示例值
serviceAccount	构建 registry 凭据时要使用的服务帐户名称。	**default**
bundle	指向镜像要获取的捆绑包 URL。	**gcr.io/tekton-releases/catalog/upstream/golang-build:0.1**
name	要从捆绑包中拉取的资源名称。	**golang-build**
kind	从捆绑包中拉取的资源类型。	**task**

如果管道或任务需要额外的参数，在管道、管道运行或任务运行规格的 **params** 部分中指定这些参数的值。**pipelineRef** 或 **taskRef** 规格的 **params** 部分必须包含解析器支持的参数。

以下示例管道运行引用 Tekton 捆绑包中的远程管道：

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: bundle-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: bundles
    params:
    - name: bundle
      value: registry.example.com:5000/simple/pipeline:latest
    - name: name
      value: hello-pipeline
    - name: kind
      value: pipeline
  params:
  - name: sample-pipeline-parameter
    value: test
  - name: username
    value: "pipelines"
```

以下示例管道引用 Tekton 捆绑包中的远程任务：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-bundle-task-reference-demo
spec:
  tasks:
  - name: "bundle-task-demo"
    taskRef:
      resolver: bundles
      params:
      - name: bundle
```

```
      value: registry.example.com:5000/advanced/task:latest
    - name: name
      value: hello-world
    - name: kind
      value: task
  params:
  - name: sample-task-parameter
    value: test
```

以下示例任务运行引用 Tekton 捆绑包中的远程任务：

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: bundle-task-reference-demo
spec:
  taskRef:
    resolver: bundles
    params:
    - name: bundle
      value: registry.example.com:5000/simple/new_task:latest
    - name: name
      value: hello-world
    - name: kind
      value: task
  params:
  - name: sample-task-parameter
    value: test
```

3.3. 从 GIT 仓库指定远程管道或任务

您可以使用 Git 解析器指定 Git 仓库中的远程管道或任务。存储库必须包含定义管道或任务的 YAML 文件。Git 解析器可以通过匿名方式或使用经过身份验证的 SCM API 来访问存储库。

3.3.1. 为匿名 Git 克隆配置 Git 解析器

如果要使用匿名 Git 克隆，您可以配置默认的 Git 修订、获取超时和默认存储库 URL，以便从 Git 存储库拉取远程管道和任务。

流程

1. 要编辑 **TektonConfig** 自定义资源，请输入以下命令：

```
$ oc edit TektonConfig config
```

2. 在 **TektonConfig** 自定义资源中，编辑 **pipeline.git-resolver-config** spec：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    git-resolver-config:
```

```
    default-revision: main (1)
    fetch-timeout: 1m (2)
    default-url: https://github.com/tektoncd/catalog.git (3)
```

1. 如果未指定，使用默认 Git 修订。
2. 单个 Git 克隆解析可能需要的最大时间，如 **1m**、**2s**、**700ms**。Red Hat OpenShift Pipelines 还在所有解析请求中强制实施全局最大超时时间为 1 分钟。
3. 如果未指定任何，匿名克隆的默认 Git 存储库 URL。

3.3.2. 为经过身份验证的 SCM API 配置 Git 解析器

对于经过身份验证的 SCM API，您必须为经过身份验证的 Git 连接设置配置。

您可以使用 **go-scm** 库支持的 Git 存储库供应商。并非所有 **go-scm** 实现都已使用 Git 解析器测试，但以下供应商已知可以正常工作：

- **github.com** 和 GitHub Enterprise
- **gitlab.com** 和自托管的 Gitlab
- Gitea
- Bitbucket 服务器
- Bitbucket 云

注意

- 您只能在集群中使用经过身份验证的 SCM API 配置一个 Git 连接。此连接可供集群的所有用户使用。集群的所有用户都可以使用您为连接配置的安全令牌访问存储库。
- 如果将 Git 解析器配置为使用经过身份验证的 SCM API，您还可以使用匿名 Git 克隆引用来检索管道和任务。

流程

1. 要编辑 **TektonConfig** 自定义资源，请输入以下命令：

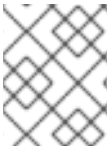
```
$ oc edit TektonConfig config
```

2. 在 **TektonConfig** 自定义资源中，编辑 **pipeline.git-resolver-config** spec：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    git-resolver-config:
      default-revision: main (1)
      fetch-timeout: 1m (2)
```

```
    scm-type: github 3
    server-url: api.internal-github.com 4
    api-token-secret-name: github-auth-secret 5
    api-token-secret-key: github-auth-key 6
    api-token-secret-namespace: github-auth-namespace 7
    default-org: tektoncd 8
```

1. 如果未指定，使用默认 Git 修订。
2. 单个 Git 克隆解析可能需要的最大时间，如 **1m**、**2s**、**700ms**。Red Hat OpenShift Pipelines 还在所有解析请求中强制实施全局最大超时时间为 1 分钟。
3. SCM 提供程序类型。
4. 用于经过身份验证的 SCM API 的基本 URL。如果您使用 **github.com**、**gitlab.com** 或 BitBucket Cloud，则不需要此设置。
5. 包含 SCM 提供程序 API 令牌的 secret 名称。
6. 包含令牌的令牌 secret 中的密钥。
7. 包含令牌 secret 的命名空间（如果不是 **default**）。
8. 可选：使用经过身份验证的 API 时存储库的默认机构。如果您没有在解析器参数中指定机构，则使用此机构。



注意

使用经过身份验证的 SCM API 需要 **scm-type**,**api-token-secret-name**, 和 **api-token-secret-key** 设置。

3.3.3. 使用 Git 解析器指定远程管道或任务

在创建管道运行时，您可以从 Git 仓库指定远程管道。在创建管道或任务运行时，您可以从 Git 仓库指定远程任务。

先决条件

- 如果要使用经过身份验证的 SCM API，您必须为 Git 解析器配置经过身份验证的 Git 连接。

流程

1. 要从 Git 仓库指定远程管道或任务，请在 **pipelineRef** 或 **taskRef** spec 中使用以下引用格式：

```
# ...
  resolver: git
  params:
  - name: url
    value: <git_repository_url>
  - name: revision
    value: <branch_name>
```

```
  - name: pathInRepo
    value: <path_in_repository>
# ...
```

表 3.3. Git 解析器支持的参数

参数	描述	示例值
url	使用匿名克隆时存储库的 URL。	**https://github.com/tektoncd/catalog.git**
repo	使用经过身份验证的 SCM API 时的存储库名称。	**test-infra**
org	使用经过身份验证的 SCM API 时存储库的机构。	**tektoncd**
revision	存储库中的 Git 修订。您可以指定分支名称、标签名称或提交 SHA 哈希。	**aeb957601cf41c012be462827053a21a420befca** **main** **v0.38.2**
pathInRepo	存储库中的 YAML 文件的路径名称。	**task/golang-build/0.3/golang-build.yaml**

注意

要匿名克隆和获取存储库，请使用 **url** 参数。要使用经过身份验证的 SCM API，请使用 **repo** 参数。不要同时指定 **url** 参数和 **repo** 参数。

如果管道或任务需要额外的参数，在管道、管道运行或任务运行规格的 **params** 部分中指定这些参数的值。**pipelineRef** 或 **taskRef** 规格的 **params** 部分必须包含解析器支持的参数。

以下示例管道运行引用 Git 存储库中的远程管道：

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: git-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: git
    params:
    - name: url
      value: https://github.com/tektoncd/catalog.git
    - name: revision
      value: main
    - name: pathInRepo
      value: pipeline/simple/0.1/simple.yaml
  params:
  - name: name
```

```
    value: "testPipelineRun"
  - name: sample-pipeline-parameter
    value: test
```

以下示例管道引用 Git 存储库中的远程任务：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-git-task-reference-demo
spec:
  tasks:
  - name: "git-task-reference-demo"
    taskRef:
      resolver: git
      params:
      - name: url
        value: https://github.com/tektoncd/catalog.git
      - name: revision
        value: main
      - name: pathInRepo
        value: task/git-clone/0.6/git-clone.yaml
    params:
    - name: sample-task-parameter
      value: test
```

以下示例任务运行引用 Git 存储库中的远程任务：

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: git-task-reference-demo
spec:
  taskRef:
    resolver: git
    params:
    - name: url
      value: https://github.com/tektoncd/catalog.git
    - name: revision
      value: main
    - name: pathInRepo
      value: task/git-clone/0.6/git-clone.yaml
  params:
  - name: sample-task-parameter
    value: test
```

3.4. 从同一集群中指定远程管道或任务

您可以使用集群解析器指定在运行 Red Hat OpenShift Pipelines 的 OpenShift Container Platform 集群上的命名空间中定义的远程管道或任务。

特别是，您可以使用集群解析器访问 OpenShift Pipelines 在其安装命名空间中提供的任务，通常是 **openshift-pipelines** 命名空间。

3.4.1. 配置集群解析器

您可以更改集群解析器的默认类型和命名空间，或者限制集群解析器可以使用的命名空间。

流程

1. 要编辑 **TektonConfig** 自定义资源，请输入以下命令：

```
$ oc edit TektonConfig config
```

2. 在 **TektonConfig** 自定义资源中，编辑 **pipeline.cluster-resolver-config** spec：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    cluster-resolver-config:
      default-kind: pipeline (1)
      default-namespace: namespace1 (2)
      allowed-namespaces: namespace1, namespace2 (3)
      blocked-namespaces: namespace3, namespace4 (4)
```

(1) 如果没有在参数中指定，则获取的默认资源类型。

(2) 如果没有在参数中指定，用于获取资源的默认命名空间。

(3) 以逗号分隔的解析器允许访问的命名空间列表。如果没有定义此密钥，则允许所有命名空间。

(4) 可选的以逗号分隔的命名空间列表，解析器会阻止访问它们。如果没有定义此密钥，则允许所有命名空间。

3.4.2. 使用集群解析器指定远程管道或任务

在创建管道运行时，您可以从同一集群中指定远程管道。在创建管道或任务运行时，您可以从同一集群中指定远程任务。

流程

- 要指定同一集群中的远程管道或任务，请在 **pipelineRef** 或 **taskRef** spec 中使用以下引用格式：

```
# ...
  resolver: cluster
  params:
  - name: name
    value: <name>
  - name: namespace
    value: <namespace>
  - name: kind
    value: [pipeline|task]
# ...
```

表 3.4. 集群解析器支持的参数

参数	描述	示例值
name	要获取的资源的名称。	**some-pipeline**
namespace	包含资源的集群中的命名空间。	**other-namespace**
kind	要获取的资源类型。	**pipeline**

如果管道或任务需要额外的参数，请在 **params** 中提供这些参数。

以下示例管道运行引用同一集群中的远程管道：

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: cluster-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: cluster
    params:
    - name: name
      value: some-pipeline
    - name: namespace
      value: test-namespace
    - name: kind
      value: pipeline
  params:
  - name: sample-pipeline-parameter
    value: test
```

以下示例管道引用同一集群中的远程任务：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-cluster-task-reference-demo
spec:
  tasks:
  - name: "cluster-task-reference-demo"
    taskRef:
      resolver: cluster
      params:
      - name: name
        value: some-task
      - name: namespace
        value: test-namespace
      - name: kind
        value: task
```

```
    params:
    - name: sample-task-parameter
      value: test
```

以下示例任务运行引用同一集群中的远程任务：

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: cluster-task-reference-demo
spec:
  taskRef:
    resolver: cluster
    params:
    - name: name
      value: some-task
    - name: namespace
      value: test-namespace
    - name: kind
      value: task
  params:
  - name: sample-task-parameter
    value: test
```

3.5. OPENSHIFT PIPELINES 命名空间中提供的任务

OpenShift Pipelines 安装包括一组可以在管道中使用的标准任务。这些任务位于 OpenShift Pipelines 安装命名空间中，通常是 **openshift-pipelines** 命名空间。您可以使用集群解析器访问任务。

从 OpenShift Pipelines 1.10 开始，**ClusterTask** 功能已弃用，并计划在以后的发行版本中删除。如果您的管道使用 **ClusterTasks**，您可以使用集群解析器使用 OpenShift Pipelines 安装命名空间中可用的任务重新创建它们。但是，与现有的 **ClusterTasks** 相比，这些任务中进行了某些更改。

您无法在 OpenShift Pipelines 安装命名空间中任何可用的任务中指定自定义执行镜像。这些任务不支持 **BUILDER_IMAGE**、**gitInitImage** 或 **KN_IMAGE** 等参数。如果要使用自定义执行镜像，请创建任务副本并通过编辑副本来替换镜像。

buildah

buildah 任务将源代码树构建到容器镜像中，然后将镜像推送到容器 registry。

buildah 任务的使用示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-image
    taskRef:
      resolver: cluster
      params:
      - name: kind
```

```
      value: task
    - name: name
      value: buildah
    - name: namespace
      value: openshift-pipelines
  params:
  - name: IMAGE
    value: $(params.IMAGE)
  workspaces:
  - name: source
    workspace: shared-workspace
# ...
```

表 3.5. buildah 任务支持的参数

参数	描述	类型	默认值
IMAGE	Buildah 要构建的完全限定容器镜像名称。	**string**	
DOCKERFILE	相对于源工作区的 **Dockerfile** （或 **Containerfile**） 的路径。	**string**	**./Dockerfile**
CONTEXT	用作上下文的目录的路径。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**

表 3.6. buildah 任务支持的工作区

Workspace	描述
source	容器构建上下文，通常是包含 **Dockerfile** 或 **Containerfile** 文件的应用源代码。

Workspace	描述
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。
rhel-entitlement	一个可选的工作区，用于提供 Buildah 用来访问 Red Hat Enterprise Linux (RHEL) 订阅的权利密钥。挂载的工作区必须包含 authorization **.pem** 和 **entitlement-key.pem** 文件。

表 3.7. buildah 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。
IMAGE_DIGEST	**string**	构建的镜像摘要。

buildah ClusterTask的更改

- 添加了 **VERBOSE** 参数。
- **BUILDER_IMAGE** 参数已被删除。

git-cli

git-cli 任务运行 **git** 命令行工具。您可以使用 **GIT_SCRIPT** 参数传递完整的 Git 命令或几个命令来运行。如果命令需要向 Git 存储库进行身份验证，例如要完成推送，您必须提供身份验证凭据。

git-cli 任务的用法示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: update-repo
spec:
# ...
  tasks:
# ...
  - name: push-to-repo
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-cli
      - name: namespace
        value: openshift-pipelines
    params:
    - name: GIT_SCRIPT
```

```
      value: "git push"
    - name: GIT_USER_NAME
      value: "Example Developer"
    - name: GIT_USER_EMAIL
      value: "developer@example.com"
    workspaces:
    - name: ssh-directory
      workspace: ssh-workspace (1)
    - name: source
      workspace: shared-workspace
  # ...
```

1 在本例中，**ssh-workspace** 必须包含 **.ssh** 目录的内容，其具有对 Git 存储库授权的有效密钥。

表 3.8. git-cli 任务支持的参数

参数	描述	类型	默认值
CRT_FILENAME	**ssl-ca-directory** 工作区中的证书颁发机构(CA)捆绑包文件名。	**string**	**ca-bundle.crt**
HTTP_PROXY	HTTP 代理服务器（非 TLS 请求）。	**string**	
HTTPS_PROXY	HTTPS 代理服务器(TLS 请求)。	**string**	
NO_PROXY	选择不使用代理 HTTP/HTTPS 请求。	**string**	
子目录	git 存储库存在 **的源** 工作区的相对路径。	**string**	
USER_HOME	pod 中 Git 用户主目录的绝对路径。	**string**	**/home/git**
DELETE_EXISTING	在完成 git 操作前，删除 **源** 工作区的任何现有内容。	**string**	**true**
详细	记录所有已执行的命令。	**string**	**false**
SSL_VERIFY	全局 **http.sslVerify** 值。除非信任远程存储库，否则请不要使用 **false**。	**string**	**true**
GIT_USER_NAME	用于执行 Git 操作的 Git 用户名。	**string**	
GIT_USER_EMAIL	用于执行 Git 操作的 Git 用户电子邮件。	**string**	
GIT_SCRIPT	要运行的 Git 脚本。	**string**	**Git 帮助**

表 3.9. git-cli 任务支持的工作区

Workspace	描述
ssh-directory	根据需要，带有私钥、**known_hosts**、配置 及其他文件的 **.ssh** 目录。**如果提供此工作区，任务会使用它来对 Git 存储库进行身份验证。将此工作区绑定到 Secret 资源，以安全存储身份验证信息。**
basic-auth	包含 **.gitconfig** 和 **.git-credentials** 文件的工作区。如果提供此工作区，任务会使用它来对 Git 存储库进行身份验证。尽可能使用 **ssh-directory** 工作区进行身份验证，而不是 **basic-auth**。将此工作区绑定到 **Secret** 资源，以安全存储身份验证信息。
ssl-ca-directory	包含 CA 证书的工作区。如果您提供此工作区，Git 将使用这些证书在与使用 HTTPS 与远程存储库交互时验证对等证书。
source	包含获取的 Git 存储库的工作区。
输入	包含需要添加到 Git 存储库中的文件的可选工作区。您可以使用 **$(workspaces.input.path)** 从脚本访问工作区，例如： **cp $(workspaces.input.path)/<file_that_i_want> .** **git add <file_that_i_want>**

表 3.10. **git-cli 任务返回的结果**

结果	类型	描述
COMMIT	**string**	位于克隆的 Git 存储库中的当前分支 HEAD 的 SHA 摘要。

git-cli ClusterTask中的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 添加了 **ssl-ca-directory** 工作区。
- **USER_HOME** 和 **VERBOSE** 参数的默认值已更改。
- 结果的名称从 **提交** 改为 **COMMIT**。

git-clone

git-clone 任务使用 Git 在工作区上初始化并克隆远程存储库。您可以在构建或以其他方式处理此源代码的管道的开头使用此任务。

git-clone 任务的用法示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-source
spec:
# ...
  tasks:
  - name: clone-repo
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
    params:
    - name: URL
      value: "https://github.com/example/repo.git"
    workspaces:
    - name: output
      workspace: shared-workspace
```

表 3.11. git-clone 任务支持的参数

参数	描述	类型	默认值
CRT_FILENAME	**ssl-ca-directory** 工作区中的证书颁发机构(CA)捆绑包文件名。	**string**	**ca-bundle.crt**
HTTP_PROXY	HTTP 代理服务器（非 TLS 请求）。	**string**	
HTTPS_PROXY	HTTPS 代理服务器(TLS 请求)。	**string**	
NO_PROXY	选择不使用代理 HTTP/HTTPS 请求。	**string**	
子目录	任务放置 Git 存储库 **的输出** 工作区中的相对路径。	**string**	
USER_HOME	pod 中 Git 用户主目录的绝对路径。	**string**	**/home/git**
DELETE_EXISTING	在运行 Git 操作前，删除默认工作区的内容（如果存在）。	**string**	**true**
详细	记录已执行的命令。	**string**	**false**
SSL_VERIFY	全局 **http.sslVerify** 值。除非信任远程存储库，否则不要将此参数设置为 **false**。	**string**	**true**
URL	Git 存储库 URL。	**string**	

参数	描述	类型	默认值
修订	要签出的修订版本，如分支或标签。	**string**	**main**
REFSPEC	任务在签出修订版本前获取的存储库的 **refspec** 字符串。	**string**	
子模块	初始化并获取 Git 子模块。	**string**	**true**
DEPTH	要抓取的提交数，"允许克隆"是一个提交。	**string**	**1**
SPARSE_CHECKOUT_DIRECTORIES	执行"稀疏签出"的目录模式列表，用逗号分开。	**string**	

表 3.12. git-clone 任务支持的工作区

Workspace	描述
ssh-directory	根据需要，带有私钥、**known_hosts**、配置 及其他文件的 **.ssh** 目录。**如果提供此工作区，任务会使用它来对 Git 存储库进行身份验证。将此工作区绑定到 Secret 资源，以安全存储身份验证信息。**
basic-auth	包含 **.gitconfig** 和 **.git-credentials** 文件的工作区。如果提供此工作区，任务会使用它来对 Git 存储库进行身份验证。尽可能使用 **ssh-directory** 工作区进行身份验证，而不是 **basic-auth**。将此工作区绑定到 **Secret** 资源，以安全存储身份验证信息。
ssl-ca-directory	包含 CA 证书的工作区。如果您提供此工作区，Git 将使用这些证书在与使用 HTTPS 与远程存储库交互时验证对等证书。
output	包含获取的 git 存储库的工作区，数据将放在工作区的根目录或由 **SUBDIRECTORY** 参数定义的相对路径中。

表 3.13. git-clone 任务返回的结果

结果	类型	描述
COMMIT	**string**	位于克隆的 Git 存储库中的当前分支 HEAD 的 SHA 摘要。
URL	**string**	克隆的存储库的 URL。

结果	类型	描述
COMMITTER_DATE	**string**	克隆的 Git 存储库中的当前分支的 HEAD 的 epoch 时间戳。

git-clone ClusterTask的更改

- 所有参数名称都改为大写。
- 所有结果名称都改为大写。
- **gitInitImage** 参数已被删除。

kn
kn 任务使用 **kn** 命令行工具在 Knative 资源上完成操作，如服务、修订或路由。

kn 任务使用示例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: kn-run
spec:
 pipelineSpec:
   tasks:
   - name: kn-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: kn
       - name: namespace
         value: openshift-pipelines
     params:
     - name: ARGS
       value: [version]
```

表 3.14. kn 任务支持的参数

参数	描述	类型	默认值
ARGS	**kn** 工具的参数。	**数组**	**- help**

从 kn ClusterTask的更改

- **KN_IMAGE** 参数已被删除。

kn-apply

kn-apply 任务将指定的镜像部署到 Knative Service。此任务使用 **kn service apply** 命令来创建或更新指定的 Knative 服务。

使用 kn-apply 任务的示例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: kn-apply-run
spec:
 pipelineSpec:
   tasks:
   - name: kn-apply-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: kn-apply
       - name: namespace
         value: openshift-pipelines
     params:
     - name: SERVICE
       value: "hello"
     - name: IMAGE
       value: "gcr.io/knative-samples/helloworld-go:latest"
```

表 3.15. kn-apply 任务支持的参数

参数	描述	类型	默认值
SERVICE	Knative 服务名称。	**string**	
IMAGE	要部署的镜像的完全限定名称。	**string**	

来自 kn-apply ClusterTask的更改

- **KN_IMAGE** 参数已被删除。

maven
maven 任务运行 Maven 构建。

使用 maven 任务的示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
```

```
  - name: build-from-source
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: maven
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.16. maven 任务支持的参数

参数	描述	类型	默认值
目标	要运行的 Maven 目标。	**数组**	**- 软件包**
MAVEN_MIRROR_URL	Maven 存储库镜像 URL。	**string**	
子目录	任务在其上运行 Maven 构建的源工作区中的子目录。	**string**	**.**

表 3.17. maven 任务支持的工作区

Workspace	描述
source	包含 Maven 项目的工作区。
server_secret	包含用于连接 Maven 服务器的 secret 的工作区，如用户名和密码。
proxy_secret	包含用于连接代理服务器的凭据的工作区，如用户名和密码。
proxy_configmap	包含代理配置值的工作区，如 **proxy_port**、**proxy_host**、**proxy_protocol**、**proxy_non_proxy_hosts**。
maven_settings	包含自定义 Maven 设置的工作空间。

Maven **ClusterTask** 的更改

- 参数名称 **CONTEXT_DIR** 被改为 **SUBDIRECTORY**。
- 工作区名称 **maven-settings** 被改为 **maven_settings**。

openshift-client

openshift-client 任务使用 **oc** 命令行界面运行命令。您可以使用此任务来管理 OpenShift Container Platform 集群。

openshift-client 任务使用示例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: openshift-client-run
spec:
 pipelineSpec:
   tasks:
   - name: openshift-client-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: openshift-client
       - name: namespace
         value: openshift-pipelines
     params:
     - name: SCRIPT
       value: "oc version"
```

表 3.18. openshift-client 任务支持的参数

参数	描述	类型	默认值
SCRIPT	要运行的 **oc** CLI 参数。	**string**	**oc help**
VERSION	要使用的 OpenShift Container Platform 版本。	**string**	**latest**

表 3.19. openshift-client 任务支持的工作区

Workspace	描述
manifest_dir	包含您要使用 **oc** 实用程序应用的清单文件的工作区。
kubeconfig_dir	一个可选工作区，您可以在其中提供 **.kube/config** 文件，其中包含用于访问集群的凭证。将此文件放在工作区的根目录中，并将其命名为 **kubeconfig**。

openshift-client ClusterTask的更改

- 工作区名称 **manifest-dir** 被改为 **manifest_dir**。
- 工作区名称 **kubeconfig-dir** 被改为 **kubeconfig_dir**。

s2i-dotnet

s2i-dotnet 任务使用 Source to Image (S2I) dotnet 构建器镜像构建源代码，该镜像可在 OpenShift Container Platform registry 中作为 **image-registry.openshift-image-registry.svc:5000/openshift/dotnet** 提供。

s2i-dotnet 任务使用示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-dotnet
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.20. s2i-dotnet 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	

参数	描述	类型	默认值
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.21. **s2i-dotnet 任务支持的工作区**

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.22. **s2i-dotnet 任务返回的结果**

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。
IMAGE_DIGEST	**string**	构建的镜像摘要。

从 s2i-dotnet ClusterTask的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-go
s2i-go 任务使用 S2I Golang 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/golang** 进行访问。

s2i-go 任务使用示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-go
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.23. s2i-go 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**

参数	描述	类型	默认值
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.24. s2i-go 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.25. s2i-go 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。
IMAGE_DIGEST	**string**	构建的镜像摘要。

s2i-go ClusterTask中的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-java
s2i-java 任务使用 S2I Java 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/java** 提供。

表 3.26. s2i-java 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	

参数	描述	类型	默认值
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.27. s2i-java 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.28. s2i-java 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。

结果	类型	描述
IMAGE_DIGEST	**string**	构建的镜像摘要。

s2i-java ClusterTask中的更改

- 添加了几个新参数。
- **BUILDER_IMAGE**,**MAVEN_ARGS_APPEND**,**MAVEN_CLEAR_REPO**, 和 **MAVEN_MIRROR_URL** 参数已被删除。您可以将 **MAVEN_ARGS_APPEND**、**MAVEN_CLEAR_REPO** 和 **MAVEN_MIRROR_URL** 值作为环境变量传递。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-nodejs

s2i-nodejs 任务使用 S2I NodeJS 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/nodejs** 提供。

s2i-nodejs 任务的用法示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-nodejs
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.29. s2i-nodejs 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	

参数	描述	类型	默认值
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.30. s2i-nodejs 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.31. s2i-nodejs 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。

结果	类型	描述
IMAGE_DIGEST	**string**	构建的镜像摘要。

s2i-nodejs ClusterTask的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-perl

s2i-perl 任务使用 S2I Perl 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/perl** 获取。

s2i-perl 任务使用示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-perl
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.32. s2i-perl 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	

参数	描述	类型	默认值
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.33. s2i-perl 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.34. s2i-perl 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。

结果	类型	描述
IMAGE_DIGEST	**string**	构建的镜像摘要。

s2i-perl ClusterTask的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-php
s2i-php 任务使用 S2I PHP 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/php** 提供。

s2i-php 任务的使用情况示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-php
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.35. s2i-php 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	

参数	描述	类型	默认值
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.36. s2i-php 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.37. s2i-php 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。

结果	类型	描述
IMAGE_DIGEST	**string**	构建的镜像摘要。

从 s2i-php ClusterTask的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-python

s2i-python 任务使用 S2I Python 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/python** 提供。

s2i-python 任务使用示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-python
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.38. s2i-python 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	

参数	描述	类型	默认值
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.39. s2i-python 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.40. s2i-python 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。

结果	类型	描述
IMAGE_DIGEST	**string**	构建的镜像摘要。

s2i-python ClusterTask中的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

s2i-ruby
s2i-ruby 任务使用 S2I Ruby 构建器镜像构建源代码，该镜像可从 OpenShift Container Platform registry 作为 **image-registry.openshift-image-registry.svc:5000/openshift/ruby** 提供。

s2i-ruby 任务的用法示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-ruby
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表 3.41. s2i-ruby 任务支持的参数

参数	描述	类型	默认值
IMAGE	S2I 进程构建的容器镜像的完全限定名称。	**string**	

参数	描述	类型	默认值
IMAGE_SCRIPTS_URL	包含构建器镜像的默认 assemble 和 run 脚本的 URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	构建过程要设置的环境变量数组，以 **KEY=VALUE** 格式列出。	**数组**	
CONTEXT	**源** 工作区中的目录的路径，以用作上下文。	**string**	**.**
STORAGE_DRIVER	设置 Buildah 存储驱动程序，以反映当前集群节点设置的设置。	**string**	**vfs**
格式	要构建的容器的格式，可以是 **oci** 或 **docker**。	**string**	**oci**
BUILD_EXTRA_ARGS	构建镜像时，**build** 命令的额外参数。	**string**	
PUSH_EXTRA_ARGS	推送镜像时，**push** 命令的额外参数。	**string**	
SKIP_PUSH	跳过将镜像推送到容器 registry。	**string**	**false**
TLS_VERIFY	TLS 验证标志，通常为 **true**。	**string**	**true**
详细	打开详细日志记录 ; 执行的所有命令都添加到日志中。	**string**	**false**
VERSION	镜像流的标签，对应于语言版本。	**string**	**latest**

表 3.42. s2i-ruby 任务支持的工作区

Workspace	描述
source	应用源代码，即 S2I 工作流的构建上下文。
dockerconfig	一个可选的工作区，用于提供 Buildah 用于访问容器 registry 的 **.docker/config.json** 文件。将文件放在工作区的根目录中，名称为 **config.json** 或 **.dockerconfigjson**。

表 3.43. s2i-ruby 任务返回的结果

结果	类型	描述
IMAGE_URL	**string**	构建的镜像的完全限定名称。

结果	类型	描述
IMAGE_DIGEST	**string**	构建的镜像摘要。

s2i-ruby ClusterTask的更改

- 添加了几个新参数。
- **BASE_IMAGE** 参数已被删除。
- 参数名称 **PATH_CONTEXT** 被改为 **CONTEXT**。
- 参数名称 **TLS_VERIFY** 被更改为 **TLSVERIFY**。
- 添加 **IMAGE_URL** 结果。

skopeo-copy

skopeo-copy 任务执行 **skopeo copy** 命令。

Skopeo 是一个用于远程容器镜像 registry 的命令行工具，不需要守护进程或其他基础架构来加载和运行镜像。**skopeo copy** 命令将镜像从一个远程 registry 复制到另一个远程 registry，例如，从内部注册表复制到生产环境 registry。Skopeo 支持使用您提供的凭据在镜像 registry 上授权。

skopeo-copy 任务的用法示例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-deploy-image
spec:
# ...
  tasks:
  - name: copy-image
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: skopeo-copy
      - name: namespace
        value: openshift-pipelines
    params:
    - name: SOURCE_IMAGE_URL
      value: "docker://internal.registry/myimage:latest"
    - name: DESTINATION_IMAGE_URL
      value: "docker://production.registry/myimage:v1.0"
    workspaces:
    - name: output
      workspace: shared-workspace
```

表 3.44. skopeo-copy 任务支持的参数

参数	描述	类型	默认值
SOURCE_IMAGE_URL	源容器镜像的完全限定名称，包括标签。	**string**	
DESTINATION_IMAGE_URL	Skopeo 将源镜像复制到的目标镜像的完全限定名称，包括标签。	**string**	
SRC_TLS_VERIFY	源 registry 的 TLS 验证标志，通常为 **true**。	**string**	**true**
DEST_TLS_VERIFY	目标 registry 的 TLS 验证标志，通常为 **true**	**string**	**true**
详细	将调试信息输出到日志。	**string**	**false**

表 3.45. skopeo-copy 任务支持的工作区

Workspace	描述
images_url	如果要复制多个镜像，请使用此工作区提供镜像 URL。

表 3.46. skopeo-copy 任务返回的结果

结果	类型	描述
SOURCE_DIGEST	**string**	源镜像的 SHA256 摘要。
DESTINATION_DIGEST	**string**	目标镜像的 SHA256 摘要。

skopeo-copy ClusterTask的更改

- 所有参数名称都改为大写。
- 添加了 **VERBOSE** 参数。
- 工作区名称从 **images-url** 改为 **images_url**。
- 添加了 **SOURCE_DIGEST** 和 **DESTINATION_DIGEST** 结果。

tkn
tkn 任务使用 tkn 对 Tekton 资源执行操作。

tkn 任务的用法示例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
```

```
 name: tkn-run
spec:
 pipelineSpec:
   tasks:
   - name: tkn-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: tkn
       - name: namespace
         value: openshift-pipelines
     params:
     - name: ARGS
```

表 3.47. tkn 任务支持的参数

参数	描述	类型	默认值
SCRIPT	要执行的 **tkn** CLI 脚本。	**string**	**tkn $@**
ARGS	要运行的 **tkn** CLI 参数。	**数组**	**- --help**

表 3.48. tkn 任务支持的工作区

Workspace	描述
kubeconfig_dir	一个可选工作区，您可以在其中提供 **.kube/config** 文件，其中包含用于访问集群的凭证。将此文件放在工作区的根目录中，并将其命名为 **kubeconfig**。

从 tkn ClusterTask的更改

- **TKN_IMAGE** 参数已被删除。
- 工作区名称已从 **kubeconfig** 改为 **kubeconfig_dir**。

3.6. 其他资源

- 在 OpenShift Pipelines 中使用 Tekton Hub

第 4 章 在 OPENSHIFT PIPELINES 中使用手动批准

您可以在管道中指定手动批准任务。当管道到达此任务时，它会从一个或多个 OpenShift Container Platform 用户暂停和等待批准。如果有任何用户选择拒绝该任务，而不是批准该任务，管道会失败。手动批准的控制器提供此功能。

重要

手动批准是技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

4.1. 启用手动批准控制器

要使用手动批准任务，您必须首先启用手动批准控制器。

先决条件

- 已在集群中安装了 Red Hat OpenShift Pipelines Operator。
- 您可以使用 **oc** 命令行工具登录到集群。
- 具有 **openshift-pipelines** 命名空间的管理员权限。

流程

1. 为 **ManualApprovalGate** 自定义资源(CR)创建名为 **manual-approval-gate-cr.yaml** 的文件：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: ManualApprovalGate
metadata:
  name: manual-approval-gate
spec:
  targetNamespace: openshift-pipelines
```

2. 输入以下命令应用 **ManualApprovalGate** CR：

```
$ oc apply -f manual-approval-gate-cr.yaml
```

3. 输入以下命令验证手动批准控制器是否正在运行：

```
$ oc get manualapprovalgates.operator.tekton.dev
```

输出示例

```
NAME                  VERSION    READY   REASON
manual-approval-gate   v0.1.0     True
```

确保 **READY** 状态为 **True**。如果不是 **True**，请等待几分钟，然后再次输入命令。控制器可能需要一些时间才能进入就绪状态。

4.2. 指定手动批准任务

您可以在管道中指定手动批准任务。当管道运行的执行达到此任务时，管道运行会停止，并从一个或多个用户等待批准。

先决条件

- 您已启用手动批准控制器。
- 已创建管道的 YAML 规格。

流程

- 在管道中指定一个 **ApprovalTask**，如下例所示：

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: example-manual-approval-pipeline
spec:
  tasks:
# ...
  - name: example-manual-approval-task
    taskRef:
      apiVersion: openshift-pipelines.org/v1alpha1
      kind: ApprovalTask
    params:
    - name: approvers
      value:
      - user1
      - user2
      - user3
    - name: description
      value: Example manual approval task - please approve or reject
    - name: numberOfApprovalsRequired
      value: '2'
    - name: timeout
      value: '60m'
# ...
```

表 4.1. 手动批准任务的参数

参数	类型	描述
approvers	数组	可以批准任务的 OpenShift Container Platform 用户。
description	string	可选：批准任务的描述。OpenShift Pipelines 为可以批准或拒绝任务的用户显示描述。

参数	类型	描述
numberOfApprovalsRequired	string	任务所需的不同用户的批准数量。
timeout	string	可选：批准的超时时间。如果任务在这个期间没有收到配置的批准数量，则管道运行会失败。默认超时为 1 小时。

4.3. 批准手动批准任务

当您运行包含批准任务的管道且执行达到批准任务时，管道运行会暂停并等待用户批准或拒绝。

用户可以使用 Web 控制台或 **opc** 命令行工具批准或拒绝任务。

如果在任务中配置任何批准者会拒绝该任务，管道运行会失败。

如果一个用户批准了该任务，但配置的批准数量仍未达到，则相同的用户可以更改为拒绝任务，管道运行会失败

4.3.1. 使用 Web 控制台批准手动批准任务

您可以使用 OpenShift Container Platform Web 控制台批准或拒绝手动批准任务。

如果您在手动批准任务中被列为 approver，且管道运行达到此任务，Web 控制台会显示通知。您可以查看需要批准并批准或拒绝这些任务的任务列表。

先决条件

- 启用了 OpenShift Pipelines 控制台插件。

流程

1. 通过完成以下任一操作来查看您可以批准的任务列表：
   - 当显示有关需要批准的任务的通知时，点此通知中的 **Go to Approvals 选项卡**。
   - 在 **Administrator** 视角菜单中，选择 **Pipelines → Pipelines**，然后点 **Approvals** 选项卡。
   - 在 **Developer** 视角菜单中，选择 **Pipelines**，然后点 **Approvals** 选项卡。
   - 在 **PipelineRun details** 窗口中，在 **Details** 选项卡中点代表手动批准任务的 rectangle。该列表仅显示此任务的批准。
   - 在 **PipelineRun details** 窗口中，点 **ApprovalTasks** 选项卡。该列表仅显示此管道运行的批准。
2. 在批准任务列表中，在代表您要批准的任务行中，点  图标，然后选择以下选项之一：
   - 要批准任务，请选择 **Approve**。

- 要拒绝该任务，请选择 **Reject**。

3. 在 **Reason** 字段中输入消息。
4. 点 **Submit**。

其他资源

- 启用 OpenShift Pipelines 控制台插件

4.3.2. 使用命令行批准手动批准任务

您可以使用 **opc** 命令行工具批准或拒绝手动批准任务。您可以查看您作为批准者的任务列表，并批准或拒绝待批准的任务。

先决条件

- 您已下载并安装 **opc** 命令行工具。这个工具包括在与 **tkn** 命令行工具相同的软件包中。
- 您可以使用 **oc** 命令行工具登录到集群。

流程

1. 输入以下命令来查看您列为批准人的手动批准任务列表：

```
$ opc approvaltask list
```

输出示例

```
NAME                                    NumberOfApprovalsRequired   PendingApprovals   Rejected
STATUS
manual-approval-pipeline-01w6e1-task-2   2                           0                  0        Approved
manual-approval-pipeline-6ywv82-task-2   2                           2                  0        Rejected
manual-approval-pipeline-90gyki-task-2   2                           2                  0        Pending
manual-approval-pipeline-jyrkb3-task-2   2                           1                  1        Rejected
```

2. 可选：要查看有关手动批准任务的信息，包括其名称、命名空间、管道运行名称、批准者列表和当前状态，请输入以下命令：

```
$ opc approvaltask describe <approval_task_name>
```

3. 根据需要批准或拒绝手动批准任务：
   - 要批准手动批准任务，请输入以下命令：

```
$ opc approvaltask approve <approval_task_name>
```

另外，您可以使用 the **-m** 参数为批准指定信息：

```
$ opc approvaltask approve <approval_task_name> -m <message>
```

   - 要拒绝手动批准任务，请输入以下命令：

```
$ opc approvaltask reject <approval_task_name>
```

另外，您可以使用 the **-m** 参数为拒绝指定信息：

```
$ opc approvaltask reject <approval_task_name> -m <message>
```

其他资源

- 安装 tkn

第 5 章 在管道中使用红帽权利

如果您有 Red Hat Enterprise Linux (RHEL)权利，您可以使用这些权利在管道中构建容器镜像。

在从 Simple Common Access (SCA)导入这个 operator 后，paption Operator 会自动管理权利。此 operator 在 **openshift-config-managed** 命名空间中提供名为 **etc-pki-entitlement** 的 secret。

您可以通过以下两种方式之一在管道中使用红帽权利：

- 手动将 secret 复制到管道的命名空间。如果您有有限的管道命名空间，则此方法会最复杂。
- 使用 Shared Resources Container Storage Interface (CSI) Driver Operator 在命名空间间自动共享 secret。

5.1. 先决条件

- 使用 **oc** 命令行工具登录到 OpenShift Container Platform 集群。
- 您可以在 OpenShift Container Platform 集群上启用 Insights Operator 功能。如果要使用 Shared Resources CSI Driver Operator 在命名空间间共享 secret，还必须启用 Shared Resources CSI 驱动程序。有关启用功能的详情，包括 Insights Operator 和共享资源 CSI 驱动程序，请参阅 使用功能门启用功能。

注意

启用 Insights Operator 后，您必须等待一些时间以确保集群使用这个 Operator 更新所有节点。您可以输入以下命令来监控所有节点的状态：

```
$ oc get nodes -w
```

要验证 Insights Operator 是否活跃，请输入以下命令检查 **insights-operator** pod 是否在 **openshift-insights** 命名空间中运行：

```
$ oc get pods -n openshift-insights
```

- 您已配置了将红帽权利导入到 Insights Operator 中。有关导入权利的详情，请参考使用 Insights Operator 导入简单内容访问权利。

注意

要验证 Insights Operator 是否使您的权利可用，并通过运行以下命令检查 **openshift-config-managed** 命名空间中是否存在 **etc-pki-entitlement** secret：

```
$ oc get secret etc-pki-entitlement -n openshift-config-managed
```

5.2. 通过手动复制 ETC-PKI-ENTITLEMENT SECRET 来使用红帽权利

您可以将 **etc-pki-entitlement** secret 从 **openshift-config-managed** 命名空间复制到管道的命名空间中。然后，您可以将管道配置为使用此 secret 用于 Buildah 任务。

先决条件

- 在您的系统上安装了 **jq** 软件包。这个软件包包括在 Red Hat Enterprise Linux (RHEL)中。

流程

1. 运行以下命令，将 **etc-pki-entitlement** secret 从 **openshift-config-managed** 命名空间复制到管道的命名空间中：

```
$ oc get secret etc-pki-entitlement -n openshift-config-managed -o json | \
  jq 'del(.metadata.resourceVersion)' | jq 'del(.metadata.creationTimestamp)' | \
  jq 'del(.metadata.uid)' | jq 'del(.metadata.namespace)' | \
  oc -n <pipeline_namespace> create -f - 1
```

1 将 **<pipeline_namespace** > 替换为管道的命名空间。

2. 在 Buildah 任务定义中，使用 **openshift-pipelines** 命名空间中提供的 **buildah** 任务或此任务的副本，并定义 **rhel-entitlement** 工作区，如下例所示。

3. 在运行 Buildah 任务的任务运行或管道运行中，将 **etc-pki-entitlement** secret 分配给 **rhel-entitlement** 工作区，如下例所示。

使用红帽权利的管道运行定义示例，包括管道和任务定义

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: buildah-pr-test
spec:
  workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
    - name: dockerconfig
      secret:
        secretName: regred
    - name: rhel-entitlement 1
      secret:
        secretName: etc-pki-entitlement
  pipelineSpec:
    workspaces:
      - name: shared-workspace
      - name: dockerconfig
      - name: rhel-entitlement 2
    tasks:
# ...
      - name: buildah
        taskRef:
          resolver: cluster
          params:
          - name: kind
```

```
    value: task
  - name: name
    value: buildah
  - name: namespace
    value: openshift-pipelines
workspaces:
- name: source
  workspace: shared-workspace
- name: dockerconfig
  workspace: dockerconfig
- name: rhel-entitlement 3
  workspace: rhel-entitlement
params:
- name: IMAGE
  value: <image_where_you_want_to_push>
```

1. 管道运行中的 **rhel-entitlement** 工作区的定义，将 **etc-pki-entitlement** secret 分配给工作区
2. 管道定义中 **rhel-entitlement** 工作区的定义
3. 任务定义中 **rhel-entitlement** 工作区的定义

5.3. 通过使用 SHARED RESOURCES CSI 驱动程序 OPERATOR 共享 SECRET 来使用红帽权利

您可以使用 Shared Resources Container Storage Interface (CSI) Driver Operator 将 **etc-pki-entitlement** secret 共享从 **openshift-config-managed** 命名空间到其他命名空间。然后，您可以将管道配置为使用此 secret 用于 Buildah 任务。

先决条件

- 您可以使用具有集群管理员权限的 **oc** 命令行工具登录到 OpenShift Container Platform 集群。
- 您可以在 OpenShift Container Platform 集群上启用 Shared Resources CSI Driver Operator。

流程

1. 运行以下命令，创建一个 **SharedSecret** 自定义资源(CR)来共享 **etc-pki-entitlement** secret：

```
$ oc apply -f - <<EOF
apiVersion: sharedresource.openshift.io/v1alpha1
kind: SharedSecret
metadata:
  name: shared-rhel-entitlement
spec:
  secretRef:
    name: etc-pki-entitlement
    namespace: openshift-config-managed
EOF
```

2. 运行以下命令，创建允许访问共享 secret 的 RBAC 角色：

```
$ oc apply -f - <<EOF
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-rhel-entitlement
  namespace: <pipeline_namespace> 1
rules:
  - apiGroups:
      - sharedresource.openshift.io
    resources:
      - sharedsecrets
    resourceNames:
      - shared-rhel-entitlement
    verbs:
      - use
EOF
```

1 将 **<pipeline_namespace>** 替换为管道的命名空间。

3. 运行以下命令，将角色分配给 **pipeline** 服务帐户：

```
$ oc create rolebinding shared-resource-rhel-entitlement --role=shared-shared-resource-rhel-entitlement \
  --serviceaccount=<pipeline-namespace>:pipeline 1
```

1 将 **<pipeline-namespace>** 替换为管道的命名空间。

注意

如果您更改了 OpenShift Pipelines 的默认服务帐户，或者在管道运行或任务运行中定义自定义服务帐户，请将角色分配给这个帐户，而不是 **管道** 帐户。

4. 在 Buildah 任务定义中，使用 **openshift-pipelines** 命名空间中提供的 **buildah** 任务或此任务的副本，并定义 **rhel-entitlement** 工作区，如下例所示。

5. 在运行 Buildah 任务的任务运行或管道运行中，将共享 secret 分配给 **rhel-entitlement** 工作区，如下例所示。

使用红帽权利的管道运行定义示例，包括管道和任务定义

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: buildah-pr-test-csi
spec:
  workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
        spec:
          accessModes:
          - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
```

```
    - name: dockerconfig
      secret:
        secretName: regred
    - name: rhel-entitlement 1
      csi:
        readOnly: true
        driver: csi.sharedresource.openshift.io
        volumeAttributes:
          sharedSecret: shared-rhel-entitlement
  pipelineSpec:
    workspaces:
    - name: shared-workspace
    - name: dockerconfig
    - name: rhel-entitlement 2
    tasks:
# ...
    - name: buildah
      taskRef:
        resolver: cluster
        params:
        - name: kind
          value: task
        - name: name
          value: buildah
        - name: namespace
          value: openshift-pipelines
      workspaces:
      - name: source
        workspace: shared-workspace
      - name: dockerconfig
        workspace: dockerconfig
      - name: rhel-entitlement 3
        workspace: rhel-entitlement
      params:
      - name: IMAGE
        value: <image_where_you_want_to_push>
```

1. 管道运行中的 **rhel-entitlement** 工作区的定义，将 **shared-rhel-entitlement** CSI 共享 secret 分配给工作区
2. 管道定义中 **rhel-entitlement** 工作区的定义
3. 任务定义中 **rhel-entitlement** 工作区的定义

5.4. 其他资源

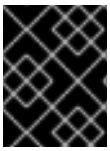
- 简单内容访问
- 使用 Insights Operator
- 使用 Insights Operator 导入简单的内容访问权利
- 共享资源 CSI Driver Operator

- 更改 OpenShift Pipelines 的默认服务帐户

第 6 章 管理未指定版本的和版本化的集群任务

作为集群管理员，安装 Red Hat OpenShift Pipelines Operator 会为每个默认集群任务创建变体，称为 *版本化的集群任务* (VCT) 和 *非版本的集群任务* (NVCT)。例如，安装 Red Hat OpenShift Pipelines Operator v1.7 创建一个 **buildah-1-7-0** VCT 和 **buildah** NVCT。

NVCT 和 VCT 具有相同的元数据、行为和规格，包括 **params**、**workspaces** 和 **steps**。但是，当禁用 Operator 或升级 Operator 时，它们的行为会有所不同。



重要

在 Red Hat OpenShift Pipelines 1.10 中，**ClusterTask** 功能已弃用，计划在以后的发行版本中删除。

6.1. 非版本和版本的集群任务之间的区别

非版本的和版本化的集群任务有不同的命名约定。另外，Red Hat OpenShift Pipelines Operator 会不同地升级它们。

表 6.1. 非版本和版本的集群任务之间的区别

	非版本的集群任务	版本的集群任务
Nomenclature	NVCT 仅包含集群任务的名称。例如，随 Operator v1.7 安装的 Buildah NVCT 的名称是 **buildah**。	VCT 包含集群任务的名称，后接为后缀的版本。例如，由 Operator v1.7 安装的 Buildah 的 VCT 的名称是 **buildah-1-7-0**。
Upgrade（升级）	升级 Operator 时，它会使用最新的更改更新非版本的集群任务。NVCT 的名称保持不变。	升级 Operator 会安装 VCT 的最新版本并保留更早的版本。VCT 的最新版本对应于升级的 Operator。例如，安装 Operator 1.7 安装 **buildah-1-7-0** 并保留 **buildah-1-6-0**。

6.2. 非版本和版本的集群任务的优点和缺陷

在将未指定版本或版本化的集群任务用作生产环境中的标准之前，集群管理员可能会考虑它们的优点和缺点。

表 6.2. 非版本和版本的集群任务的优点和缺陷

集群任务	优点	缺点

集群任务	优点	缺点
非版本的集群任务 (NVCT)	• 如果您希望使用最新更新和程序错误修复部署管道，请使用 NVCT。 • 升级 Operator 会升级非版本的集群任务，这会消耗比多个版本的集群任务更少的资源。	如果您部署使用 NVCT 的管道，如果自动升级的集群任务不向后兼容，则它们可能会在 Operator 升级后中断。
版本化的集群任务 (VCT)	• 如果您的目的是在生产环境中提供稳定的管道，请使用 VCT。 • 之前的版本会保留在集群中，即使安装了集群任务的更新的版本。您可以继续使用较早的集群任务。	• 如果您继续使用集群任务的早期版本，您可能会错过最新的功能和关键安全更新。 • 早期版本的集群任务，它们不正常消耗集群资源。 • * 升级后，Operator 无法管理更早的 VCT。您可以使用 **oc delete clustertask** 命令手动删除更早的 VCT，但您无法恢复它。

6.3. 禁用未指定版本和版本的集群任务

作为集群管理员，您可以禁用安装 OpenShift Pipelines Operator 的集群任务。

流程

1. 要删除所有非版本的集群任务和最新版本的集群任务，请编辑 **TektonConfig** 自定义资源定义 (CRD)并将 **spec.addon.params** 中的 **clusterTasks** 参数设置为 **false**。

   TektonConfig CR 示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
  - name: createRbacResource
    value: "false"
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
    - name: clusterTasks
      value: "false"
...
```

当您禁用集群任务时，Operator 会删除所有非版本的集群任务，且只从集群中删除最新版本的集群任务。

注意

重新启用集群任务将安装非版本的集群任务。

2. 可选：要删除早期版本的集群任务，请使用以下方法之一：

   a. 要删除个别较早版本的集群任务，请使用 **oc delete clustertask** 命令，后面是版本化的集群任务名称。例如：

   ```
   $ oc delete clustertask buildah-1-6-0
   ```

   b. 要删除由旧版本 Operator 创建的所有版本集群任务，您可以删除对应的安装程序设置。例如：

   ```
   $ oc delete tektoninstallerset versioned-clustertask-1-6-k98as
   ```

   小心

   如果您删除旧版本的集群任务，则无法恢复它。您只能恢复当前创建的 Operator 版本和未版本的集群任务。